ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Universidad Politécnica de Cartagena

Telecommunications Engineering – Bachelor's degree final thesis

Department of Electrical, Electronics and Information Engineering "Guglielmo Marconi" (DEI), NetLAB (Network Laboratory)
University of Bologna, Italy.

STUDY OF 5G NETWORKS BY THE USE OF A SIMULATOR BASED ON A 3-LEVEL QoS SCHEDULING MODEL IN HYBRID OPTICAL SWITCHES

Author:

Nerea Cilleruelo Méndez

Thesis advisors:

Carla Raffaelli
Federico Tonini
Rafael Toledo Moreo

Cartagena, February 25

"Tutte le verità sono facile da capire una volta che sono state rivelate.

Il difficile è scoprirle."

~Galileo Galilei

Acknowledgments

I would like to thank all the people who supported me in writing this thesis.

First of all, I would like to express my gratitude to my advisor, Carla Raffaelli, and co-advisor, Federico Tonini: thank you so much for this opportunity and for your patience with my English.

I also would like to thank my friends, new and old friends, for all your support, especially to say thanks to my friend Pablo Lucas for sharing the Erasmus experience at the same time that me and make the process easier.

Last, but for sure not least, I would like to thank my family. Mum, dad and sister: you always support me in my choices and encourage me.

During this time in Italy working on this thesis I have learned a lot so I thank all the people who encouraged me to having this Erasmus experience.

Contents

List of Tables

List of Figures

LIST OF ACRONYMS

1G	First generation of mobile telecommunications technology
2G	Second generation of mobile telecommunications technology
3G	Third generation of mobile telecommunications technology
4G	Fourth generation of mobile telecommunications technology
5G	Fifth generation of mobile telecommunications technology
AON	Active Optical Network
AWG	Arrayed Wavelength Grating
BBU	Base Band Unit
BE	Best Effort
BER	Bit Error Ratio
BS	Base Station
C-RAN	Cloud/Centralized Radio Access Network
CPRI	Common Public Radio Interface
CoE	CPRI over Ethernet
CWDM	Coarse Wavelength Division Multiplexing
D-RoF	Digital Radio over fiber (CPRI/OBSAI)
D-RAN	Distributed RAN
DWDM	Dense Wavelength Division Multiplexing
E2E	End to End
F-RAN	Fog RAN
FTTH	Fiber To The Home
GE	Gigabit Ethernet
GSM	Global System for Mobile communications
GST	Guaranteed Service Transport
H-RAN	Heterogeneous RAN
HetNets	Heterogeneous Networks
IHON	Integrated Hybrid Optical Network
LAN	Local Area Network
LTE	Long Term Evolution
MIMO	Multiple Input Multiple Output
NG-PON2	Next Generation – PON2
OADM	Optical Add-Drop Multiplexer

OBSAI	Open Base Station Architecture Initiative
ODFs	Optical Distribution Frames
OTN	Optical Transport Network
PDV	Packet Delay Variation
PLR	Packet Loss Ratio
PONs	Passive Optical Networks
QoS	Quality of Service
RAN	Radio Access Network
RAT	Radio Access Technology
RE	Radio Equipment
REC	Radio Equipment Control
RF	Radio Frequency
RoE	Radio over Ethernet
RRH or RRU	Remote Radio Head
RT	Real Time
RTT	Round-Trip Time or Round-Trip Delay Time
RU	Radio Unit
SM	Statistical Multiplexing
TD-SCDMA	Time Division Synchronous Code Division Multiple Access
TDD	Time Division Duplex
TWDM	Time Wavelength Division Multiplexing PON
Traditional RAN	Traditional Radio Access Network
UD-WDM-PON	Ultra-Dense Wavelength Division Multiplexing PON
UMTS	Universal Mobile Telecommunications System
WiMAX	Worldwide Interoperability for Microwave Access
WDM	Wavelength Division Multiplexing

Summary

The development of mobile telecommunication is a constant process and a challenge to enhancing mobile network quality of service. There is a high demand to add new features and improve higher data rates, greater reliability, energy efficiency and security as well as cost reduction. Beginning with First Generation (1G) networks to the promising Fifth Generation (5G) which could be implemented by 2020.

The evolution in the traffic on the Internet has greatly increased in recent years. A large number of applications and devices have appeared demanding different requirements to the current system. Prevalent 4G generation (LTE) systems development will continue with new 5G generation radio technology, which will provide us to access to any kind of information anywhere and anytime with very low latency and jitter. Due to the capacity limitations found in present communication networks, a new architecture which could bring us benefits to get much more information appears as an alternative: Centralized Radio Access Network Architecture (C-RAN). And with it, the fronthaul network and the Common Public Radio Interface (CPRI) solution.

This thesis will present the application of some 5G fronthaul concepts to a Hybrid Integrated Optical Network (IHON) event-driven simulator programmed in C. Concepts as Common Public Radio Interface over Ethernet (CPRIoE), backhaul and fronthaul traffic or 3-Level Quality of Service (QoS) scheduling for optical aggregation in data centers are studied. The key feature concepts shall be examined and their impacts on the efficiency network shall be analyzed. One of the biggest challenges for this is to get the highest traffic success rate in the network when it is working with fronthaul and backhaul functionalities in the same infrastructure.

Chapter 1

Introduction

The future 5G generation radio technology will result in a huge potential. This technology will allow services that require high bandwidth in mobility and will promote powerful applications such as IoT (Internet of Things), connected vehicles, virtual reality, gaming, intelligent transport, ultimate video quality, medical operations without the presence of the surgeon (telemedicine) or, in short, any application in real time.

1.1. Fifth Generation (5G) technology: the next generation in mobile connectivity

"5G will need to deliver massive capacity, because the current support of 5 billion users through mobile network systems will have to expand to support billions of applications and hundreds of billions of machines as well" [1]. In this way, the new system will provide a service delivery of 10 GB/second over the air, enabling up to 100 times faster data rates, 1000 times higher capacity and energy consumption.

The evolution in cellular networks has evolved as a result of the growing demand for new features, beginning with the First Generation (1G) cellular networks to Fifth Generation (5G). The 1G provided the basic mobile voice service based on analog radio transmission techniques, with no security and poor-quality communications. It employed Frequency Division Multiple Access (FDMA) to multiplex traffic flows. The 2G marks the transition from analogue to digital, which improved the handling of calls, more connections could be made at the same time in

the same bandwidth and integrate other additional services such as the short message service. It employed either time (Time Division Multiple Access, TDMA) or code (Code Division Multiple Access, CDMA) multiplexing. The technology called Global System for Mobile Communication (GSM) was the first to facilitate digital voice and data as well as roaming. It was deployed to provide a single standard and enable services throughout Europe, allowing the customer to go from one place to another. GSM uses TDMA technology and it has been in development, indeed, new technologies based on it leads to advanced systems. Later, in 3G systems, a distributed Base Station (BS) was introduced and the Universal Mobile Telecommunications System (UMTS) standard supported the increase in data rates, facilitated growth and greater voice and data capacity. Then, the data is sent through the technology called Packet Switching and the voice calls are transmitted by circuit switching. The present 4G is based entirely on IP. The main objective of 4G technology is to provide high speed, capacity and quality as well as security and low cost services for voice and data services, multimedia and internet over IP. Long Term Evolution (LTE) is the technology developed and it can be used by wireless modems, smart phones and other devices. To provide wireless services anytime and anywhere, terminal mobility is a key factor in 4G.

The main difference from preceding generations is the fact that the new 5G radio technology should provide reliability, very high capacity and speed, imperceptible latency and low power, but it also requires devices with a certain quality to deliver audiovisual content with the high quality these devices enable, e.g., the new generation of TV devices is 4k-enabled.

On the one hand, C-RAN (Cloud/Centralized Radio Access Network) is one of the popular network architectures, which is being studied to get the mentioned features on the network. Furthermore, optical switching technologies could be a good option for data center interconnection networks. In C-RAN design, the radio network segment, also called fronthaul, is under study. However, fronthaul network includes some tight technical requirements which are necessary to perform. The Base Station (BS) also changes depending on the type of network, but it is studied in the following chapters

On the other hand, Common Public Radio Interface (CPRI) can be mention as the protocol to encapsulate data in fronthaul segment.

In the following chapters these concepts are explained in detail.

1.2. Objectives of the Thesis

The work presented in this thesis focuses in the utilization of the model and simulator developed in [2] to increase network efficiency when traditional packet based backhaul traffic and CPRIoE are transported over the same link, that is to say,

when it is working with fronthaul and backhaul funcionalities in the same infrastructure.

An analysys of these techniques is proposed by simulation and the numerical results are evaluated.

The 3-level priority model supports packet-based integrated services and, moreover, hibrid optical switches are implemented in order to lead the network traffic. Fundamentally, the target is to study the transport of radio signals over Ethernet signals using hybrid optical switches; in technical terms, it means to save resources filling the GAP with backhaul packets and trying to get the highest fronthaul traffic success rate in the network. And, to conclude, at the end of the thesis, the packets are divided to get even better succesful transmision of information.

The objectives of this work are to perform and analyze the following points:

- Know about the evolution of mobile networks, the advantages of using C-RAN architecture and fronthaul/backhaul segments.

- Collect the fronthaul requirements and solutions.

- Identify the CPRI standard and study the benefits that bring into CPRI over Ethernet.

- Make the IHON network out. Study how Integrated Hybrid technology in Ethernet switches works and the 3-LEVEL priority Quality of Service (QoS) traffic model.

- Examine the traffic on the network by the use of a simulator in which some features are added to get the highest fronthaul traffic success rate in the network and evaluate the results.

1.3. Thesis Outline

The thesis is organized as follows:

Chapter 2. RAN Evolution: Cloud RAN architecture for 5G networks.
This introductory chapter describes the evolution from Traditional Radio Access Network (RAN) architecture to C-RAN, emphasizing on the C-RAN advantages to use it as a future option for 5G and some traffic features.

Chapter 3: Fronthaul network options in C-RAN.
The requirements and solutions of fronthaul networks are presented here focusing on the key performance elements: capacity, latency, synchronization and jitter. The transport network technologies between the Base Band Unit (BBU) Pool and the Remote Radio Head (RRH) are explained.

Chapter 4: Reference network model.
In this chapter, the scenario based on an Integrated Hybrid Optical Network (IHON) is explained and the CPRI encapsulation over Ethernet is described. The CPRIoE parameters and equations are explained in order to understand how the specification works and to relate this notation with the variables used in the simulator.

Chapter 5: Simulation environment and model.
The event-based simulator is explained in detail as well as the queuing model representing the output interface of the hybrid switch. An introductory example of how the simulation model works is presented and some initial graphs are shown.
Furthermore, in this chapter, the characterization of the simulator is carried out; several functionalities are implemented in order to evaluate the network traffic.

Chapter 6: Numerical results and graphs.
This chapter presents the results obtained from the simulator. Graphs, comparisons and tables of the performance of the network are shown to evaluate the changes introduced.

Chapter 7: Conclusions and future work.
This final part of the thesis wraps up and summarizes the conclusions. Some suggestions or future lines of study are presented for further investigation that weren´t covered in this work.

Chapter 2

Radio Access Network Evolution: Cloud RAN architecture for 5G networks

Mobile communications are a growing sector. In view of the high volume of mobile data transmission, operators have to increase network capacity. This high volume may also result in a substantial number of novel mobile network architectures, e.g. C-RAN, D-RAN, H-RAN or F-RAN.

Thus, to satisfy user demands, that is to increase network capacity, there are possible future options such as mentioned C-RAN, small cells or by implementing techniques as MIMO (Multiple Input Multiple Output) or Massive MIMO. However, C-RAN is easier to upgrade and repair than the other candidates [3]. This thesis will focus on C-RAN, but also can be used these other architectures.

Moreover, saving costs and energy consumption become a necessity to optimize the network and with C-RAN it can be possible.

Let us start by looking at the traditional architecture and its limitations.

2.1. Traditional Architecture (RAN)

In this architecture, Base Stations had the equipment in the base of the antenna tower. The BTS incorporated these functionalities: radio and baseband processing. It implies that each BTS had an all-in-one architecture, an architecture where power, analog and digital functions are housed in a single unit. They were equipped with its own cooling system, battery, monitoring system, and so on.

Traditional Base Stations (BS) contains:

- Base Band Unit (BBU): digital signal processing.
- Radio Unit (RU): contains the Radio Frequency (RF) transmit and receive components. It is connected to the antenna by coax feeder.

Each BTS processes and transmits its own signal to and from the user. As seen from Figure 1, extracted from [4], BBU and RU are contained in the same unit. Both elements are co-located in one container.

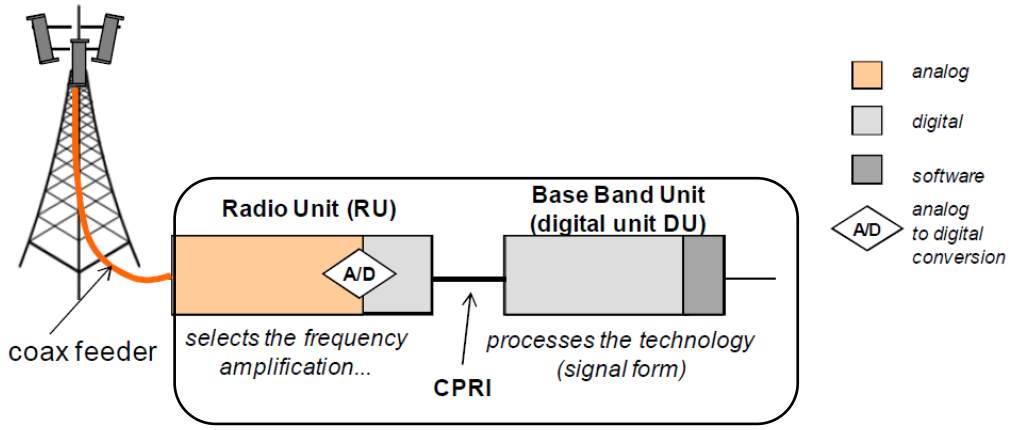


Fig. 1: Traditional Base Station RU and BBU.

The antenna is located a few meters of the radio module by coaxial cables, which provide high losses.

Additionally, CPRI is the protocol to transport data between RU and BBU.

This architecture was useful for 1G and 2G mobile networks but now is obsolete and wastes resources. The new approach is moving towards divided functionalities.

2.2. Centralized Radio Access Network Architecture (C-RAN)

C-RAN is a novel mobile network architecture for cellular networks where the letter C means: Cloud, Centralized processing, Cooperative radio, Collaborative or Clean. It was proposed by China Mobile Communications Corporation (CMCC) in 2011.

This architecture appears as solution to the underutilized RAN to solve capacity problems and separates the functionalities. It is based on distributed Base Stations.

2.2.1. System Architecture

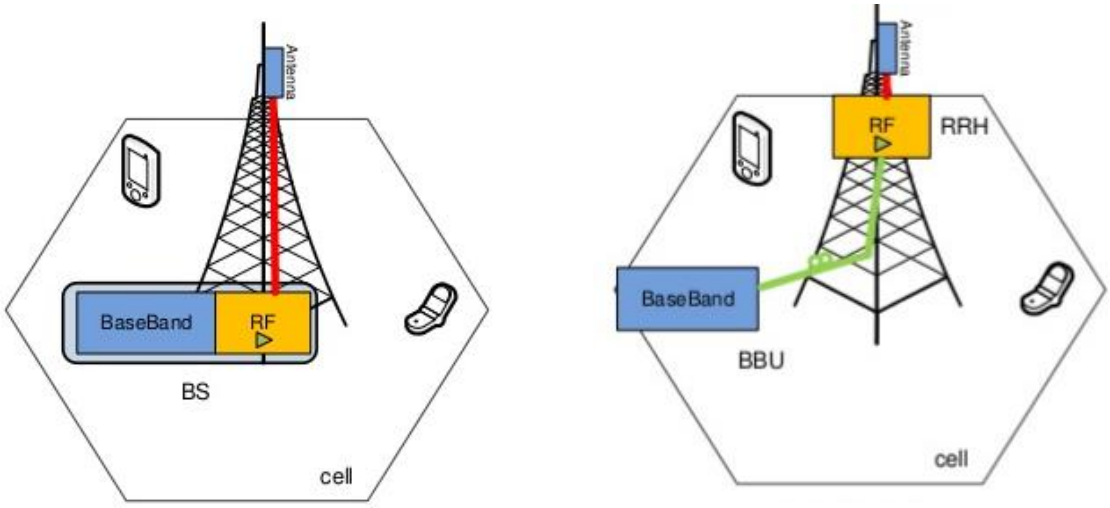
On the one hand, now the RU is called Remote Radio Head (RRH) or Remote Radio Unit (RRU) and it is located near to the antenna, on top of the tower or mast [3]. And on the other hand, the Digital Unit is called BBU, where takes part the baseband signal processing and it is separated from RRH.

The BBU transforms the digital signal into an analog signal ready to be transmitted by the wireless network. There is one BBU per radio access technology [4].

The RRH transmits the RF signal to the user by the air.

Furthermore, the aim of this architecture is to pool all the BBUs from the different Base Station into one unit, a centralized BBU Pool [3]. In this way, C-RAN network is able to adapt to non-uniform traffic and utilizes the resources. A BBU Pool is shared between adjacent cells.

A comparison between traditional RAN Base Station and C-RAN Base Station with RRH is shown in Figure 2, extracted from [5].



a) Traditional Base Station. b) Base Station with RRH.

Fig. 2: Base Station Evolution.

A simplified C-RAN network architecture is presented in the following Figure 3, extracted from [5], where the BBU Pool contains the BBUs from a certain number of base stations and the CPRI segment is “stretched”.

This new network segment is called fronthaul: the segment between the RRU location and the BBU. Optical fiber can be used here. These optical fiber links use digital baseband interfaces such us CPRI or Open Base Station Architecture Initiative (OBSAI), but in this thesis I will focus on CPRI.

Therefore, CPRI is the radio interface protocol used in this segment for data transmission and it has some specific requirements, which are explained in detail in the next chapter. The purpose of CPRI is to allow the replacement of a copper or coaxial cable connection between the RRH and the BBU, so the connection can be made to a remote location and more convenient.

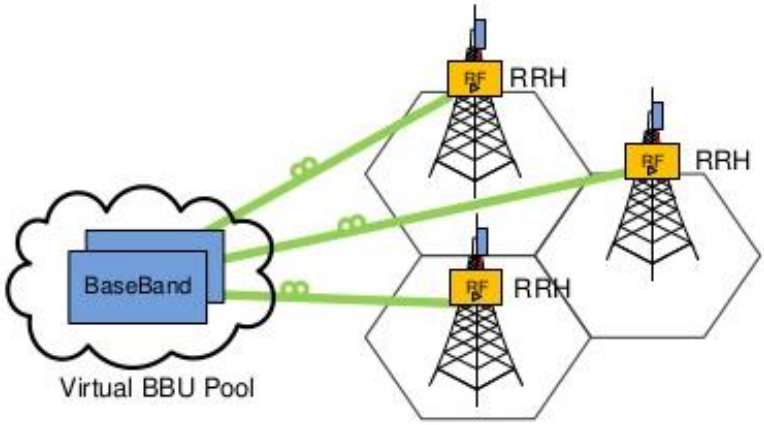


Fig. 3: C-RAN with RRHs and its Virtualized/Cloud/Centralized BBU Pool

In addition, backhaul network can be added to this scenario. The concept of backhaul refers to the segment between the core network and the edge of the subnetworks. In this segment can be used: fiber, cooper or microwave. Figure 4 extracted from [7], illustrates the entire network working with both sections: backhaul and fronthaul to build future mobile network. As we can see, the mobile network architecture is split into three parts: Radio Access Network (RAN) or fronthaul, backhaul network and core network.

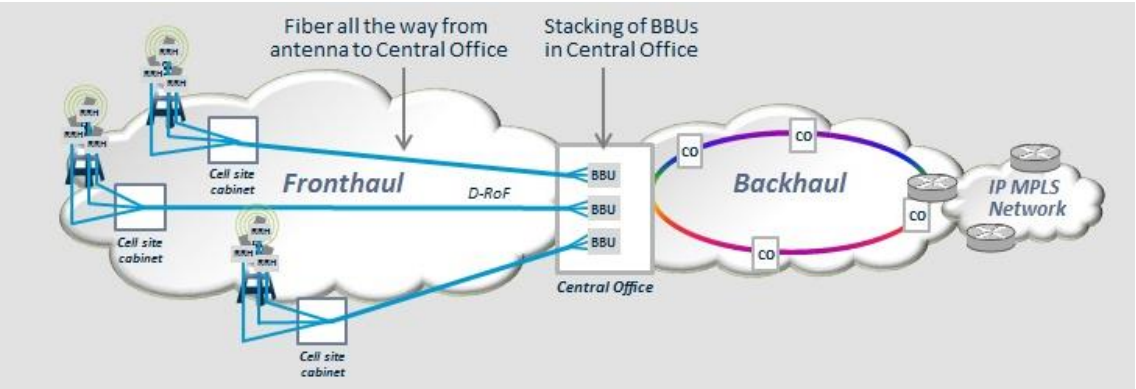


Fig. 4: C-ran architecture with fronthaul and backhaul

2.2.2. Traffic features

From another angle, users load changes during the day and processing power can be wasted in some areas. With this architecture, the utilization rate can be improved.

The baseband processing capacity of the BBU pool will be smaller than the sum of capacities of single base stations, remaining a type communication link sharing to have dynamic use of resources [3].

The communications channel is divided into a number of variable bitrate digital channels or data streams. In this way, the link sharing is adapted to the instantaneous traffic demands of the data streams that are transferred over each channel. As a result, when the link utilization is improved, it is called **statistical multiplexing gain**.

Then, as a definition, we can say: statistical multiplexing gain is the ratio between the sum of capacities from single base stations to what capacity is required in the BBU pool [3]. When the network size or the traffic intensity is higher, the gain increases.

It is used in packet switched computer networks. Each stream is divided into packets. The packets may be delivered following, for example, some scheduling rules for queuing or some guaranteed quality of service. Hence, resources could be managed and allocated dynamically on demand.

In [3] an analysis of statistical multiplexed gain of BBUs in C-RAN is performed:
"4 times less BBUs are needed for user data processing in a C-RAN compared to a traditional RAN for specific traffic patterns. The model does not include mobile standard protocols processing. After including protocol processing, we concluded that the statistical multiplexing gain varies between 1.2 and 1.6 depending on traffic mix, thereby enabling saving of 17% - 38% (in compute resources)".
Therefore:

* The utilization of the computed resources depends on the user distribution and data traffic.
* It is necessary a flexible mapping between RRH and BBU to adjust to different traffic and to get the maximum statistical multiplexing gain.

2.2.3. Advantages

In conclusion, the main advantages introduced by C-RAN approach can be summarized as follow:

- Using baseband processing centralized it is able to adapt to non-uniform traffic and utilizes the resources. Network flexibility is increased.
- As BBUs are located in one pool, they can interact with lower delays. Furthermore, if they are located in a single location, it can reduce repair cost and renting cost.
- RRH near to the antenna saves power and provide low losses. Moreover, RRHs placed up on the top of the tower doesn't need cooling.
- The cost of deployment and operation in the base stations is reduced. Base Stations more efficient.
- Few BBUs are needed in C-RAN which reduces the cost of network operation. Decrease power consumption compared with traditional RAN.
- Distance between BBU and RRH can be higher compared with traditional RAN, allowing a better location for the BBU equipment.
- The response time of application servers is shorter using BBU pools.
- Network security is increased.

Network performance is improved (less delay, non-uniform traffic, higher network capacity…); hence, the utilization of the network is more efficient, flexible and reduces costs.

Chapter 3

Fronthaul network options in C-RAN

In spite of C-RAN advantages, it has some stringent requirements for packet-based network in the fronthaul segment (fronthaul streams). The following subsections explain these requirements and the different transport solutions which can appear to solve these requirements. All they are good candidates for CPRI transport.

3.1. Fronthaul requirements

The fronthaul segment can measure up to 25 km, but depending on this distance, the requirements will be more stringent or more relaxed, i.e. the separation distance between the BBU and the RRH has a large effect on this. A summary of the technical requirements are featured below [8].

- Capacity
  High capacity is necessary in fronthaul. It implies the traffic streams need a high bitrate in order of gigabits per second.
  Using CPRI, the data rates go from 614.4Mbit/s up to 24.33Gbit/s, but it depends on the Radio Access Technology (RAT), carrier bandwidth and MIMO implementation.

  For example, using LTE with 20MHz carrier, the CPRI bit-rate can be calculated according to the following formula, extracted from [9]:

  $$R_{CPRI} = Ns * NAnt * Rs * 2NRes * OCW * OLC \qquad (1)$$

  Ns: number of sectors
  NAnt: number of MIMO elements per sector
  Rs: sampling rate (30.72 MHz for a single 20 MHz carrier)
  NRes: number of bits per sample (15 bits per sample)
  OCW: overhead introduced by CPRI control words (one control word for 15 words of payload)
  OLC: line coding overhead (10/8 Byte)

Sectors	MIMO	CPRI bitrate (Gbps)
1	1	1.228 (option 2)
1	2	2.457 (option 3)
1	4	4.915 (option 5)
1	8	9.830 (option 7)
2	1	2.457 (option 3)
2	2	4.915 (option 5)
2	4	9.830 (option 7)
3	4	12.165 (option 9)

Table 1. CPRI transport capacity for different configurations using a 20MHz carrier.

For one LTE sector configured as 4x4 MIMO with 20 MHz carrier bandwidth requires 4.915 Gbps as well as with two LTE sectors configures as 2x2 MIMO.
In short, more sectors mean a high CPRI rate, and more MIMO elements also mean a high CPRI rate.

- Latency
  Fronthaul network in C-RAN needs to have a very low latency to be able to implement applications in real time. The strict timing condition between BS and BBU also depends on the considered RAT. Latency requirement is one of the most important specifications because it limits how far the fronthaul network can be extended.

  The total latency can be measured as the sum of two parts:
  1. Latency due to the adaptation of fronthaul signals into the RAN infrastructure services (caused by CPRI transmission/reception interfaces)
  2. Latency due to the contribution of signal propagation along the RAN (defining the maximum distance between BBU hostel and cell sites).

  In the picture below extracted from [10], latency parts are shown.

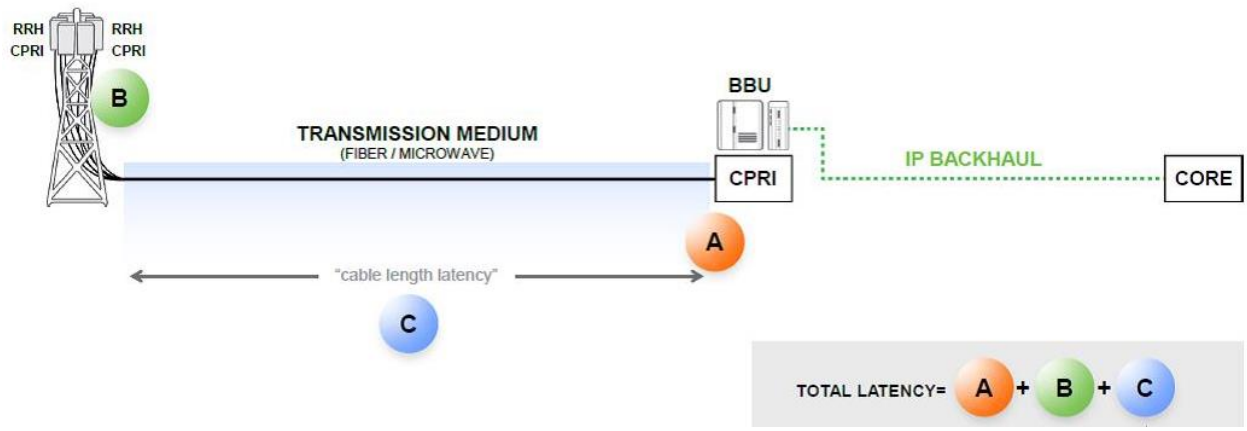


Fig. 5: CPRI latency route

The total latency, the Packet Delay Variation (PDV) and the Packet Loss Ratio (PLR) are fixed by standard as well as an assumption for the geographical distance between RRH and BBU. It is shown in Table 2 [6].

Feature	Value
Maximum End to End Latency/RTT (including fiber length, PDV, bridged delays)	100 μ s - 400 μ s (250 μ s for optical networks)
Latency budget (BBU to RRH, including fiber length, PDV, bridged delays)	50 μ s (for data rate 1-10Gbps)
Latency budget (excluding cable, BBU to RRH)	5 μ s (for data rate 1-10Gbps)
Maximum PDV	5 μ or10% of E2E latency
PLR	10^{-6}-10^{-9}
Geographical distance between RRH and BBU	<20km (25km for optical networks)

Table 2. Fixed fronthaul features

- Synchronization and jitter

  Synchronization issues should be solved to provide a correct adjustment between transmitter and receiver.
  There must be no mismatch between the transmitter clock and the receiver clock. Furthermore, bit errors and jitter should also be taken into account when talking about synchronization performance.

  Synchronization mechanisms in C-RAN BS are different (due to the changes introduced in the BS network architecture) by comparison with traditional BS. With traditional RAN there was a single clock generator which is feed by the BS. Now with C-RAN, the RRH clock generator is synchronized to the bit clock of the received CPRI signal and it is responsible to transmit the baseband radio signals in both directions [8].

  The jitter requirement (or delay jitter) is a measure of smoothness of the baseband data (short term variations in the timing of a repetitive signal). It means that if jitter affects the CPRI signals, it will have significant implications for the precision of the clock frequency generation.

  Different types of synchronization exist [11]:

  1. Frequency synchronization:
  This type of synchronization is necessary for mobile systems to match the time between two rising edges of the clock. It reduces signal distortion.

  2. Phase synchronization:
  This method is used for several rising edges happen in the same time.

For example, it is used in the case of Time Division Duplex (TDD) systems where the transmission (uplink and downlink) uses the same frequency but different time slots. Correct phase synchronization in BS is needed to avoid interference between signals.

Fig. 6. Types of synchronization

Some synchronization features in fronthaul networks as the Bit Error Ratio (BER), Frequency Error Contribution or time requirements for BS are shown in the following table, extracted from [8]:

Feature	Value
Maximum BER	10^{-12}
Maximum Frequency Error Contribution	2 ppb

Feature	Frequency	Time
LTE-A FDD	±50 ppm(wide area BS) ±100ppb(local area BS) ±250 ppb(home BS)	-
LTE-A TDD	-	±5μs(cell with radius>3km) ±1.5μs(cell with radius ≤3km)
MIMO	-	≤65ns
CPRI	±2ppb	±16.6276ns

Table 3. Synchronization features and time requirements.

Using CPRI specification, the jitter introduced by this protocol is fixed to 2 ppb (parts per billion) and it shouldn't be greater than this value to the frequency accuracy budget.
Based on these fronthaul requirements, there are different research directions for C-RAN and a wide range of fronthaul solutions.

3.2. Fronthaul solutions

In fronthaul segment, many options are being studied. Different interfaces, transport network technologies and topologies can be employed. This section summarizes the several existing solutions in C-RAN for fronthaul traffic [12]. The radio-link between RRH-BBU Pool can be realized through the electrical, optical or wireless domain depending on the interface requirements, but the most common solution is the use of optical networks.

3.2.1. Dedicated fiber

This approach deploys fiber (fiber pairs or a single directional fiber) between each RRH and the BBU over point to point; no additional optical transport equipment is needed. Figure 7 and 8 extracted from [14] and [15] respectively, illustrates the solution. Any interface can be used here because it's no necessary to encapsulate (encapsulated latency is not added).

But the main drawback of this solution is the cost of deploying the fiber network. Though, if fiber is deployed in a ring topology, we can take advantage of this physical aspect. This scenario is useful when the operator has a large available installed fiber. Otherwise the cost will be very high. Because of this, the dedicated fiber is not convenient to deploy a large fronthaul network. Apart from that, the fronthaul latency is zero due to the point to point connection, excellent latency for the fronthaul requirements.

It is a passive solution because the point to point connection is routed by Optical Distribution Frames (ODFs) without any type of extra power supply [8].

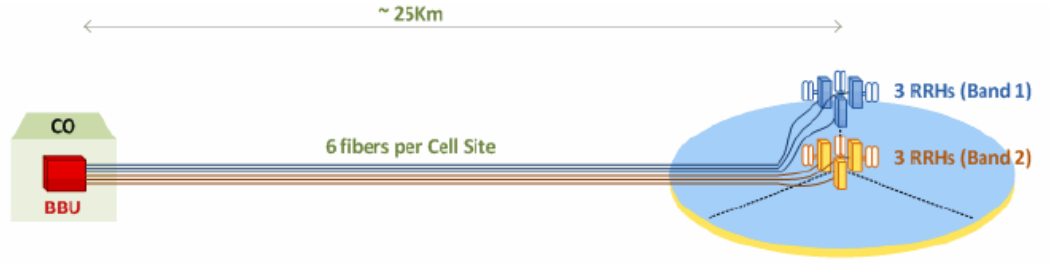


Fig. 7. Fronthaul link with dedicated fiber [14]

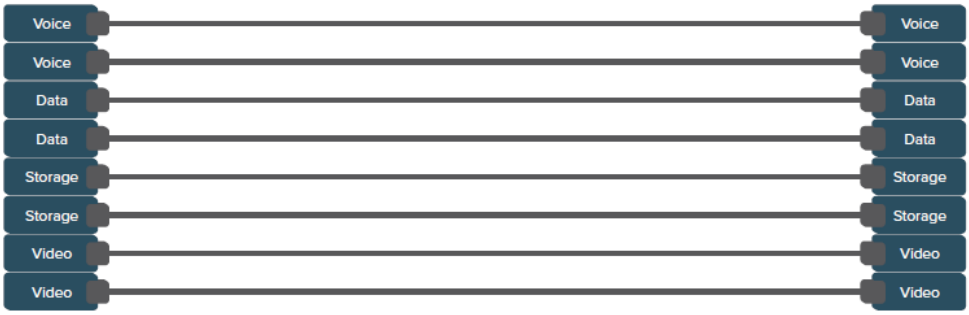


Fig. 8. Point to point connections: individual fiber per channel [15]

3.2.2. Passive/Active Wavelength Division Multiplexing (WDM)

The WDM technologies are based on Digital Radio over Fiber (D-RoF) technology. It allows transmitting independent digital radio signals over the same fiber through optical carriers of different wavelengths as shown in Figure 9 extracted from [13]. In this way, it is easier to deal with the high number of signals multiplexing them in one fiber link; WDM can save fiber consumption and improves the bandwidth.

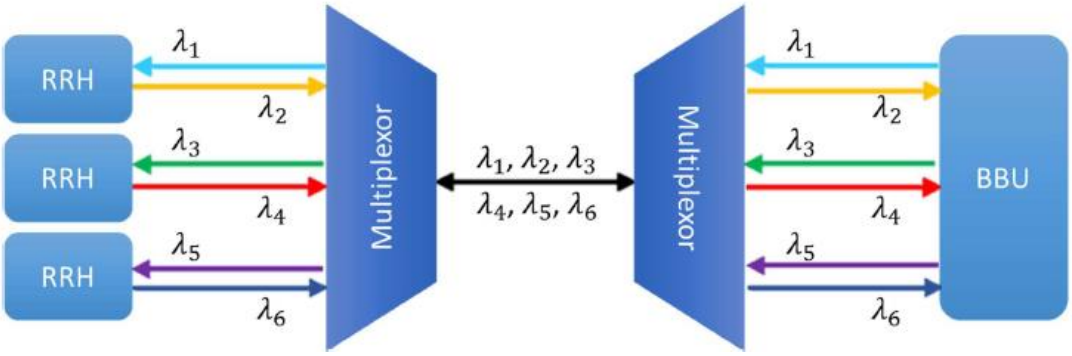


Fig. 9. WDM concept

40-80 optical wavelengths can be transmitted per link (per single optical fiber) [3].But it's necessary to add some equipment (transceivers or multiplexers) at the ends of the link to achieve the multiplexing as shown in Figure 10 and Figure 11 extracted from [14] and [15] references.

Fig. 10. Fronthaul link with WDM link

Fig. 11. Multiplexing combines multiple channels on a single fiber

WDM can be classified in active or passive depending on if the equipment/transport options needs power supply or not [8]. The passive solution doesn't need additional

energy in the intermediate nodes, translating to lower operational and maintenance costs. The active one requires power supply in the intermediate nodes or end points.

- *Passive WDM*
  The WDM transceiver resides in the data switch and its output connects to an unpowered multiplexer that combines and distributes the signals.

- *Active WDM*
  It employs a transponder separate from the switch which needs power supply and an external device inside is used for Optical Add/Drop Multiplexer (OADM) purpose. The WDM transponder changes the wavelength of the received fiber (λ0) into a specific wavelength (λ1, λ2 …). After that, the WDM multiplexer receives the specific wavelengths, it combines them and distributes the signals. The added components introduce an asymmetric latency in both directions and it requires two modules for uplink and downlink links: this WDM solution is more complex.

Below illustrate the difference between a WDM transponder and a WDM multiplexer [13]:

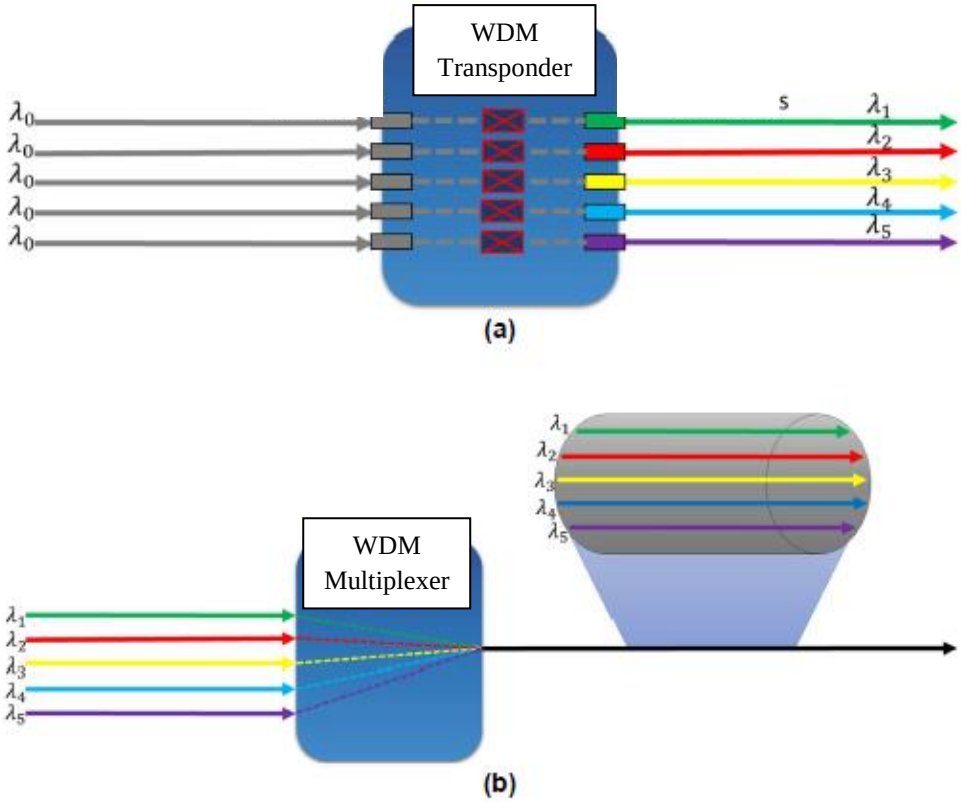


Fig. 12. a) WDM transponder and b) WDM multiplexer

In addition, depending on the network, there are two WDM implementations types [16]:

- *Coarse Wavelength Division Multiplexing – CWDM (Passive WDM).*
  It is appropriate for Time Division Synchronous Code Division Multiple Access (TD-SCDMA) or for short term fronthaul deployments (distances shorter than 70km) and it can be used to reduce costs. It is capable of multiplexing 16/18 wavelengths channels into a fiber. A single fiber with bidirectional transmission can be employed to save fiber/wavelength resources.
- *Dense Wavelength Division Multiplexing – DWDM.*
  It is a solution which assigned automatically and passively the wavelength supporting up to 80 simultaneous wavelengths of each channel. Dense WDM is appropriate for larger aggregate transport requirements (distances between 40km and 70 km) and it can handle higher speed protocols. It is suitable for LTE.

An example of both implementations techniques are shown in Figure 13 extracted from [16]. CWDM systems can provide 20 nm spacing between transmitted wavelengths multiplexing a few channels over the fiber link (approximately 16-18 channels). With DWDM, spacing between wavelengths can be even less than 1 nm multiplexing a high amount of channels over the fiber (approximately 80-90 channels) [13].

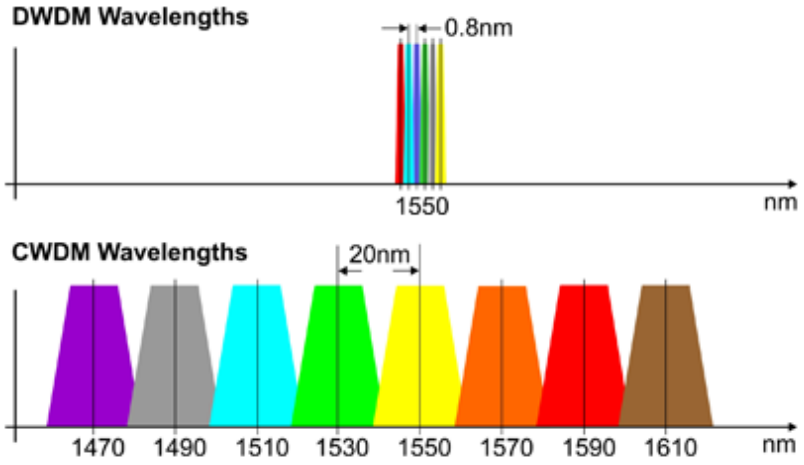


Fig. 13. WDM implementations

Wavelength Division Multiplexing is the preferred solution of the operators to migrate the current fiber network to the centralized architecture (using Fiber To The Home - FTTH to deploy the network in a cost-effective manner). Even though it is true that the WDM solutions reduce the amount of fiber and solve the latency issues, it also has some disadvantages as the additional transport equipment or little system adaptability. Furthermore, some issues occur in these networks as Raman scattering, crosstalk… Some proposals for mentioned effects are explained in [17], [18] and [19] references.

At this point, PONs concept can be mentioned. PONs means Passive Optical Networks and it can be utilized to carry any type of traffic on a common fiber using a passive optical splitter. To support sufficient data capacity there is a telecommunications network standard known as Next Generation Passive Optical Network (NG-PON2). Currently, the standard allows working simultaneously with Time Division Multiplexing and Wavelength Division Multiplexing, such as TWDM-PON and UD-WDM-PON (Ultra Dense WDM) technologies respectively. The PON data stream is converted to a service as Ethernet or CPRI. This solution can save fiber consumption since it is designed to include previous optical architectures and it reduces energy compared to Active Optical Networks (AON).

3.2.3. Optical Transport Network (OTN)

By introducing optical networks, it gives us a standard format to transport different protocols over the network. It is proposed to oversee the signals and to enable carriers to serve a higher number of customers ensuring reliability. The main problem is that OTN is not able to perform Statistical Multiplexing (SM).

In essence, the traffic flows are mapped into wavelengths using TDM-over-WDM technology. It supports Ethernet and CPRI/OBSAI protocols, but for now is difficult to operate with CPRI due to the lack of standardization.

For example, CPRI traffic flows are encapsulated and transported over the OTN signal hierarchy (multiplexed on the fronthaul). A new module is added to achieve this: OTN muxponders. First, in this signal hierarchy, CPRI data is mapped into OTN low-level containers; after that is multiplexed into higher layer signals and transmitted on different wavelength channels [8].

Some publications, such as [3] reference, introduce OTN concept within the explanation of Active/Passive WDM because OTN architecture is similar to active WDM, but there are certain differences: this OTN technology is based on ITU Recommendation G.709, moreover, it is a standard based on customer multiplexing which enables to reduce the number of wavelengths required, and therefore it increases fiber utilization.

A list of the added functions can be found below, extracted from [12].

* Standard based carrier-grade functions- per client and line OAM.
* Forward Error Correction (FEC) as well as Control and Management Functions (C/M).
* Multi-service support to combine different interfaces on the same infrastructure.
* Capable of managing DWDM transport + single fiber supporting 40 to 90 wavelengths + bidirectional transmission.

* OTN Muxponder (increase the cost of the network introducing these additional transport equipment).
* OTN wrappers (electronic switches) or Arrayed Wavelength Grating AWG/OADM (optical switches).

Additionally, asymmetric latency must be considered as an important OTN issue due to the active equipment introduced. This is a problem that must be solved in order to meet the requirements explained in previous sections.

3.2.4. Microwave

Microwave is a solution when fiber is not available, for instance, when the access to the geographical location is difficult. It may be fine for Heterogeneous Networks (HetNets) and for short distance CPRI fronthaul transport [8]. HetNets is a connection network of computers and other devices with different operating systems and protocols which uses multiple types of access nodes. For example, a wireless network that provides a service over a wireless LAN and that is capable of maintaining the service when it is switched to a cellular network (heterogeneous wireless network).

This technology could support the CPRI line bit rate options and can be cost effective. It also has low latency and great flexibility. Besides this, some important limitations are explained in [8].

3.2.5. Ethernet

Ethernet is used in Local Area Network (LAN) which supports higher bit rates and longer link distances. This asynchronous technology provides collision detection and access by carrier detection, besides it divides data streams in frames, making it perfect to use CPRI. Data streams are packetized in an Ethernet packet transmitted on an Ethernet link. Headers are added to these packets. Then, Ethernet can be used to transport CPRI frames and it is called CPRI over Ethernet (CoE) explained in detail in Section 4.1.1. Some remarkable advantages about Ethernet are the low cost of equipment and the use of statistical multiplexing gain, both good advantages for a 5G network.

All of these transport solutions are potential candidates to use CPRI interface, but Ethernet has become the most popular to perform a flexible, low cost and high capacity network. It is due to the adaptability to encapsulate CPRI frames in Ethernet packets. This is why the thesis focuses on the fronthaul transport of radio signals over Ethernet signals, more precisely CPRI over Ethernet (CoE). In the next chapter, the proposed network is explained as well as the CPRI protocol and Integrated Hybrid Optical Network (IHON) and switches.

Chapter 4

Reference network model

In this work, the network simulation concerned to the thesis is focused on the model proposed by [2] using the notation proposed by [20]. For instance, a scenario could be the one in Figure 14 extracted from [21], an Integrated Hybrid Optical Network (IHON), also called fusion solution, with a ring topology which can work with fronthaul and backhaul functionalities in the same infrastructure. The proposed solution combines circuit-switching and packet-switching properties into a single architecture because is necessary to achieve fully integrated systems. This kind of network is able to offer high throughput efficiency due to Statistical Multiplexing traffic on transport wavelengths.

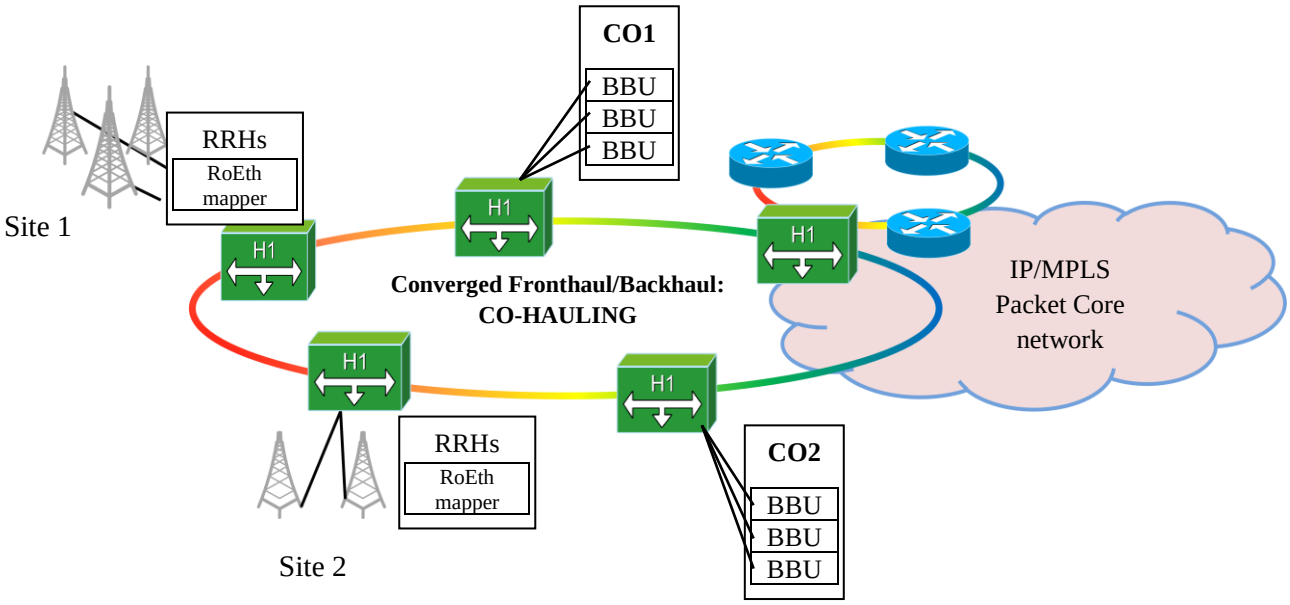


Fig. 14. IHON scenario based on a ring topology

As shown in Figure 14, the Central Offices (CO) can allocate a number of BBU (building the BBU pool), as has been argued above, to get lower delays and load balancing. CWDM or DWDM can be used as the candidate for optical network

technology. As previously mentioned, the fronthaul radio format used for data transmission mapping data in Ethernet frames, between RRH and BBU, is called CPRI and it is discussed in section 4.1.

The main idea of the IHON concept is to use packet switched nodes to transport Guaranteed Service Transport (GST) traffic and Statistical Multiplexing (SM) traffic classes, in addition to encapsulating CPRI data over Ethernet in the fronthaul segment. GST packets follow preasigned wavelengths from the sender to the receiver while SM packets are encapsulated in the empty gaps to improve link utilization. Thus, the network can be defined as a Guaranteed service Ethernet-based that uses WDM co-hauling.

As will be explained in detail, the traffic service classes consider for Quality of Service (QoS) aggregation model with 3-level priority can be classified in: Guaranteed Service (GS), Best Effort (BE) and Real Time (RT). The traffic has been analyzed on a 10GE output wavelength.

In the following subsections, concepts as CPRI, QoS model or optical aggregation in data centers using hybrid switches are discussed in more detail according to this network topology.

4.1. What is CPRI?

The Common Public Radio Interface (CPRI) arises from the need to make best use of the potential flexibility of the BBU-RRH segment. CPRI is the most well-known current transport standard which was formed in 2003 by industry cooperation (Ericsson, Huawei, NEC Corporation, Alcatel and Nokia). Currently, the most recent CPRI specification is version 7.0 [22].

Initially, CPRI was defined as an internal BS interface to allow antenna functions to be moved away from the baseband processing. Due to the future expected demand, it will be stretched and used over links of several kilometers. Now, the protocol is standardized to transport sampled RF data in the fronthaul of mobile networks. The interface can be used for radio standards as GSM, UMTS, LTE and WiMAX. Concerning the topologies, CPRI can support tree, ring or chain topologies but the management of these networks has not yet been thoroughly researched.

There are other interfaces like Open Base Station Architecture Initiative (OBSAI) or Open Radio Equipment Interface (ORI), but the most digital deployments use CPRI.

4.1.1. CPRI over Ethernet (CoE)

The use of Ethernet for transport in the fronthaul segment is being studied. Encapsulating CPRI frames onto Ethernet packets (CoE) makes possible to share a common Ethernet link with several CoE flows in which we can fill the empty gap with different traffic such as backhaul packets. It will be the process in which this

thesis will focus to see whether we can improve the throughput link. An example of architecture, indicated in Figure 15 extracted from [20], is composed by three RRH-BBU Pool links carrying CPRI flows packetized over Ethernet. It is necessary to add some extra equipment as well as Ethernet switches to multiplex the packets. The Ethernet signals are extracted from the CPRI equipment.

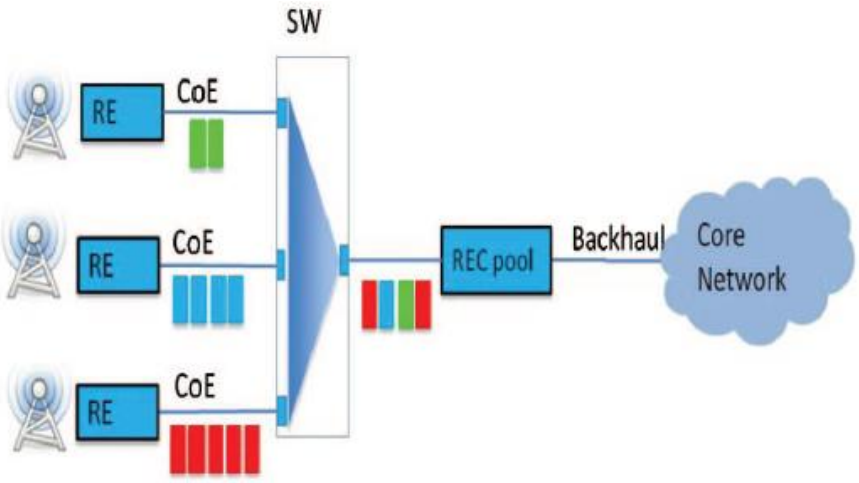


Fig. 15. Fronthaul architecture using CoE.

CPRI is expensive to deploy and also entails stringent requirements, as has already been mentioned under chapter 3. These delay and jitter requirements can be satisfied with high speed fronthaul solutions. CPRI ensure high throughput and low latency. Using an Ethernet-based mobile fronthaul could be less costly than using other technologies and more easily reconfigurable.

Line rate options

CPRI defines a continuous data flow which needs high-speed data rates to satisfy the high data transmission in fronthaul. RRH module generates the CPRI frames. The bandwidth of the RRH-BBU link is fixed, what changes is the line rate, setting by the standard. Table 4 indicates the line rate options available [20]. Line rates options from 614.4 Mbps (Option 1) up to 24330.2 Mbps (Option 10).

Line Rate Options	Line Rate (Mbps)
Option 1	614.4
Option 2	1228.8
Option 3	2457.6
Option 4	3072
Option 5	4915.2
Option 6	6144
Option 7	9830.4
Option 8	10137.6
Option 9	12165.1
Option 10	24330.2

Table 4. CPRI line rates

Frame format

The interface sends sampled IQ data in a frame format. The frame structure of CPRI is shown in Figure 16 extracted from [20], where the bytes per word change depending on the line rate option, see Table 5 [20].

The CPRI radio frame is made by 150 hyper frames and the duration is fixed to 10 ms. A hyper frame is made by 256 basic frames. The duration of each basic frame is fixed to 260 ns and is formed by 16 words; the word length depends on the line rate. One CPRI basic frame is the minimum data that can be encapsulated into the Ethernet.

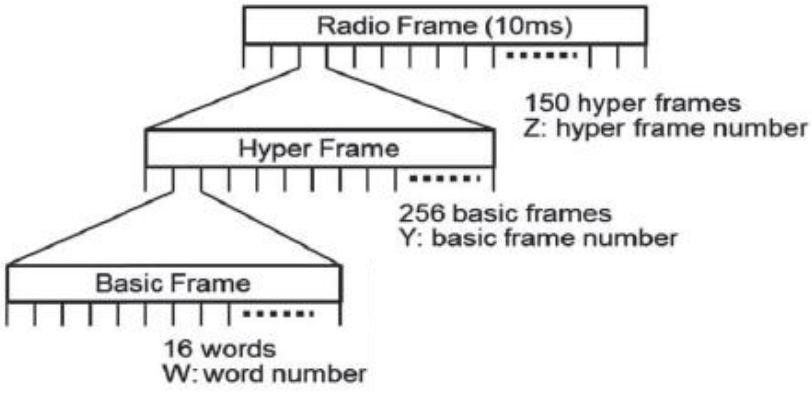


Fig. 16. Frame composition of CPRI

Line Rate Options	Bytes per word
Option 1	1
Option 2	2
Option 3	4
Option 4	5
Option 5	8
Option 6	10
Option 7	16
Option 8	20
Option 9	24
Option 10	48

Table 5. Bytes per word for each line rate

Furthermore, CPRI supports 8B/10B or 64B/66B encoding options. The thesis considers 8B/10B option [20]. For example, a CPRI option 1 basic frame has the following structure [14]:

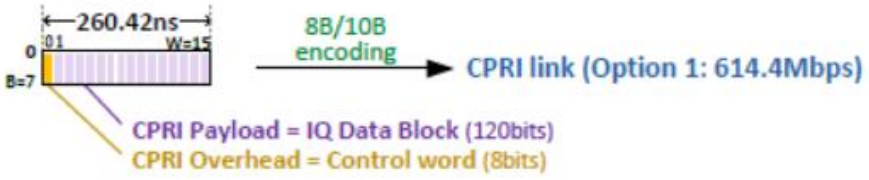


Fig. 17. CPRI option 1 basic frame structure

To get an idea of the basic frames per option, the image below extracted from [14] shows different options and the respective payload. Multiplying a number of basic frames, the CPRI data in an Ethernet frame is obtained.

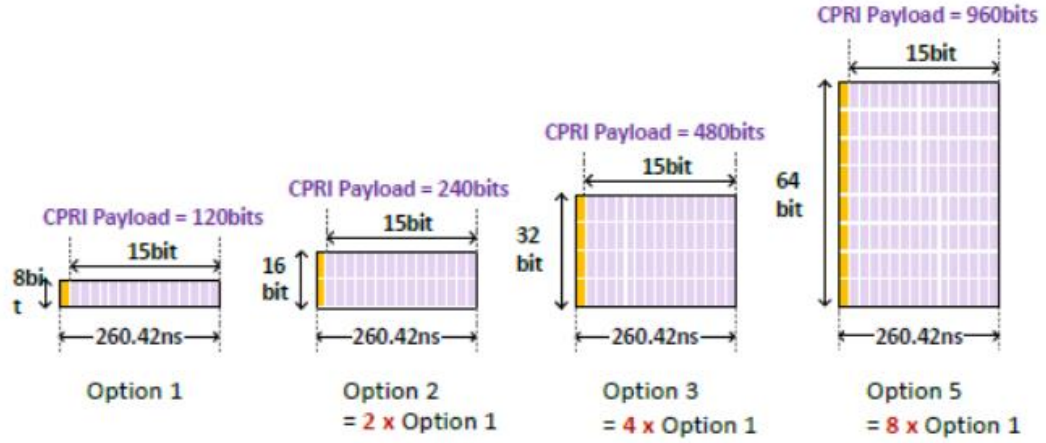


Fig. 18. CPRI basic frame structures per option

CoE encapsulation

CPRI flows are sequentially packetized into Ethernet packets, which mean mapping between CPRI and Ethernet frames and the need for an additional header to encapsulate data. The CPRI data is added to the following Ethernet structure frame with an additional RoE header [20]:

Preamble (7Bytes)	SFD (1Byte)	Dest MAC Addr (6 Bytes)	Src MAC Addr (6 Bytes)	Ether Type (2Bytes)	RoE Hdr (6Bytes)	FCS (4Bytes)	IPG (12Bytes)	Ethernet Payload

Fig. 19. CoE packet format

* Preamble - 7B
  This field is used by the receiver for a correct synchronization
* Start of Frame Delimiter (SFD)- 1B
  It indicates the start of the frame.
* Additional source MAC address -6B
  It contains the sending station address.
* Additional destination MAC address -6B
  It contains the adders of destination station

* Ethernet type – 2B
* RoE Header – 6B
  It contains different subfields: version, packet type, start of the frame, flow id, timestamp select field, timestamp and optional header space.
* Frame Check Sequence (FCS) – 4B
  Bits attached to the end of the Ethernet frame to verify the information through an incorrect frame check sequence or checksum
* Inter Packet GAP (IPG)- 12B
  It is the time between packets

4.1.2. CoE mapping notation

An overview of the parameters used to define CPRI encapsulation over Ethernet is shown below, extracted from [20]. The equations based on the encapsulation are explained with these parameters and are set out below.

Parameter	Description	Value
T_B	Length of Basic CPRI frame [s]	260 ns
L_E	Length of Ethernet Frame [bit]	Eq. (7)
T_E	Time of Ethernet Frame [s]	Eq. (8)
T_{encap}	Encapsulations Delays [s]	Eq. (4)
L_p	Ethernet Payload Size [bit]	Eq. (2)
R_{CPRI}	CPRI Line Bit Rate [bit/s]	Eq. (1)
T_{EOH}	Header Overhead per Ethernet Frame [s]	Eq. (5)
T_{totHOH}	Total Ethernet Header Overhead [s]	Eq. (6)
L_{EH}	Ethernet Header Size [bit]	44B
N_B	Number of CPRI Basic Frames	Eq. (3)
N_E	Number of Ethernet Frames in a Radio Frame	-
T_{totEOH}	Total CoE Overheads [s]	Eq. (10)
R_E	Ethernet Rate [bit per second]	10 Gbps
T_{hop}	Hop Delays [s]	Eq. (9)
T_{GAP}	Time available to put additional data [s]	Eq. (11)

Table 6. Notation to describe mapping between CPRI and Ethernet

An example of the CPRI encapsulation over Ethernet transport is shown in Figure 20 extracted from [20]. In this example, Encapsulation INPUT shows the CPRI frames which are going to be encapsulated in the Ethernet frame – Encapsulation OUTPUT. The input is at the CPRI line rate (R_{CPRI}) and the output is at the Ethernet rate (R_E). In this example, there are four basic frames to be encapsulated (N_B).

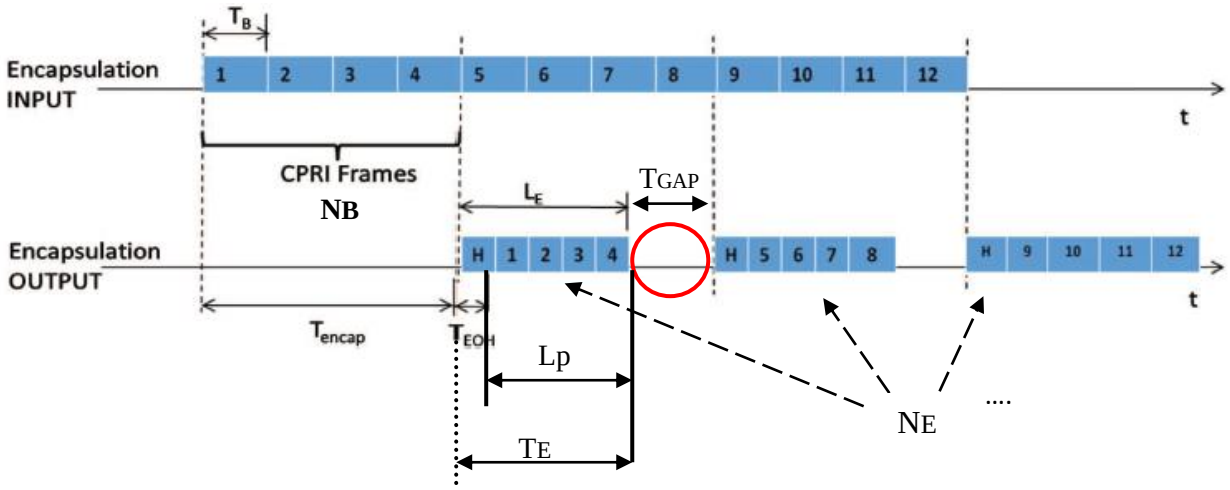


Fig. 20. CPRI encapsulation over Ethernet transport.

The CPRI Line Bit Rate (RCPRI) can be calculated with the equation (1) explained in Chapter 3. And the Ethernet Line Rate (RE) is fixed to 10Gbps because it's being used a 10G Ethernet.

The Ethernet payload size (Lp) is calculated as:

$$Lp = NB * RCPRI * TB \qquad (2)$$

where TB is fixed to 260 ns and the number of CPRI Basic Frames –NB– can be computed as:

$$1 < NB < \frac{Lp}{RCPRI * TB} \qquad (3)$$

The lower bound for NB is one because it is the minimum number of basic frames that can be encapsulated. And the upper bound is determined by the maximum payload size for an Ethernet frame –Lp = 1500B–.

The encapsulation delay (Tencap) is the time needed to encapsulate the CPRI data into the Ethernet payload and is calculated as:

$$Tencap = NB * TB = \frac{Lp}{RCPRI} \qquad (4)$$

where the maximum duration of Tencap is given for the maximum payload size for an Ethernet frame –Lp = 1500B–.

TEOH is the duration of the header overhead per Ethernet frame that in this case is computed as:

$$TEOH = \frac{LEH}{RE} = \frac{352\ bits}{10Gbps} = 3.52\ x10^{-8}\ \text{s} \qquad (5)$$

where the Ethernet header size (LEH) is set to 44B (352 bits) .

And, the total duration of the headers overhead depends on the number of Ethernet frames in a radio frame (NE). Total Ethernet header overhead (TtotHOH) is computed as:

$$TtotHOH = NE * \frac{LEH}{RE} = NE * TEOH = NE * 3.52x10^{-8} \qquad (6)$$

The length Ethernet frame (LE) includes the Etherned payload (Lp) plus the Ethernet header (LEH):

$$LE = Lp + LEH = NB * RCPRI * TB + LEH \qquad (7)$$

Thus, the respective time of Ethernet frame is computed as:

$$TE = TEOH + \frac{Lp}{RE} = 3.52x10^{-8} + \frac{NB*TB*RCPRI}{10Gbps} \quad (8)$$

Due to the equipment used (Ethernet switches) to process the packets, a delay is introduced. Hop delay (Thop) value depends on the switch forwarding method. In this thesis, is considered a store-and-forward switch (Ethernet switches will be explained in more detail below). Thop is introduced as

$$Thop = \frac{LE}{RE} \qquad (9)$$

Finally, the total CoE overhead (TtotEOH) can be calculated as:

$$TtotEOH = TtotHOH + Thop \qquad (10)$$

In order to improve the performance, the empty time that is not used per Ethernet frame can be useful to put into that GAP an additional data, in this case, backhaul packets. This GAP time can be obtained as:

$$TGAP = Tencap - TE = (NB * TB) - TE = \frac{Lp}{RCPRI} - (TEOH + \frac{Lp}{RE}) \quad (11)$$

Thus, it can be seen that depending on the number of encapsulated CPRI frames (NB), the Lp value changes, and therefore also LE, making the GAP time shorter or larger. NB has an effect on Tencap, TE and TGAP. In this way, data that fill the GAP could improve or not the information transfer.

4.2. Integrated Hybrid Optical Network (IHON)

This section looks more closely at the IHON network or fusion solution which blends circuit and packet switching sharing the same physical links, e.g. the same wavelength. This packet-based technique allows us to use the free GAP between Guaranteed Service Transport (GST) packets when is used a model of quality of service to schedule traffic in Ethernet hybrid aggregation nodes or switches. The target is to insert Best Effort (BE) or Real Time (RT) packets in the empty GAP focusing in how hybrid switch handle the traffic. All this will be explained more fully later in the following sections.

4.2.1. 3-Level Priority Quality of Service (QoS)

QoS model enables to provide better service to certain traffic and limiting other kind of traffic. It provides different treatment to the flows, giving the highest priority to the Guaranteed Service (GS) in this case.

The idea is to use packet switched nodes to transport Guaranteed Service (GS) traffic and Statistical Multiplexing (SM) traffic classes. GS packets follow preasigned wavelengths from the sender to the receiver while SM packets are encapsulated in the empty gaps to improve link utilization. The introduction of SM packets is useful to exploit the inefficient utilization of bandwidth (free GAPs) which means high utilization of the channel (high throughput).

The traffic service classes consider for QoS aggregation model with 3-level priority can be classified in: Guaranteed Service (GS), Best Effort (BE) and Real Time (RT) [2]. The CPRI flows are designated as the Guaranteed Service, whilst the Best Effort traffic is indicated as the Backhaul traffic in this thesis. To identify which priority has the traffic, a field in the Ethernet frame is used.

Some of the characteristic features of these service classes are explained below [24].

* Guaranteed Service: reserved resources for each flow from source to destination. It shouldn't be lost inside the network.
  - -GS traffic has the highest priority
  - -Time transparency
  - -Low delay and no losses
  - -E.g.: high definition video streaming to end users

* Best Effort: basic connectivity with no guarantees. It should have a low packet loss.
  - -traffic with no requirements or priority
  - -packets stored in queues
  - -E.g.: free e-mail, web and storage services
* Real Time: There mustn't be any delay.
  - -time transparency
  - -limited packet loss
  - -real time service
  - -E.g.: Voice conversational services

4.2.2. Data Center Network Architecture: Integrated Hybrid technology in Ethernet switches

The architecture of the proposed model needs the introduction of optical packet switching technologies for data center interconnection to support and satisfy the network requirements. Requirements such as the downsizing of interconnection complexity, save energy consumption, high throughput or successful QoS.

The switch architecture considered in this thesis is shown below, extracted from [2].

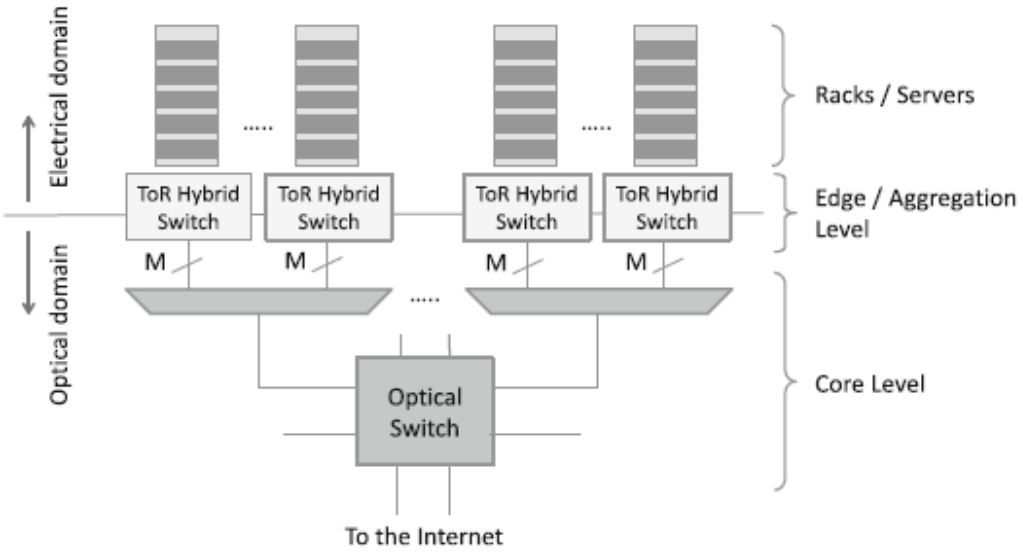


Fig. 21. Data center interconnection with hybrid aggregation switches

This model uses racks to store the traffic generated and interconnect the electrical and optical domain to send the traffic to the optical switch to be forwarded. Each hybrid switch carries different traffic.

A detailed example of how the hybrid optical switch operates in a single channel (M=1) is shown in Figure 22 extracted from [23]. The switch inserts SM packets at an output wavelength where the GS packets have the priority. This occurs because of the GS gap detector, which detects the empty gaps in between de GS packets and the SM scheduler, which inserts the SM packets (before, it checks the head of the SM packets stored in the queue) if they fit in the gaps. This packet class detection can be possible due to coding techniques. In this way, the throughput of the channel is improved.
Using this kind of IHON switches, SM packets can use the physical wavelengths provisioned for the GS packets.

Furthermore, a fixed delay is introduced on the GS burst to avoid collisions. This delay is explained in details in section 5.4.3.

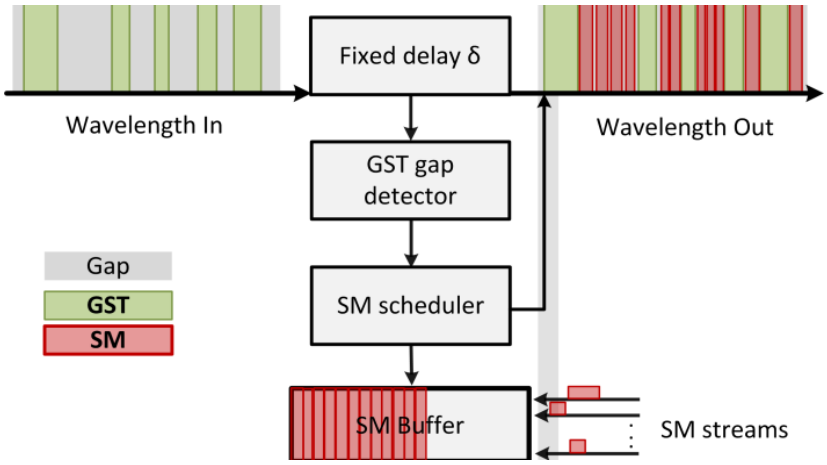


Fig. 22. Hybrid optical switch for SM packets insertion in a single channel

For instance, the performance of the proposed model can be seen in Figure 23, where is shown an example of the channel occupancy of four channels extracted from the [2] reference. It takes into account the RT traffic, although in the thesis is not employed. RT traffic is not queued because it's a real time service, then it has to be forwarded. And there is always BE traffic in the queue for this example.

The BE and RT packets are multiplexed and transmitted through the channel when there are empty periods due to the fact that there is not any GS packet to transmit from the GS source.

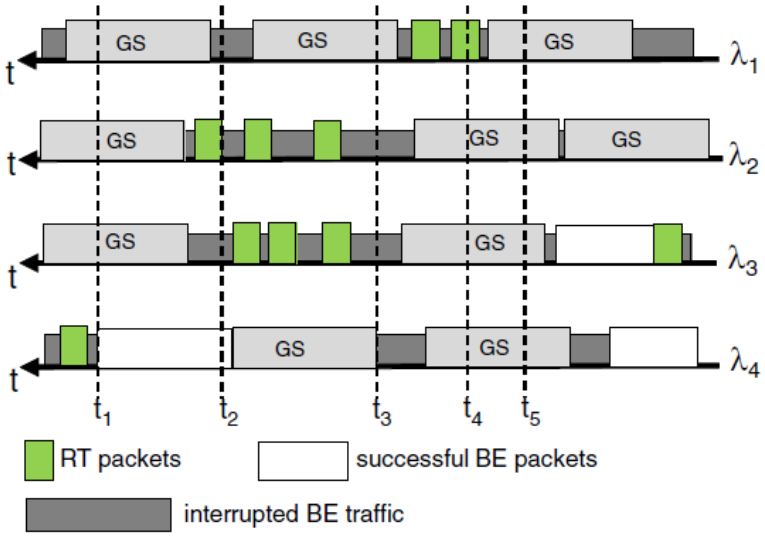


Fig. 23. Example of channel occupancy of four channels

The behavior can be explained as follows:

-If any GS packet is forwarded and any RT packet arrives, the BE packet can be successfully sent through the channel as illustrated in the fourth channel between t1 and t2.

-If any GS packet is forwarded but some RT packets arrives, these packets choose a channel and interrupt the BE packets as in the second channel for t2.

-If some GS packet arrives, it has the priority over the rest of the classes packets and is forwarded in its pre-assigned wavelength interrupting the BE packets as we see in the fourth channel between t3 and t5.

-If all the channels are occupied by GS packets as occurs for t5, an alleged RT arriving will be lost.

The following chapter explains the analytical hybrid switch model representing the output interface with the queues and the different traffic classes explained before: an application of the previous contents to a simulation.

Chapter 5

Simulation Environment and Model

5.1. Why to simulate?

Nowadays, the evolution process to change to some new mobile system technology needs advanced studies. Furthermore, the electronic devices and developed instrument use to be expensive hence, changes cannot be taken lightly.

This is why a simulation tool of the network is tested and evaluated. The analysis of the wanted requirements gives us a model to start working with several scenarios.

5.2. General features

The proposed interface is explained below as well as the model parameters. The event-based simulator is based on the architecture proposed in [2] and the original code developed in C language is written at the end of the thesis, under Appendix I.

5.2.1. Analytical Multi-service queuing model representing the output interface of the hybrid switch

A model which takes into consideration only GS and BE classes of traffic can be graphically represented as follows. In Figure 24, the hybrid switch optical output interface with two classes of traffic is presented.

Each GS traffic source is associated to a channel (i=1, 2,…, M). Each channel carries the GS packets traffic through different output wavelengths. As already explained, the GS packets have priority and they are forwarded to the respective output. Then, the scheduler inserts the BE packets in the vacant gaps. Thus, GS traffic is guaranteed without any loss and without any delay caused by the rest of the traffic, and the BE traffic is added as much as possible. The queue can also be seen in the figure for BE packets.

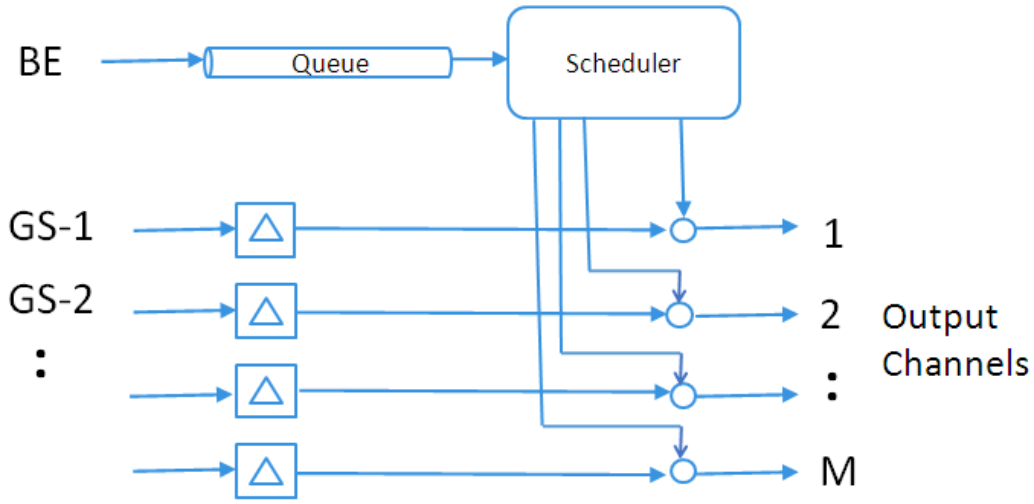


Fig. 24. Output interface model with M channels

There are some important features to consider:

* GS burst can interrupt BE packets in transmission, GS packets have the priority
* The interrupted BE packets need to be transmitted again. They are considered as a new packet getting into the queue
* With a GS arrival at the input, BE packets are not allowed to transmit by that channel, but they can be scheduled through other channel if it is free.
* After the end of the transmitted GS burst plus the fixed delay, a BE packet can start to be transmitted.

5.2.2. Model parameters

Below is a summary of the parameters used in the simulator model, a description and some behavior features [2].

Parameter	Description
M	Number of output channels
n <= M	Number of GS burst being simultaneously transmitted at the generic instant t
h = M - n	Number of output channels available to BE traffic at the generic instant t
m	Average number of channels available to BE packets
$\mathbf{Tg^{OFF}}$	Average OFF period of a GS source (one source per channel)
$\boldsymbol{\lambda g = \frac{1}{Tg^{OFF}}}$	GS burst arrival rate during an OFF period
λb	BE packet arrival rate (one source per interface)

θ g = $\frac{1}{\mu g}$	Average GS burst service time (also average ON period)
θ b = $\frac{1}{\mu b}$	Average BE packet service time
ρ g = $\frac{\theta g}{(\theta g + Tg^{OFF})}$	Offered GS load per channel
ρ b = $\frac{Ab}{M}$	Offered BE load per channel
Ab = λb* θ b	Offered BE load per interface

Table 7. Simulator parameters

1. Independent and identical ON/OFF Gs sources generate GS arrivals with a negative exponential distribution of ON/OFF periods. There are M sources generating GS traffic. Each GS source generates the traffic ρ g, which feeds each output channel.

2. BE arrivals follow independent Poisson processes i.e., the service time is exponentially distributed for BE packets. The BE source generates BE traffic with intensity Ab. These BE packets are scheduled and transmitted to the output interface (to the M channels).

Total output interface traffic: M* ρ g + Ab

5.2.3. Simulator input/output arrays

To run the simulator, some arguments should be inserted. The input array is:

<seed><N_samples><N_servers><rho_GS><serv_GS><load_RT><serv_RT><load_BE><serv_BE>

where:

<seed> = whatever = <0>
<N_samples> = 100 million = <100.000.000>
<N_servers> = 1,2 and 5 number of servers are used (M). The total number that can be used is 32 servers.
<rho_GS> = varies from 0 to 1
<serv_GS> = θg = TE (GS service time = TON) varies
<load_RT> = 0.0 in this thesis
<serv_RT> = 0.00032
<load_BE> = Ab = M*ρ_b
<serv_BE> = θb = 160ns, 320ns, 480ns, 640ns or 800ns

And the results are obtained in the following format:

<RT_loss_rate><BE_int_rate><BE_succ_rate><servint><servsuccBE><servBE><avgutil[0]/Nserv><avgutil[1]/Nserv><avgutil[2]/Nserv><avgutilBEint/Nserv><totBEutil/Nserv><w[2]>

where:

1. <RT_loss_rate> = RT loss rate
2. <BE_int_rate> = BE interruption rate
3. <BE_succ_rate> = BE success rate
4. <servint> = BE interrupted service time
5. <servsuccBE> = BE successful service time
6. <servBE> = BE service time on channel
7. <avgutil[0]/Nserv> = GS channel utilization
8. <avgutil[1]/Nserv> = RT channel utilization
9. <avgutil[2]/Nserv> = BE channel utilization of successful packets (BE throughput)
10. <avgutilBEint/Nserv> BE channel of interrupted packets
11. <totBEutil/Nserv> BE channel utilization (successful+interrupted)
12. <w[2]> BE packet average waiting time

According to the simulator, the traffic classes are ordered as

MAX_CLASSES = [GS RT BE …]
[0 1 2 … 10]

5.2.4. Concepts to be analysed

To study the performance of the simulator, this thesis analyses the following terms:

- Success rate for BE traffic (%)
  This result is the ratio between the number of BE packets successfully sent (not interrupted by GST) and the total BE packets on the output channel.

- Throughput (Bps)
  This concept refers to the channel utilization level. In this case, we can evaluate the channel utilization of successful BE packets.

Thr (interface) = Thr(channel#1) + Thr(channel#2) + Thr(channel#3) … = = Thr(%) * 10Gbps * M

  *Moreover, the channel utilization depends on the ρ_b value. For example, if ρ_b = 0.1 means that a 10% of the channel can be used, and taking into account a channel of 10Gbps: 10Gbps * 0.1 = 1 Gbps.
  And, if ρ_b = 1 (100%): 10Gbps * 1 = 10 Gbps.

- Average waiting time for BE traffic (s)
  This term refers to the time that BE packets spend in the queue until they are transmitted.

5.2.5. Simulator performance

A first example to start understanding how the simulator works, could be running it to represent the BE success rate for CPRI option 1 and 6.

Lp is the Ethernet payload size so we can run it for the following values: 200 B, 400 B, 600 B, 800 B, 1000 B, 1200 B or 1400 Bytes. Furthermore, the traffic is analyzed on a 10GE output wavelength

The correct value for the parameters can be calculated using the CPRI equations seen before in section 4.1.2., getting the following table. For CRPI Option 1 (Line Rate = 614.4Mbps):

NB	TB[ns]	RCPRI[Gbps]	RE[Gbps]	Lp[Byte]	Lp Aprox[Byte]	TE(=thetaG)[ns]	Tencap[ns]	rhoG	Delta	Tgap[ns]
10	260	0,6144	10	199,68	200	195,2	2604,166667	0,0749568	99,2	2309,76667
20	260	0,6144	10	399,36	400	355,2	5208,333333	0,0681984	99,2	4753,93333
30	260	0,6144	10	599,04	600	515,2	7812,5	0,0659456	99,2	7198,1
40	260	0,6144	10	798,72	800	675,2	10416,66667	0,0648192	99,2	9642,26667
50	260	0,6144	10	998,4	1000	835,2	13020,83333	0,06414336	99,2	12086,4333
60	260	0,6144	10	1198,08	1200	995,2	15625	0,0636928	99,2	14530,6
70	260	0,6144	10	1397,76	1400	1155,2	18229,16667	0,063370971	99,2	16974,7667

Table 8. Parameters using CPRI option 1.

As can be seen from Table 8, we obtain the respective Tgap for each payload, which increases as payload increase. Greater Tgap means largest time to add more BE packets. We can also observe that Tencap is getting bigger as the payload increases.

In this case is using one server (M=1) and θb is 160ns. ρ_g and θg vary, thus, the input array should be changed for each payload:

<0><100000000><1><ρ_g>< θg><0><0.00032><0.1><0.00000016>

If we do the same for CPRI option 6 (Line Rate = 6144 Mbps), the table below is obtained:

NB	TB [ns]	RCPRI [Gbps]	RE [Gbps]	LP [Byte]	LP (approx.) [Byte]	TE (=thetaG) [ns]	Tencap [ns]	rhoG	Delta	Tgap
1	260	6,144	10	199,68	200	195,2	260,4166667	0,749568	99,2	-33,9833
2	260	6,144	10	399,36	400	355,2	520,8333333	0,681984	99,2	66,43333
3	260	6,144	10	599,04	600	515,2	781,25	0,659456	99,2	166,85
4	260	6,144	10	798,72	800	675,2	1041,666667	0,648192	99,2	267,2667
5	260	6,144	10	998,4	1000	835,2	1302,083333	0,6414336	99,2	367,6833
6	260	6,144	10	1198,08	1200	995,2	1562,5	0,636928	99,2	468,1
7	260	6,144	10	1397,76	1400	1155,2	1822,916667	0,6337097	99,2	568,5167

Table 9. Parameters using CPRI option 6.

Now, Tgap and Tencap values are smaller comparing them with the values obtained for option 1. Furthermore, we should note that for Lp =200B the TGAP is negative, then any BE packet can be transmitted.

For both options, the following graph is obtained:

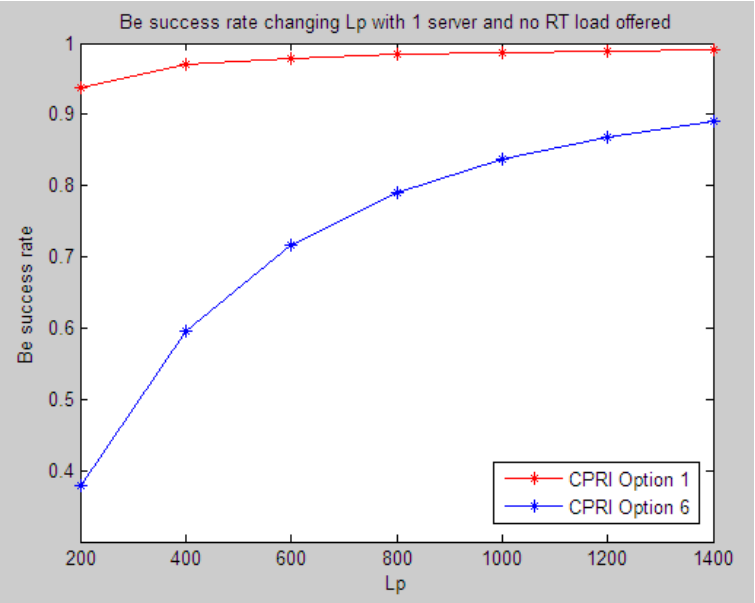


Fig. 25. Success rate of BE traffic (%) as a function of Lp, varying the CPRI option with θb = 160ns, $\rho_b = 0.1$ and M = 1. No RT load offered.

1. With option 6, the gap is small so the success of sending BE packets through the empty gaps will be smaller than the success of sending packets with option 1 because of the largest gap that option 1 provides.

2. Raised BE success rate is obtained with higher payload due to the large Ethernet frame it provides. Higher payload means higher Tencap, then higher Tgap.
Thus, the BE success rate is always more elevated for Lp=1400B than for Lp=200B.

5.3. Characterization of the simulator

Several functionalities are implemented in the simulator in order to evaluate the usage of the transport network. The following subsections explain the changes and the related code.

5.3.1. Service time function

The original simulator follows an exponentially distribution for both BE and GS packets. Now the idea is to change it so that GS packets follow a deterministic distribution, as shown in Figure 26. A deterministic distribution is useful in order to improve the channel utilization because you know the GS frames times and the TGAP periods have the same length. We are changing the burstiness to make it synchronous. The distribution for BE packets doesn't change.

To this aim, the GS service time has to be changed i.e., the θg parameter (TON).

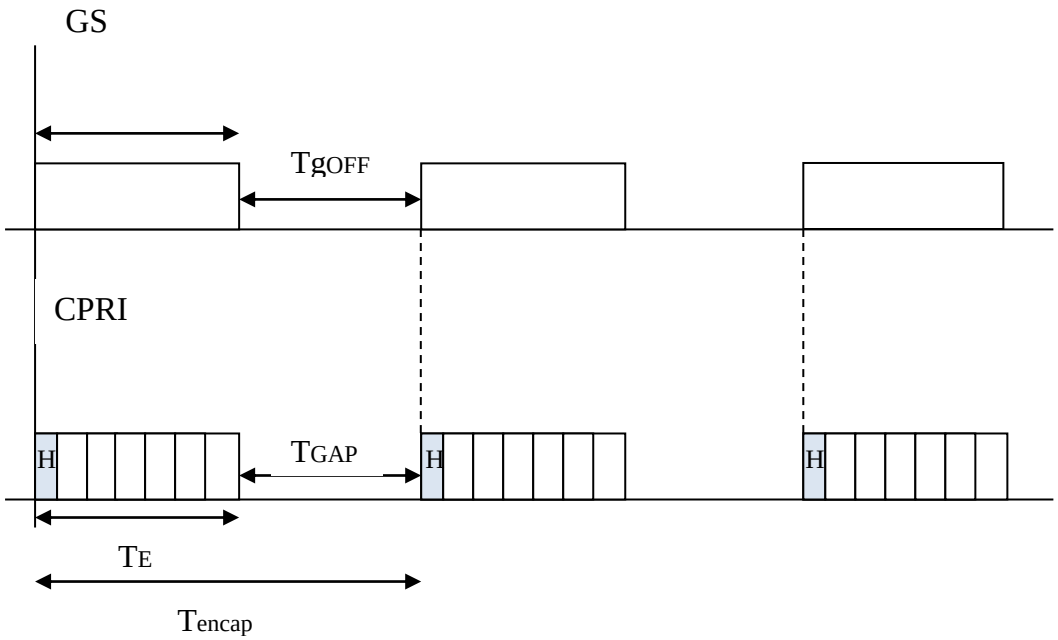


Fig. 26. CPRI encapsulation with deterministic GS service time

Using the following relations to compute T_{GAP}, we obtain TGAP.

$$T_E = Tencap - T_{GAP} \quad \text{(i)}$$

$$TgON = \frac{1}{\mu g} = \theta g \quad \text{(ii)}$$

$$TgOFF = T_{GAP} \quad \text{(iii)}$$

$$\lambda g = \frac{1}{TgOFF} \quad \text{(iv)}$$

$$\rho_g = \frac{T_E}{Tencap} = \frac{\theta g}{\theta g + TgOFF} = \frac{TgON}{TgON + TgOFF} = \frac{T_E}{T_E + T_{GAP}} = \frac{\theta g}{\theta g + T_{GAP}}$$

$$\boldsymbol{T_{GAP} = TgOFF = \frac{\theta g(1-\rho_g)}{\rho_g} = \theta g\left(\frac{1}{\rho_g} - 1\right) = \frac{1}{\mu g}\left(\frac{1}{\rho_g} - 1\right)}$$

```
// This function generates an instance of the service time random variable. Arguments:
pclass = class of the customer

double serv(char pclass) {
        if (pclass == 0){              //GS return a determistic service time
                return 1.0/mu[pclass];
        }
        else if(pclass == 2) {//BE return a random service time
                return expon(mu[pclass]);
        }
        else {                         // RT: return a fixed service time
                return 1.0/mu[pclass];
        }
}
```

```
// Get the load and average service time for each class from the following arguments and
set mu = 1/service and lambda = rho*mu (different for GS due to on-off source)
        for (j = 0; j < C; j++) {
                mu[j] = 1.0/atof(argv[4+2*j+1]);
                if (j == 0) {
                        Toff = 1.0/mu[j]*(1.0/atof(argv[4+2*j])-1.0);
                        lambda[j] = 1.0/(Toff+1.0/mu[j]);
                } else {
                        lambda[j] = atof(argv[4+2*j])*mu[j];
                }
        }
```

where:

argv[5] = θg

argv[4] = ρ_g

} Input arguments

5.3.2. Service time function for different BE packet lengths

At this point, the change involves the BE service time. Now, the BE packet length follows an empirical distribution: we have different packet lengths with different probabilities [21] and the consequently duration time. The maximum length of the packet (Lmax) is 1518 Bytes.

Packet length	Rate	Probability	Time
64B (512bits)	10Gbps	0.45	51.2ns
594B (4752bits)		0.1	475ns
1318B (10544bits)		0.05	1054.4ns
1418B (11344bits)		0.05	1134.4ns
1518B (12144bits)		0.35	1214.4ns

Table 10. BE packet length, probability and service time

It's necessary to change the previous code adding the new features:

```
// This function generates an instance of the service time random variable. Arguments:
        pclass = class of the customer
double serv(char pclass) {
        if (pclass == 0){              //GS return an determistic service time
                return 1.0/mu[pclass];
        }
        else if(pclass == 2) {//BE return a random service time(of a
                        previous list based on the probabilities)
                return get_BE_service_time_discrete();
        }
        else {                  // RT: return a fixed service time
                return 1.0/mu[pclass];
        }
}
```

```
//This function generates different service time for BE packets, which have different
packet length and probabilities
double get_BE_service_time_discrete(){
int random = rand() % 101;
double return_time; //probability(%)=[45 10 5 5 35]-->[45 55 60 65 100];
if (random <=45){
        return_time=0.000000051;
        }
        else if(random<=55){
                return_time=0.000000475;
                }
                else if(random<=60){
                        return_time=0.000001054;
                        }
                        else if(random<=65){
                                return_time=0.000001134;
                                }
                                else if(random<=100){
                                        return_time=0.000001214;
                                        }
return return_time;
}
```

5.3.3. Fixed delay

GS packets are delayed before being transmitted to the output channel. They are delayed for a fixed time Δ, which is a parameter that depends on the bit rate and the maximum length of RT packet service time. Delay Δ should be smaller than T_{GAP}.

It allows that the GAP detector finds the length of the empty time in order to avoid the pre-emption of the BE packets transmission as is shown in Figure 27 [12].

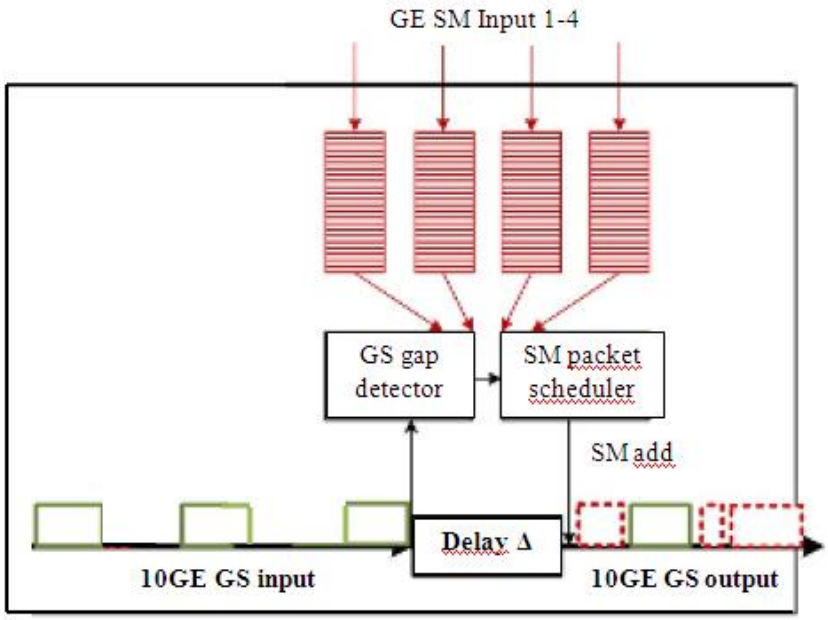


Fig. 27. Aggregation scheme of GS and BE packets with fixed delay

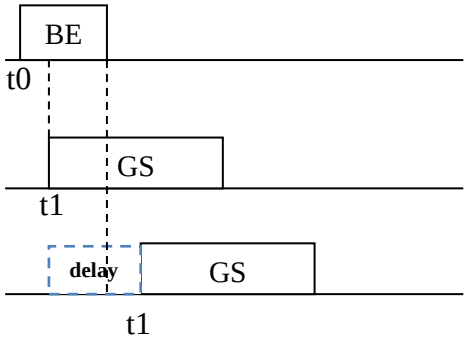
In this way, a BE packet, which is being forwarded, can be sent without any collision with the following GS packet. We have to take into account that the delay Δ should be smaller than T_{GAP}. For example, the delay in a channel of capacity C of a maximum length SM packet Lmax, can be expressed as: $\Delta = \frac{Lmax}{C}$

If any GS packet is detected and the channel is free at time t0, the SM packet scheduler can start to transmit a BE packet of size L<=Lmax as in Figure 28(a). Let us assume a GS packet is detected at time t1, after sending the BE packet (t1>t0), Figure 28(b). There would be a collision without the delay; but adding this time, the GS packet is delayed the sufficient time to allow always the BE packet transmission and after this, it will start to be forwarded the GS packet, Figure 28(c) [21].

Fig. 28a).

Fig. 28b).

Fig. 28c).



Thus, $t1 + \frac{Lmax}{C} > t0 + \frac{L}{C}$

In this thesis, the value for this fixed delay is 99.2ns, which is a standard extracted from [2] and corresponds with the smallest fragment (124B) that can be pre-empted. Then, $T_{GAP} = Tencap - T_E - \Delta$

```
if (argc < 6) {
        fprintf(stderr,"Usage: %s <seed> <N_samples> <N_servers>   <rho_GS> <serv_GS>
<load_RT> <serv_RT> <load_BE> <serv_BE> <fix_delay>\n",argv[0]);

                exit(-1);
        }

//Fixed delay to avoid collisions between SM packets and GS burst
//This value is introduced as input, const char *x=argv[10];
        double fixed_delay = atof(argv[10]);
        printf("Fixed delay = %1.20f",fixed_delay);
```

5.3.4. BE packets successfully sent

Some counters are added to know how many packets are successfully transmitted and their percentage as well as the interrupted packets.

1. Total number of BE packets successfully sent depending on the different packet length:

$$BE\_success[i] = BEarrivals[i] - BEinterrupted[i]$$

where i=0,1,2,3,4 math with 51ns, 475ns, 1054ns, 1134ns and 1214ns

2. To see what happens for each size Lp, the ratio of BE interrupted packets over the total arrivals for each packet length can be calculated as:

$$Ratio1 = BEinterrupted[i]/arrivals\_BE[i]$$

// Ratio approximate to 1 means all packets are interrupted
// Ratio approximate to 0 means all packets are successfully sent (not interrupted)

3. To see what happens for each size Lp, the ratio of BE interrupted packets over the total BE arrivals can be calculated as:

$$Ratio2 = BEinterrupted[]/arrivalsBE[k=2]$$

4. SuccessfulRatio1 = 1 – Ratio1
   SuccessfulRatio2 = 1 – Ratio2

```
//This function generates different service time for BE packets, which have different
packet length and probabilities
double get_BE_service_time_discrete(){
int random = rand() % 101;
double return_time;        //probability=[45 10 5 5 35]--> [45 55 60 65 100];

if (random <=45){
        arrivals_BE[0] = arrivals_BE[0] + 1;
        return_time=0.000000051;
        }
        else if(random<=55){
                arrivals_BE[1] = arrivals_BE[1] + 1;
                return_time=0.000000475;
                }
                else if(random<=60){
                        arrivals_BE[2] = arrivals_BE[2] + 1;
                        return_time=0.000001054;
                        }
                        else if(random<=65){
                                arrivals_BE[3] = arrivals_BE[3] + 1;
                                return_time=0.000001134;
                                }
                                else if(random<=100){
                                        arrivals_BE[4] = arrivals_BE[4] + 1;
                                        return_time=0.000001214;
                                        }
return return_time;
}
```

```
//This function returns the array position-->[0 1...5]=[51ns...1214.4ns]
int get_BE_position(double service_time){
int return position;

if (service_time == 0.000000051){
        return_position=0;
        }
        else if(service_time == 0.000000475){
                return position=1;
                }
                else if(service_time == 0.000001054){
                        return_position=2;
                        }
                        else if(service_time == 0.000001134){
                                return position=3;
                                }
                                else if(service_time == 0.000001214){
                                        return_position=4;
                                        }
return return_position;
}

void initialize() {
        ...

        arrivals_BE[0] = 0;
        arrivals_BE[1] = 0;
        arrivals_BE[2] = 0;
        arrivals_BE[3] = 0;
        arrivals_BE[4] = 0;
        BE_interrupted_distribution[0] = 0;
        BE_interrupted_distribution[1] = 0;
        BE_interrupted_distribution[2] = 0;
        BE_interrupted_distribution[3] = 0;
        BE_interrupted_distribution[4] = 0;
}

if (e->type == ARRIVAL) {
// The event is an arrival: increment the counters and the state
        tot_arrivals++; arrivals[e->pclass]++;
        if (e->pclass == 0 || e->pclass == 1 || (e->pclass == 2 && q[e->pclass] == NULL))
{
// Schedule the new packet/burst and return the output channel assigned to it
        outputch = schedule_arrival(now,e->pclass,e->inputch);
        if (outputch != -1) { // Channel found!
        if (now < tfree[outputch]) {
        // An ongoing transmission seems to be interrupted
        if (lastpktclass[outputch] != 2) {
// No interruption of GS or RT by new GS because of the fixed delay
        if (tfree[outputch]-now > fixed_delay) {
        fprintf(stdout,"ERROR: GS burst or RT packet interrupted!!!\n");
        fprintf(stdout,"%d\n",lastpktclass[outputch]);
                exit(-1);
        }
        } else {  // BE is always interrupted by RT
// BE is interrupted by GS only if BE ends transmission after the fixed delay
        if (e->pclass == 1 || (e->pclass == 0 && (tfree[outputch]-now > fixed_delay))){
                ...

                int position = get BE position(e1->service);
                BE_interrupted_distribution[position] =
                BE_interrupted_distribution[position] + 1;

                }
        }
}
```

5.3.5. Segmentation of packets

Despite the previous changes regarding the different BE packet length used, it can be proved what happens if we split packets and use a fixed length for all packets. In this way, it could bring us some benefits as a throughput approximate to 1 as shown in Figure 29. The channel will be always transmitting packets.

Fig. 29. High channel utilization (high throughput)

But we should count with the headers added to each packet. This headers waste a time sending useless bytes. Then, is it better to use a fixed length?

The target is to split the previous packets until getting approximately the same length before sending them through the channel. The packets of 51ns don't need to be divided because we are going to assume 99.2ns as the new fixed service time for all divided packets.

The header is 38B = 7B (preamble) +1B (SFD) +6B (source MAC address) +6B (destination MAC address) +2B (Ethernet type) +4B (FCS) +12B (IPG). Each time we split a packet we should add 38B. An example is shown below in Figure 30.

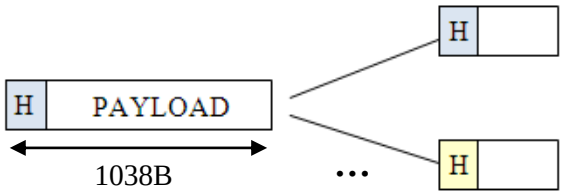


Fig. 30. Packet segmentation

If we had a packet which length is 1038B, 1000B would correspond to the payload and 38B to the header. Each time we split the packet we should add 38B to this length: 1038B+38B+38B…

To compute the payload and the number of packets:

1. Basic payload = $\frac{99.2\text{ns} * 10\text{Gbps}}{8\ bits}$ – Header

2. Payload = $\frac{service\ time * 10Gbps}{8\ bits}$ - Header

3. Packets number = $\left\lceil \frac{\text{Payload}}{\text{Basic Payload}} \right\rceil$

Furthermore, it is necessary to add some identifiers (sequence IDs) to the fragments to count the headers.

```
//define the header you add when you split a packet - 38B
#define HEADER 38
#define OUTPUT_LINE_RATE 10000000000

// Structure representing an event
struct event {
        double time;      // Event occurrence time
        char type;        // Event type
        char pclass;      // Class of the customer involved in the event
        char inputch;     // Input channel(relevant for classGS,outp chann for DEPARTURES)
        double service;   // Service time, in case the event is an arrival
        int sequenceID;   // fragments of the same packet have the same ID
        event *next;      // Pointer to the next event in the list
};
// Structure representing a customer waiting in the queue
struct customer {
        double t_arr;     // Time of arrival
        double service;   // Service time
        int sequenceID;   // fragments of the same packet have the same ID
        customer *next;   // Pointer to the next customer in the list
};
float new_service_time_BE;
double timeNextArrivalEvent;
int arrivalsFragments;
int header_count_departed;//counts the number of fragment headers (departures)
int header_count_total;//counts the number of headers

void insert_new_event(double time, char type, char pclass, char inputch, double service,
int sequenceID) {
        // Create the new event using the provided arguments
        w3 = (struct event *) malloc(sizeof(struct event));
        w3->time = time;
        w3->type = type;
        w3->pclass = pclass;
        w3->inputch = inputch;
        w3->service = service;
        w3->sequenceID = sequenceID;
        ...
}
double get_BE_service_time_discrete(){
int random = rand() % 101;
double return_time;        //probability=[45 10 5 5 35]--> [45 55 60 65 100];
if (random <=45){
  return_time=0.000000051;
  arrivals_BE[0] = arrivals_BE[0] + 1;
  }
  else if(random<=55){
        return_time=0.000000475;
        float payload_basic=((new_service_time_BE*OUTPUT_LINE_RATE)/8)- HEADER;
        float payload=((return_time*OUTPUT_LINE_RATE)/8)-HEADER;
        int number_packets=ceil(double(payload/payload_basic));
        arrivals_BE[1] = arrivals_BE[1] + number_packets;
        }
        else if(random<=60){
          return_time=0.000001054;
          float payload_basic=((new_service_time_BE*OUTPUT_LINE_RATE)/8)- HEADER;
          float payload=((return_time*OUTPUT_LINE_RATE)/8)-HEADER;
          int number_packets=ceil(double(payload/payload_basic));
          arrivals_BE[2] = arrivals_BE[2] + number_packets;
            }
            else if(random<=65){
              return_time=0.000001134;
              float payload_basic=((new_service_time_BE*OUTPUT_LINE_RATE)/8)- HEADER;
              float payload=((return_time*OUTPUT_LINE_RATE)/8)-HEADER;
              int number_packets=ceil(double(payload/payload_basic));
              arrivals_BE[3] = arrivals_BE[3] + number_packets;
                }
                else if(random<=100){
                 return_time=0.000001214;
                 float payload_basic=((new_service_time_BE*OUTPUT_LINE_RATE)/8)- HEADER;
                 float payload=((return_time*OUTPUT_LINE_RATE)/8)-HEADER;
                 int number_packets=ceil(double(payload/payload_basic));
                 arrivals_BE[4] = arrivals_BE[4] + number_packets;
                   }
        return return_time;
}
```

```
// Initialization function
  void initialize() {
  int j,ch;
  // Initialize all the global variables and counters
...
  header_count_departed = 0;  header_count_total = 0;
...
// Create a GS arrival for each input channel (one on-off source per channel)
for (ch = 0; ch < Nserv; ch++) {
        double temp = serv(j);//test
        printf("INIT GS service time = %d",temp);
        insert_new_event(expon(1.0/Toff),ARRIVAL,j,ch,temp,0);
        }
} else {
// Create an arrival for each remaining class (RT and BE, one source per class)
        double temp = serv(j);//test
        printf("INIT BE service time = %1.20f",temp);

if(j==1){
     insert_new_event(expon(lambda[j]),ARRIVAL,j,-1,temp,0);
        }
if(j==2){
     timeNextArrivalEvent = expon(lambda[j]);
     insert_new_event(timeNextArrivalEvent,ARRIVAL,j,-1,fixed_delay,header_count_total);
        //initilized with a packet size equal to fixed_delay to avoid collision (largest
        fragment size)
        }
}

// Start processing the first event in the list and set the current time
        e = get_event();
        now = e->time;
        if(e->type == ARRIVAL){//The event is an arrival:increment the counters and the state
                if(e->pclass == 2){
                        arrivalsFragments++;
                        if(timeNextArrivalEvent==now){//changed to avoid to count fragments
                                tot_arrivals++; arrivals[e->pclass]++;
                        }
                }else{
                        tot_arrivals++; arrivals[e->pclass]++;
                }
if (e->pclass == 0 || e->pclass == 1 || (e->pclass == 2 && q[e->pclass] == NULL)) {
                ...
// The packet/burst begins transmission and a departure event is created and inserted to
the list
                ...
                if (e->pclass == 0) {
                // GS: add a fixed delay equal to a RT packet time
                        tfree[outputch] += fixed_delay;
                }
                lastpktclass[outputch] = e->pclass;
insert_new_event(tfree[outputch],DEPARTURE,e->pclass,outputch,e->service,e->sequenceID);
        } else {
// Channel not found (will never happen to GS bursts)
        // RT packet is blocked, BE packet is queued
                if (e->pclass == 1) {
                        RT_blocked++;
                }
                else if (e->pclass == 2) {
                        k[e->pclass]++;
                        append(e->pclass,now,e->service,e->sequenceID);
                }
        }
} else {// BE packet is queued
        k[e->pclass]++;
        append(e->pclass,now,e->service,e->sequenceID);
        }
if (tot_arrivals < N) {  // There are more arrivals to be generated: create a new one
and insert it in the event list
    if (e->pclass == 0) {
                ...
    insert_new_event(now + e->service + val,ARRIVAL,e->pclass,e->inputch,temp,0);

    } else // RT or BE: generate inter-arrival time
        double temp = serv(e->pclass);//test
        new_service_time_BE=0.0000000992;
        double interarrival = expon(lambda[e->pclass]);
```

```
        if(timeNextArrivalEvent == now){
                if(temp > new_service_time_BE){
                float payload_basic= ((new_service_time_BE*OUTPUT_LINE_RATE)/8)- HEADER;

                float payload=((temp*OUTPUT_LINE_RATE)/8)-HEADER;
                int number_packets=ceil(double(payload/payload_basic));

                int i=0;
                for(i=1; i<=number_packets; i++){
                   insert new event(interarrival+(e->time+((i-
1)*new_service_time_BE)),ARRIVAL,2,e->inputch,new_service_time_BE,header_count_total);
                   timeNextArrivalEvent = interarrival+now;
                     }
                }
        else{
                 insert new event(now + interarrival,ARRIVAL,e->pclass,e-
>inputch,temp,header_count_total);
                timeNextArrivalEvent = interarrival+now;
                }
        header_count_total = header_count_total + 1;
        }
}

} else if (e->type == DEPARTURE) {
// The event is a successful departure: increment the counters and decrement the state
                ...
                //counts the real packets, header_count_departed is incremented only when
all the fragments of a packet are departed.
                if(header_count_departed < e->sequenceID){
                        header_count_departed = header_count_departed + 1;
                }
        }
        lastdeptime = now;
        if (now >= tfree[e->inputch] && q[2] != NULL) { // Output channel is free and BE
queue is not empty: put the next queued customer in service
                j = 0;
                while (q[j] == NULL) j++;  // Find the first non-empty waiting list
(should always be j=2)
                ...
                insert_new_event(tfree[outputch],DEPARTURE,j,outputch,c->service,c-
>sequenceID);  // Create a departure event and insert it into the list
                ...

if (DEBUG == 0)
        printf("w[2] %1.20f\n\n",w[2]);

        // Update and print some variables
        for (j = 0; j < C; j++) {
        //tot_w = tot_w + w[j];
        // Update the sum of the overall waiting time
        //we should account for all fragments when j==2
        if(j==2){
                w[j] = w[j]/(arrivalsFragments-k[j]);
        }
        else{
w[j] = w[j]/(arrivals[j]-k[j]);  // Compute the mean waiting time for class j
        }
        printf("w[2] %1.20f\n\n",w[2]);

BE_int_rate = (double)BE_interrupted/(arrivalsFragments-k[2]);
BE_succ_rate = (double)BE_successful/(arrivalsFragments-k[2]);
```

Chapter 6

Numerical Results and Graphs

In this chapter, the results, which are obtained by simulation of the previous changes, are analyzed. Several graphs show the performance of BE traffic and its impact on the efficiency network. Moreover, some useful comparisons are realized between the simulation changes.

6.1. Service time function results

6.1.1. Success rate of the BE traffic

First, the idea is to compare the BE success rate for different BE service times. As it has been explained in the previous section the GS distribution is deterministic then, the results are analyzed changing the BE distribution. The graphs show the results by using BE exponential distribution in one hand and the new distribution *get_BE_service_time_discrete()* in the other hand. With this new BE function, the packets duration are 51 ns, 475 ns, 1054 ns, 1134 ns and 1214 ns.

Figure 31 shows the success rate of the BE traffic as a function of Lp, for the first six CPRI options. The colored traces show the results using BE exponential distribution varying Lb and the black trace shows the results for the new distribution. Only one server is used and $\rho_b = 1$, which means that there are always BE packets in the queue to be transmitted.

The input parameters for the options are shown in Annex II. For option 7 the gap is so small that we cannot put any payload inside, thus we don't consider neither it nor any of the following options. There is also a problem using option 6 with Lp=200B because the gap is too small to fill it with packets so we cannot send an packet, then, the BE success rate is 0 as we see in the graph.

As we can see in the tables, encapsulation delay increases with larger Lp then, it will increase the size of the gap. Furthermore, TE is also higher with larger Lp values.
The service time θb is changed for the first distribution, where 160ns, 320ns, 480ns, 640ns and 800ns are the values and for the second distribution θb is 604.75ns because we are changing the inter-arrival time with the new function.

The values for Lb are as follows:

Lb (Bytes) = 160ns*10Gbps = 200B
Lb(Bytes) = 320ns*10Gbps = 400B
Lb(Bytes) = 480ns*10Gbps = 600B
Lb(Bytes) = 640ns*10Gbps = 800B
Lb(Bytes) = 800ns*10Gbps = 1000B

604.75 ns is the average service time:

$\theta b = 51*45 + 475*10 + 1054*5 + 1134*5 + 1214*35 = \frac{60475}{100} = 604.75ns$

Lb(Bytes) = 604.75ns *10Gbps = 755.93B

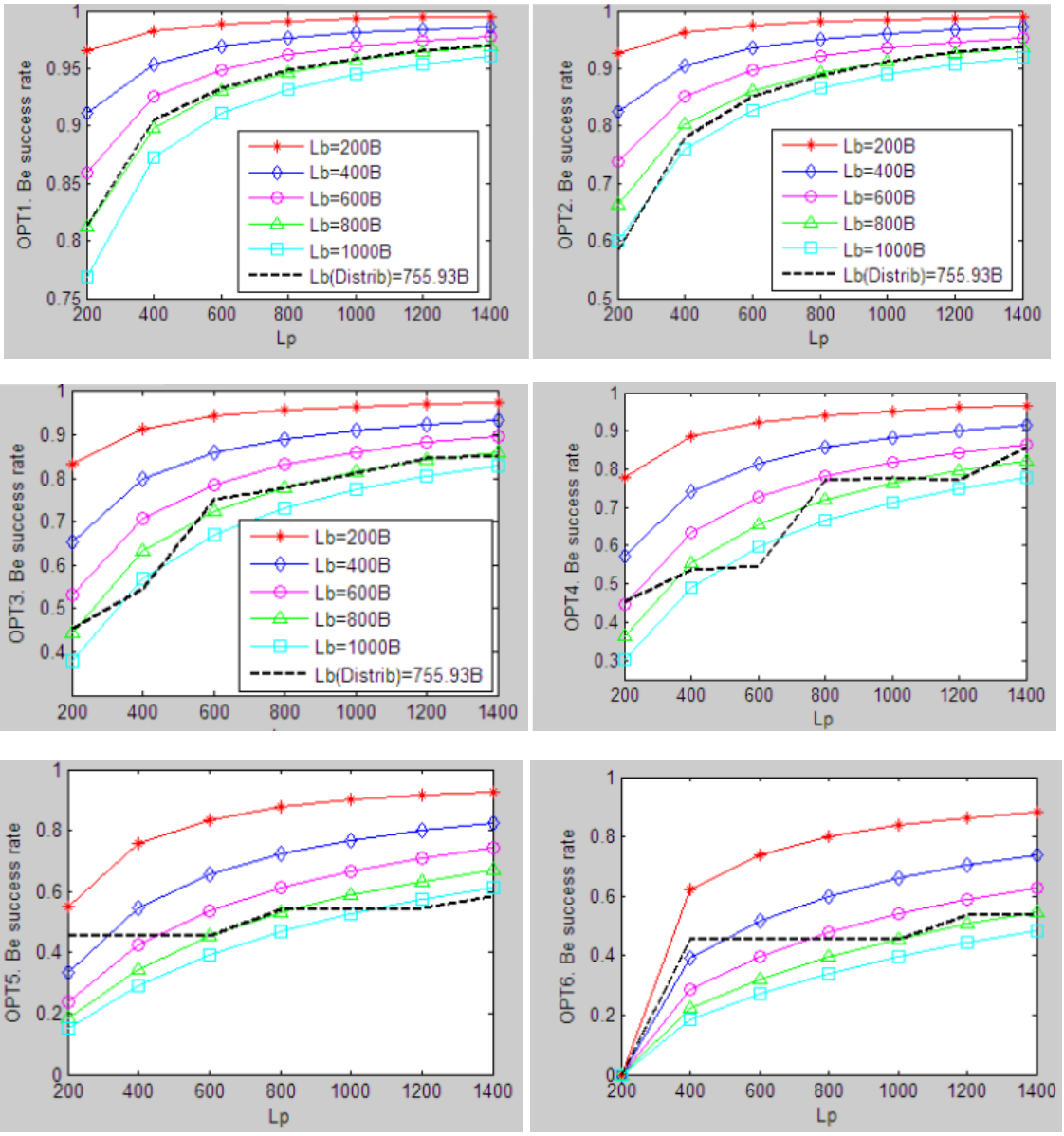


Fig. 31. BE success rate as a function of Lp for different BE payloads using CPRI option 1 to option 6 and both distributions

As a consequence of the small gap that the last options provide, the BE success rate, which depends on the gap size, is lower as we approach to option 6. In the case of

option 1 the BE success rate is high due to the large gap size thus, the gap can be filled with many packets. With option 6, the gap is small, hence the gap is filled with a few packets, and then the rate is low. We can observe this behavior for both distributions.

The growing traces occur because the gap is larger as Lp is higher. With Lp = 200B we can suit less packets than using Lp = 1400B, then the success rate of the BE traffic is greater for Lp=1400B.

Regarding θb, when its value is low the graphs show high BE success rate, as for example in the red trace. This is because with lower inter-arrival time θb, the packets arrive fast and are sent quickly but if θb is higher, the time between packets will be large and the BE success rate will decrease.

The results for the new distribution (black trace) aren't worse than the results of the worst-case for the other distribution but for some options is better the BE exponential distribution. The slope changes are due to the packet collision; for example we can see it in the graph for option 4, where in some points the results don't grow properly until the next change in Lp.

In the following graph are shown the curves just for the new distribution, the previous black traces together in one graph:

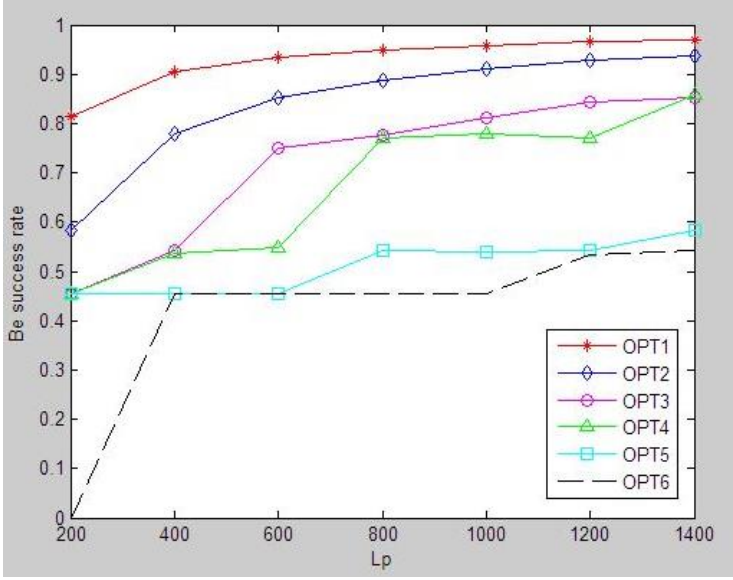


Fig. 32. BE success rate as a function of Lp for different options using the new distribution

We can compare the new distribution results with the trace of 800B of the exponential distribution, because they use approximately the same Lb.

1. Exponential function – θb=640ns – Lb=800B

2. New distribution – θb=604.75ns – Lb= 755.93B

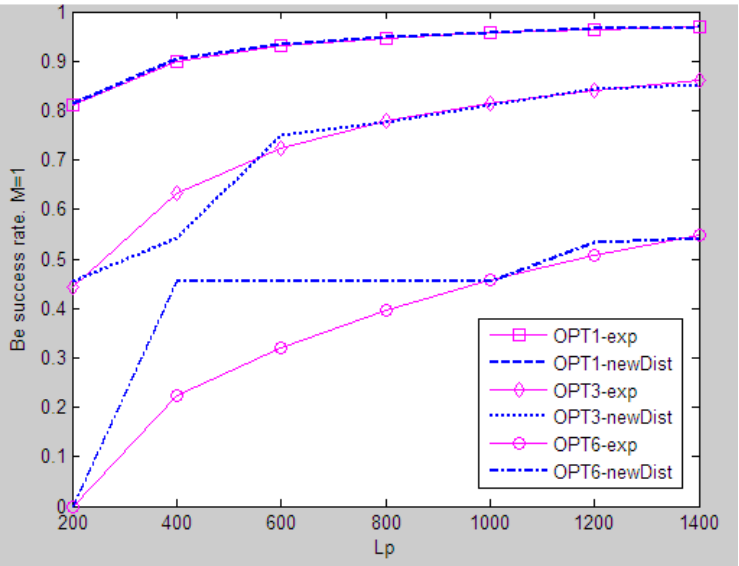
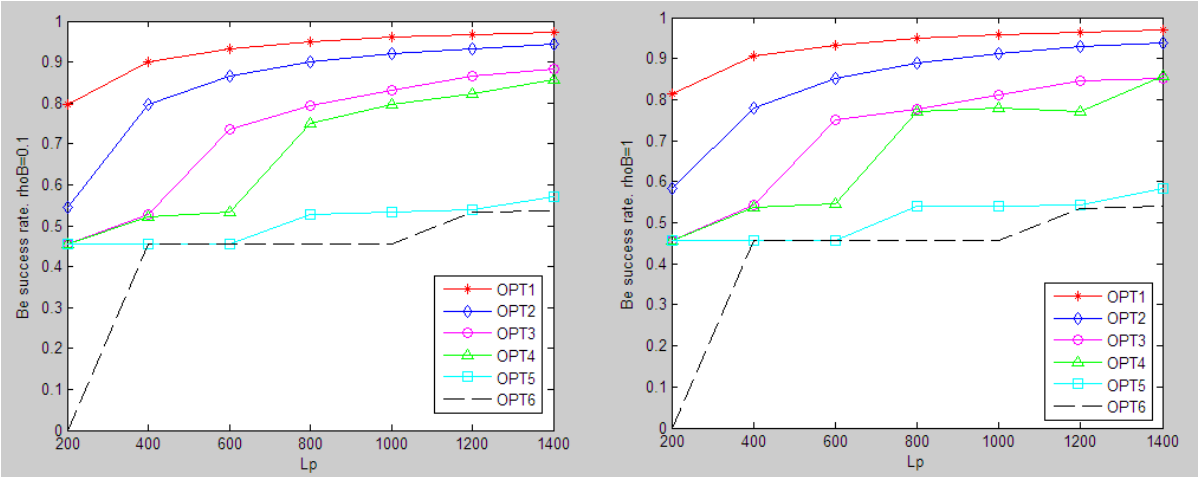


Fig. 33. Comparison between BE success rates for both distributions

As we see in Figure 33, values of the new distribution are above the exponential distribution results, getting a better performance.

In addition, ρ_b may vary from 0 to 1. ρ_b is the offered BE load per channel, and this relates to the amount of packets in the queue, where 1 is the maximum and means that there are always packets in the queue. The graphs below shown both performances but they don't show major changes, just a few more collisions with $\rho_b = 1$, for example as we see for option 4.



a) b)

Fig. 34. BE success rate as a function of Lp for different options using the new distribution. a) $\rho_b = 0.1$, b) $\rho_b = 1$

6.1.2. BE throughput

The BE channel utilization or throughput is also measured. The results for the new distribution are shown in Figure 35, where the last two options indicate a low utilization but to have a real vision of this parameter we can compare it with the results of the exponential distribution, as in Figure 36. This comparison is done as in Figure 32, with the following traces:

1. Exponential function – θb=640ns – Lb=800B

2. New distribution – θb=604.75ns – Lb= 755.93B

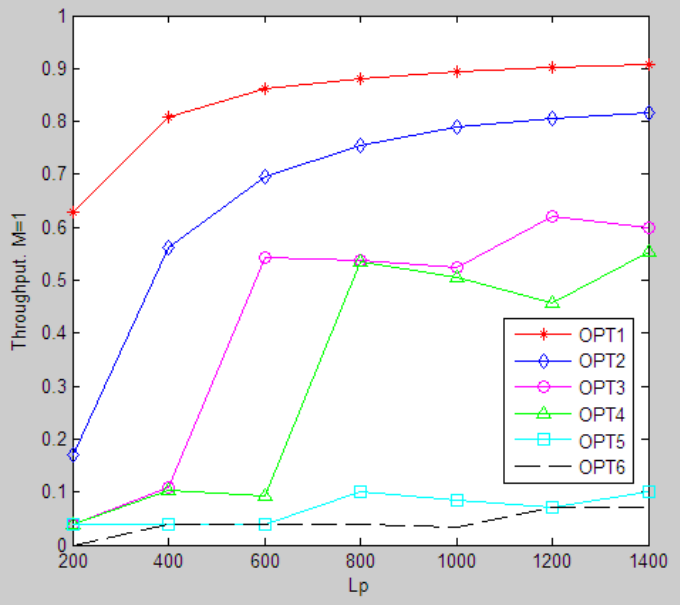


Fig. 35. BE throughput as a function of Lp for different options and the new distribution. $\rho_b = 1$

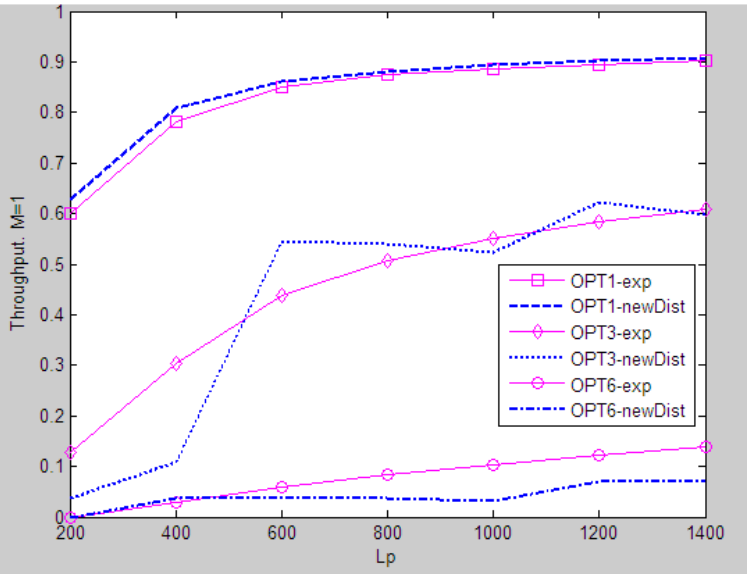
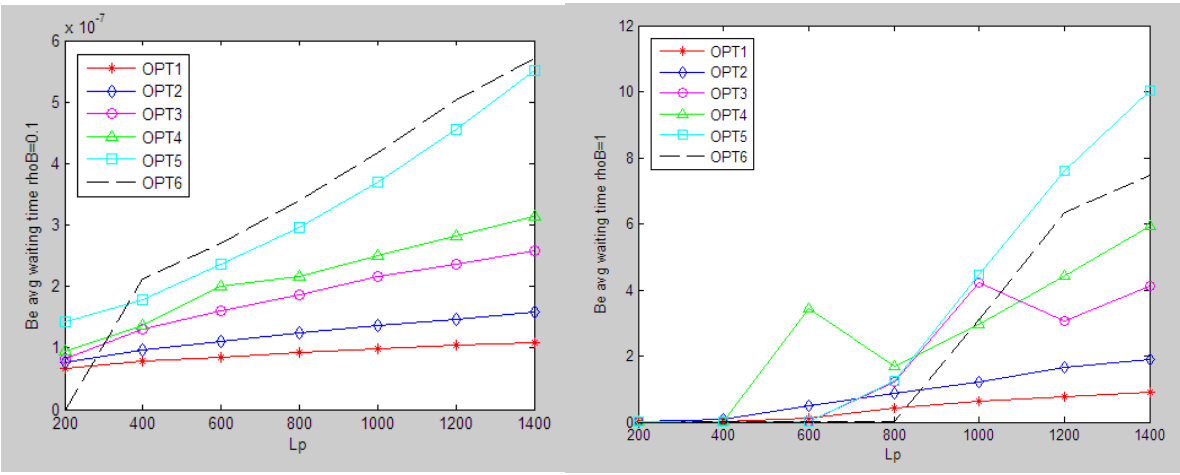


Fig. 36. Comparison between throughputs for both distributions

The throughput follows the trend of the success rate. In some cases it is better than the exponential distribution throughput but for the last options it is worse. The influence of the option chosen is meaningful as well as the payload Lp.

6.1.3. BE packet average waiting time

The model explained in Chapter 5 involves queues for BE traffic. We can measure the time that packets spend in the queue until they are transmitted. For example, if there is one server and $\rho_b = 0.1$ the queue is not saturated and the packets spend a little time in the queue as we see in Figure 37(a), in the order of 10^{-7} seconds. But with the queue always receiving packets, always full, the time reaches the order of seconds and it is considerably greater than the previous one, as in Figure 37(b).



a) b)

Fig. 37. BE packet average waiting time as a function of Lp for different options using the new distribution. a) $\rho_b = 0.1$, b) $\rho_b = 1$

This parameter can also be measured having several output channels: one, two or five output channels, for example. The larger channel number you have, the shortest time that the packets will spent in the queue.

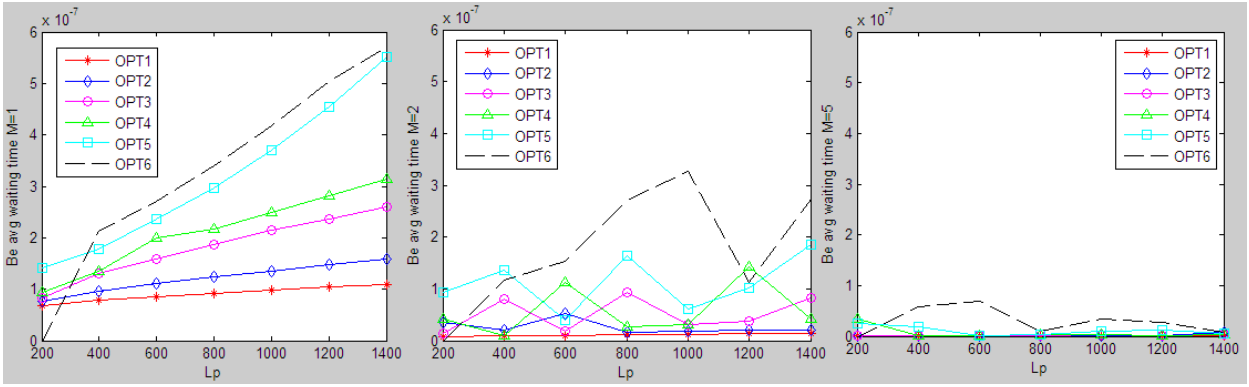


Fig. 38. BE packet average waiting time depending on the servers number. M=1, 2 or 5.

Figure 38 shows this behavior, with five output channels the time spend in the queue by the packets decrease if you compare with the results for one and two servers.

6.2. Interrupted packet rates and graphs

Some useful information could be the percent of interrupted packets depending the duration of packets. In this way, we can asses if the packet duration is appropriate and the performance is suitable. The definitions of the different rates were explained before in section 5.3.4. The following graphs show the results for different rates and CPRI options.

$$Ratio1 = BEinterrupted[i]/arrivals\_BE[i]$$

Varies from 0 to 1:
// Ratio approximate to 1 means all packets are interrupted
// Ratio approximate to 0 means all packets are successfully sent

CPRI Option 1

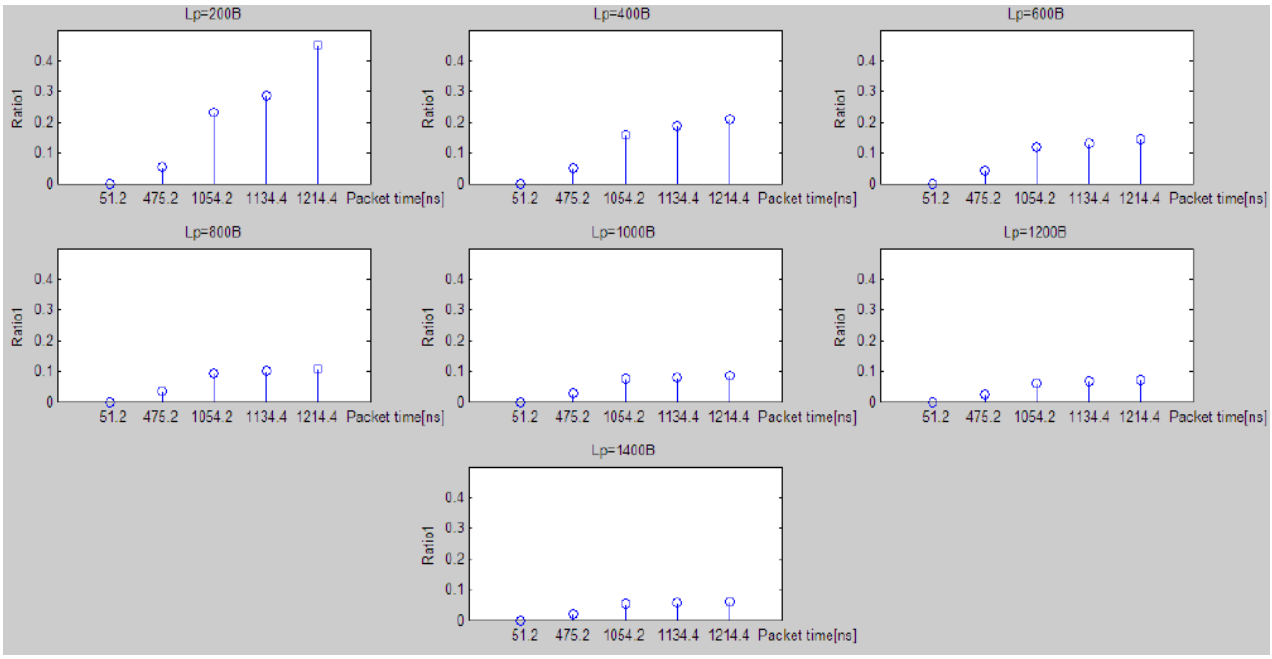


Fig. 39. Ratio1 CPRI Option 1

As Lp is higher, there are fewer interrupted packets due to the larger gap. Moreover, packets, which duration time is small, have less difficulty to suit in the channel, and larger duration packets have higher probability to be interrupted.

CPRI Option 3

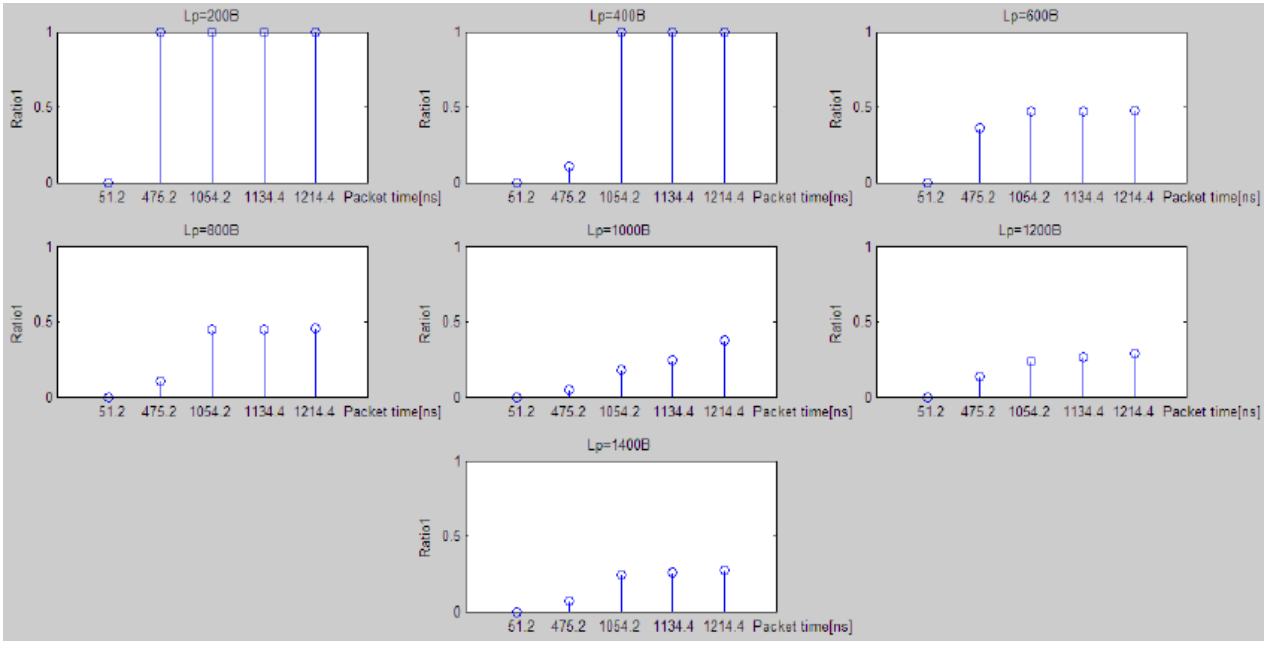


Fig. 40. Ratio1 CPRI Option 3

As we approach CPRI option 6, the gap becomes smaller then, larger packets are interrupted with elevate probability. It is the case show in Figure 40 with Lp=200B, the gap is only filled with 51.2ns packets, the smallest one. All other packets are interrupted. But with Lp=1400B the gap is larger than 200B option and thus the interrupted packet rate for larger packets decreases.

CPRI Option 6

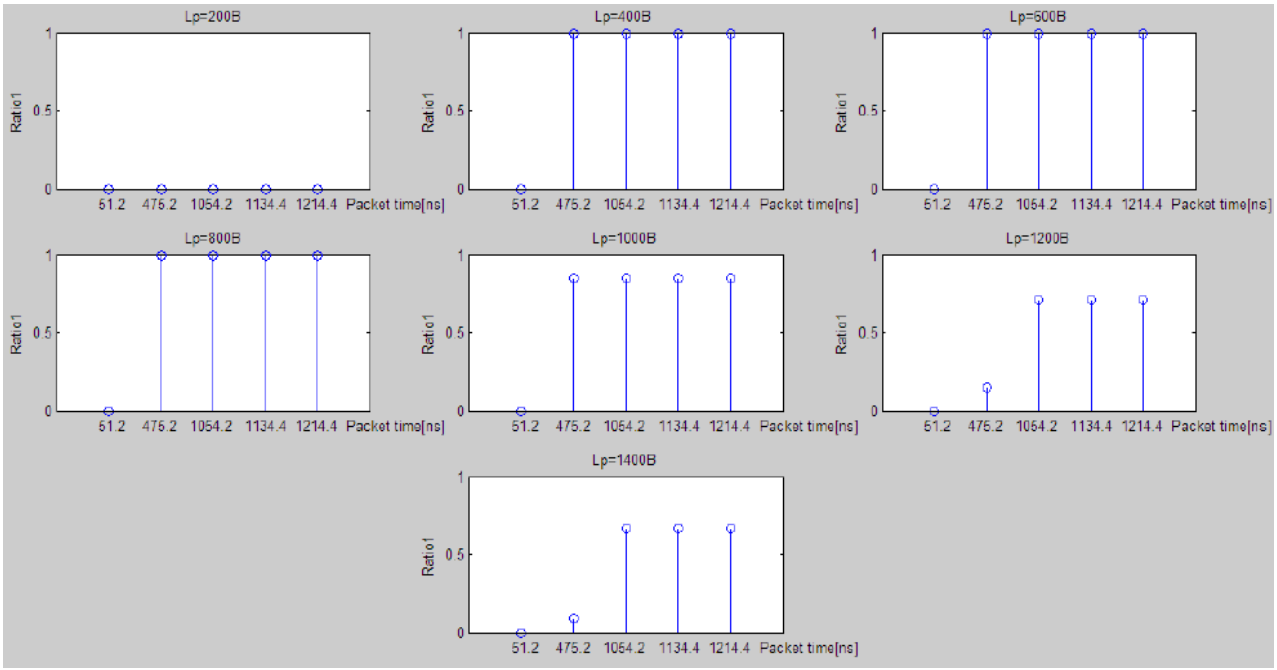


Fig. 41. Ratio1 CPRI Option 6

For option 6, which is the option with the smallest gap, we see several graphs with elevate interrupted packet rate. With Lp=200B we cannot transmit any packet, then it is zero interrupted packet rate and zero successful packet rate. With Lp=1400B, almost all packets are transmitted for the first two packet durations but for the last three durations, the interrupted packet rate continues to be elevate.

We can also calculate the interrupted packet rate over the total BE arrivals:

Ratio2 = BEinterrupted[]/arrivalsBE[k=2]

Varies from 0 to 1:
// Ratio approximate to 1 means all packets are interrupted
// Ratio approximate to 0 means all packets are successfully sent

CPRI Option 1

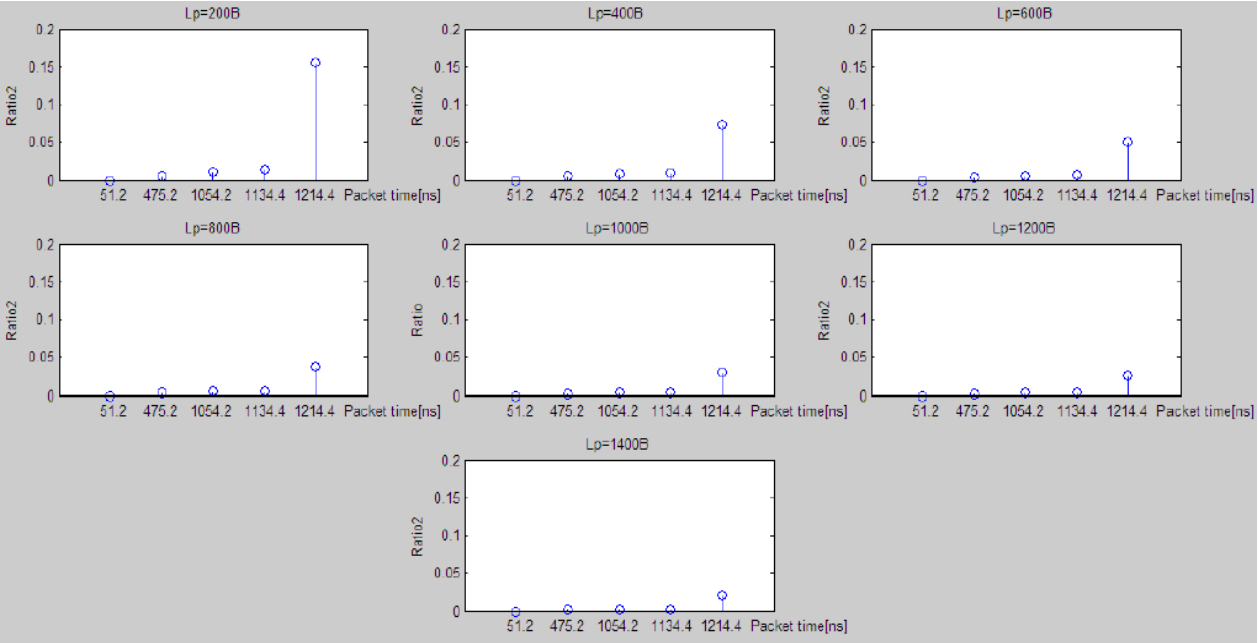


Fig. 42. Ratio2 CPRI Option 1

We can see in the graphs that the highest interrupted packet rate over total BE arrivals is for 1214ns, which is the largest packet. Good performance with Lp=1400B seeing that the results are quite well.

CPRI Option 3

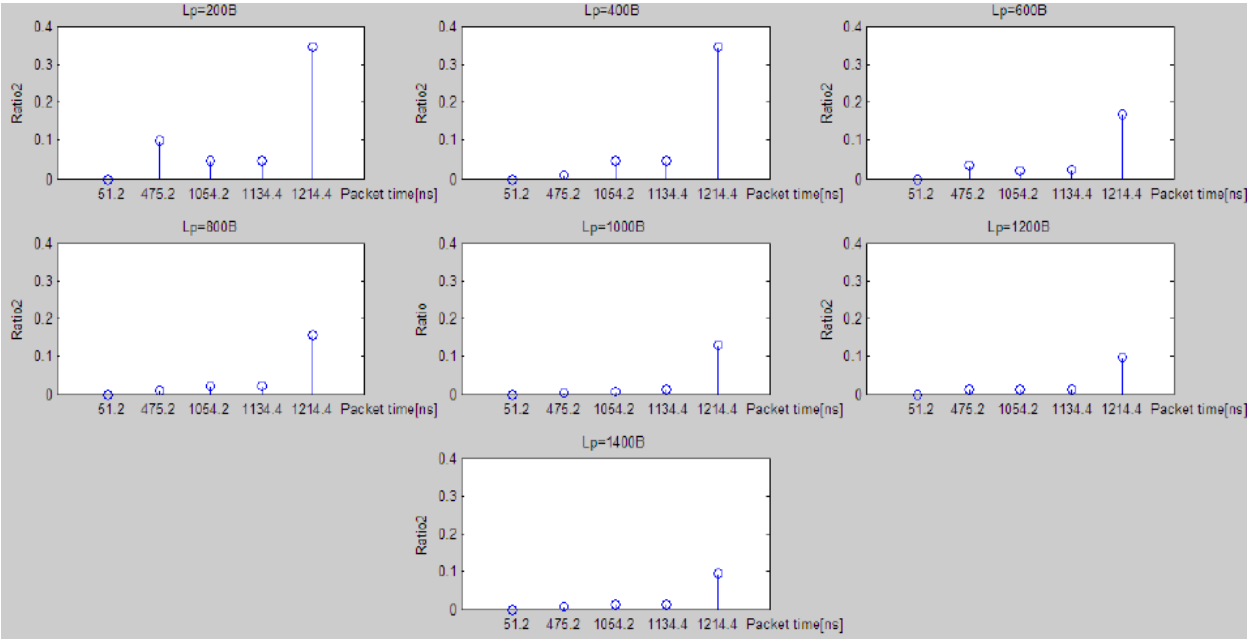


Fig. 43. Ratio2 CPRI Option 3

Regarding option 3, we can see the same behavior than the previous option except for Lp=200B where there are an increase in interrupted packets rate.

CPRI Option 6

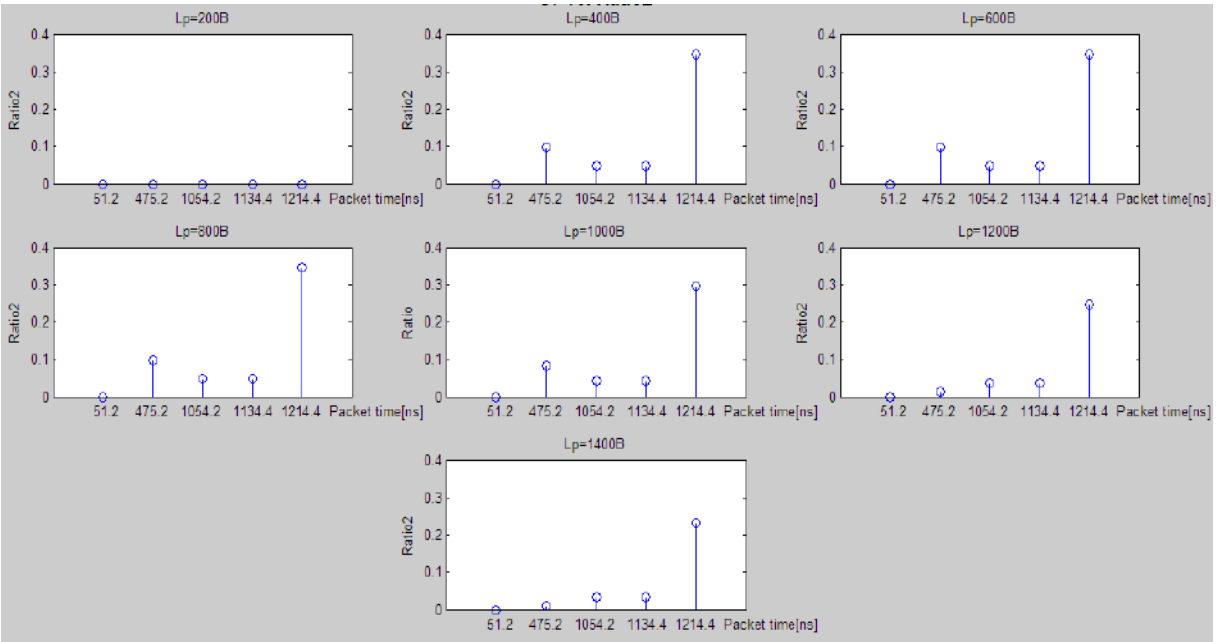


Fig. 44. Ratio2 CPRI Option 6

Same behavior is seen as before graph. The rate is higher because we are using option 6. Then we can say that more interrupted packets over the total are 475ns and 1214 ns packets.

Other kind of graphs could be the successful packets rate but it is just the inverse of interrupted packets rate. For example, where we have 0.6 of interrupted packet rate, we are going to have 0.4 of successful packet rate, and vice versa. I think it is unnecessary to show more graphs about it.

$$SuccessfulRatio1 = 1 - Ratio1$$
$$SuccessfulRatio2 = 1 - Ratio2$$

But a comparison can be useful to see the changes. For instance, Figure 45 shows the Ratio 1 for four CPRI options. We can see the progressive worsening of sent packets depending on each duration.

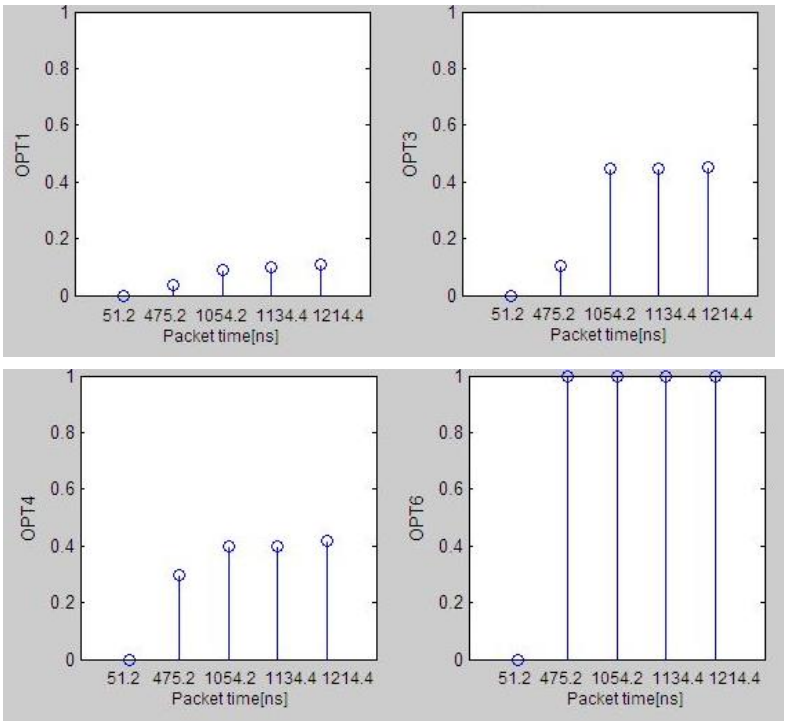


Fig. 45. Comparison Ratio 1

Or we can also see the following graphs, where the Ratio 2 is shown. The interrupted packet rate over the total BE arrivals is remarkable for packets of 1214ns duration.

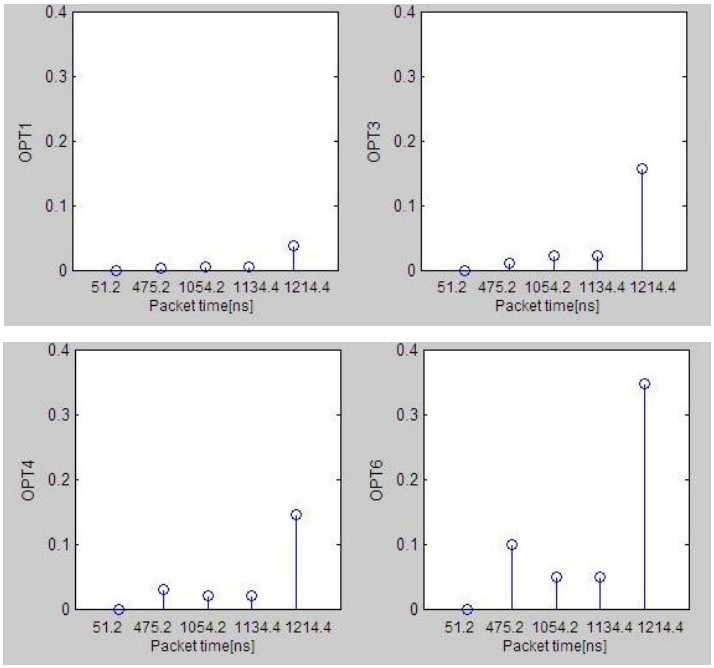


Fig. 46. Comparison Ratio 2

6.3. Split packets graphs

In this section, a fixed length for all packets can be proved. To get this, we should split the packets until a fixed duration: 99.2ns approximately. It may bring us some benefits as a throughput approximate to 1. But we have to take into account the headers added to each fragment. The headers waste a time sending useless bytes. But, how great is this time?

For example for CPRI option 1, we obtained the following BE success rate and throughput, both really high and appropriate to the expected behavior.

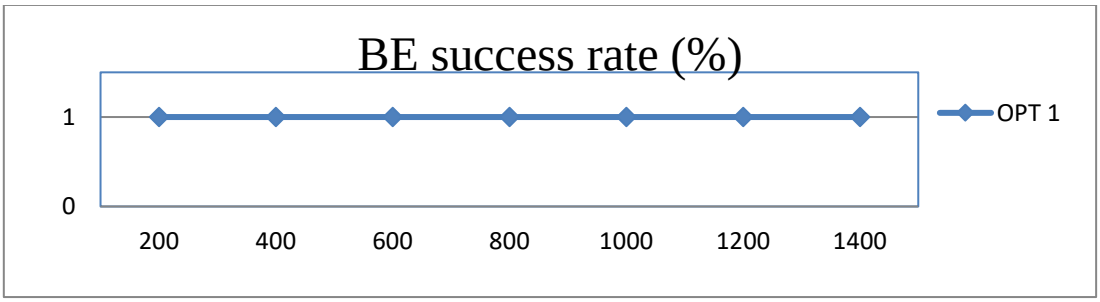


Fig. 47. BE successs rate with fragmentation. CPRRI Option 1

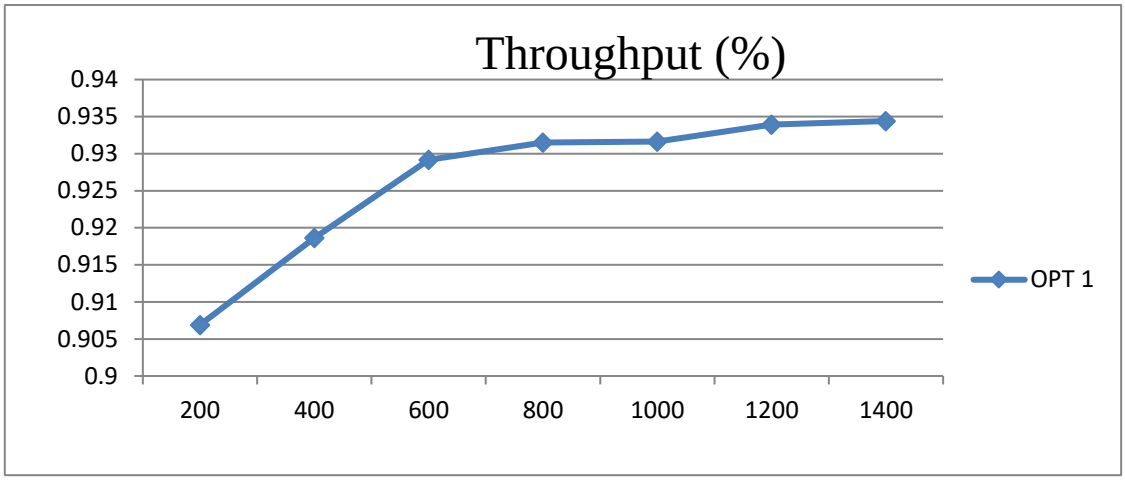


Fig. 48. Throughput with fragmentation. CPRRI Option 1

But due to the high number of fragments waiting to be forwarded, the BE packet average waiting time is substantial as we see in Figure 49.

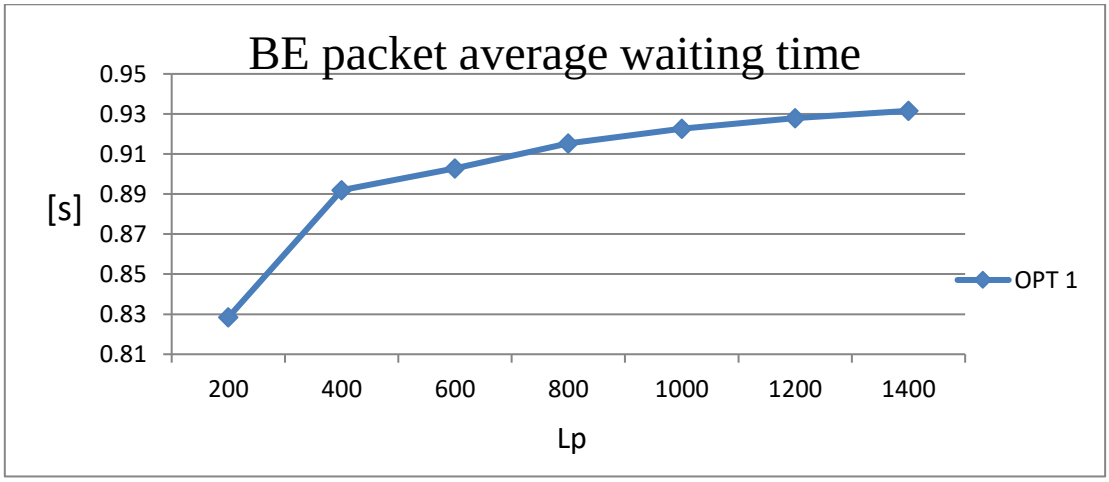


Fig. 49. BE packet average waiting time with fragmentation. CPRRI Option 1

Then, the comparison between the bytes used in fragment headers and the bytes used in packet headers without fragmentation are contrasted in Figure 50. There a huge amount of bytes wasted in fragment headers that it entails much more time to send the same useful information.

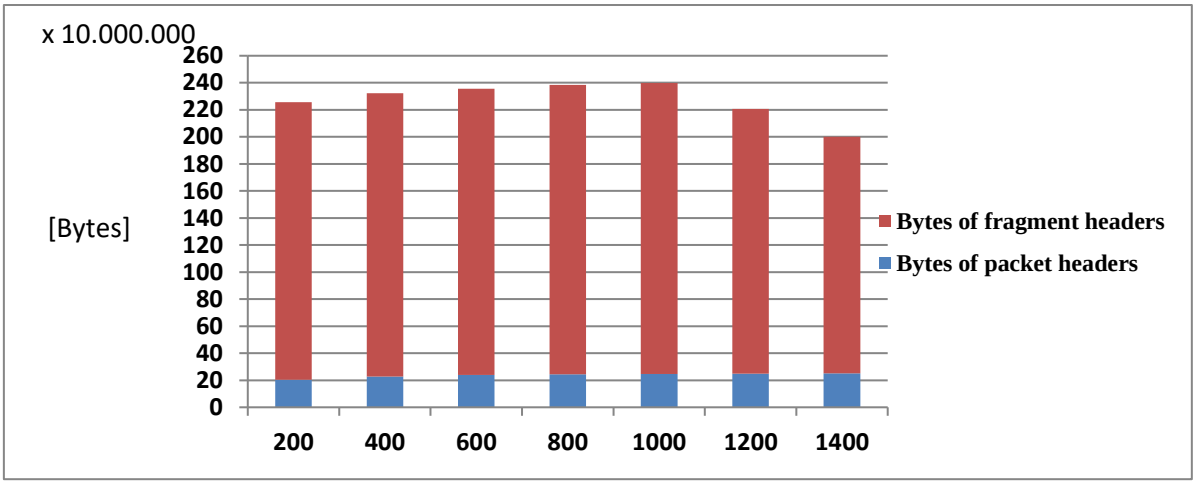


Fig. 50. Headers comparison. CPRI Option 1

With CPRI option 6, the results are similar. BE success rate to 1 as expected, low throughput due to option 6 and high BE packet average waiting time on the order of seconds.

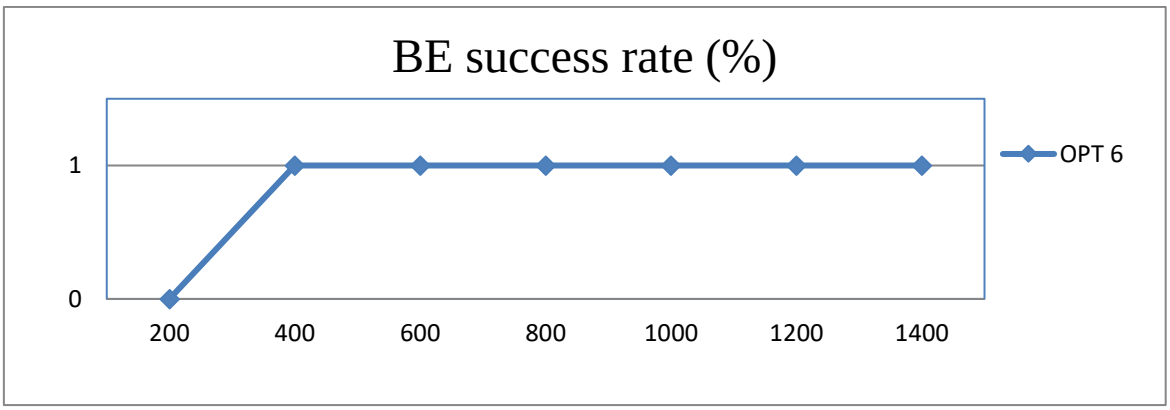


Fig. 51. BE success rate with fragmentation. CPRRI Option 6

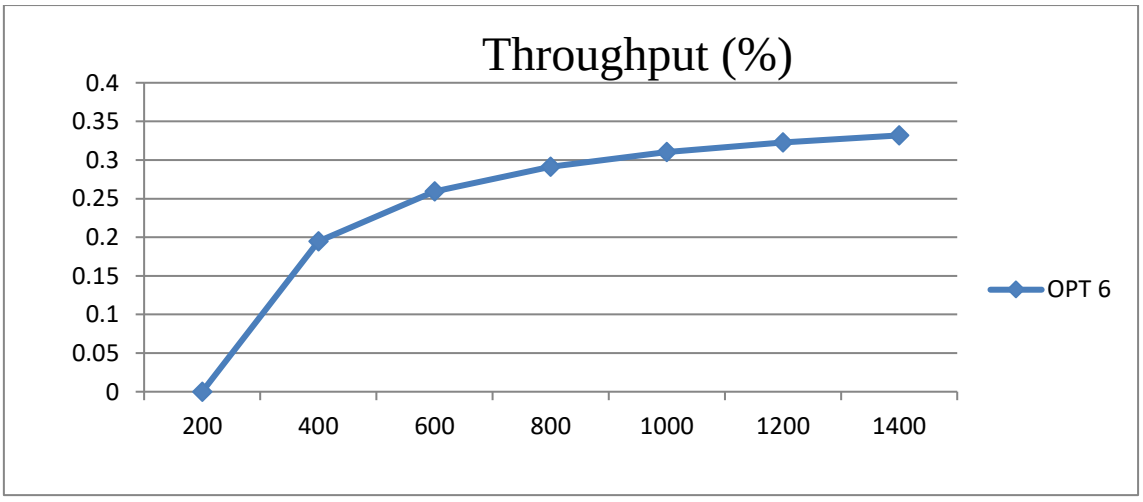


Fig. 52. Throughput with fragmentation. CPRRI Option 6

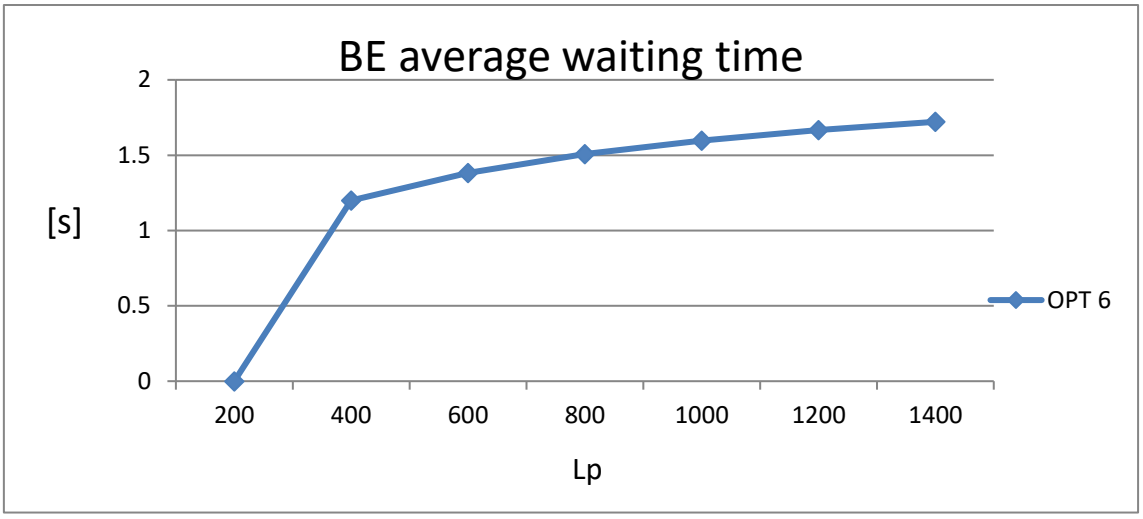


Fig. 53. BE packet average waiting time with fragmentation. CPRRI Option 6

Comparison between header bytes follows the same behavior. There a huge amount of bytes wasted in fragment headers but in this case the values are smaller due to the short gap this option provides. Figure 54 shows this comparison.

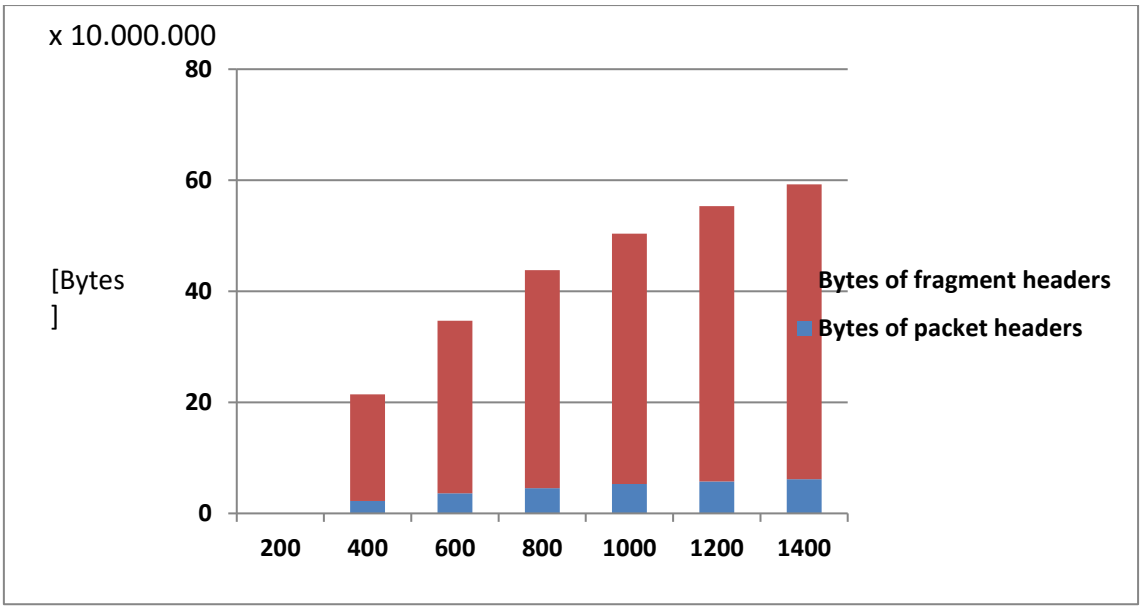


Fig. 54. Headers comparison. CPRI Option 6

Now, it is easy to calculate the header time employed to send the same amount of useful information with and without fragmentation.

CPRI Options	Lp	Headers time without fragmentation [s]	Headers time with fragmentation [s]	Wasted time (%)
Option 1	200	0.163	1.400	88.331
	400	0.182	1.566	88.335
	600	0.191	1.641	88.336
	800	0.195	1.675	88.336
	1000	0.197	1.694	88.329
	1200	0.199	1.711	88.342
	1400	0.201	1.720	88.335
Option 2	200	0.111	0.957	88.336
	400	0.136	1.166	88.333
	600	0.146	1.257	88.341
	800	0.152	1.309	88.341
	1000	0.156	1.345	88.344
	1200	0.159	1.365	88.334
	1400	0.161	1.388	88.333
Option 5	200	0.020	0.176	88.324
	400	0.032	0.274	88.333
	600	0.051	0.435	88.350
	800	0.061	0.521	88.336
	1000	0.062	0.538	88.333
	1200	0.070	0.602	88.341
	1400	0.073	0.634	88.340
Option 6	200	0	0	0
	400	0.017	0.153	88.334
	600	0.028	0.248	88.354
	800	0.036	0.313	88.347
	1000	0.042	0.361	88.324
	1200	0.046	0.396	88.334
	1400	0.049	0.424	88.338

Table 11. Header times and wasted time (%)

The table above shows the header time employed to send the same amount of useful information with and without fragmentation. Depending on the option, the time is larger or shorter because of the gap in which the packets are suit. The 88% of the time sending fragmentation packets is wasted. It is a lot of useless time. With packets fragmentation we can ensure a 100% of packets sent successfully but the time needed for that is too high. It is not worthwhile using this method.

Chapter 7

Conclusions and future work

7.1. Conclusions

The thesis presents several results for a simulator which is based in the CPRI over Ethernet transport model and integrated hybrid technology in Ethernet switches. It allows three services classes but only two are performed. Fronthaul traffic and backhaul traffic use the same optical Ethernet channel. CPRI over Ethernet is expected to provide many benefits to fronthaul networks.

The simulator performance has been studied by means of different parameters and CPRI options. Results as measured by success rate of BE packets, throughput or interrupted packets rate have seen evaluated. Furthermore, different scenarios or methods have been studied. The concept of collision avoidance has also been discussed.

Remarkable results are shown for CPRI option 1 as well as the performance for packets of duration smaller than or equal to 1134 ns. We also have to mention the different results depending on the Ethernet payload size, which are better when to approach 1400 Bytes.

It is noteworthy that with our new distribution, the results for CPRI option 5 and 6 can prove quite deficient. The throughput for these cases is lower than 10% of utilization. In some CPRI options, the BE success rate has been higher than using the BE exponential distribution, but only with some Ethernet payload sizes.

Moreover, regarding the average waiting time of BE packets in the queue, obviously, it decreases when more than one server are employed.

Additionally, the extra headers added with fragmentation have been measured. The results have demonstrated elevate wasted time in headers for this method.

7.2. Future work

As future lines of study, we can mention eCPRI specification which is the last update after CPRI technology.

Regarding the thesis work, CPRI compression could be useful to provide additional efficiency. Furthermore, we could consider packets with a size that is the same as the gap instead of multiple packets filling the gap. In addition, another idea could be run the simulator taking into account the Real Time (RT) traffic and see what happens to the BE success rate; it probably will decrease a few.

Some parameters could also be analyzed until find the optimal value, for example this is the case of the offered load per channel.

Another key challenge is to evaluate the real cost of the model used in a real environment as well as the network requirements explained in the papers; requirements such us latency, synchronization and jitter and so on. Reliability and security of the guaranteed service could be studied in detail too.

Simultaneously, different network topologies or mobile network architectures as H-RAN or F-RAN could be researched.

ANNEXES

ANNEX I. Original version code of the simulator.

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

// Define the event types
#define ARRIVAL 1
#define DEPARTURE 2
// Define the maximum number of traffic classes allowed by the simulator
#define MAX_CLASSES 10
// Define the maximum number of servers allowed by the simulator
#define MAX_NSERV 64
// Define whether or not to print debug messages
#define DEBUG 0

// Structure representing an event
struct event {
	double time;	// Event occurrence time
	char type;	// Event type
	char pclass;	// Class of the customer involved in the event
	char inputch;	// Input channel (relevant for class GS only) (output
channel for DEPARTURES)
	double service;	// Service time, in case the event is an arrival
	event *next;	// Pointer to the next event in the list
};

// Structure representing a customer waiting in the queue
struct customer {
	double t_arr;	// Time of arrival
	double service;	// Service time
	customer *next;	// Pointer to the next customer in the list
};

// Event list
struct event *event_list;
// Lists of waiting customers
struct customer *q[MAX_CLASSES];
// Pointers to the last customers in the queues
struct customer *lastq[MAX_CLASSES];

// Global variables
int seed; // Random seed
int N;	// Number of arrivals to be simulated
int C;	// Number of traffic classes
int Nserv;// Number of servers (e.g., wavelengths, channels, etc.)
int k[MAX_CLASSES]; // State array of the system = number of customers of each
class currently in the system
int tot_arrivals, tot_departures; // Counters for the total number of
arrivals/departures
int RT_blocked, BE_interrupted, BE_successful; // Counters for blocked RT and
interrupted/successful BE packets
int arrivals[MAX_CLASSES]; // Array of counters for the number of arrivals of
each class
int departures[MAX_CLASSES][MAX_NSERV]; // Array of counters for the number of
departures of each class per channel
double tfree[MAX_NSERV]; // Array of times when channels become idle
char lastpktclass[MAX_NSERV]; // Array with the class of last scheduled
service per channel
double now;	// Current time
double tot_w; // Overall mean waiting time
double w[MAX_CLASSES]; // Array of mean waiting times for each class
double lambda[MAX_CLASSES], mu[MAX_CLASSES]; // Arrays of arrival and service
rates for each class
```

```
double Toff; // Average GS off period (Ton = 1/mu[0])
double fixed_delay; // Fixed delay for GS bursts (equal to RT packet time)
double RT_loss_rate, BE_int_rate, BE_succ_rate; // RT loss prob. and BE
interruption/success prob.
double util[MAX_CLASSES][MAX_NSERV]; // Channel utilization per class
double utilBEint[MAX_NSERV]; // Channel utilization of interrupter DE traffic
double avgutil[MAX_CLASSES], avgutilBEint; // Average channel utilization
double servsucc[MAX_CLASSES][MAX_NSERV]; // Average successful service time
per class
double servint; // Average interrupted BE service time on the channel
double servsuccBE; // Average successful BE service time
double servBE; // Average BE service time on the channel (both successful and
interrupted)
double tot_serv_traffic[MAX_CLASSES]; // Total served traffic per class
int num_eventsBE[MAX_CLASSES]; // Number of events happening to BE packets by
each class (interruptions from GS and RT or successes for BE)

// This function inserts a new event in the event list ordered by time
// Arguments:
//     time = occurrence time of the new event
//     type = type of the new event
//     pclass = class of the customer involved in the new event
//     inputch = input channel, in case of GS arrival, 0 otherwise
//     service = service time of the arriving customer, in case the event is an
arrival, 0 otherwise
void insert_new_event(double time, char type, char pclass, char inputch,
double service) {
       // Declare some useful pointers
       struct event *w1,*w2,*w3;

       // Create the new event using the provided arguments
       w3 = (struct event *) malloc(sizeof(struct event));
       w3->time = time;
       w3->type = type;
       w3->pclass = pclass;
       w3->inputch = inputch;
       w3->service = service;
       if (DEBUG) fprintf(stderr,"DEBUG 12: New event object created:
%d\n",w3);

// Insert the new event in the event list keeping the correct order
       if(event_list == NULL) {  // The event list is empty: set it to point to
the new event
              w3->next = NULL;
              event_list = w3;
              if (DEBUG) fprintf(stderr,"DEBUG 13: Event list was empty, new
event added to list head: %d\n",event_list);
       } else if (event_list->time > w3->time) {  // The new event must be
inserted at the head of the list
              w3->next = event_list;
              event_list = w3;
              if (DEBUG) fprintf(stderr,"DEBUG 14: Event list was not empty,
new event added to list head: %d\n",event_list);
       } else {  // In all the other cases, move pointers w1 and w2 along the
list until the correct point is found,
                 // then insert the new event
              w1 = event_list;
              w2 = event_list->next;
              while ((w2 != NULL) && (w2->time <= w3->time)) {
                     w1 = w2;
                     w2 = w2->next;
              }
              w1->next = w3;
              w3->next = w2;
              if (DEBUG) fprintf(stderr,"DEBUG 15: Event list was not empty,
new event added to list\n",event_list);
       }
```

```
	if (DEBUG) fprintf(stderr,"DEBUG 01: New event inserted: time = %1.6f,
type = %s, class = %d, inputch = %d, service = %1.6f, next = %d\n",w3-
>time,(w3->type == 1 ? "ARRIVAL" : "DEPARTURE"),w3->pclass,w3->inputch,w3-
>service,w3->next);
}

// This function extracts the first event from the event list
struct event * get_event() {
	// Declare some useful pointers
	struct event *w3;
	// The list is empty and it should not be: generate an error
	if (event_list == NULL) {
		fprintf(stdout,"ERROR: event list is empty when it should not be
(get_event)\n");
		exit(-1);
	}

	// Return the first event from the list and update the list pointer
	w3 = event_list;
	event_list = w3->next;
	if (DEBUG) fprintf(stderr,"DEBUG 02: Next event extracted: time = %1.6f,
type = %s, class = %d, inputch = %d, service = %1.6f, next = %d\n",w3-
>time,(w3->type == 1 ? "ARRIVAL" : "DEPARTURE"),w3->pclass,w3->inputch,w3-
>service,w3->next);

	return w3;
}

// This function removes an event from the event list based on the given
arguments
struct event * remove_event(double time, char type, char pclass) {
	// Declare some useful pointers
	struct event *w1,*w2,*w3;

	if (event_list == NULL) {
	// The list is empty and it should not be: generate an error
		fprintf(stdout,"ERROR: event list is empty when it should not be
(remove_event)\n");
		exit(-1);
	} else if (event_list->time == time && event_list->type == type &&
event_list->pclass == pclass) {
		// The event to be removed is at the head of the list
		w3 = event_list;
		event_list = w3->next;
		if (DEBUG) fprintf(stderr,"DEBUG 21: Event to be removed found at
the head of the list: %d\n",w3);
	} else {
		// In all the other cases, move pointers w1 and w2 along the list
until the event is found, then remove the event
		w1 = event_list;
		w2 = event_list->next;
		while ((w2 != NULL) && !(w2->time == time && w2->type == type &&
w2->pclass == pclass)) {
			w1 = w2;
			w2 = w2->next;
		}
		if (w2 == NULL) {
			// Event not found: generate an error
			fprintf(stdout,"ERROR: event not found in the list
(remove_event)\n");
			exit(-1);
		} else {// Event found: remove it
			w3 = w2;
			w1->next = w2->next;
			if (DEBUG) fprintf(stderr,"DEBUG 22: Event to be removed
found in the list: %d\n",w3);
		}
	}
```

```
	// Return the event
	if (DEBUG) fprintf(stderr,"DEBUG 23: Event removed: time = %1.6f, type =
%s, class = %d, inputch = %d, service = %1.6f, next = %d\n",w3->time,(w3->type
== 1 ? "ARRIVAL" : "DEPARTURE"),w3->pclass,w3->inputch,w3->service,w3->next);
	return w3;
}

// This function appends a new customer to the list of waiting customers
according to the class
// Arguments:
//     pclass = class of the customer
//     time = arrival time of the customer
//     service = service time of the customer
void append(char pclass, double time, double service) {
	// Declare some useful pointers
	struct customer *w1,*w2,*w3;

	// Create the customer using the provided arguments
	w3 = (struct customer *) malloc(sizeof(struct customer));
	w3->t_arr = time;
	w3->service = service;

	// Append the customer to the relevant class waiting list
	if(q[pclass] == NULL) {  // The list is empty
		w3->next = NULL;
		q[pclass] = w3;
		lastq[pclass] = w3;
	} else {  // Append it to the end of the list
		w1 = lastq[pclass];
		w2 = lastq[pclass]->next;
		while (w2 != NULL) {
			w1 = w2;
			w2 = w2->next;
		}
		w1->next = w3;
		w3->next = w2;
		lastq[pclass] = w3;
	}
	if (DEBUG) fprintf(stderr,"DEBUG 03: New customer added to queue %d:
time = %1.6f, service = %1.6f, next = %d\n",pclass,w3->t_arr,w3->service,w3-
>next);
}

// This function extracts the first customer from the customer list
// Arguments:
//     pclass = class of the customer
struct customer * get_customer(char pclass) {
	// Declare some useful pointers
	struct customer *w3;

	// The list is empty and it should not be: generate an error
	if (q[pclass] == NULL) {
		fprintf(stdout,"ERROR: customer list of class %d is empty when it
should not be\n",pclass);
		exit(-1);
	}

	// Return the first customer from the list and update the list pointer
	w3 = q[pclass];
	q[pclass] = w3->next;
	if (DEBUG) fprintf(stderr,"DEBUG 04: Next customer extracted from queue
%d: time = %1.6f, service = %1.6f, next = %d\n",pclass,w3->t_arr,w3-
>service,w3->next);
	return w3;
}

// This function generates an instance of an exponential random variable
```

```
// Arguments: param = parameter of the exponential distribution, i.e., inverse
of the mean value
double expon(double param) {
	// Declare some useful variables
	int rnd_num;
	double unif,val;
	// Generate a uniform random number between 0 and 1
	rnd_num = rand();
	if (rnd_num == RAND_MAX) rnd_num--;
	unif = (double)rnd_num/RAND_MAX;
	// Transform the uniform random variable into an exponential one using
the inverse function rule
	val = -log(1.0-unif)/param;
	//if (DEBUG) fprintf(stderr,"DEBUG 05: New exponential random variable
with parameter %1.6f: generated value = %1.6f\n",param,val);

	return val;
}

// This function generates an instance of the service time random variable
// Arguments: pclass = class of the customer
double serv(char pclass) {
	if (pclass == 0 || pclass == 2) {
		// GS or BE: return an exponential service time
		return expon(mu[pclass]);
	} else {
		// RT: return a fixed service time
		return 1.0/mu[pclass];
	}
}

// Initialization function
void initialize() {
	int j,ch;
	// Initialize all the global variables and counters
	now = 0.0;
	tot_arrivals = 0; tot_departures = 0; RT_blocked = 0; BE_interrupted =
0; BE_successful = 0; tot_w = 0.0;
	servint = 0.0; servsuccBE = 0.0; servBE = 0.0;
	for (j = 0; j < C; j++) {
		k[j] = 0; arrivals[j] = 0; w[j] = 0.0;
		q[j] = NULL;
// Initialize the list of events by inserting the first arrival for each class
		if (j == 0) {
// Create a GS arrival for each input channel (one on-off source per channel)
			for (ch = 0; ch < Nserv; ch++) {
	insert_new_event(expon(1.0/Toff),ARRIVAL,j,ch,serv(j));
			}
		} else {
//Create an arrival for each remaining class (RT and BE, one source per class)
			insert_new_event(expon(lambda[j]),ARRIVAL,j,-1,serv(j));
		}
		for (ch = 0; ch < Nserv; ch++) {
			util[j][ch] = 0.0;
			servsucc[j][ch] = 0.0;
			departures[j][ch] = 0;
		}
		tot_serv_traffic[j] = 0.0;
		avgutil[j] = 0.0;
		num_eventsBE[j] = 0;
	}
	for (ch = 0; ch < Nserv; ch++) {
		tfree[ch] = 0.0;
		lastpktclass[ch] = 0;
		utilBEint[ch] = 0.0;
	}
	avgutilBEint = 0.0;
}
```

```
// This function schedules a new arrival based on its class (and input
channel, in case of GS arrival)
// Arguments:
//     time = arrival time
//     pclass = class of the arrival
//     inputch = input channel if GS arrival
double schedule_arrival(double time, char pclass, char inputch) {
       int outputch, rnd_num, startch, ch, chBE, i;
       char found, foundBE;
       if (DEBUG) fprintf(stderr,"DEBUG 18: scheduling packet of class %d
arriving at time %1.6f on channel %d...\n",pclass,time,inputch);

       if (pclass == 0) {
       // GS arrival: output channel is the same as the input channel
              outputch = inputch;
              if (DEBUG) fprintf(stderr,"DEBUG 18 GS: Channel %d: tfree =
%1.6f, lastpktclass = %d\n",outputch,tfree[outputch],lastpktclass[outputch]);

       } else {
// Scan the output channels starting with a randomly selected one;
              rnd_num = rand();
              if (rnd_num == RAND_MAX) rnd_num--;
              startch = (int)(((double)rnd_num/RAND_MAX)*Nserv);

              // Look for the first empty channel, but keep also the first
channel with BE transmission
              found = 0; foundBE = 0;
              for (i=0; i<Nserv && found == 0; i++) {
                     ch = (startch+i)%Nserv;
                     if (DEBUG) fprintf(stderr,"DEBUG 18 RT/BE: Channel %d:
tfree = %1.6f, lastpktclass = %d",ch,tfree[ch],lastpktclass[ch]);
                     if (time >= tfree[ch]) {// Found first empty channel
                            outputch = ch;
                            found = 1;
                            if (DEBUG) fprintf(stderr,"\t*** First empty channel
found ***");
                     } else if (lastpktclass[ch] == 2 && foundBE == 0) {
                            // Found first channel occupied by a BE packet
                            chBE = ch;
                            foundBE = 1;
                            if (DEBUG) fprintf(stderr,"\t*** First channel
occupied by BE packet found ***");
                     }
                     if (DEBUG) fprintf(stderr,"\n");
              }
              if (DEBUG && found == 1) {
                     while (i < Nserv) {
                            ch = (startch+i)%Nserv;
                            fprintf(stderr,"DEBUG 18 RT/BE: Channel %d: tfree =
%1.6f, lastpktclass = %d\n",ch,tfree[ch],lastpktclass[ch]);
                            i++;
                     }
              }
              // What to do if no channel is found empty
              if (pclass == 1) {
// RT arrival: preemts first BE packet found, otherwise is blocked
                     if (found == 0 && foundBE == 1)
                            outputch = chBE;
                     else if (found == 0 && foundBE == 0)
                            outputch = -1;
              } else if (pclass == 2) {
                     // BE arrival: queued if no channel is found empty
                     if (found == 0)
                            outputch = -1;
              }
       }
       if (DEBUG) fprintf(stderr,"DEBUG 18: Output channel = %d\n",outputch);
// Returns the assigned output channel or -1 if no channel is available
```

```
	return outputch;
}

// Main program
int main(int argc, char* argv[]) {
	// Define some useful variables and pointers
	int j, pp = 0, pp_old = 0;
	char outputch, ch;
	struct event *e, *aux, *e1;
	struct customer *c, *caux;
	double totutil = 0.0, totBEutil = 0.0, lastdeptime = 0.0;

//Check the command line arguments: must be at least 6, including the
simulator executable.Otherwise generate an error
	if (argc < 6) {
		fprintf(stderr,"Usage: %s <seed> <N_samples> <N_servers> <rho_GS>
<serv_GS> <load_RT> <serv_RT> <load_BE> <serv_BE>\n",argv[0]);
		exit(-1);
	}

	// Get the random generator seed from the first argument
	seed = atoi(argv[1]);
	// If the provided seed is zero, use the current timestamp
	if (seed == 0) seed = time(NULL);
	srand(seed);
	// Get the number of customers to be simulated from the second argument
	N = atoi(argv[2]);
	// Get the number of servers from the third argument (no more than
MAX_NSERV)
	Nserv = atoi(argv[3]);
	if (Nserv > MAX_NSERV) Nserv = MAX_NSERV;
	// Set the number of classes to 3
	// 0:GS - 1:RT - 2:BE
	C = 3;

	// Check if there are enough arguments for each class (load + average
service time for each class)
	// Otherwise generate an error
	if (argc < 2*C+4) {
		fprintf(stderr,"Missing load and service time for classes %d to
%d\n",(argc-4)/2,C-1);
		exit(-1);
	}
	// Get the load and average service time for each class from the
following arguments and set mu = 1/service and lambda = rho*mu (different for
GS due to on-off source)
	for (j = 0; j < C; j++) {
		mu[j] = 1.0/atof(argv[4+2*j+1]);
		if (j == 0) {
			Toff = 1.0/mu[j]*(1.0/atof(argv[4+2*j])-1.0);
			lambda[j] = 1.0/(Toff+1.0/mu[j]);
		} else {
			lambda[j] = atof(argv[4+2*j])*mu[j];
		}
	}
	// Set the fixed GS delay equal to RT packet time
	fixed_delay = 1.0/mu[1];

	// Initialize the simulator
	initialize();
	// Start the main loop
	//while (tot_departures + RT_blocked + BE_interrupted < N) {  // Loop
until all the generated customers have left the system
	while (tot_arrivals < N) {  // Loop until all generated customers have
arrived to the system

if (DEBUG) {  // Print lots of information about the lists if DEBUG is true
		fprintf(stderr,"DEBUG 17: Cycling...\n");
```

```
                fprintf(stderr,"          Current event list:\n");
                fprintf(stderr,"          ");
                aux = event_list;
                while (aux != NULL) {
                        fprintf(stderr,"[ addr = %d, time = %1.6f, next = %d ]-----
>",aux,aux->time,aux->next);
                        aux = aux->next;
                }
                fprintf(stderr,"\n");
        }
        if (DEBUG) {
                fprintf(stderr,"          Current customer list:\n");
                for (j=0;j<C;j++) {
                        fprintf(stderr,"          q[%d]----->",j);
                        caux = q[j];
                        while (caux != NULL) {
                                fprintf(stderr,"[ addr = %d, time = %1.6f, next = %d
]----->",caux,caux->t_arr,caux->next);
                                caux = caux->next;
                        }
                        fprintf(stderr,"\n");
                }
        }
// Start processing the first event in the list and set the current time
        e = get_event();
        now = e->time;
        if (DEBUG) fprintf(stderr,"DEBUG 10: Current time = %1.6f\n",now);

        if (e->type == ARRIVAL) {  // The event is an arrival: increment the
counters and the state
                tot_arrivals++; arrivals[e->pclass]++;
                if (DEBUG) fprintf(stderr,"DEBUG 06: Arrival: time = %1.6f, class
= %d, inputch = %d, service = %1.6f\n",e->time,e->pclass,e->inputch,e-
>service);
                if (e->pclass == 0 || e->pclass == 1 || (e->pclass == 2 && q[e-
>pclass] == NULL)) { // Schedule the new packet/burst and return the output
channel assigned to it
                        outputch = schedule_arrival(now,e->pclass,e->inputch);
                        if (outputch != -1) { // Channel found!
                                if (now < tfree[outputch]) {
                                // An ongoing transmission seems to be interrupted
                                        if (lastpktclass[outputch] != 2) {
// No interruption of GS or RT by new GS because of the fixed delay
                                                if (DEBUG) fprintf(stderr,"DEBUG 20:
Packet of class %d interrupted: OK if %1.20f is smaller than
%1.20f\n",lastpktclass[outputch],tfree[outputch]-now,fixed_delay);
                                                if (tfree[outputch]-now> fixed_delay) {
                                                        fprintf(stdout,"ERROR: GS burst
or RT packet interrupted!!!\n");
                                                        exit(-1);
                                                }
                                        } else {// BE is always interrupted by RT
// BE is interrupted by GS only if BE ends transmission after the fixed delay
                                                if (e->pclass == 1 || (e->pclass == 0
&& (tfree[outputch]-now > fixed_delay))) {
                                                        k[2]--; BE_interrupted++;
                                                        num_eventsBE[e->pclass]++;
                                        e1 = remove_event(tfree[outputch],DEPARTURE,2);
                                        servint += e1->service - (e1->time - now);
                                        servBE += e1->service - (e1->time - now);
                                //servint += e1->service;
                        utilBEint[outputch] += e1->service - (e1->time - now);
                                                        free(e1);
                                                }
                                        }
                                }// The packet/burst begins transmission and a
departure event is created and inserted to the list
                                k[e->pclass]++;
```

```
                        tfree[outputch] = now + e->service;
                        if (e->pclass == 0) {
                        // GS: add a fixed delay equal to a RT packet time
                              tfree[outputch] += fixed_delay;
                        }
                        lastpktclass[outputch] = e->pclass;
                        insert_new_event(tfree[outputch],DEPARTURE,e-
>pclass,outputch,e->service);
                  } else { // Channel not found (will never happen to GS
bursts). RT packet is blocked, BE packet is queued
                  if (DEBUG) fprintf(stderr,"DEBUG 19: Channel not
found...\n");
                  if (e->pclass == 1) {
                        if (DEBUG) fprintf(stderr,"DEBUG 19: RT packet
blocked!!!\n");
                        RT_blocked++;
                  }
                  else if (e->pclass == 2) {
                        if (DEBUG) fprintf(stderr,"DEBUG 19: BE packet
queued!!!\n");
                        k[e->pclass]++;
                        append(e->pclass,now,e->service);
                  }
            }
      } else {  // BE packet is queued
            k[e->pclass]++;
            append(e->pclass,now,e->service);
      }
      if (tot_arrivals < N) { // There are more arrivals to be generated:
create a new one and insert it in the event list
            if(e->pclass == 0){// GS: generate off period after current burst
                  insert_new_event(now + e->service +
expon(1.0/Toff),ARRIVAL,e->pclass,e->inputch,serv(e->pclass));
            } else { // RT or BE: generate inter-arrival time
                  insert_new_event(now + expon(lambda[e->pclass]),ARRIVAL,e-
>pclass,e->inputch,serv(e->pclass));
            }
      }
} else if (e->type == DEPARTURE) { // The event is a successful departure:
increment the counters and decrement the state
      k[e->pclass]--; tot_departures++; departures[e->pclass][e->inputch]++;
      if (DEBUG) fprintf(stderr,"DEBUG 16: Departure: time = %1.6f, class =
%d, inputch = %d, service = %1.6f\n",e->time,e->pclass,e->inputch,e->service);
      util[e->pclass][e->inputch] += e->service;
      servsucc[e->pclass][e->inputch] += e->service;
      if (e->pclass == 2) {
            servsuccBE += e->service;
            servBE += e->service;
            BE_successful++;
            num_eventsBE[e->pclass]++;
      }
      lastdeptime = now;
      if (now >= tfree[e->inputch] && q[2] != NULL) { // Output channel is
free and BE queue is not empty: put the next queued customer in service
            j = 0;
            while (q[j] == NULL) j++;  // Find the first non-empty waiting
list (should always be j=2)
            c = get_customer(j);
            w[j] = w[j] + (now - c->t_arr);  // Update the sum of the waiting
time for class j
// Schedule the new BE packet and return the output channel assigned to it
            outputch = schedule_arrival(now,j,-1);
            if (outputch == -1) {// Something went wrong: exit!
                  fprintf(stdout,"ERROR: all channels are reported busy while
at least one should not be\n");
                  exit(-1);
            }
            tfree[outputch] = now + c->service;
```

```
                lastpktclass[outputch] = j;
                insert_new_event(tfree[outputch],DEPARTURE,j,outputch,c-
>service);  // Create a departure event and insert it into the list
        if (DEBUG) fprintf(stderr,"DEBUG 07: Removing customer object\n");
                if (c == NULL) {
                        fprintf(stdout,"ERROR: customer object is NULL when it
should not be\n");
                        exit(-1);
                }
                free(c); //Remove the customer object and free memory space
                }
        }
        if (DEBUG) fprintf(stderr,"DEBUG 08: Removing event object\n");
        if (e == NULL) {
                fprintf(stdout,"ERROR: event object is NULL when it should not
be\n");
                exit(-1);
        }
        free(e);  // Remove the event object and free memory space

        if (DEBUG) {
                for (j = 0; j < C; j++) {
                        fprintf(stderr,"DEBUG 09: Current state class %d =
%d\n",j,k[j]);
                }
        }
        if (DEBUG) fprintf(stderr,"DEBUG 11: Total arrivals = %d, Total
departures = %d, Total RT blocked = %d, Total BE interrupted =
%d\n",tot_arrivals,tot_departures,RT_blocked,BE_interrupted);
        if (DEBUG) fprintf(stderr,"-----------------------------------------------
-----------------------------------\n\n");
        if (DEBUG == 0) {
                pp = (int)((double)tot_arrivals/N*100);
                if (pp > pp_old) {
                        fprintf(stderr,"Perc. completed: %3d\r",pp);
                        pp_old = pp;
                }
        }
}  // End of the main loop
if (DEBUG == 0)
        fprintf(stderr,"\nDone!\n");
// Update and print some variables
for (j = 0; j < C; j++) {
//tot_w = tot_w + w[j];  // Update the sum of the overall waiting time
        w[j] = w[j]/(arrivals[j]-k[j]);  // Compute the mean waiting time for
class j
        // The previous average should be made on the number of customers not in
the queue (i.e., those already departed plus those currently in service)
        // The difference is negligible if arrivals[j] >> Nserv
        for (ch = 0; ch < Nserv; ch++) {
                util[j][ch] = util[j][ch]/lastdeptime;
                servsucc[j][ch] = servsucc[j][ch]/departures[j][ch];
                tot_serv_traffic[j] += util[j][ch];
        }
}
// ***** TO BE CORRECTED *****
//tot_w = tot_w/N;  // Compute the mean overall waiting time
/*
printf("# Class\tMean waiting time\n");
for (j = 0; j < C; j++) {
        printf("%d\t%1.6f\n",j,w[j]);
}
printf("\n# Overall\t%1.6f\n",tot_w);
*/

RT_loss_rate = (double)RT_blocked/arrivals[1];
BE_int_rate = (double)BE_interrupted/(arrivals[2]-k[2]);
BE_succ_rate = (double)BE_successful/(arrivals[2]-k[2]);
```

```
servint = servint/BE_interrupted;
servsuccBE = servsuccBE/BE_successful;
servBE = servBE/(BE_interrupted+BE_successful);

//printf("Total arrivals = %d, Total departures = %d, Total RT blocked = %d,
Total BE interrupted = %d, k[0] = %d, k[1] = %d, k[2] =
%d\n",tot_arrivals,tot_departures,RT_blocked,BE_interrupted,k[0],k[1],k[2]);
      //printf("\n");
      // Print channel utilization per class
      //printf("#
Channel\trho_GS\t\trho_RT\t\trho_BE\t\trho_BE_int\trho_tot\n");
for (ch = 0; ch < Nserv; ch++) {
      utilBEint[ch] = utilBEint[ch]/lastdeptime;
      avgutilBEint += utilBEint[ch];
      totutil += utilBEint[ch];
      totBEutil += utilBEint[ch];
      //printf("%d\t",ch);
      for (j = 0; j < C; j++) {//printf("\t%1.10f",util[j][ch]);
            avgutil[j] += util[j][ch];
            totutil += util[j][ch];
            }
      totBEutil += util[2][ch];
      //printf("\t%1.10f\t%1.10f\n",utilBEint[ch],totutil);
}

      // Print:
      //  1. RT loss rate
      //  2. BE interruption rate
      //  3. BE success rate
      //  4. BE interrupted service time
      //  5. BE successful service time
      //  6. BE service time on channel (average of 4 and 5)
      //  7. GS channel utilization
      //  8. RT channel utilization
      //  9. BE channel utilization of successful packets (BE throughput)
      // 10. BE channel utilization of interrupted packets
      // 11. BE channel utilization (both successful and interrupted)
      // 12. BE packet average waiting time

printf("%1.20f\t%1.20f\t%1.20f\t%1.20f\t%1.20f\t%1.20f\t",RT_loss_rate,BE_int_
rate,BE_succ_rate,servint,servsuccBE,servBE);
      for (j = 0; j < C; j++) {
            printf("%1.20f\t",avgutil[j]/Nserv);
      }
      printf("%1.20f\t%1.20f\t",avgutilBEint/Nserv,totBEutil/Nserv);
      printf("%1.20f\n",w[2]);

      int totnuminterr = 0;
//double avgNserv = Nserv*(1.0-atof(argv[4]))-atof(argv[6])*(1.0-
RT_loss_rate);
      //printf("\n\nAverage # of BE servers: %1.6f\n",avgNserv);
      //printf("# interr. from class GS: %d (freq: %1.6f, lambda =
%1.6f)\n",num_eventsBE[0],(double)num_eventsBE[0]/lastdeptime,avgNserv/Toff);
      //printf("# interr. from class RT: %d (freq: %1.6f, lambda =
%1.6f)\n",num_eventsBE[1],(double)num_eventsBE[1]/lastdeptime,lambda[1]);
      //printf("# succ. BE departures: %d (freq: %1.6f, mu =
%1.6f)\n",num_eventsBE[2],(double)num_eventsBE[2]/lastdeptime,avgNserv*mu[2]);
      totnuminterr = num_eventsBE[0]+num_eventsBE[1]+num_eventsBE[2];
      //printf("\n-----------------------------\ntotal: %d (counters:
%d)\n\n",totnuminterr,BE_interrupted+BE_successful);
      //printf("%1.20f\t%1.20f\t%1.20f\t%1.20f\n",(double)num_eventsBE[0]/last
deptime,(double)num_eventsBE[1]/lastdeptime,(double)num_eventsBE[2]/lastdeptim
e,(double)totnuminterr/lastdeptime);
      //printf("-----------------------\nclass %d total served traffic =
%1.6f\n\n",j,tot_serv_traffic[j]);
      //printf("class %d channel %d\tInt. serv. time = %1.20f\tutil =
%1.20f\n",2,ch,servint,0.0);
}
```

Annex II. Input parameters for CPRI Option 1 to Option 7.

CPRI Option 1

NB	TB[ns]	RCPRI[Gbps]	RE[Gbps]	Lp[Byte]	Lp Aprox[Byte]	TE(=thetaG)[ns]	Tencap[ns]	rhoG	Delta	Tgap[ns]
10	260	0,6144	10	199,68	200	195,2	2604,166667	0,0749568	99,2	2309,76667
20	260	0,6144	10	399,36	400	355,2	5208,333333	0,0681984	99,2	4753,93333
30	260	0,6144	10	599,04	600	515,2	7812,5	0,0659456	99,2	7198,1
40	260	0,6144	10	798,72	800	675,2	10416,66667	0,0648192	99,2	9642,26667
50	260	0,6144	10	998,4	1000	835,2	13020,83333	0,06414336	99,2	12086,4333
60	260	0,6144	10	1198,08	1200	995,2	15625	0,0636928	99,2	14530,6
70	260	0,6144	10	1397,76	1400	1155,2	18229,16667	0,06337097	99,2	16974,7667

CPRI Option 2

NB	TB[ns]	RCPRI[Gbps]	RE[Gbps]	Lp[Byte]	Lp Aprox[Byte]	TE(=thetaG)[ns]	Tencap[ns]	rhoG	Delta	Tgap[ns]
5	260	1,2288	10	199,68	200	195,2	1302,083333	0,1499136	99,2	1007,68333
10	260	1,2288	10	399,36	400	355,2	2604,166667	0,1363968	99,2	2149,76667
15	260	1,2288	10	599,04	600	515,2	3906,25	0,1318912	99,2	3291,85
20	260	1,2288	10	798,72	800	675,2	5208,333333	0,1296384	99,2	4433,93333
25	260	1,2288	10	998,4	1000	835,2	6510,416667	0,12828672	99,2	5576,01667
30	260	1,2288	10	1198,08	1200	995,2	7812,5	0,1273856	99,2	6718,1
35	260	1,2288	10	1397,76	1400	1155,2	9114,583333	0,12674194	99,2	7860,18333

CPRI Option 3

NB	TB[ns]	RCPRI[Gbps]	RE[Gbps]	Lp[Byte]	Lp Aprox[Byte]	TE(=thetaG)[	Tencap[ns]	rhoG	Delta	Tgap[ns]
2	260	2,4576	10	159,744	200	195,2	651,041667	0,2998272	99,2	356,641667
5	260	2,4576	10	399,36	400	355,2	1302,08333	0,2727936	99,2	847,683333
7	260	2,4576	10	559,104	600	515,2	1953,125	0,2637824	99,2	1338,725
10	260	2,4576	10	798,72	800	675,2	2604,16667	0,2592768	99,2	1829,76667
12	260	2,4576	10	958,464	1000	835,2	3255,20833	0,25657344	99,2	2320,80833
15	260	2,4576	10	1198,08	1200	995,2	3906,25	0,2547712	99,2	2811,85
17	260	2,4576	10	1357,824	1400	1155,2	4557,29167	0,25348389	99,2	3302,89167

CPRI Option 4

NB	TB[ns]	RCPRI[Gbps]	RE[Gbps]	Lp[Byte]	Lp Aprox[Byte]	TE(=thetaG)[	Tencap[ns]	rhoG	Delta	Tgap[ns]
2	260	3,072	10	199,68	200	195,2	520,833333	0,374784	99,2	226,433333
4	260	3,072	10	399,36	400	355,2	1041,66667	0,340992	99,2	587,266667
6	260	3,072	10	599,04	600	515,2	1562,5	0,329728	99,2	948,1
8	260	3,072	10	798,72	800	675,2	2083,33333	0,324096	99,2	1308,93333
10	260	3,072	10	998,4	1000	835,2	2604,16667	0,3207168	99,2	1669,76667
12	260	3,072	10	1198,08	1200	995,2	3125	0,318464	99,2	2030,6
14	260	3,072	10	1397,76	1400	1155,2	3645,83333	0,31685486	99,2	2391,43333

CPRI Option 5

NB	TB[ns]	RCPRI[Gbps]	RE[Gbps]	Lp[Byte]	Lp Aprox[Byt	TE(=thetaG)[	Tencap[ns]	rhoG	Delta	Tgap[ns]
1	260	4,9152	10	159,744	200	195,2	325,520833	0,5996544	99,2	31,1208333
2	260	4,9152	10	319,488	400	355,2	651,041667	0,5455872	99,2	196,641667
3	260	4,9152	10	479,232	600	515,2	976,5625	0,5275648	99,2	362,1625
5	260	4,9152	10	798,72	800	675,2	1302,08333	0,5185536	99,2	527,683333
6	260	4,9152	10	958,464	1000	835,2	1627,60417	0,51314688	99,2	693,204167
7	260	4,9152	10	1118,208	1200	995,2	1953,125	0,5095424	99,2	858,725
8	260	4,9152	10	1277,952	1400	1155,2	2278,64583	0,50696777	99,2	1024,24583

CPRI Option 6

NB	TB [ns]	RCPRI [Gbps]	RE [Gbps]	LP [Byte]	LP (approx.) [Byte]	TE (=thetaG) [ns]	Tencap [ns]	rhoG	Delta	Tgap
1	260	6,144	10	199,68	200	195,2	260,4166667	0,749568	99,2	-33,9833
2	260	6,144	10	399,36	400	355,2	520,8333333	0,681984	99,2	66,43333
3	260	6,144	10	599,04	600	515,2	781,25	0,659456	99,2	166,85
4	260	6,144	10	798,72	800	675,2	1041,666667	0,648192	99,2	267,2667
5	260	6,144	10	998,4	1000	835,2	1302,083333	0,6414336	99,2	367,6833
6	260	6,144	10	1198,08	1200	995,2	1562,5	0,636928	99,2	468,1
7	260	6,144	10	1397,76	1400	1155,2	1822,916667	0,63370971	99,2	568,5167

CPRI Option 7

NB	TB [ns]	RCPRI [Gbps]	RE [Gbps]	LP [Byte]	LP (approx.) [	TE (=thetaG)	Tencap [ns]	rhoG	Delta	Tgap
0	260	9,8304	10	0	200	195,2	162,760417	1,1993088	99,2	-131,63958
1	260	9,8304	10	319,488	400	355,2	325,520833	1,0911744	99,2	-128,87917
1	260	9,8304	10	319,488	600	515,2	488,28125	1,0551296	99,2	-126,11875
2	260	9,8304	10	638,976	800	675,2	651,041667	1,0371072	99,2	-123,35833
3	260	9,8304	10	958,464	1000	835,2	813,802083	1,02629376	99,2	-120,59792
3	260	9,8304	10	958,464	1200	995,2	976,5625	1,0190848	99,2	-117,8375
4	260	9,8304	10	1277,952	1400	1155,2	1139,32292	1,01393554	99,2	-115,07708

References

[1] Franka, L: IoT-enhanced Entertainment: A new use case for 5G Fixed Wireless Access. http://www.diva-portal.se/smash/get/diva2:1034576/FULLTEXT01.pdf KTH Stockholm Royal Institute of Technology School of Computer Science and Communication. (2016)

[2] Cerroni, W., Raffaelli, C.:Analytical model of quality of service scheduling for optical aggregation in data centers. (2014)

[3] Checko, A., Christiansen, H. L., Yan, Y., Scolari, L., Kardaras, G., Berger, M. S., & Dittman, L.: Cloud RAN for Mobile Networks – a Technology Overview. IEEE Communications Surveys and Tutorials, 17(1), pp. 405~426. (2014)

[4] Pizzinat, A., Chanclou, P., Diallo, T., Saliou, F.: Things you should know about fronthaul. (2014)

[5] Checko, A.: Cloud RAN fronthaul. (2015)

[6] Hadush Hailu, D., Gebrekrstos Iema, G., Bjornstad, S.: QoS Performance of Integrated Hybrid Optical Network in Mobile Fronthaul networks.
Ethiopian Institute of Technology-Mekelle(EiT-M), pp. 189~204. (2017)

[7] Baldry, J.: Staying in Sync with Mobile Fronthaul and the Migration to Cloud-RAN. (2016)

[8] Hadush Hailu, D., G. Gebrehaweria, B., H. Kebede, S., G. Lema, G., T. Tesfamariam, G.: Mobile fronthaul transport options in C-RAN and emerging research directions: A comprehensive study.
Ethiopian Institute of Technology-Mekelle(EiT-M), pp. 40~52. (2018)

[9] Fiorani, M., Skubic, B., Martensson, J., Valcarenghi, L., Castoldi, P., Wosinska, L., Monti, P.: On the design of 5G transport networks. (2015)

[10] Sevillano, F.: 5 things you should know about fronthaul. (2015)

[11] Bladsjö, D., Hogan, M., Ruffini, S.: Synchronization Aspects in LTE Small Cells. IEEE Communications Magazine, Ericsson. (2013)

[12] Hadush Hailu, D.: Cloud Radio Access Networks(C-RAN) and optical Mobile backhaul and fronthaul.
Norwegian University of Science and Technology. (2016)

[13] Madrazo, J.: Análisis y studio de soluciones para fronthaul radio,
Universitat Oberta de Catalunya. (2018)

[14] Shin, S., Harrison J (Netmanias): Why WDM is essential in C-RAN fronthaul networks?-Ultra high CPRI link capacity. (2014)

[15] Smartoptics: The basics of Wavelength Division Multiplexing, WDM. Oslo, Norway. (2018)

[16] Abate, A.: CWDM Case Study. Line Systems, Inc. https://www.omnitron-systems.com/downloads/case_study/line_systems_case_study.php

[17] J. Li, H. He, W. Hu: Power Depletion and Crosstalk Induced by Stimulated Raman Scattering in WDM Fronthaul. IEEE Photonics Technology Letters, V28, N10. (2016)

[18] N. Cheng, L. Zhou, X. Liu: Reflective Crosstalk Cancellation in Self-Seeded WDM PON for Mobile Fronthaul/Backhaul. Journal of lightwave technology, V34, N8. (2016)

[19] T. Diallo, A. Pizzinat, F. Sliou: Self-Seeded DWDM Solution for Fronthaul Links in Centralized Radio Access Network. Journal of lightwave technology, V34, N21. (2016)

[20] Chitimalla, D., Kondepu, K., Valcarenghi, L., Tornatore, M., Mukherjee, B.: 5G Fronthaul- Latency and Jitter Studies of CPRI Over Ethernet. Journal of Optical Communications and Networking, V9, N2, pp. 172~182. (2017)

[21] Veisllari, R., Bjornstad, S., Braute, J.P., Bozorgebrahimi, K., Raffaelli, C.: Field-trial demonstration of cost efficient sub-wavelength service through integrated packet/circuit hybrid network. IEEE/OSA Journal of Optical Communications and Networking, V7, N3, pp. 379~387. (2015)

[22] Common Public Radio Interface; Interface Specification. CPRI Specification V7.0. (2015)

[23] Veisllari, R., Bjornstad, S., P. Braute, J., Bozorgebrahimi, K., Raffaelli, C.: Field-Trial Demonstration of Cost Efficient Sub-wavelenght Service Through Integrated Packet/Circuit Hybrid Network. Journal of Optical Communications and Networking, V7, N3. (2015)

[24] Raffaelli, C., Stol, N., Savi, M.: 3-Level Integrated Hybrid Optical Network (3LIHON) to Meet Future QoS Requirements. IEEE Globecom 2011

[25] IEEE Stnadard P802.1CM: Time Sensitive Networking for Fronthaul.