Sleep mode scheduling technique for energy saving in TDM-PONs

MARTINA FIAMMENGO

Master of Science Thesis
Stockholm, Sweden 2011

TRITA-ICT-EX-2011:17
Acknowledgments

This thesis project has been developed between September 2010 and January 2011 at Ericsson Research in Stockholm, Kista. There are many people that I would like to thank for helping me during my thesis.

First of all I would like to thank my industrial supervisor in Ericsson, Alexander Lindström. He has guided and supported me throughout my work and has always been willing to waste his time to help me. I have also received significant help from my examiner, Prof. Lena Wosinska and from my supervisor at KTH, Dr. Paolo Monti. They have always supported me and provided me with very precious advice regarding both the project and the writing of the report. Then I would like to thank my supervisor in Politecnico di Torino, Prof. Fabio Neri for his comments about the thesis work.

Furthermore I would like to thank Jiajia Chen for supporting and helping me with her advice. Then I would like to thank Björn Skubic for the useful discussions we had during our meetings.

Last but not least I would like to thank my boyfriend Rocco Luciano Grimaldi for his unfailing support throughout my thesis work. Finally I would like to thank Min Chen, Andreas Tsopelas and Ourania Lympouridou for making my staying in Ericsson so pleasant. I had a very good time with you all.
## Contents

Abstract ........................................................................................................................................... 1
Sammanfattning ......................................................... 3
List of figures ................................................................. 5
List of acronyms ......................................................... 7

1 Introduction ................................................................. 9
  1.1 Triple-play services and their impact on sleep mode technique ........................................ 10
  1.2 Organization of the thesis .................................................. 12

2 Passive optical networks ........................................... 15
  2.1 Time division multiplexing PONs ......................................... 18
  2.2 TDM-PON standards .................................................... 21

3 Power saving techniques via sleep mode in TDM PONs .................................. 23
  3.1 Classification of TDM-PON power saving techniques ................................................ 23
  3.2 State of the art of TDM-PON power saving techniques ................................................. 26
    3.2.1 Characteristics of the sleep state ........................................... 27
    3.2.2 Energy conservation limitations due to the ONU transceiver ................................ 27
    3.2.3 Duration of the sleep state .................................................. 28
    3.2.4 Conditions triggering the sleep state ......................................... 31
    3.2.5 Conditions triggering the return to the active state ............................................ 32
    3.2.6 Messaging schemes supporting the sleep mode ................................................ 33
    3.2.7 Solutions for improving the ONU wake-up time ............................................ 34

4 Proposal of a new sleep mode scheduling technique ........................................... 39
  4.1 Operational scheme ..................................................... 39
  4.2 Control mechanism ..................................................... 41
    4.2.1 Prediction of idle periods .................................................. 42
    4.2.2 Scheduling of sleep periods ................................................. 44
    4.2.3 Additional features ......................................................... 47
  4.3 Simulations ................................................................. 50
  4.4 Hardware support ....................................................... 60

5 Implementation of the proposed technique .................................................. 67
  5.1 Multiplexing and demultiplexing loop .................................................. 68
  5.2 Implementation of the sleep controllers ................................................ 71
6 Analysis of measurements ........................................................................................................75
   6.1 Constant rate traffic ...........................................................................................................76
   6.2 Bursty traffic .......................................................................................................................78
   6.3 Double-rate traffic ..............................................................................................................80
   6.4 Bursty traffic with superimposed background traffic ..........................................................83
   6.5 Traffic with superimposed rates ........................................................................................85
7 Conclusions and future work ..................................................................................................89
References ....................................................................................................................................93
Appendix I – Matlab codes ........................................................................................................95
Appendix II – Schematics of the blocks inside the sleep controller ...........................................119
Appendix III – Experimental results ........................................................................................127
Abstract

Nowadays energy efficiency of telecommunication networks is receiving more attention than in the past for natural reasons. The situation is critical especially for access networks that typically include many end-user devices that consume a lot of energy.

This thesis proposes a novel sleep mode technique for TDM-PONs that turns off the ONUs (placed at the customer premises) when certain traffic conditions are verified. The objective is to achieve an improved energy efficiency without impacting too much the Quality of Service perceived by end-users. The whole operation is managed at the OLT (placed at the provider central office) and the length of the sleep time periods is computed employing a statistical method. An approximated version has been implemented in hardware for proof of concept.

The obtained results show that the proposed sleep mode technique has good performances for some test cases while it should be avoided for others.
Sammanfattning

Idag får energieffektivitet i telekommunikationsnät mer uppmärksamhet än tidigare på grund av naturliga anledningar. Situationen är särskilt kritisk för accessnät som vanligt omfattar många slutanvändares apparater som förbrukar mycket energi.

Denna avhandling föreslår en ny sleep mode teknik för TDM-PONs som stänger av ONUs (placerade i näheten av kunder) när vissa trafikförhållanden bekräftas. Målet är att uppnå en förbättrad energieffektivitet utan att försämra för mycket Quality of Service som slutanvändarna uppfattar. Hela operationen sköts i OLT (placerad vid leverantörs centrala kontor) och längden på sovperioder beräknas genom en statistik metod. En approximerad version har genomförts i hårdvara för proof-of-concept skull.

De erhållna resultaten visar att den föreslagna sleep mode teknik har goda prestanda för vissa testfall men den bör undvikas i andra fall.
List of figures

Figure 2.1 Generic structure of a PON solution .................................................................14
Figure 2.2 Generic architecture of a PON ...........................................................................15
Figure 2.3 Time division multiplexing mechanism for downstream transmission ............17
Figure 2.4 Time division multiplexing mechanism for upstream transmission ..................18
Figure 2.5 Generic structure of a TDM PON optical line terminal .................................19
Figure 2.6 Generic structure of a TDM PON optical network unit ..................................20
Figure 3.1 Sleep period as a function of frame inter-arrival time [15] .................................29
Figure 3.2 Sleep period as a function of exponentially smoothed average of 
inter-arrival times [16] .....................................................................................................29
Figure 3.3 Architecture of the analog front-end of the ONU receiver [9] ..........................34
Figure 3.4 First proposal for a faster ONU receiver [9] ....................................................35
Figure 3.5 Second proposal for a faster ONU receiver [9] ..................................................36
Figure 4.1 Three scenarios for the operational scheme ......................................................41
Figure 4.2 Sleep period as a function of exponentially smoothed average of 
inter-arrival times for delay-sensitive traffic ..................................................................46
Figure 4.3 Scheduling of sleep periods for delay-insensitive traffic .................................48
Figure 4.4 Scheduling of sleep periods in case of prediction errors .................................49
Figure 4.5 Ratio of sleep time as a function of average inter-arrival time .......................52
Figure 4.6 Power consumption as a function of average inter-arrival time .....................54
Figure 4.7 Power consumption as a function if average inter-arrival time with increased 
wake-up time ................................................................................................................55
Figure 4.8 Power consumption as a function of average inter-arrival time with reduced 
maximum sleep period ..................................................................................................56
Figure 4.9 Average additional latency as a function of average inter-arrival time ..........57
Figure 4.10 Worst case additional latency as a function of average inter-arrival time ..........58
Figure 4.11 Average inter-arrival time as a function of time .............................................59
Figure 4.12 Ratio of sleep time as a function of time and average inter-arrival time ..........60
Figure 4.13 Power consumption as a function of time and average inter-arrival time ......61
Figure 4.14 Average additional latency as a function of time and average inter-arrival 
time ..................................................................................................................................61
Figure 4.15 Worst case additional latency as a function of time and average inter-arrival 
time ..................................................................................................................................62
Figure 4.16 Schematic of hardware support at OLT .........................................................63
Figure 4.17 Schematic of the sleep controller block.........................................................66
Figure 5.1 Experimental setup ..................................................................................69
Figure 5.2 Support loop structure .........................................................................71
Figure 5.3 Sleep controller structure .....................................................................73
Figure 5.4 First type of sleep block ......................................................................74
Figure 5.5 Second type of sleep block ..................................................................75
Figure 6.1 Percentage of saved power with constant rate traffic .........................78
Figure 6.2 Average additional latency with constant rate traffic .........................79
Figure 6.3 Worst case additional latency with constant rate traffic .....................79
Figure 6.4 Percentage of saved power with bursty traffic ...................................81
Figure 6.5 Average additional latency with bursty traffic ....................................82
Figure 6.6 Worst case additional latency with bursty traffic ...............................82
Figure 6.7 Percentage of saved power with double-rate traffic .........................83
Figure 6.8 Average additional latency with double-rate traffic ............................84
Figure 6.9 Worst case additional latency with double-rate traffic .......................85
Figure 6.10 Percentage of saved power with bursty traffic superimposed to constant rate traffic .................................................................86
Figure 6.11 Average additional latency with bursty traffic superimposed to constant rate traffic .................................................................87
Figure 6.12 Worst case additional latency with bursty traffic superimposed to constant rate traffic .................................................................87
Figure 6.13 Percentage of saved power with superimposed rates traffic ............89
Figure 6.14 Average additional latency with superimposed rates traffic .............90
Figure 6.15 Worst case additional latency with superimposed rates traffic ..........90
List of acronyms

AON: Active Optical Network
BM-CDR: Burst Mode Clock and Data Recovery
BPON: Broadband Passive Optical Network
CDR: Clock and Data Recovery
CM-CDR: Continuous Mode Clock and Data Recovery
CO: Central Office
DBA: Dynamic Bandwidth Assignment
EPON: Ethernet Passive Optical Network
FIFO: First In First Out
FPGA: Field-Programmable Gate Array
FTTH: Fiber To The Home
GEM: GPON Encapsulation Method
GPON: Gigabit-capable Passive Optical Network
IEEE: Institute of Electrical and Electronics Engineers
IP: Internet Protocol
IPTV: Internet Protocol TeleVision
ITU-T: Telecommunication standardization sector of the International Telecommunications Union
JIT-SC: Just In Time Sleep Control
LIA: Limiting Amplifier
LO: Local Oscillator
MAC: Media Access Control
MPCPDU: Multi-Point Control Protocol Data Unit
MPEG: Moving Picture Experts Group
MTBF: Mean Time Between Failures
OLT: Optical Line Terminal
ONU: Optical Network Unit
P2MP: Point-to-Multi-Point
PCS: Physical Coding Sublayer
PD: PhotoDiode
PLOAM: Physical Layer Operation, Administration and Maintenance
PMA: Physical Medium Attachment
PMD: Physical Medium Dependent
PON: Passive Optical Network
POTS: Plain Old Telephone Service
QoS: Quality of Service
RTD: Round Trip Delay
SFP: Small Form-factor Pluggable
SNI: System Network Interface
SPW: Sleep and Periodic Wake-up
TCP: Transmission Control Protocol
TDMA: Time Division Multiple Access
TDM-PON: Time Division Multiplexing Passive Optical Network
TE: Terminal Equipment
TIA: TransImpedance Amplifier
TP: Twisted Pair
TSE: Triple-Speed Ethernet
UNI: User Network Interface
VoIP: Voice over Internet Protocol
WDM-PON: Wavelength Division Multiplexing Passive Optical Network
1 Introduction

During the most recent years the bandwidth demand for telecommunication services has enormously increased. As a consequence network plants consume higher and higher amounts of energy. According to the experts a rapid further growth of networks is about to take place, posing new challenges to the telecommunication industry that will have to face an increased power consumption of the networks [1]. Thus low-power solutions are becoming a crucial topic both in the fight against global warming and in the control of operational expenses. At the same time operators and customers are getting more and more interested in environmentally sustainable technologies. Therefore standardization bodies and equipment vendors have started to include power saving among the first points in their agendas, in order to provide green telecommunication solutions in the near future.

The concern about the increasing power consumption is particularly urgent for access networks. They are the last (or first) segment of a telecommunication network connecting the provider central office (CO) to end users and constitute the largest part of a network. They require more power than core or metro networks because they involve a larger amount of active devices, which is proportional to the number of end users.

In line with this scenario, ITU-T and IEEE have opened a discussion about energy-saving potential of telecommunication networks. The study groups of both organizations have also focused on energy-saving techniques for passive optical networks (PONs), a type of access network. Some of the most popular approaches discussed exploit those periods of time characterized by a lack or reduction of traffic in order to turn off (completely or partially) the equipment placed at the customer premises. These power saving methods are commonly known as sleep mode techniques. Special attention is given to the possibility of introducing a sleep mode technique for the EPON (Ethernet PON) and GPON (Gigabit-capable PON) standards. Both organizations have studied different power-efficiency protocol schemes [2] [3], but neither the performances nor the methods to set the necessary control parameters have been fully investigated.

The main goal of a sleep mode technique is to enhance the energy efficiency of the network in a completely seamless way. This means that the presence of a power saving mechanism should not affect the quality of the user experience. The perceived quality of service must remain unchanged when passing from normal operation to sleep mode and backwards.
This ability to seamlessly enter and exit the ONU (optical network unit) sleep state is quite tricky to accomplish. In fact, while the ONU is in sleep mode, it can neither receive nor transmit any traffic, thus causing severe impairments on service quality, should data be transmitted to it. One way to mitigate this problem is to estimate the length of the period during which the ONU will be idle (neither receiving nor transmitting any frames) and then instruct it accordingly. In this way the ONU would become active just in time to deal with the incoming traffic and the service quality would be preserved.

Unfortunately this is not so easily achieved in a TDM-PON (time division multiplexing PON) network. In fact the length of inactivity periods strongly depends on the particular applications that the customer is using and on the traffic load conditions throughout the whole network. Despite these factors, still it is possible to obtain an estimation of the idle period. This can be achieved by exploiting the statistical properties of the traffic, but some error must be tolerated. The tolerance level mainly depends on the service-specific requirements because different service types usually require different frame rates and show different sensitivities to traffic delay.

This thesis investigates and proposes a sleep mode technique where the length of the sleep periods for each ONU is computed with a statistical method by monitoring the inter-arrival times between downstream frames. The scheduling mechanism is then divided into two different methods in order to preserve quality of service when delay-sensitive services are active but save more energy when the traffic is not so critical in terms of latency. The proposed technique is then simulated with Matlab and finally an approximated version of the whole technique is implemented using a board with an FPGA as a platform. In this way both simulation results and measurements will be available in order to evaluate the performances of the proposed technique.

1.1 TRIPLE-PLAY SERVICES AND THEIR IMPACT ON SLEEP MODE TECHNIQUE

TDM-PON networks support triple-play services, namely data, video and voice.
Data are transported at the network layer using the Internet Protocol (IP). IP usually provides best-effort services. The term best-effort describes the fact that the network does not guarantee that data is delivered or that a certain quality of service is maintained. All users obtain unspecified bit rate and delivery time, depending on current network load. Anyway guaranteed delivery can still be provided by higher layer protocols (transport layer) like TCP (transmission control protocol). [4]

Video contents can be transported through an optical network either as an analog television signal or as a digital signal exploiting the IP protocol. In the recent years digital IPTV (internet protocol television) services are replacing traditional analog TV services. With this approach the TV content is compressed using a MPEG (moving pictures experts group) codec (usually MPEG-2 or MPEG-4) and transported using the IP protocol with the IP packets being encapsulated into Ethernet frames.

Voice services can be supported either by means of traditional circuit-switched techniques or exploiting the IP protocol. Voice services are moving to the VoIP (voice over internet protocol) solution, just like video services are shifting to IPTV. In fact thanks to VoIP, users can easily enrich their phone calls with other contents, such as messages or videos.

The migration from traditional video and voice services to IPTV and VoIP allows for the unification of TV, voice and data distribution networks towards a single all-IP network. From the operators’ point of view, this has the advantage of reducing both the costs and the management complexity of the network.

The sleep mode technique proposed in this thesis determines the length of the ONU sleep period using a statistical approach. Obviously this method can incur in some estimation errors. Such errors translate into additional traffic delay. Thus it would be useful to determine how much additional delay the different services can tolerate. In this way it would be possible to apply stricter or more relaxed constraints to the estimation method depending on which service is active.

Unfortunately this is quite complex. When trying to extract different QoS (quality of service) requirements for different services, one must be aware of the fact that these are actually service bunches. This means that an application usually involves different service components, each with different QoS requirements [5]. Tele-learning applications for example can include interactive video telephone sessions with the teacher, download of learning material, messaging
sessions and web browsing [5]. Furthermore a customer could use more services at the same time. For example, one could have a video phone call, implying real-time transmission of voice and video contents, and at the same time browse web pages.

Despite this complex scenario, most critical service conditions have some characteristics in common. According to the analysis conducted in [5] and [6], the most delay-sensitive applications require a high level of interactivity and/or distribution of real-time contents. Examples can be video phone calls, multi-player online games or tele-commerce. All these delay-sensitive services require an IP packet transfer delay of less than 100 ms [5], [6]. The sleeping mode technique must take care of these critical situations in such a way that the quality of the user experience is not worsened.

The presence of critical delay-sensitive services sets some limitations on the energy conservation achievable via sleep mode. In fact, because of their reduced delay tolerance, sometimes the sleep technique must become less aggressive in order to guarantee a certain maximum delay. In this way the quality of the user experience is preserved but the power consumption is increased.

1.2 ORGANIZATION OF THE THESIS

This report consists of seven chapters each covering a different aspect of the thesis work.

- Chapter 1 introduces the topic of the thesis, together with its motivations and goals.
- Chapter 2 briefly reviews PONs and in particular TDM-PONs.
- Chapter 3 presents the sleep mode techniques that can be found in literature. The chapter goes through the main issues that one must deal with when developing a sleep mode technique and for each of these the solutions proposed by researchers around the world are illustrated. Chapter 4 illustrates the sleep mode technique proposed by this thesis work. At first the chapter illustrates how the prediction and scheduling mechanisms work, then the Matlab simulations are discussed and finally a possible hardware implementation is shown.
- Chapter 5 describes the implementation of the technique employed for proof-of-concept purposes.
- Chapter 6 presents the results of the measurements and discusses the performances of the proposed technique.
- Chapter 7 closes the report presenting the conclusions and some suggestions about possible future work.
2 Passive optical networks

A modern telecommunication network typically consists of three main portions: backbone/core network, metro/regional network and access network. The access network represents the last segment of the connection and it links the service provider’s central office (CO) to end users. This section is often called the last mile.

Different kinds of access technologies exist and they can be divided into two main categories: wired and wireless. Among the wired solutions, a very popular kind of network is FTTH (fiber-to-the-home) that uses optical fibers to deliver telecommunication services up to the boundaries of living spaces. It can be based on different underlying technologies among which PONs are the dominant solution. PONs constitute the prevailing choice for fiber access networks because they are characterized by low cost and low power consumption. These features come as the result of the combination of two main factors, namely the passivity of components and the minimal number of transceivers.

First of all PONs involve only passive elements in their plants. In fact the splitters used to broadcast the signal to all end users are just passive devices. This means that they do not need to be powered up to operate, leading to a lower power consumption, when compared to AONs (active optical networks). Furthermore passive components require little maintenance and have a high mean time between failures (MTBF).

Secondly, the structure of PONs (shown in Figure 2.1) manages to minimize the number of transceivers that constitute one of the main sources of power consumption. The optical fiber supports transmission of optical signals, while the signal processing is accomplished through digital electronics. Thus a transceiver is required at each side of every connection in order to convert the optical signal into a digital one and backwards. As a large number of end users are usually present in an access network, using a pure point-to-point connection based on optical fibers would result in a very expensive solution. In fact 2N transceivers, where N is the number of end users, would be required.
The PON solution manages to reduce this number thanks to the structure shown in Figure 2.1:

![Figure 2.1 Generic structure of a PON solution](image)

In PONs a single strand of fiber goes out to a passive optical splitter where the signal is multiplexed to \( N \) different lines, being \( N \) the number of customers. This point-to-multipoint (P2MP) topology requires just one feeder fiber line and it allows for the minimum number of transceivers that is \( N+1 \).

A large portion of the network infrastructure is shared among different users, thus also the relative costs can be shared. This makes PONs a low-cost per customer solution. Furthermore PONs can support a broad range of applications including triple play (voice, data and video services) over a single fiber.
The generic architecture of a PON is sketched in Figure 2.2:

![Figure 2.2 Generic architecture of a PON](image)

The optical line terminal (OLT) is placed at the central office node. It sends and receives messages and data to and from the connected optical network units (ONUs). These can be placed at the user nodes or in the subscriber neighbourhood to terminate the optical fiber transmission line and provide electrical signals over metallic lines to the subscribers. Thus the ONUs receive data from the OLT and convert the optical signal into an electrical one.

The optical transceivers placed at OLT and ONUs typically use different wavelengths for transmitting and receiving the optical signals. In this way the downstream (from the OLT to ONUs) and the upstream (from the ONUs to the OLT) transmissions can share the same physical link. The downstream channel has a broadcast nature. In the downstream direction PONs behave like point-to-multipoint networks. Thus the OLT can manage the whole available bandwidth. The upstream link, instead, is a multipoint-to-point connection. All ONUs share the same channel to communicate with the OLT. Thus ONUs can transmit only towards the OLT and they can not detect other ONUs transmission. The main problem due to such a connection is that the data transmitted by different ONUs may collide. Thus a channel separation mechanism is needed in order to fairly share bandwidth resources and avoid data collisions. The OLT controls and manages this mechanism so that ONUs can correctly perform upstream transmission.
Two categories of PONs exist, each characterized by a different multiplexing method of the transported signals. Time division multiplexing (TDM) PONs multiplex the messages associated to different ONUs in the time domain. This means that frames addressed to different ONUs are transmitted at different times in the downstream direction, while upstream transmission is performed assigning different time slots to the ONUs according to TDMA (time division multiple access) algorithms. Wavelength division multiplexing (WDM) PONs, instead, make use of a different wavelength for each transmission channel established between the OLT and one ONU. Thus frames addressed to and coming from different ONUs travel on different wavelengths and making signals multiplexed in the wavelength domain.

2.1 TIME DIVISION MULTIPLEXING PONS

In TDM PONs signals are multiplexed in the time domain and are distributed throughout the network by means of passive power splitters placed at the network’s nodes.

As shown in Figure 2.3, in the downstream direction, the OLT schedules traffic into timeslots, depending on which ONU the frames are addressed to. These time slots can vary from a...
microseconds to a milliseconds range, according to the standard implemented. The same identical signal will be broadcast by the power splitters to all ONUs. Every ONU will then be able to recognize its own data thanks to an address label embedded in the signal itself and will discard all other data. This broadcasting feature implies a weak security of the transmission and some encryption mechanism must be provided in order to protect the delivered contents.

As shown in Figure 2.4, in the upstream direction all ONUs share the same only channel. As a result collisions may occur between data sent by different ONUs. In order to prevent this, TDM-PON standards propose time division multiple access schemes. According to these, the upstream transmission channel is divided into separate time slots, each assigned to one ONU according to some algorithms.

![Figure 2.4 Time division multiplexing mechanism for upstream transmission](image)

In order to communicate to ONUs which time slot they have been assigned for upstream transmission, downstream traffic carries grants that schedule upstream traffic. This grant distribution must take into account the different propagation times required to reach each ONU. In order to deal with this issue, PON standards define the so called ranging mechanism. At first it manages the measurement of the logical distance between each ONU and the OLT. In this way the grants scheduling the upstream traffic can be adjusted accordingly to the transmission time so that upstream frames sent by different ONUs do not collide.
As ONUs receive all the frames with constant power, they can be equipped with simpler receivers leading to reduced costs. On the contrary the OLT receives frames at different powers, making the recovery of synchronization much more difficult. Therefore a burst-mode receiver is required at the OLT. This component is more complex than the one placed at the ONU and more expensive.

A typical optical line terminal (OLT) in TDM technology has the following structure (Figure 2.5):

![Figure 2.5 Generic structure of a TDM PON optical line terminal](image)

Multiple OLT cards like the one sketched above can be placed at the CO (central office). Each OLT card has its own MAC (medium access control) and PMD (physical media dependent) layers and serves a separate PON. The service network interface (SNI) acts as the interface between the OLT and the backbone network. The service adaptation layer provides conversion between the backbone signal formats and PON section signals. The MAC layer schedules the right to use the shared optical link in order to prevent collisions among different ONUs transmitting upstream. The PMD layer defines the optical transceiver and the wavelength diplexer.
On the other hand, a typical optical network unit (ONU) in TDM technology has the following structure (Figure 2.6):

![Diagram of a TDM PON optical network unit](image)

**Figure 2.6 Generic structure of a TDM PON optical network unit**

The PMD and MAC layers have approximately the same functions as in the OLT. The only difference is that, while the ONU’s MAC acts like a slave, the OLT’s MAC acts like a master during the assignment of time slots over the shared physical medium. The service MUX/DMUX provides multiplexing functions for different client interfaces. The service adaptation layer provides conversion between the signal format required for the client equipment connection and the PON signal format. An ONU may provide multiple user network interfaces (UNI) for different types of services (data, voice) and each UNI may support a different signal format and require its own particular adaptation service.

### 2.2 TDM-PON STANDARDS

Among other things, standards define the characteristics of the physical layer, the downstream and upstream frame formats, the upstream grant distribution mechanism and the ranging procedure. The most common commercial standards are G-PON and EPON.
G-PON stands for gigabit-capable PON and is covered by the ITU-T G.984 series standards. It is an evolution of the B-PON (broadband PON) standard supporting higher data rates. Both upstream and downstream transmissions are structured into 125 µs fixed size frames. These are encapsulated using GEM (GPON encapsulation mode), a technique that provides fragmentation of the payload data together with multiplexing functions. This feature allows for an easier adaptation to different data formats. Each downstream frame carries an overhead containing a frame synchronization sequence, PLOAM (physical layer operation administration and management) messages and the so called upstream bandwidth map. This field defines for the ONUs the time slots during which they are allowed to transmit traffic in the upstream direction.

EPON stands for Ethernet PON and is covered by the IEEE 802.3 standard. The data traffic is encapsulated into Ethernet frames and operation runs at standard Ethernet speeds. EPON has been optimized for Ethernet packet transport and variable length frames are employed at the transport layer. Therefore circuit emulation is needed to support fixed bandwidth TDM circuits. MPCPDUs (multipoint control protocol data units) are particular types of Ethernet frames used for some control purposes. For example they are used to perform ONU discovery and ranging functions. Furthermore they also provide support for the arbitration mechanism of upstream medium access among multiple ONUs.
3 Power saving techniques via sleep mode in TDM PONS

Power conservation and CO₂ footprint reduction have become increasingly important aspects in designing access networks [1]. This concern is mainly due to the desire of improving the performances and service availability of battery-powered operations and of reducing the costs related to power consumption both in terms of global warming and of network costs [2]. A large portion of the total power consumption is due to the activity of ONUs located at the user premises. Thus a number of different solutions have been proposed in order to reduce the power required by an ONU. These solutions usually exploit the fact that the ONUs are rarely used at their full potential and thus a lot of functionalities can be unpowered when inactive. According to a study about power saving in PONs conducted by ITU-T [2], those techniques that operate during main power failures are of primary interest because they allow to reduce the size and cost of backup batteries. Secondly, obviously, it is important to reduce the average power consumption at all times. Furthermore these objectives should be reached sacrificing neither quality nor availability of service and some basic services like POTS (plain old telephone service) should always be available.

It is also important to keep in mind that these power saving techniques can require modifications of the hardware at the OLT and/or at the ONU, implying higher complexity and costs. This requirement is not a real issue if it impacts only the OLT while it can become a serious limit to the feasibility of these solutions if the additional hardware is required at ONUs. In fact the ONUs cost must be kept low as it is not shared among the customers.

3.1 CLASSIFICATION OF TDM-PON POWER SAVING TECHNIQUES

The most popular and discussed solutions to improve the power consumption on the ONU side can be classified into: ONU power shedding, ONU dozing, ONU deep sleep and ONU fast sleep techniques. This section presents each of these techniques according to the classification and description reported in [2].
ONU power shedding

The power shedding technique exploits the fact that some ONU’s functionalities may be inactive or at least non-essential for a certain period of time and thus can receive a reduced amount of power or be switched off. This effort to reduce the consumed power still keeps the optical link fully operational. This means that the ONU transceiver is always active.

Traditional power shedding mode is applied only in case of main AC power supply failure. According to this solution, each interface type of the ONU is associated to a particular shedding class. Each class is characterized by a static time parameter indicating the interval separating the moment when the relative interface type’s support must be switched off from the moment of main power failure.

An extended power shedding technique has also been proposed in order to apply it to more situations than just AC power failure. This solution allows the customer to inform the operator about specific time periods during which certain services are not used. In this way the operator has the opportunity to turn off those interfaces at the user’s ONU that are not required. This solution can be desirable for both the operator and the customer: while the operator does not need to decide when the ONU interfaces must be powered down, the customers can control their own consumptions.

In any case it is the OLT that is responsible for controlling and managing the power saving service. The ONU removes and restores the power when prescribed by the OLT.

One advantage of power shedding is that it is a well understood and experimented technique because it is already largely applied in cellular phones, laptop PCs and monitors industries. Furthermore, according to an ITU-T study, power shedding can save over 70% of active power for a typical North-American ONU while the size of the backup battery can be reduced by more than 50% [2].
ONU dozing

The ONU dozing technique prescribes that the ONU transmitter can be powered off for certain periods of time while the ONU receiver must remain on all the time. This means that a dozing ONU ignores its upstream allocations as long as it has no traffic to send but it keeps the downstream link fully operational. This last fact allows continuous delivering of traffic to the customer premises equipment. A dozing ONU can be waken up by a specific OLT request or by a local stimulus. Examples of stimuli can be the reception of upstream traffic from any UNI or traffic generation due to an internal process. In the meantime the OLT must send upstream grants to the dozing ONUs, without expecting any answer, so that they can recover immediately when they have traffic to send.

ONU deep sleep mode

The deep sleep technique is characterized by the fact that the transceiver and most functionalities of the ONU remain completely off for the entire duration of the power save state. Just some basic functions remain optionally active, like activity detection and some local timing. This fact allows this solution to reach maximal power saving.

A deeply sleeping ONU can wake up when it is switched on by the customer or when a local timer has expired. The OLT must be informed of the ONU’s transition to the deep sleep state, in order to avoid unnecessary alarming. While the ONU is deeply sleeping, the OLT can decide whether to keep on transmitting or discard downstream traffic. It can also allocate upstream traffic for the sleeping ONU but it should not expect any answer. Anyway the OLT should allocate regular narrow upstream grants to the sleeping ONU so that it can wake up and recover in a reasonable time.

This technique is especially useful in particular situations. For example, when the customer switches the terminal equipment off or when the service provider believes that the loss of service generated by the deep sleep technique can be tolerated.
ONU fast sleep mode

The ONU fast sleep technique prescribes that, while in the power save state, an ONU goes through a sequence of sleep cycles, each composed by a sleep and an active period. During sleep periods the ONU behaves just as if it was in deep sleep mode, meaning that it is completely powered off, apart for some timing and activity detection functions that remain active. During active periods, instead, the ONU is normally active. The transitions between these two different periods are synchronized among all ONUs in fast sleep mode and they are controlled by the OLT. In fact an ONU enters the sleep period when it receives the related message from the OLT. As for the opposite state transition, an ONU wakes up when its timer expires and generates a wake-up signal after a time prescribed by the OLT. After the ONU has woken up it enters a synchronization state before recovering completely to normal operation. While an ONU is sleeping, the OLT buffers the downstream traffic addressed to it, so that it can be delivered as soon as the ONU wakes up.

3.2 STATE OF THE ART OF TDM-PON POWER SAVING TECHNIQUES

Even though a TDM PON access network can experience a low average utilization, it must still provide support for high peak data rates. In particular, the OLT broadcasts downstream traffic to all ONUs that remain active even if they are not the destination of any data. Thus the transceivers located at ONUs are oversized when compared to the average data rate. Therefore it would be very sensible to power off ONU transceivers at points when they are not receiving user traffic. This means entering the so-called sleep state of the ONU in order to reduce power consumption.
3.2.1 Characteristics of the sleep state

A sleep mode solution allowing for minimal energy consumption, represents a scenario where during the sleep state the ONU transceiver is completely powered off. Actually, some parts of the back-end digital circuit, such as the clock and volatile memory, are kept on and thus the ONU still consumes some power even during the sleep period.

In the method described in patent [7], if a downstream service begins while the ONU is still sleeping, the received data will be stored in a buffer at the OLT for later transmission with no packets being lost. In this way downstream traffic will exhibit an increased latency, bounded within the sleep period. According to [7] the layers above the MAC layer are unaware of the temporary inactivity of lower layers thanks to the buffering mechanism done at the MAC layer.

3.2.2 Energy conservation limitations due to the ONU transceiver

The ONU’s transition from the sleep state to the active state (wake-up) is not instantaneous but it requires a finite amount of time, an overhead. This overhead is partly due to resynchronization between the OLT and the ONU. In fact, in order to return to full operation after a sleep period, the physical layer needs to determine the level of gain required for the amplifying stages, clock frequency and phase while the MAC layer needs to resynchronize line framing.

The time required to recover frame synchronization strictly depends on the length and the structure of the frames. Thus it depends on the standard that is implemented. The other portion of the wake-up time, instead, is due to the ONU transceiver that needs to recover the OLT clock and turn on its laser again. In TDM-PONs, ONUs keep their local clock synchronization with the OLT clock by recovering it from the continuous bit stream they receive from the OLT. This is accomplished by means of the particular block called clock and data recovery (CDR) circuit. If an ONU stops receiving downstream transmission because it is in the sleep state, ONU clock synchronization is lost and the transceiver needs a certain clock recovery time. As a result, the
improvement in energy conservation via sleep mode is limited because the ONUs spend quite a long time (~2 ms) awake performing synchronization duties [8].

The sleep mode technique should take into account the wake-up delay introduced by the ONU transceiver. This can be done by setting a constraint on the shortest allowable sleep period that can be scheduled.

### 3.2.3 Duration of the sleep state

The optimal length of a sleep period depends on the traffic load, the desired energy saving and the QoS requirements. If scheduling of sleep periods is considered valuable even when the QoS requirements are quite stringent, then short sleep periods are preferable. The QoS requirements are met if the sleep state duration does not exceed the maximum tolerable additional queuing delay due to the presence of the sleep mode technique. But the length of sleep periods has a lower bound. This is set by the wake-up time required by the ONU transceiver to return to the active state. In fact, in order to have a benefit in terms of energy conservation, the sleep period must be longer than the wake-up time. On the other hand, if energy saving is the primary objective and QoS impairments are not so stringent, long sleep periods should be scheduled. These long periods worsen delay and jitter performances, thus the QoS requirements set an upper bound for the duration of the sleep state. In conclusion, the length of the sleep period should be determined as a result of the trade-off between QoS requirements and desired energy conservation. [9]

The sleep state duration can be determined once and kept fixed for all ONUs or it can be tailored to fit different ONUs and/or changed in time in order to allow for a dynamic adaptation to real-time traffic conditions. The second alternative leads to a better energy efficiency but it requires a certain overhead in terms of coordination and communication between the OLT and the ONUs. Furthermore it would be desirable to synchronize sleeping periods for all ONUs entering the sleep state in order to reduce the messaging overhead.
In [10] the authors suggest a method for allowing the OLT alteration of the frequency of the ONU transition to the active state according to current network load. They state that this can be accomplished exploiting existing protocols and they provide a possible application to EPON as an example. In that paper a formula is proposed for computing the timeout value of the OLT timer triggering the WAKEUP messages to the ONU:

\[
\text{Timeout} = \frac{\text{FrameLength} \cdot 8}{\text{Bandwidth} \cdot \text{Ratio}}
\]

\text{Formula 3.1}

FrameLength is the frame length in bytes, Bandwidth is the bandwidth used by every ONU and Ratio is a coefficient defining the rate at which the system should slow down the generation of keep-alive messages from OLT to ONU. This coefficient is computed just once setting Bandwidth to the maximum allowed value (maximum traffic load) and setting Timeout to the value prescribed by the current standard. This way of choosing Ratio assures backward compatibility. In fact those ONUs that do not support this adaptive power saving technique can operate with the standard timeout value, while the ones supporting it can indicate this feature to the OLT during the ranging process. The measurement of the current bandwidth usage for the given ONU can be obtained exploiting the messages used for DBA (dynamic bandwidth assignment). The authors of [10] state that along with the decrease in bandwidth use, the frequency of the keep-alive mechanism diminishes. This then leads to a minimization of the time the ONU is on and to a reduction of power consumption. However they also mention that the ratio between Timeout and Bandwidth is useful to facilitate calculations but it actually still needs experimental verifications.

The authors in [11] suggest a simpler method for determining the duration of the sleep period though still providing some adaptation to the traffic conditions. According to their proposal, the sleep period \( T_s \) is computed depending on the downstream frame inter-arrival time \( i \). The function that allows determining \( T_s \) once that \( i \) is known is sketched in Figure 3.1. The threshold values \( i_{th1} \) and \( i_{th2} \) as well as the sleep times \( T_{s1} \) and \( T_{s2} \) must be carefully chosen on the basis of the maximum allowable traffic delay and on the average traffic load.
A similar method is adopted also in [12]. Here the authors use another function to determine the length of the sleep period, where the chosen length can vary within a minimum and maximum value, as shown in Figure 3.2:

Figure 3.1 Sleep period as function of the frame inter-arrival time [11]

Figure 3.2 Sleep period as function of exponentially smoothed average of inter-arrival times [12]
The \( i_{\text{average}} \) variable used as x-axis in Figure 3.2 is obtained as an exponentially smoothed average of the inter-arrival times between downstream frames addressed to a specified ONU, according to the following formula:

\[
i_{\text{average},k} = c \cdot i_{k-1} + (1 - c) \cdot i_{\text{average},k-1}
\]

Formula 3.2

where \( i_{k-1} \) indicates the frame inter-arrival time at time \( k-1 \), \( i_{\text{average},k-1} \) indicates the average of inter-arrival times at time \( k-1 \) and \( c \) is a smoothing factor \( (0 \leq c \leq 1) \).

In order to avoid long queuing times (and thus high traffic latency) in the OLT or ONU buffers, the authors in [8] proposed a so called just-in-time sleep control (JIT-SC) scheme. This scheme is based on current DBA algorithms but it further provides variable sleep time assignment. The information about the sleep period allocation is carried in those sections of the frame that are usually employed for DBA purposes. For example, in EPON this is accomplished through GATE messages, while in GPON it is the upstream bandwidth map field of the downstream frame that contains the information about the scheduling of the sleep period.

With JIT-SC, if the upstream and downstream traffic have similar loads, the OLT controller tries to match the downstream traffic slot with the allocated upstream traffic slot. The authors claim that in this way minimum downstream OLT backlog can be expected, thanks to the exploitation of results known from opposite upstream DBA algorithms. In this way the ONU performs sending and receiving operations at the same time and then it can enter the sleep state, avoiding the storage of downstream frames at the OLT. If traffic load is asymmetrical or if significant backlog is accumulated at OLT, the OLT controller can schedule a shorter sleep time, thus allowing a more flexible downstream traffic scheduling.

### 3.2.4 Conditions triggering the sleep state

The condition that is usually adopted for entering the sleep state is the absence of messages to be sent downstream (from the OLT to the ONUs) or to be sent upstream (from the user to the OLT), meaning that no service is active.
In order to trigger the entering of the ONU into the sleep state, the system activity (or inactivity) must be monitored. The patent [7] suggests a few ways to determine the network activity. The first method prescribes to meter the traffic flowing through the ONU, the second one suggests monitoring upper-layer control messages indicating termination and initiation of traffic, while according to the third one external indications of system activity should be probed. In [13] the authors suggest another method to control the level of activity of the downstream traffic towards a specified ONU. The method exploits the downstream frame interval as key parameter triggering the start of the sleep state. A certain threshold value needs to be determined on the basis of desired traffic latency and average traffic bandwidth. They also mention that another triggering parameter could be the downstream queue length at OLT. Also the authors of [12] suggest monitoring the downstream frames inter-arrival time in order to trigger the ONU sleep state. But they propose a slightly different method. In fact an exponentially smoothed average is computed on the time interval between frames. Then this value is compared to carefully defined thresholds so that the sleep condition is triggered.

It would be desirable to guarantee the possibility of returning to the active state (wake-up) either when downstream or upstream traffic need to be delivered. Thus the wake-up procedure should support both OLT and ONU initiation. Even though it would be better in terms of energy conservation to keep the ONU transceiver off during the whole sleep state, it could be necessary to wake it up periodically in order to collect potential wake-up signals from the OLT if OLT-initiated wake-up needs to be supported. This fact can strongly impact the maximum achievable energy saving.

### 3.2.5 Conditions triggering the return to the active state

It would be desirable to return to the active state (wake-up) both in case of incoming traffic from the network side (downstream) and in case of traffic coming from the user side (upstream). In this case the waking-up procedure should therefore support initiation by both the OLT and the ONU. As an alternative, the wake-up of ONUs can be triggered by the expiry of a pre-charged timer. This simplifies the wake-up procedure, but still there are some issues to deal with. For example, when it is necessary to guarantee wake-up in case of urgent incoming traffic (either from the network or from the user), interruption of the sleep period must be supported.
An issue related to ONU wake-up is the clock drift (Δ). This is a phase error of the ONU clock with respect to the OLT clock, accumulated during the sleep period. In fact, the sleep duration is timed by a local oscillator inside the ONU that goes in “free-running” mode while the ONU is sleeping and thus accumulates phase errors. Because of this phase drift, the ONU may wake up too late with respect to what the OLT expects and miss a downstream frame. In order to mitigate this problem, the authors of [8] suggest instructing the ONU to wake up a Δ earlier than scheduled so that no messages are lost.

### 3.2.6 Messaging schemes supporting the sleep mode

The transitions between the active and the sleep states of the ONU can be initiated by the OLT or by the ONU itself, but these are not the only existing alternatives. It is actually possible to drive state transitions making use of timers (at OLT or at ONU) pre-charged with a certain time value, triggering the transition when they expire. The second solution does not allow much flexibility but requires less communication overhead between OLT and ONU.

In [14] the author proposes a control message flow allowing both OLT- and ONU-initiated transitions. The transition from the active state to the sleep state can be initiated by the ONU sending a sleep request (containing the maximum allowed sleep period) to the OLT. The OLT then accepts it sending back a sleep message containing the specification on the actual sleep period. This transition can also be initiated by the OLT that can directly send the sleep message to the ONU. When the sleep period has expired, ONU wakes up and immediately sends another sleep request to the OLT. If the OLT has no traffic to send, it answers with a sleep message. On the contrary, if OLT has downstream traffic for the specified ONU it denies the sleep request sending the data. In this case an OLT-initiated wake-up takes place. If the ONU realizes that it has upstream traffic to send, it waits until the sleep period has expired. Then, instead of sending a sleep request to the OLT, the ONU sends a message indicating that it has traffic to send. This mechanism provides ONU-initiated wake-up.

The authors of [13] suggest a similar SPW (sleep and periodic wake-up) operation with some differences. The transition to the sleep state is OLT-initiated by a request message sent periodically to the ONU every time its sleep period expires. This message also contains the duration of next sleep period. The value of this period can be a finite value or a zero value,
depending on the current downstream frame interval. In the second case the request message actually is a wake-up request and provides OLT-initiated wake-up. When the ONU detects that upstream traffic is received on the UNI, it refuses the OLT request by sending an appropriate message to the OLT. This provides ONU-initiated wake-up.

Then it is also very important to guarantee that the OLT is aware of the ONUs’ state at all time, otherwise service continuity and availability would be seriously penalized. This can be achieved by making use of acknowledgement messages, as suggested in [7]. The proposed solution is very similar to the one suggested in [14] but it provides some more expedients. To avoid lack of coordination between OLT and ONU, every ONU request is sent three times. Furthermore the ONU keeps on sending its request until it receives an acknowledgement from the OLT, even if the conditions that generated that request are no longer valid. This acknowledged handshaking guarantees that in case of message failures, service continuity and/or availability are not affected. In fact only two kinds of failures can happen: an ONU sleep request is lost and ONU remains active or an ONU wake-up request is lost and the OLT thinks that the ONU is sleeping while it is active. In both cases the only impairment is constituted by a less energy-efficient operation.

3.2.7 Solutions for improving the ONU wake-up time

The current architecture of the receiver front-end analog circuit in an ONU transceiver is sketched in Figure 3.3.

The authors of [15] proposed two different solutions (shown in Figure 3.4 and 3.5) for implementing the CDR (clock and data recovery) block in order to achieve a shorter clock recovery time and thus improve the overhead time occurring at ONU’s wake-up. In fact they claim that having a shorter overhead has better effects on energy efficiency than further reducing the energy consumption during sleep mode. This would be because the scheduled sleep periods are expected to be quite short in TDM networks.
The first proposed implementation of the CDR circuit is sketched in Figure 3.4:

In this solution clock synchronization is kept within a certain tolerance during the sleep period by means of a local oscillator (LO), allowing for a faster clock recovery. This is further accomplished by replacing the continuous mode (CM) CDR circuit with a burst mode (BM) one.
that adapts faster. When the ONU enters the sleep state, the mode switching block receives a SLEEP signal and it selects the LO path instead of the data path as input to BM-CDR. The counter then uses the BM-CDR clock to time the sleep period. When it expires, it sends a WAKEUP signal to the mode switching block that switches back to the data path as input to the BM-CDR. The authors claim that this solution has low additional costs because BM-CDR circuits rely on similar semiconductor manufacturing processes and materials as CM-CDR ones and the additional local oscillator is a low-cost component. Anyway, as it is desirable to keep the ONU cost as low as possible, the authors suggested a second solution that does not add any new component to the ONU. This is sketched in Figure 3.5:

In this solution clock synchronization is maintained because the CM-CDR circuit, the avalanche photodiode, the transimpedance amplifier (TIA) and the limiting amplifier (LIA) are kept on during the sleep period. A mode switching block selects the counter input when the ONU enters the sleep state. The counter then uses the recovered clock to time the sleep period. When it expires, it sends a WAKEUP signal to the mode switching block that switches the CM-CDR output to the DMUX in order to resume the normal receiving operation. This second implementation achieves only a minor energy saving because it does not deactivate the high-speed front-end components during the sleep period, but does not either impose additional cost.

Figure 3.5 Second proposal for a faster ONU receiver [9]
Another method improving the ONU wake-up time is proposed in [7]. In this case the time that is reduced is the one required by the resynchronization of line framing. In order to accomplish this, a parallel state machine instead of a serial one can be implemented. In this proposal, several sync events are checked concurrently. In case of false-pattern detection the machine is not cleared but it just needs to return one state back, improving the lock time. Re-ranging is not necessary: as suggested in [7] the detected changes in Round Trip Delay (RTD) can be sent to the ONU by the OLT just after the end of the sleep period. Therefore the ranging procedure does not need to be repeated.
4 Proposal of a new sleep mode scheduling technique

The new sleep mode scheduling technique that is illustrated in this chapter could be classified as a fast sleep technique, according to the scheme defined in [2] and reported in section 3.1. The sleep cycles are set by monitoring downstream traffic flows towards the ONUs. The ONUs sleep cycles are independent and not synchronized with each other. The sleep state can be triggered at different times and can have a different duration for every ONU. The length of a sleep period is computed by means of a statistical analysis of inter-arrival times between downstream frames addressed to a specific ONU. This is performed at the OLT, heading to OLT-initiated triggering of the sleep state. The actual scheduling of sleep cycles is divided into two approaches depending on the level of delay sensitivity of active services.

4.1 OPERATIONAL SCHEME

The operational scheme adopted in order to support the proposed sleep mode scheduling is quite similar to the one proposed in [12] and [13]. Its functioning is explained below and illustrated in Figure 4.1.

The sleep state of the ONU is triggered by a message received from the OLT. Thus the sleep operation is OLT-initiated. This SLEEP message is generated at the OLT when certain conditions on downstream traffic towards the specified ONU are detected. It contains the instruction about the length of the sleep period.

When the ONU receives the SLEEP message, it decides whether to accept (Figure 4.1 (a)) or refuse (Figure 4.1 (c)) the OLT request. This decision depends on the presence of traffic to be sent on the upstream link. If there is no traffic that needs to be sent upstream, the ONU accepts the SLEEP request sending an ACK message, otherwise it refuses it by sending a NACK message. In the first case the ONU enters the sleep state while in the second case the normal operation is resumed until a new SLEEP message is generated at the OLT.
During the sleep period the OLT buffers all the incoming downstream frames addressed to the sleeping ONU. If the estimation error of the sleep period is too big, this buffering mechanism could result into an overflow implying packet loss. It would be preferable to have an OLT implementation having separate queuing buffers for each ONU. In fact, with a shared buffer implementation, the overflow caused by one sleeping ONU would cause degraded performances for all served ONUs. With private buffers, instead, only the service to the sleeping ONU would be affected.

During the sleep period the OLT keeps on periodically granting some upstream bandwidth allocations to sleeping ONUs. In this way an ONU will be able to send its WAKEUP signal when the sleep period has expired, or when it needs to pre-maturely wake-up.

The sleep period is timed by a timer located at the ONU running on a local oscillator.

The ONU can wake up either upon expiration of its timer or upon detection of incoming upstream traffic, like in Figure 4.1 (b). In order to provide these features it is necessary to maintain powered up both the timer and some activity detector at the ONU during the sleep period. Upon waking-up the ONU sends a WAKEUP message to the OLT.

When the OLT receives the WAKEUP message from the ONU, it can send the downstream traffic stored in the buffer. Normal operation is resumed until a new SLEEP request is generated.

According to this operational scheme, three different scenarios may occur. These are sketched in Figure 4.1.

A fundamental requirement in this handshaking scheme is that the OLT is always aware of the actual state of the ONU. This can fail if a message of the operational scheme is lost. For example, if the SLEEP request does not reach the ONU, the ONU will remain active and the OLT will buffer the associated downstream traffic while waiting for an ACK/NACK message. These messages will never arrive thus causing an overflow of the buffer with subsequent loss of frames. In order to prevent this kind of situations, every message is sent three times, so that the probability that it gets lost is reduced. A similar technique has been already proposed in [7].
According to this operational scheme, three different scenarios may occur. These are sketched in figure 4.1:

If a G-PON standard is implemented, the messages required by the proposed operational scheme can be implemented as PLOAM messages. The signalling method can be similar to the one presented in [7].

### 4.2 CONTROL MECHANISM

The control mechanism takes care of triggering the forwarding of sleep requests and of computing the length of sleep periods. In the proposed technique both issues are coped with by monitoring the inter-arrival times of downstream frames addressed to a specified ONU. This is a key parameter since the ONU is idle between two consecutive frame arrivals, if upstream transmission is neglected.

The statistical method used to compute the length of the sleep periods can be adapted to different delay requirements. The method providing the identification of delay-sensitive services needs further investigation. For example, the upstream traffic could be monitored in
order to detect those services characterized by a high interactivity that makes them very delay-sensitive.

Some additional features are also provided. The OLT buffers are monitored to prevent the dispatch of a sleep request in case a downstream frame reaches the OLT before the request has been sent. Furthermore, those situations when no services are active can be detected allowing for a more power aggressive method.

All functions related to the control mechanism operate at the OLT. Thus the OLT must be provided with additional functionality by adding appropriate hardware components.

**4.2.1 Prediction of idle periods**

The core of the control mechanism is based on monitoring the inter-arrival times of downstream frames addressed to a specified ONU. A statistical approach is adopted in order to predict the next inter-arrival time on the basis of the past behaviour of downstream traffic.

At first, I assume that the frames addressed to a specified ONU arrive at the OLT following a Poisson distribution. Thus the associated inter-arrival times will follow an exponential distribution. The mean value can change over time because of changes in running applications or in traffic load. Thus, I assume that the inter-arrival times follow a series of exponential distributions with different mean values.

The first step of the proposed method is the estimation of the mean value of the current distribution of inter-arrival times. This is accomplished by means of an arithmetic average computed on \( n \) samples:

\[
average(k) = \frac{\sum_{j=0}^{n-1} iat(k-j)}{n}, k \geq n-1
\]

**Formula 4.1**
where \( k \) is a discrete variable counting the arrival events of downstream frames addressed to a specified ONU at the OLT and \( iat(x) \) is the inter-arrival time between the \( (x-1)^{th} \) and the \( x^{th} \) arrival.

The second step is the actual estimation of the next inter-arrival time. This is accomplished by means of an exponentially smoothed average. The general expression for an exponentially smoothed average is:

\[
ESA(t) = c \cdot Y(t-1) + (1-c) \cdot ESA(t-1) , \quad t > 0 , \quad ESA(0) = 0
\]

**Formula 4.2**

where \( t \) is a discrete variable indicating discrete time samples, \( c \) is a coefficient representing the degree of weighting decrease, with a value between 0 and 1 and \( Y(t-1) \) is the observation at time \( t-1 \).

This type of average assigns different weight factors to the considered samples that decrease exponentially. Thus it gives more weight to the most recent samples. The coefficient \( c \) can be expressed in terms of \( N \) time periods:

\[
c = \frac{2}{N + 1}
\]

**Formula 4.3**

My method adopts the exponentially smoothed average as an estimation of the next inter-arrival time for two reasons. Firstly, the ratio between the weights assigned to the previous value of the average and to the last sample can be easily modified by tuning the coefficient \( c \). This means that the weight of the past history can be changed through \( c \). Thus the effective width of the sampling window can be dynamically adapted by means of a modification of the weights. Secondly, the fact that the most recent samples have higher weights, makes the exponentially smoothed average adapt quickly to changing traffic conditions.

In the proposed technique the coefficient \( c \) is tuned accordingly to the number of past samples that are considered useful for the estimation. This is accomplished by directly associating this number to the number of time periods \( N \).
If the arithmetic average changes by more than one time its value, then I infer that the mean value of the distribution of inter-arrival times is changing, producing a different distribution. Thus only the samples inside the new distribution should be considered in order to obtain a good estimation of the next inter-arrival time because this will be part of the current distribution. If the condition of change of the mean value is detected, the sample history of the exponentially smoothed average (all previous samples) is discarded by setting $N$ to 1 and, consequently, $c$ to 1.

The described technique allows the exponentially smoothed average to quickly adapt to a changing traffic load. For example, this can happen when the user activates a new service. If such a condition does not occur, a greater number of samples are considered. This is accomplished by increasing $N$ by one every time a new inter-arrival time inside the current distribution is sampled. Thanks to this technique the exponentially smoothed average takes into account all the samples inside the current distribution. By doing so, the presence of some regular patterns in the traffic can be exploited in order to provide a better estimation of the next inter-arrival time. The logical expression implementing the described mechanism is reported below:

$$\text{if } (\left|\text{average}(k-1) - \text{average}(k-2)\right| > \text{average}(k-1) \text{ or } \left|\text{average}(k-1) - \text{average}(k-2)\right| > \text{average}(k-2))$$

$$\text{ESA}(k) = \text{iat}(k-1)$$

$$\text{reset} = k - 1$$

$$\text{else}$$

$$\text{ESA}(k) = \frac{2}{k - \text{reset} + 1} \cdot \text{iat}(k-1) + \left(1 - \frac{2}{k - \text{reset} + 1}\right) \cdot \text{ESA}(k-1)$$

**Formula 4.4**

### 4.2.2 Scheduling of sleep periods

The $ESA$ value is considered to be a prediction of the next inter-arrival time. Its value is used both to trigger the sleep state and to compute its duration. The actual details of this mechanism depend on what kind of services are running.
If delay-sensitive services are active, then the sleep technique adopts a more conservative approach: a maximum additional delay is guaranteed but energy saving is limited. On the contrary, if the active services are not so critical in terms of delay, a more aggressive technique can be adopted. In this case energy saving is enhanced but the additional delay due to OLT buffering is worse.

**Method for delay-sensitive traffic**

The described scheduling method is quite similar to the one proposed in [12]. If delay-sensitive services are detected, the length of the sleep period is computed according to the following function:

![Figure 4.2 Sleep period as a function of exponentially smoothed average of inter-arrival times for delay-sensitive traffic](image)

$T_1$ corresponds to the minimum sleep period worth scheduling while $T_2$ corresponds to the maximum sleep period that can be scheduled. As the ONU will actually be turned off for the sleep period reduced by the wake-up time, the minimum sleep period worth scheduling must
be longer than the wake-up time. Therefore $T_1$ is set to an integer multiple of the wake-up time. This value is considered as the minimum schedulable sleep period. The value of $T_2$ is set equal to the maximum added forwarding delay that could potentially be introduced. In fact the maximum queuing delay caused by the sleep technique is equal to the maximum sleep period that can be scheduled.

If the value of $ESA$ is below a certain threshold $thr_1$, then the length of the sleep period $T_{sleep}$ is zero and no sleep requests are issued. Otherwise, a sleep request containing a certain sleep period length is issued. The sleep period length indicated in this request will depend on the $ESA$. When the $ESA$ value is between thresholds $thr_1$ and $thr_2$ the sleep period will be half of the $ESA$, otherwise it will equal $T_2$. Thus, the values of the two thresholds $thr_1$ and $thr_2$ are set respectively to $2*T_1$ and $2*T_2$.

**Method for delay-insensitive traffic**

If the active services are not so critical from a delay point of view, a more aggressive technique is adopted. Here no maximum length for the schedulable sleep periods exists.

Sleep requests are triggered when the $ESA$ value crosses a certain threshold. This threshold is equal to $thr_1$ used in the method employed in case of delay-sensitive traffic.

The computation of the sleep periods included in the sleep requests is divided into two phases, as shown in Figure 4.3. In the first phase the sleep period is assigned half the value of the $ESA$. If a new downstream frame for the specified ONU does not arrive during this first sleep period, the second phase starts. Here the sleep period is set to a fraction of the $ESA$ value lower than one half. These shorter sleep periods are then repeatedly rescheduled until a new downstream frame for the sleeping ONU arrives at the OLT. The resulting scheduling of sleep periods is sketched in Figure 4.3.
This mechanism offers some protection against estimation errors deriving from the computation of the exponentially smoothed average, thanks to the rescheduling of shorter sleep periods after a longer one.

If the shorter sleep period computed as a small fraction of the ESA value results to be shorter than the minimum schedulable sleep period, than I infer that it would not be worth scheduling. Thus the sleep period is upgraded to the minimum sleep cycle value. In my proposal this is set to an integer multiple of the wake-up time, as mentioned in the previous section.

4.2.3 Additional features

This section presents some features that have been added to the proposed sleep mode technique in order to improve its performances.
Coping with errors

The typical sleep period scheduling situation is sketched in Figure 4.4 for both delay-sensitive and delay-insensitive traffic:

As shown in Figure 4.4, the sleep request containing $T_3$ is triggered at $k+1$ but, as the ONU is still sleeping because $T_2$ has not expired yet, $T_3$ will be actually dispatched at time $x$. But $T_3$ is a prediction of the time between $k+1$ and $k+2$, not between $x$ and $k+2$. If no actions are taken the errors will sum up as time goes by. Thus the delay generated by $T_2$ must be subtracted from $T_3$ before issuing the associated sleep request.
Buffers monitoring

The worst-case queuing additional delay is introduced when a downstream frame for a certain ONU arrives at the OLT just after the triggering of a sleep request. This undesirable situation can be partially avoided by monitoring the OLT buffer.

The inter-arrival times are measured at the service network interface (SNI), before the frames can enter the OLT buffer. Also sleep requests are generated before the OLT buffer and they must enter it before being sent to ONUs. In my proposal a flag is set to 1 when a sleep request is triggered and inserted into the OLT buffer. A different flag exists for every served ONU. When the sleep request exits the buffer and is sent to the ONU, the associated flag is reset to 0. Thus a flag is 1 only while a sleep request for the associated ONU is stored in the OLT buffer.

Every time a downstream frame arrives, the value of the flag associated to the addressed ONU is checked. If it is 1, then the sleep request stored in the buffer would generate a long queuing delay because a new frame has already arrived at the OLT. Thus a signal is sent to the OLT buffer instructing it to cancel the stored sleep request, so that the newly arrived frame can be normally sent. On the contrary, if the flag is 0, the sleep request is dispatched to the ONU as usual.

Idle ONUs

Another feature is added in order to detect those situations characterized by the absence of active services. These cases constitute a great opportunity for energy saving since idle periods are usually long. In order to take advantage of these situations, the output of the timer that measures inter-arrival times is continuously compared with a threshold. In my proposal, this is set to a quite high value, around 10 s. When the instantaneous value of $i_{at}$ crosses this threshold, a sleep request is triggered. This will instruct the ONU to sleep for a long time, such that the wake-up time is negligible. My suggestion is to set it to 5 s. These long sleep periods are repeatedly scheduled until a new downstream frame addressed to the specified ONU arrives and normal sleep operation is resumed.
4.3 SIMULATIONS

This section has the purpose of showing the results of some simulations testing both power and latency performances of the proposed sleep mode technique. The Matlab code used for the simulations (reported in Appendix I) generates distributions of inter-arrival times emulating the downstream traffic transmitted from the OLT to one ONU. Then the proposed sleep mode scheduling is implemented in the code with some approximations in order to extract both power and latency performances.

In order to evaluate the performance of the prediction method itself, also a simplified version of the proposed technique is implemented in the Matlab codes. This method uses the arithmetic average as the prediction for frame inter-arrival times, skipping the usage and adjustment of the exponentially smoothed average. The scheduling method, instead, is kept unchanged since the purpose of the comparison is to understand whether the proposed prediction method is valuable or not.

In the simulations the frame inter-arrival times follow an exponential distribution, assuming that downstream frames arrive at the ONU according to a Poisson distribution.

As mentioned above, the proposed sleep mode technique is not completely described in the Matlab codes since some approximations have been made. First of all, the presence of upstream traffic is discarded and the messaging between the OLT and the ONU is not included in the simulations. Thus the feature described in section 4.2.3 allowing for the erasing of waiting sleep requests in the OLT buffers could not be simulated. In fact the sleep requests are immediately executed since no messaging is present in the simulations. Furthermore, the function detecting idle periods in the downstream transmission (described in section 4.2.3) is not present in the simulations since the generated distribution would never allow the activation of this mechanism.

In order to actually implement the proposed technique, all the control parameters presented as parametric in the previous sections had to assume some specific values. Firstly I chose to compute the arithmetic average on eight samples ($n = 8$) (formula 4.1). Then I set the minimum schedulable sleep period to six times the wake-up time. Finally I used one sixteenth of the prediction as the value for the shorter sleep periods scheduled in case of the delay-insensitive traffic (Figure 4.3).
In order to obtain some numeric results describing the power and latency performances of the proposed technique, some simulations parameters need to be set. Therefore I assumed that the ONU wake-up time is equal to 1 ms, the power consumed by an active ONU is 10 W and the power consumed by an ONU in the sleep state is 2 W. All simulations are repeated ten times and then the results are averaged over this number.

In the first set of simulations the proposed technique and the simplified one are applied to a number of exponential distributions of inter-arrival times. Each distribution is made of 100 samples and is characterized by a different mean value.

Figure 4.5 shows the ratio of time spent in the sleep state as a function of the mean values of the distributions of inter-arrival times. The term *Tech.1* stands for *technique 1* indicating the proposed sleep mode technique. On the other hand the term *Tech.2* stands for *technique 2* indicating the simplified method using the arithmetic average as prediction of the inter-arrival times. The results obtained for each technique are distinguished depending on which scheduling method is used: the one for delay-sensitive traffic (*d.sens.t.* in the figure) or the one for delay-insensitive traffic (*d.ins.t.*).

If the average frame inter-arrival time is below \(~4\) ms, the sleep time is zero since the ONU is never required to sleep. The graph shows that *technique 1* starts to trigger some sleep periods for lower average inter-arrival times than *technique 2*. This is due to the presence of some inter-
arrival times quite longer than the average that cause the collapsing of the exponentially smoothed average to the last sampled value. This fact leads technique 1 to trigger some sleep periods earlier than the simplified technique. When the average frame inter-arrival time gets higher than ~6 ms, also technique 2 starts to turn off the ONU. From this point and up to ~10.5 ms average inter-arrival time, the figure shows that technique 2 makes the ONU sleep for a longer time than technique 1. This is due to the fact that, once the prediction based on the arithmetic average has crossed the triggering threshold, it decreases less easily than the prediction adopted in technique 1. In fact the exponentially smoothed average could be lowered drastically by the presence of an inter-arrival time much lower than the average value. In the interval between ~10.5 ms and ~80 ms average inter-arrival time, technique 1 shows better energy-saving capabilities than technique 2. The first technique, in fact, can adapt quicker to the changing conditions of the traffic without the risk of resetting its prediction to a value below the triggering threshold since the average values of the inter-arrival times are quite far from it. The figure also shows that when the average inter-arrival time gets very high, the two techniques tend to show the same performances in terms of power.

Figure 4.5 also shows that the scheduling methods used for delay-sensitive and delay-insensitive traffic give similar ratios of sleep time up to an average inter-arrival time of ~100 ms. Above this value the scheduling used for delay-insensitive traffic gives better power performances since it can schedule significantly longer sleep periods than the other scheduling method that is characterized by a maximum allowed sleep period.

The results presented in Figure 4.5 are reflected in Figure 4.6 that plots the power consumed by one ONU as a function of the average frame inter-arrival time.

As mentioned for Figure 4.5, also Figure 4.6 shows that the scheduling method adopted for delay-insensitive traffic is convenient only when the mean value of the inter-arrival times is bigger than ~100 ms. Anyway it is important to point out that this behaviour strongly depends on the simulation parameters. If the ONU transceiver has a longer wake-up time than the one adopted in the simulations, the power saving advantages in case of delay-insensitive traffic appear also at lower values of the mean inter-arrival time. In fact in this case the advantage of having longer sleep period is enhanced since the portion of time spent to wake up the ONU is bigger. This fact can be shown by comparing the power performances of the two scheduling methods plotted in Figure 4.7, obtained setting the wake-up time to 5 ms.
If the wake-up time is increased, the scheduling method used in case of delay-insensitive traffic results to be convenient for lower values of the mean inter-arrival time, as shown in Figure 4.7. But if the average inter-arrival time is lower than ~60 ms, then the method for delay-sensitive
traffic is more convenient in terms of energy saving. This is due to the fact that below this value in case of delay-insensitive traffic shorter sleep periods are scheduled on average because the method prescribes the scheduling of sleep periods equal to one sixteenth of the prediction after a longer one (Figure 4.3).

A similar phenomenon occurs when the delay-sensitive traffic requires a lower tolerance on the additional queuing delay. In this case the maximum schedulable period in case of delay-sensitive traffic is shorter, worsening the power performances. Thus also in this case the advantages of the more aggressive technique appear at lower values of the mean inter-arrival time. This fact is shown in Figure 4.8, obtained setting the maximum tolerable additional delay for delay-sensitive traffic to 5 ms:

![Figure 4.8 Power consumption as a function of average inter-arrival time with reduced maximum sleep period](image)

Figure 4.9 shows the average additional latency caused by the presence of the sleep mode technique as a function of the average frame inter-arrival time. As already noticed in Figure 4.5, also Figure 4.9 shows that when the average inter-arrival time is below ~6 ms, the sleep mode technique having a prediction based on the arithmetic average (technique 2) does not trigger any sleep period, while technique 1 does. Anyway the average additional latency in this zone can be considered negligible since it is lower than 1 ms. Apart for an initial area (between ~6 ms and ~10 ms average inter-arrival time) where technique 1 shows better latency performances than technique 2, the two techniques show similar latency performances, according to the figure.
Comparing the scheduling methods used for delay-sensitive and delay-insensitive traffic, Figure 4.9 shows that the method for delay-insensitive traffic leads to less latency impairments than the method for delay-sensitive traffic up to ~100 ms of average inter-arrival time. Above this value the situation switches to the opposite. In the first zone it is more likely that the two scheduling methods produce sleep periods of similar length. In case of delay-insensitive traffic the additional latency will be lower because of the scheduling of shorter sleeping periods after the long one (Figure 4.3). This feature is not present in the scheduling method used for delay-sensitive traffic, implying a bigger latency. On the other hand when the average inter-arrival time gets higher, the scheduling method used in case of delay-sensitive traffic has better latency performances thanks to the presence of a maximum schedulable sleep period.

Similar results regarding the comparison between the two scheduling methods used for delay-sensitive and delay-insensitive traffic are also showed in Figure 4.10. Here the worst case additional latency is plotted as a function of the average inter-arrival time. Figure 4.10 shows that in case of delay-insensitive traffic technique 2 results in a lower worst case latency than technique 1. In fact at some points the exponentially smoothed average used as prediction in technique 1 can collapse to an inter-arrival time much bigger than the average value of the distribution because of the jumps of the arithmetic average value. If, after this long inter-arrival time, a much shorter one happens, then a big additional delay for the traffic is generated.
In the second set of simulations the proposed sleep mode technique and its simplified version are applied to an exponential distribution of inter-arrival times characterized by a mean value that changes over time. The evolution of the mean value is plotted in Figure 4.11 as a function of time:
The data describing power and latency performances have been averaged over the periods of time characterized by the same mean inter-arrival time, implying that a different result is produced every time the mean value of the distribution changes.

Figure 4.12 plots the evolution of the ratio of sleep time as the average inter-arrival time changes over time.

![Figure 4.12 Ratio of sleep time as a function of time and average inter-arrival time](image)

The power performances of the two sleep mode techniques characterized by different prediction methods are comparable, as shown in Figure 4.12. The only significant difference can be noticed when the average inter-arrival time passes from 150 ms to 9 ms. In this case technique 1 gives a lower sleep time ratio. This is due to the fact that the prediction technique based on the tuning of the exponentially smoothed average, quickly adapts to the new distribution, scheduling shorter sleep periods almost immediately. On the other hand technique 2 requires more time in order to adapt to the new distribution, resulting in a higher sleep time ratio.

These results are reflected also in Figure 4.13 that plots the evolution of power consumption.
Figure 4.13 Power consumption as a function of time and average inter-arrival time.

Figure 4.14 plots the evolution of the average additional latency over time.

Figure 4.14 Average additional latency as a function of time and average inter-arrival time.
The graph shows that technique 2, whose prediction method is constituted by an arithmetical average, has in general worse average latency performances than technique 1. The main differences are visible when the average inter-arrival time is equal to 270 ms and 9 ms. This fact happens because the prediction adopted in technique 1 can adapt more quickly to changing traffic conditions thanks to the tuning of the exponentially smoothed average.

A different scenario appears in Figure 4.15 that plots the evolution of the worst case additional latency.

![Figure 4.15](image-url)  
**Figure 4.15** Worst case additional latency as a function of time and average inter-arrival time
4.4 HARDWARE SUPPORT

The arithmetics and control operations required by the fast sleep technique are performed at the OLT node. Thus some additional hardware must be provided in order to perform the required actions. The block that I propose to add at the OLT is sketched in Figure 4.16, considering a split ratio of four:

![Figure 4.16 Schematic of hardware support at OLT](image)

**Inputs**

The input signals are CLOCK, ONU_ID and a number of CRITICAL, WAKEUP and SENT signals equal to the split ratio of the PON.
The CLOCK signal is the 8 kHz reference clock located at the OLT node.

The ONU_ID signal is extracted from downstream frames and it indicates which ONU the current frame is addressed to.

The CRITICAL signals indicate whether the services active for a specified ONU are very sensitive to queuing delay (high value) or not (low value). How the CRITICAL signals are actually produced needs to be further investigated.

The WAKEUP signals indicate whether a specified ONU is in the active state (high value) or in the sleep state (low value). The signal is de-asserted when the specified ONU accepts a sleep request and an acknowledgement is received at the OLT and asserted when a wakeup message is received at the OLT from the specified ONU.

The SENT signals indicate whether a sleep request addressed to a specified ONU has been dispatched. It is produced at the OLT buffer and it assumes a high value for a finite period of time (a clock period) every time a sleep request for the specified ONU exits the buffer and is sent to the ONU.

**Outputs**

The output signals are TRIGGER, TSLEEP, ID and a number of DELETE signals equal to the split ratio of the PON.

The TRIGGER signal indicates when a sleep request should be created by assuming a high value for a finite period of time.

The TSLEEP signal indicates the length of the sleep period that should be included in the sleep request triggered by the TRIGGER signal. The number indicated corresponds to the number of clock periods during which the ONU should remain in the sleep state.

The ID signal indicates which ONU is the destination of the sleep request triggered by the TRIGGER signal.
Hardware blocks

The hardware at the OLT is made up of different blocks: a decoder, an arbiter, two multiplexers, and a number of buffers and sleep controllers equal to the split ratio of the PON.

The decoder receives the ONU_ID signal and every time this is asserted it raises the ARRIVAL output associated to the addressed ONU for a finite period of time. In this way every ARRIVAL signal will show a positive impulse every time a downstream frame addressed to the associated ONU arrives at the OLT. Thus these impulses correspond to the arrival events.

The arbiter controls the multiplexers in such a way that their outputs correspond to the signals produced by the sleep controller that is triggering a sleep request. If more than one sleep controller is triggering a sleep request at the same time, then an arbitration policy must be implemented. In this case the requests that are not immediately forwarded must be stored in buffers placed at the inputs of the multiplexers. The arbiter will serve all the sleep requests produced in one cycle before serving any request produced in the next cycle. If no sleep controller is triggering a sleep request, then the output signal TRIGGER will have a low value. The arbiter also takes care of indicating which ONU is associated to the current sleep request by means of the output signal ID.

Every sleep controller implements the actions required by the fast sleep technique. This block measures inter-arrival times exploiting the ARRIVAL signal and performs the required arithmetical and control operations in order to produce the TRIGGER signal that triggers sleep requests and the TSLEEP signal indicating the length of the sleep period, associated to a specified ONU. While performing these actions it takes into account whether delay-sensitive services are active (CRITICAL input signal) and it checks the status of the ONU (WAKEUP input signal). The sleep controller also implements the additional feature regarding the monitoring of the OLT buffer. By checking the value of the SENT input signal, it produces a DELETE signal that informs the OLT buffer when it should delete the stored sleep request addressed to the associated ONU.
The sleep controller block

The structure of the sleep controller block is sketched in Figure 4.17:

![Figure 4.17 Schematic of the sleep controller block](image)

The timer block measures the inter-arrival times between the downstream frames addressed to the specified ONU. The produced measurement $IAT$ is expressed as a number of clock periods, where the clock is the 8 kHz clock present at the OLT.

The average block performs the arithmetical average on the measured inter-arrival times. This block produces the value of the arithmetic average at time $k$ together with its previous value at time $k-1$. The discrete values of the time variable correspond to the arrival events. Also the most recent value of inter-arrival time is produced as an output.
The detector block checks when the arithmetic average changes by more than one time its value. If this condition is verified, the RESTART output is asserted, informing the exponentially smoothed block that the coefficient $c$ of the exponentially smoothed average should be reset to 1. The exponentially smoothed block computes the exponentially smoothed average of the inter-arrival times. In the meantime it keeps on updating the coefficient $c$ according to the instructions received from the RESTART signal. Finally it produces the ESA signal that is the prediction for the next inter-arrival time.

The controller block performs the scheduling mechanism proposed in section 4.2.2. It produces a TRIGGER signal so that this is asserted when a sleep request should be triggered. In the meantime it also produces a TSLEEP signal indicating the desired length of the sleep period in correspondence of every impulse of the TRIGGER signal. In order to perform the described actions, the controller block takes into account the additional queuing delay generated by the last sleep request, by means of the signal CORRECTION. The produced output signals also depend on whether the active services are sensitive to the queuing delay, whether the ONU is in the active state and whether the condition of idle link has been detected.

The corrector block measures the additional queuing delay caused by the sleeping periods on downstream frames. This delay is indicated in the output signal CORRECTION that is updated every time a new downstream frame addressed to the specified ONU arrives.

The delete detector block implements the additional feature associated with the monitoring of the OLT buffer. If a downstream frame addressed to the specified ONU arrives after a sleep request has been triggered but before its dispatch, the DELETE signal is asserted. This instructs the OLT buffer to cancel the stored sleep request.

The idle detector block implements the additional feature detecting idle links. If the inter-arrival time IAT measured by the timer block crosses a certain threshold, then the signal LONG is asserted. This instructs the controller block to trigger a sleep request characterized by a fixed long sleep period.

The detailed schematics of all the blocks that constitute the sleep controller block are reported in Appendix II.
Hardware support at the ONUs

In order to support the proposed sleep mode technique, every ONU must be provided with a timer in order to time the sleep periods instructed by the OLT. The timer is charged with the value included in the sleep request. Then it starts a countdown that is clocked by a local oscillator tuned on the same frequency of the OLT clock (8 kHz). When the countdown reaches zero, the ONU reactivates all its equipment and then sends a WAKEUP message to the OLT.
5 Implementation of the proposed technique

The experimental setup supporting the implementation of the proposed sleep technique is sketched in Figure 5.1:

Traffic is generated by the Ixia XM12 High Performance Chassis, using a port operating at 1 Gbps. The IxExplorer software provides a Graphic User Interface for managing the Ixia traffic generator. In this way it is possible to set some parameters characterizing the traffic, such as rate, frame size and MAC addresses. The generated traffic is transported over an Ethernet TP cable to an SFP (small form-factor pluggable) board. Here a physical media dependent (PMD) device converts the received traffic into a differential signal compliant with the 1.25 Gbps serial interface on the Altera board. The signals are transported to the Altera board (Transceiver Signal Integrity Development Kit, Stratix IV GT Edition) on coaxial cables. At this point the traffic enters the hardware loaded into the FPGA (field-programmable gate array) mounted on
the board, where the sleep mode function is actually emulated. The programming of the FPGA is accomplished by means of the Altera design tool Quartus II. Afterwards the traffic is sent all the way back to the port in the Ixia Chassis with the reverse operation.

Latency measurements and traffic statistics are collected by means via the IxExplorer interface. Data describing the percentage of time spent in the sleep state are collected by sampling the content of some hardware registers in the FPGA. This had been done using Signal Tap Logic Analyzer, a tool included in Quartus II.

The experimental implementation does not include all the functionalities of the proposed sleep mode function. First of all, the presence of upstream traffic is discarded. Thus the interruption mechanism of the sleep period performed by the ONUs is not emulated. Furthermore, the messaging operation between the OLT and the ONUs is not implemented. The sleep requests are triggered and the sleep periods are computed only on the basis of downstream traffic and are always immediately executed. Therefore the possibility of erasing the sleep requests while they are waiting to be sent to the ONUs is neglected in this experimental implementation. Finally the subtraction of the wake-up time from every scheduled sleep period is not executed, since no real ONU is present in the hardware and the wake-up time is zero.

5.1 MULTIPLEXING AND DEMULTIPLEXING LOOP

The implementation of the sleep mode function requires the presence of a support hardware in the background. This is made up of a loop having multiplexing and demultiplexing capabilities.

At first, the traffic is multiplexed into 16 fifos, emulating the split ratio of a PON, and then demultiplexed back into a single flow. The execution of the sleep requests is emulated by blocking the emptying of the fifos.
This support loop is sketched in Figure 5.2.

![Figure 5.2 Support loop structure](image)

The communication with the serial interface is managed by the Triple-Speed Ethernet (TSE) block, that is a configurable Altera’s IP (intellectual property) compliant with the IEEE 802.3 standard [16]. This combines the features of a physical medium attachment (PMA), a 100BASE-X/SGMII physical coding sublayer (PCS) and a 10/100/1000 Mbps Ethernet media access controller (MAC). Also another Altera’s IP is instantiated in the design. It is the ALTGX RECONFIG block that performs the offset cancellation on the receiver channel of the TSE block. The finite state machine takes care of initializing the content of the control registers of the TSE block. The remaining hardware behaves like a MAC client communicating with the TSE IP. This communication is based on an interface supporting the transport of packets, specified by the following five signals clocked on a 125 MHz clock:

- data (8 bits)
- data valid (1 bit)
- start of packet (1 bit)
- end of packet (1 bit)
- error (5 bits for the RX channel, 1 bit for the TX channel).
The decoder block extracts the MAC destination address from each packet. This information is interpreted as the indication of the addressed ONU. As a split ratio equal to 16 is implemented, only 16 MAC addresses are considered to be valid (from 0 to 15). The packets having a MAC destination address higher than 15 are discarded and lost. The extracted address is used for two purposes. Firstly, it is used as a selection signal by the muxing block in order to multiplex the packets into the correct fifo. Secondly, this information is used to produce the control signals indicating the arrival of a packet addressed to a particular ONU. In fact the 16 sleep controllers (one for each emulated ONU) require arrival signals in order to be able to measure the inter-arrival times between frames addressed to a specified ONU.

The muxing block takes care of multiplexing the traffic into the correct fifo, based on the address information received from the decoder block. Furthermore, it drives the write signals that control the writing operations of the fifos.

The fifos are implemented using an Altera’s IP. They can contain 16384 bytes of data and they have the same clock domain for both writing and reading procedures.

The demuxing block takes care of re-building a single stream of packets. It manages the emptying of the fifos, that is performed using a fair loop-like arbitration mechanism. Therefore it drives the read signals that control the reading operations of the fifos. The reading procedure can be blocked by the control signals coming from the sleep controller blocks that indicate the execution of the sleep requests. Furthermore the demuxing block takes care of informing the sleep controllers block when the emulated ONUs are in the active state and thus new sleep requests can be issued.

Finally, the sleep controller blocks are the ones in charge of the actual implementation of the proposed sleep mode technique. On the basis of the measurements of the frame inter-arrival times, 16 control signals are produced able to block the emptying procedure of the fifos for the computed sleep period. In this way, the behaviour of the proposed sleep mode technique is emulated.
5.2 IMPLEMENTATION OF THE SLEEP CONTROLLERS

The number of instantiated sleep controllers is equal to the number of ONUs existing in the emulated PON. Each sleep controller drives a control signal that remains asserted during the entire sleep period each time a sleep request is triggered. This signal blocks the emptying of the fifo containing the frames addressed to the emulated ONU. In this way, by means of latency measurements, it is possible to test the impairments on quality of service due to the proposed sleep mode function.

Each sleep controller is built in two blocks, as sketched in Figure 5.3.

![Figure 5.3 Sleep controller structure](image)

The sleep block measures frame inter-arrival times exploiting the arrival signal. This information is then elaborated in order to drive the triggering of the sleep request and compute the length of the sleep period. These operations are performed in different ways depending on the value of the critical signal, indicating the presence of delay-sensitive traffic. Sleep requests can be triggered only when the emulated ONU is in the active state (awake). This information is carried by the wakeup signal.

The fifo controller drives the sleep signal that blocks the emptying of the fifo when asserted.

For the same reasons explained in section 4.3, also a design characterized by a simplified prediction method has been implemented for experimental purposes.
The first type of sleep block implements the method described in chapter 4. Its structure is sketched in Figure 5.4:

![Figure 5.4 First type of sleep block](image)

The functionalities of the blocks are the ones described in section 4.4. Anyway the experimental implementation shows some differences compared to the actual implementation proposed for a PON. First of all the delete detector block is missing. This is due to the fact that the sleep requests are immediately executed and thus the erasing of waiting requests is neglected.

The clock frequency employed in the experimental implementation is not equal to 8 kHz, that is the frequency provided at the OLT. The clock runs at 125 MHz, instead. Therefore the signals carrying a time measure information are extended to 30 bits, instead of 16 bits. The critical input signal should be high when delay-sensitive traffic is present and low otherwise. As a detection mechanism able to distinguish the different types of active services has not been implemented, this input is provided by means of a switch on the board that can be set manually.
The second type of sleep block differs from the first one for the method employed to predict the next frame inter-arrival time. In the second implementation the result of the arithmetical average is directly used as the value of the prediction. The structure of this sleep block is sketched in Figure 5.5:

In order to actually implement in hardware the sleep blocks, all the control parameters presented as parametric in the proposed sleep mode technique had to assume some specific values. Therefore I chose to set the same values used for the simulations reported in section 4.3.
6 Analysis of measurements

The purpose of the measurements is to test both the impact on quality of service and the power saving capabilities due to the presence of the proposed sleep mode technique. Furthermore the design described in chapter 4 is compared with a different version of it, as explained in section 5.2. Therefore the two implementations have been loaded into the FPGA one at a time and their performances compared in order to discuss the actual advantages and disadvantages of the proposed sleep mode technique.

Traffic latency and the percentage of time that ONUs spend in the sleep state have been measured. The IxExplorer interface allows to collect latency measurements on the frames going through the hardware implemented into the FPGA. The data associated to energy saving, instead, can be extracted by sampling the values of some counters loaded into the FPGA. These counters take care of measuring both the time spent in the sleep state by the ONUs and the number of triggered sleep requests. The sampling is performed by means of Signal Tap II Logic Analyzer, a tool included in the Altera’s Quartus II design tool.

Different types of traffic have been generated in order to obtain measurements relative to a number of different traffic conditions and provide a wider analysis. Measurements are divided into five sets depending on the general shape of the traffic. Each set is then divided into three configurations characterized by different values assigned to the parameters defining the traffic. For each configuration the performances of the two designs described in section 5.2 are tested, for both delay-sensitive and delay-insensitive traffic conditions.

The data reported in the following sections are obtained making the same assumptions used for the simulations, thus the wake-up time is set to 1 ms and the power consumed by one ONU is 10 W in the active state and 2 W in the sleep state.
6.1 CONSTANT RATE TRAFFIC

The first set of experiments is characterized by the presence of constant rate traffic between the OLT and the ONUs. The graphs reported below show the percentages of saved power, the average additional latency and the worst case latency for each sleep mode technique, both in case of delay-sensitive and delay-insensitive traffic. The results are reported as a function of the inter-arrival time characterizing the traffic rate.

Figure 6.1 shows the percentages of saved power.

![Graph showing percentages of saved power with constant rate traffic](image)

First of all Figure 6.1 shows that when the traffic rate is high the conditions controlling the triggering of the sleep requests are never verified and the ONUs are always active. Thus, in that case, no energy saving is possible. In case of constant rate traffic, Figure 6.1 shows that the scheduling method employed for delay-insensitive traffic can lead to higher energy saving when the traffic rate is quite low thanks to the scheduling of longer sleep periods.
Figure 6.2 shows the average additional latency while Figure 6.3 shows the worst case additional latency.
Regarding latency impairments, Figures 6.2 and 6.3 show that traffic delay can be smaller when the delay-insensitive traffic method is used if the frame inter-arrival time is equal to or lower than twice the maximum schedulable sleep period for the delay-sensitive traffic method. This happens because, while with delay-sensitive traffic the same sleep period is scheduled all the time, in case of delay-insensitive traffic shorter sleep periods are scheduled after the longer one (Figure 4.3). Instead, if the traffic has a lower rate, then latency is lower when the method for delay-sensitive traffic is employed, thanks to the presence of a maximum schedulable sleep period.

Figures 6.1, 6.2, 6.3 also show that when the traffic has a constant rate technique 1 and technique 2 produce more or less the same results, both in terms of energy-saving and latency impairments.

### 6.2 BURSTY TRAFFIC

In this set of measurements the generated traffic is made of bursts separated by some idle periods. Each configuration is characterized by a different number of frames included in each burst. The frame inter-arrival time within a burst and the idle period, instead, are kept constant to 1 ms and 1 s respectively.

Figure 6.4 shows the percentages of power saved by each ONU. The x-labels indicate how many frames are contained in each burst. In case of bursty traffic Figure 6.4 shows that energy-saving is much more aggressive when the method for delay-insensitive traffic is employed. The graph also shows that, in case of delay-insensitive traffic, technique 1 leads to much higher energy-saving than technique 2.
Figures 6.4 and 6.5 show, respectively, the additional average latency and the worst case additional latency.

Figure 6.4 Percentage of saved power with bursty traffic

Figure 6.5 Average additional latency with bursty traffic
When the traffic is bursty, Figures 5.5 and 5.6 show that technique 2 gives much better latency performances than technique 1. This happens because the proposed technique makes the prediction for the next inter-arrival time collapse to the last acquired value when the idle period is sampled. This fact causes severe impairments on latency because a long sleep period is scheduled just at the beginning of the burst. With technique 2, instead, this does not happen because the idle period is mediated with other seven samples, thus avoiding the scheduling of such a long sleep period.

### 6.3 DOUBLE-RATE TRAFFIC

In this set of experiments the traffic is characterized by two rates, each active for a certain number of frames. The two rates keep on alternating themselves all the time. In the following graphs the x-labels indicate the number of frames and the rates characterizing the two phases of the traffic shape.
Figure 6.7 plots the ratio of power that can be saved.

![Figure 6.7 Percentage of saved power with double-rate traffic](image)

In case of traffic characterized by two alternating rates, Figure 6.7 shows that if the difference between the rates is high, then the method for delay-insensitive traffic has better energy-saving capabilities than the delay-sensitive traffic method. The situation switches to the opposite when the rate difference gets lower. The figure also shows that technique 1 gives higher energy-saving than technique 2, especially when the difference between the rates is not too small.

Figures 6.8 and 6.9 show respectively the average additional latency and the worst case additional latency. Figure 6.8 shows that if the average frame inter-arrival time is higher than twice the maximum sleep period allowed in case of delay-sensitive traffic, then the method used for delay-sensitive traffic shows better latency performances. Otherwise it is the delay-insensitive traffic method the one that guarantees the lower latency. The reasons for this are the same as the ones explained for the constant rate traffic case, in section 6.1. Furthermore, Figure 6.8 and 6.9 show that technique 1 causes shorter latencies when the rate difference is high, while the two technique result in similar impairments otherwise. In fact the proposed method can adapt the length of the scheduled sleep periods quicker when big and sudden changes in the inter-arrival times distribution are detected. Thus it is able to save more energy when passing
from a high rate to a lower one (Figure 6.7), while it can limit latency impairments in the opposite situation (Figures 6.8 and 6.9).

**Figure 6.8** Average additional latency with double-rate traffic

![Average additional latency graph](image)

**Figure 6.9** Worst case additional latency with double-rate traffic

![Worst case additional latency graph](image)
6.4 BURSTY TRAFFIC WITH SUPERIMPOSED BACKGROUND TRAFFIC

In this set of experiments a bursty traffic is superimposed to a constant rate traffic. The x-labels in the following graphs indicate the number of frames and the inter-arrival time within each burst as well as the inter-arrival time characterizing the background constant rate traffic. The idle periods between the bursts are fixed to 2 s.

Figure 6.10 shows the percentages of power that can be saved.

![Graph showing percentages of saved power with bursty traffic superimposed to constant rate traffic.](image)

The graph shows that technique 1 and technique 2 have similar power-saving capabilities. A different result is obtained with the third traffic configuration, where technique 1 manages to save more energy than technique 2 in case of delay-insensitive traffic.

Furthermore Figure 6.10 shows that when a bursty traffic is superimposed to a constant rate one, the scheduling method employed for delay-sensitive traffic usually leads to better power performances than the delay-insensitive traffic method. This is probably due to the same reasons explained in section 6.1 for the constant rate traffic case.
Figures 6.11 and 6.12 show the average additional latency and the worst case additional latency, respectively.

Figure 6.11 Average additional latency with bursty traffic superimposed to a constant rate traffic

Figure 6.12 Worst case additional latency with bursty traffic superimposed to a constant rate traffic
Figures 6.11 and 6.12 show that, in case of bursty traffic superimposed to a background constant rate traffic, the delay-insensitive traffic method leads to better latency performances than the method employed for delay-sensitive traffic.

When bursty traffic is superimposed to constant rate traffic, latency performances are generally better for technique 1 than for technique 2, as shown in Figures 6.11 and 6.12. In fact, when the traffic is not that regular, like in this test case, the sleep mode technique based on the proposed prediction method has the capability to follow the behaviour of inter-arrival times more rapidly than the simplified technique.

### 6.5 TRAFFIC WITH SUPERIMPOSED RATES

In this set of experiments the generated traffic tries to emulate the presence of almost random inter-arrival times. Thus multiple constant rates are superimposed. The x-labels of the following graphs indicate the inter-arrival times characterizing each superimposed rate for each different configuration.

Figure 6.13 shows the percentages of saved power. The graph shows that the scheduling method used in case of delay-sensitive traffic can save more energy than the method used for delay-insensitive traffic. Furthermore technique 1 leads to better power performances than technique 2.
Figures 6.13 and 6.15 show the average additional latency and the worst case additional latency, respectively.
Figure 6.15 Worst case additional latency with superimposed rates traffic

The figures show that the delay-insensitive traffic method produces shorter average latencies but longer worst-case latencies than the method used in case of delay-sensitive traffic.

Figures 6.14 and 6.15 show that, when the distribution of inter-arrival times is almost random, technique 1 has better latency performances than technique 2, especially when the average traffic rate is higher. If the rate gets lower, then only the worst-case latency results to be better for technique 2, while the average latency still remains shorter for the proposed technique.
7 Conclusions and future work

Power consumption of telecommunication networks is becoming a more and more urgent issue that needs to be faced. During the last years both the bandwidth demand per user and the number of connected customers have significantly increased, implying a need for more energy-hungry networks able to support the new requirements. The situation is critical especially for access networks, the so called last mile connecting the provider central office to final users. The target for this thesis is constituted by TDM-PONs, a kind of access network that multiplexes the transported signals within the time domain. They employ optical fiber as transport medium and use passive devices in order to distribute the signals through the network, to and from the users.

Researchers around the world and standardization bodies have investigated different solutions providing energy saving capabilities in TDM-PONs. Many of these are based on the so called sleep mode techniques. The ONUs (optical network units, components placed at the customer premises) are turned off (partially or completely) during certain periods of time according to the mechanism specified by the particular technique. In this way it is possible to save considerable amounts of energy, being the ONUs the main cause for power consumption in TDM-PONs.

This thesis investigated and proposed a new sleep mode technique for TDM-PONs. The objective is to achieve improved power performances without impacting too much the quality of service perceived by the users. According to the proposed technique an ONU is turned off when certain conditions on the traffic addressed to it are verified. The whole operation is managed at the OLT (optical line terminal, placed at the central office), where the hardware necessary to implement the proposed mechanism is placed. The length of the sleep time periods is computed here and a statistical method monitoring the inter-arrival times between downstream frames is employed for this purpose. The proposed mechanism is made up by two different sub-methods in order to be able to preserve quality of service when delay-critical services are active but save more energy when the traffic is not so sensitive to delay.

The results of the Matlab simulations show that if the frame arrivals follow a Poisson distribution, the proposed technique (technique 1) generally leads to higher energy saving than the technique based on a simplified prediction method (technique 2). This becomes more and more evident as the traffic rate gets lower. On the other hand technique 2 usually produces less
additional latency on the traffic than technique 1. The situation changes when the frame inter-arrival times follow an exponential distribution with a mean value that changes over time. In this case the proposed technique shows better latency performances than technique 2 in the majority of the tests. This happens thanks to the prediction method adopted in technique 1 that adapts more quickly to changing traffic conditions.

The analysis of the experimental results shows that the proposed sleep mode technique can lead to good energy-saving features while still limiting the QoS impairments due to additional traffic latency. Unfortunately this is not true for all traffic cases: sometimes the proposed sleep mode technique should be avoided unless it is improved.

When the traffic shows a quite regular pattern (tests in sections 6.1 and 6.3) the proposed sleep mode technique (technique 1) has in general better or equal performances than its simplified version (technique 2) both in terms of energy-efficiency and traffic latency. It can save almost 68% of the energy with an average inter-arrival time between frames of 20 ms, providing a worst-case latency of about 2 ms (the assumptions reported in section 6.1 hold).

The proposed sleep mode technique performs well also in case of more irregular traffic patterns (tests in sections 5.4 and 5.5). It can reach energy-saving percentages around 70% without producing too big additional latencies for the frames, especially in the average case.

A totally different scenario appears when the traffic has a bursty nature. In this case the proposed sleep mode technique should be avoided because the impairments on QoS are almost unacceptable (almost 0,5 s additional delay with an inter-bursts gap of 1 s). Another solution could be to improve the proposed technique, as explained in the following section.

Apart for some particular cases where the average frame inter-arrival time is comparable to the maximum sleep period allowed when delay-sensitive services are active, the scheduling method employed in case of delay-sensitive traffic provides better protection of QoS requirements. Furthermore this generally happens without degrading too much the energy-saving performances. Therefore, if power consumption is not that critical, the proposed sleep mode technique could be implemented including only the scheduling method thought for delay-sensitive traffic. In this way a small slice of energy-saving would be lost, but a quite significant improvement in latency performances would be gained. On the other hand, if it is necessary to save as much power as possible, the proposed sleep mode technique should include both scheduling methods, for delay-sensitive and delay-insensitive services.
Both the theoretical investigation on sleep mode possibilities and the mechanisms managing the proposed sleep mode technique, can be further developed and improved.

From the point of view of theoretical study, the development of a sleep mode technique for TDM-PONs needs an investigation of the traffic shape produced by the activity of triple-play services. A more detailed knowledge of the realistic behaviour of the traffic would provide a significant help during the design of an energy saving method via sleep mode. Such a background could indicate the possible design choices. In fact knowing the shape of the traffic, some solutions could result clearly unsuitable and be excluded. At the same time some hints could be provided by pointing out the characteristics of the traffic that could be exploited to the advantage of energy saving.

Concerning the sleep mode technique proposed in this thesis, there are still several points that should be developed and clarified.

First of all a mechanism to detect the presence of delay-sensitive services should be provided. In fact, in order to switch between the two described control methods, it is necessary to identify when delay-sensitive applications are active. One possibility is to consider critical those services that require a high level of interactivity, like video telephony (VoIP) or online games. In order to measure interactivity it would be useful to monitor upstream traffic. This could be achieved by checking the content of the messages sent by ONUs indicating the status of its queuing buffer for DBA purposes. These messages are encoded according to different schemes depending on which standard is implemented. By exploiting PON transmission scheduling protocols, the upstream activity and thus the level of interactivity of the current service could be sensed. Another way to monitor upstream traffic could rely only on frame count. The number of received upstream frames over a period of time could be measured at the OLT. Then this number could be compared to the number of downstream frames. If they are comparable, then a high interactive service is active.

Secondly, a dozing of the upstream channel could be combined with the proposed sleep mode technique. The described technique affects both the downstream and the upstream channels at the same time. The dozing technique, instead, would only affect the upstream channel. The purpose of this combination would be to exploit the energy saving opportunities deriving from the asymmetry of upstream and downstream traffic. In fact this condition is verified for the majority of situations. The dozing technique could be similar to the one discussed in [2]. The ONU dozing could be triggered internally to the ONU when the queuing buffer is empty and
no upstream traffic needs to be sent. While the ONU is in the doze state, its transmitter could be powered off but its receiver could remain active. Thus the ONU would keep on receiving upstream grants but the OLT should not expect any guaranteed answer from the ONU. The ONU could stop the doze state and return to the active state when it detects the presence of some upstream traffic or if it receives a request from the OLT that needs a response. The time required to recover would be shorter than the wake-up time required in the fast sleep technique. In fact neither clock synchronization nor line framing synchronization need to be recovered thanks to the fact that the ONU receiver would remain on during dozing and would not loose this information.

Finally some additional features should be added to the proposed sleep mode technique in order to prevent the big worst-case latency produced when the traffic is bursty. In fact the presence of a long idle period between bursts is detected as a change in the mean value of the distribution of frame inter-arrival times. Thus the sample history involved in the computation of the exponentially smoothed average is erased and the prediction of the next gap between frames collapses to the last sample, that is the idle period between bursts. In this way a long sleep period is scheduled just at the beginning of the burst, causing a huge penalty in terms of latency. For example, this could be avoided changing the condition that causes the reset of the exponentially smoothed average computation. When a longer inter-arrival time is sampled, the condition of jumping mean value (formula 4.4 in section 4.2.1) should be verified for two consecutive times and not just once.

Regarding the experimental implementation, it would be desirable to manage to design an hardware structure able to emulate all the features of the proposed sleep mode technique. In particular it would be very interesting to include the presence of upstream traffic, together with the interruption mechanism performed by the ONUs. Also the emulation of the messaging between OLT and ONUs should be added together with the associated timing penalties. In this way, also the feature described in section 4.2.3 that allows the erasing of sleep requests waiting in the OLT buffers could be tested. A complete implementation would provide more realistic measurements together with deeper and more reliable conclusions about the proposed technique.
References


[7] PMC-Sierra Ltd., Methods and devices for reducing power consumption in a passive optical network while maintaining service continuity, US2009/0263127 A1


[9] B. Skubic, D. Hood, Evaluation of ONU power saving modes for XG-PON, IEEE Network Mag., Accepted for publication in March/April issue 2011


[18] F. Neri, Passive Optical Networks, a.a. 2009/10 [material from the course Optical networks, Politecnico di Torino]
Appendix I

Matlab codes

In this section the Matlab codes used for the simulation of the proposed sleep technique are reported.

A1.1 Matlab code used for the first set of simulations

```matlab
rand('seed',9);
randn('seed',9);
sleep_ratio_1_sen = zeros(37,10);
sleep_ratio_1_ins = zeros(37,10);
sleep_ratio_2_sen = zeros(37,10);
sleep_ratio_2_ins = zeros(37,10);
avg_power_1_sen = zeros(37,10);
avg_power_1_ins = zeros(37,10);
avg_power_2_sen = zeros(37,10);
avg_power_2_ins = zeros(37,10);
avg_latency_1_sen = zeros(37,10);
avg_latency_1_ins = zeros(37,10);
avg_latency_2_sen = zeros(37,10);
avg_latency_2_ins = zeros(37,10);
worst_latency_1_sen = zeros(37,10);
worst_latency_1_ins = zeros(37,10);
worst_latency_2_sen = zeros(37,10);
worst_latency_2_ins = zeros(37,10);

avg_sleep_ratio_1_sen = zeros(37,1);
avg_sleep_ratio_1_ins = zeros(37,1);
avg_sleep_ratio_2_sen = zeros(37,1);
avg_sleep_ratio_2_ins = zeros(37,1);
avg_avg_power_1_sen = zeros(37,1);
avg_avg_power_1_ins = zeros(37,1);
avg_avg_power_2_sen = zeros(37,1);
avg_avg_power_2_ins = zeros(37,1);
avg_avg_latency_1_sen = zeros(37,1);
avg_avg_latency_1_ins = zeros(37,1);
avg_avg_latency_2_sen = zeros(37,1);
avg_avg_latency_2_ins = zeros(37,1);

%PARAMETERS IN ms OR W

tolerance = 10;
```
wakeup = 1;
pactive = 10;
psleep = 2;

for index=1:1:10

%GENERATION OF THE DISTRIBUTIONS
distribution = zeros(100,37);
distribution(:,1) = exprnd(0.1,100,1);
distribution(:,2) = exprnd(0.2,100,1);
distribution(:,3) = exprnd(0.3,100,1);
distribution(:,4) = exprnd(0.4,100,1);
distribution(:,5) = exprnd(0.5,100,1);
distribution(:,6) = exprnd(0.6,100,1);
distribution(:,7) = exprnd(0.7,100,1);
distribution(:,8) = exprnd(0.8,100,1);
distribution(:,9) = exprnd(0.9,100,1);
distribution(:,10) = exprnd(1,100,1);
distribution(:,11) = exprnd(2,100,1);
distribution(:,12) = exprnd(3,100,1);
distribution(:,13) = exprnd(4,100,1);
distribution(:,14) = exprnd(5,100,1);
distribution(:,15) = exprnd(6,100,1);
distribution(:,16) = exprnd(7,100,1);
distribution(:,17) = exprnd(8,100,1);
distribution(:,18) = exprnd(9,100,1);
distribution(:,19) = exprnd(10,100,1);
distribution(:,20) = exprnd(20,100,1);
distribution(:,21) = exprnd(30,100,1);
distribution(:,22) = exprnd(40,100,1);
distribution(:,23) = exprnd(50,100,1);
distribution(:,24) = exprnd(60,100,1);
distribution(:,25) = exprnd(70,100,1);
distribution(:,26) = exprnd(80,100,1);
distribution(:,27) = exprnd(90,100,1);
distribution(:,28) = exprnd(100,100,1);
distribution(:,29) = exprnd(200,100,1);
distribution(:,30) = exprnd(300,100,1);
distribution(:,31) = exprnd(400,100,1);
distribution(:,32) = exprnd(500,100,1);
distribution(:,33) = exprnd(600,100,1);
distribution(:,34) = exprnd(700,100,1);
distribution(:,35) = exprnd(800,100,1);
distribution(:,36) = exprnd(900,100,1);
distribution(:,37) = exprnd(1000,100,1);

%PROPOSED METHOD

%Computation of the arithmetic average on eight samples
average = zeros(100,37);
for j=1:1:37
  average(1,j) = distribution(1,j)/8;
  average(2,j) = (distribution(1,j)+distribution(2,j))/8;
  average(3,j) = (distribution(1,j)+distribution(2,j)+distribution(3,j))/8;
  average(4,j) = (distribution(1,j)+distribution(2,j)+distribution(3,j)+distribution(4,j))/8;
  average(5,j) = (distribution(1,j)+distribution(2,j)+distribution(3,j)+distribution(4,j)+distribution(5,j))/8;
  average(6,j) = (distribution(1,j)+distribution(2,j)+distribution(3,j)+distribution(4,j)+distribution(5,j)+distribution(6,j))/8;
average(7, j) =
(distribution(1, j)+distribution(2, j)+distribution(3, j)+distribution(4, j)+distribution
(5, j)+distribution(6, j)+distribution(7, j))/8;
for i=8:1:100
    average(i, j) = (distribution(i, j)+distribution(i-1, j)+distribution(i-
-2, j)+distribution(i-3, j)+distribution(i-4, j)+distribution(i-5, j)+distribution(i-
-6, j)+distribution(i-7, j))/8;
end
end

%Computation of the adaptive exponentially smoothed average
prediction = zeros(100,37);
reset = zeros(1,37);
for j=1:1:37
    for i = 1:1:100
        if i>= 2
            if abs(average(i,j)-average(i-1,j))>average(i,j) || abs(average(i,j)-
average(i-1,j))>average(i-1,j)
                prediction(i,j) = distribution(i,j);
                reset(j) = i;
            else
                prediction(i,j) = 2*distribution(i, j)/(i-reset(j)+2) + (1-2/(i-
reset(j)+2))*prediction(i-1,j);
            end
        else
            prediction(i,j) = distribution(i,j);
        end
    end
end

%Delay-sensitive traffic case
ratio = zeros(37,1);
power = zeros(37,1);
delay = zeros(37,1);
delay_max = zeros (37,1);
for j=1:1:37
    time_sleep = 0;
time = 0;
tsleep_1 = 0;
tsleep_2 = 0;
n_sleep = 0;
trigger_time = 0;
correction = 0;
for i=2:1:100
    time = time + distribution(i-1,j);
    if (trigger_time+tsleep_1+n_sleep*tsleep_2)<=time
        trigger_time = time;
correction = 0;
    else
        correction = trigger_time + tsleep_1 + n_sleep*tsleep_2 - time;
        trigger_time = trigger_time + tsleep_1 + n_sleep*tsleep_2;
delay(j) = delay(j) + correction;
    if correction > delay_max(j)
        delay_max(j) = correction;
    end
end
if prediction(i-1,j)<12*wakeup
    tsleep_1 = 0;
tsleep_2 = 0;
n_sleep = 0;
end
if prediction(i-1,j)>=12*wakeup && prediction(i-1,j)<=2*tolerance
    tsleep_1 = prediction(i-1,j)/2 - correction;
tsleep_2 = prediction(i-1,j)/2;
\[ n_{sleep} = \text{ceil}((\text{distribution}(i,j) - tsleep_2)/tsleep_2); \]
\[ \text{time}_{sleep} = \text{time}_{sleep} + tsleep_1 - \text{wakeup} + n_{sleep}*(tsleep_2 - \text{wakeup}); \]
\]
\[
\text{end} \]
\[
\text{if} \ \text{prediction}(i-1,j) > 2*\text{tolerance} \\
\hspace{1cm} tsleep_1 = \text{tolerance} - \text{correction}; \\
\hspace{1cm} tsleep_2 = \text{tolerance}; \\
\hspace{1cm} n_{sleep} = \text{ceil}((\text{distribution}(i,j) - tsleep_2)/tsleep_2); \\
\hspace{1cm} \text{time}_{sleep} = \text{time}_{sleep} + tsleep_1 - \text{wakeup} + n_{sleep}*(tsleep_2 - \text{wakeup}); \\
\]
\[
\text{end} \]
\[
\text{ratio}(j) = \text{time}_{sleep}/(\text{time} + \text{distribution}(100,j)); \\
\text{power}(j) = \text{pactive} - \text{ratio}(j)\*(\text{pactive} - \text{psleep}); \\
\text{delay}(j) = \text{delay}(j)/100; \\
\]
\[
\text{end} \]
\[
\text{sleep}_{ratio}\_1\_sen(:, \text{index}) = \text{ratio}; \\
\text{avg}_{power}\_1\_sen(:, \text{index}) = \text{power}; \\
\text{avg}_{latency}\_1\_sen(:, \text{index}) = \text{delay}; \\
\text{worst}_{latency}\_1\_sen(:, \text{index}) = \text{delay}_{max}; \\
\]
\[
\%\text{Delay-insensitive traffic case} \\
\text{ratio} = \text{zeros}(37,1); \\
\text{power} = \text{zeros}(37,1); \\
\text{delay} = \text{zeros}(37,1); \\
\text{delay}_{max} = \text{zeros}(37,1); \\
\text{for} \ j=1:1:37 \\
\hspace{1cm} \text{time}_{sleep} = 0; \\
\hspace{1cm} \text{time} = 0; \\
\hspace{1cm} tsleep_1 = 0; \\
\hspace{1cm} tsleep_2 = 0; \\
\hspace{1cm} n_{sleep} = 0; \\
\hspace{1cm} \text{trigger}_{time} = 0; \\
\hspace{1cm} \text{correction} = 0; \\
\hspace{1cm} \text{for} \ i=2:1:100 \\
\hspace{2cm} \text{time} = \text{time} + \text{distribution}(i-1,j); \\
\hspace{2cm} \text{if} \ (\text{trigger}_{time}+tsleep_1+n_{sleep}*tsleep_2) <= \text{time} \\
\hspace{3cm} \text{trigger}_{time} = \text{time}; \\
\hspace{3cm} \text{correction} = 0; \\
\hspace{2cm} \text{else} \\
\hspace{3cm} \text{correction} = \text{trigger}_{time} + tsleep_1 + n_{sleep}*tsleep_2 - \text{time}; \\
\hspace{3cm} \text{trigger}_{time} = \text{trigger}_{time} + tsleep_1 + n_{sleep}*tsleep_2; \\
\hspace{3cm} \text{delay}(j) = \text{delay}(j) + \text{correction}; \\
\hspace{3cm} \text{if} \ \text{correction} > \text{delay}_{max}(j) \\
\hspace{4cm} \text{delay}_{max}(j) = \text{correction}; \\
\hspace{2cm} \text{end} \\
\hspace{2cm} \text{end} \\
\hspace{1cm} \text{if} \ \text{prediction}(i-1,j) < 12*\text{wakeup} \\
\hspace{2cm} tsleep_1 = 0; \\
\hspace{2cm} tsleep_2 = 0; \\
\hspace{2cm} n_{sleep} = 0; \\
\hspace{1cm} \text{end} \\
\hspace{1cm} \text{if} \ \text{prediction}(i-1,j) >= 12*\text{wakeup} \\
\hspace{2cm} tsleep_1 = \text{prediction}(i-1,j)/2 - \text{correction}; \\
\hspace{2cm} tsleep_2 = \text{prediction}(i-1,j)/16; \\
\hspace{2cm} \text{if} \ \text{tsleep}_2 < 6*\text{wakeup} \\
\hspace{3cm} \text{tsleep}_2 = 6*\text{wakeup}; \\
\hspace{2cm} \text{end} \\
\hspace{2cm} n_{sleep} = \text{ceil}((\text{distribution}(i,j) - tsleep_1 - \text{correction})/tsleep_2); \\
\hspace{2cm} \text{time}_{sleep} = \text{time}_{sleep} + tsleep_1 - \text{wakeup} + n_{sleep}*(tsleep_2 - \text{wakeup}); \\
\hspace{1cm} \text{end} \\
\hspace{1cm} \text{end} \\
\hspace{1cm} \text{end} \\
\hspace{1cm} \text{end} \\
\text{end} \]
\[ \text{ratio}(j) = \frac{\text{time} \times \text{sleep}}{\text{time} + \text{distribution}(100,j)}; \]
\[ \text{power}(j) = \text{pactive} - \text{ratio}(j) \times (\text{pactive} - \text{psleep}); \]
\[ \text{delay}(j) = \text{delay}(j)/100; \]

\text{sleep\_ratio\_1\_ins(:,index) = ratio;}
\text{avg\_power\_1\_ins(:,index) = power;}
\text{avg\_latency\_1\_ins(:,index) = delay;}
\text{worst\_latency\_1\_ins(:,index) = delay\_max;}

\%SIMPLIFIED METHOD

\%Computation of the arithmetic average on eight samples
\text{average = zeros(100,37);}
\text{for } j=1:1:37
  \text{average}(1,j) = \text{distribution}(1,j)/8;
  \text{average}(2,j) = (\text{distribution}(1,j)+\text{distribution}(2,j))/8;
  \text{average}(3,j) = (\text{distribution}(1,j)+\text{distribution}(2,j)+\text{distribution}(3,j))/8;
  \text{average}(4,j) = (\text{distribution}(1,j)+\text{distribution}(2,j)+\text{distribution}(3,j)+\text{distribution}(4,j))/8;
  \text{average}(5,j) = (\text{distribution}(1,j)+\text{distribution}(2,j)+\text{distribution}(3,j)+\text{distribution}(4,j)+\text{distribution}(5,j))/8;
  \text{average}(6,j) = (\text{distribution}(1,j)+\text{distribution}(2,j)+\text{distribution}(3,j)+\text{distribution}(4,j)+\text{distribution}(5,j)+\text{distribution}(6,j))/8;
  \text{average}(7,j) = (\text{distribution}(1,j)+\text{distribution}(2,j)+\text{distribution}(3,j)+\text{distribution}(4,j)+\text{distribution}(5,j)+\text{distribution}(6,j)+\text{distribution}(7,j))/8;
\text{for } i=8:1:100
  \text{average}(i,j) = (\text{distribution}(i,j)+\text{distribution}(i-1,j)+\text{distribution}(i-2,j)+\text{distribution}(i-3,j)+\text{distribution}(i-4,j)+\text{distribution}(i-5,j)+\text{distribution}(i-6,j)+\text{distribution}(i-7,j))/8;
\text{end}
\text{end}

\text{prediction = zeros(100,37);}
\text{prediction = average;}

\%Delay-sensitive traffic case
\text{ratio = zeros(37,1);}
\text{power = zeros(37,1);}
\text{delay = zeros(37,1);}
\text{delay\_max = zeros(37,1);}
\text{for } j=1:1:37
  \text{time} = 0;
  \text{time} \times \text{sleep} = 0;
  \text{tsleep} = 0;
  \text{tsleep} = 0;
  \text{n} = 0;
  \text{trigger\_time} = 0;
  \text{correction} = 0;
\text{for } i=2:1:100
  \text{time} = \text{time} + \text{distribution}(i-1,j);
  \text{if } (\text{trigger\_time}+\text{tsleep}+\text{n}\times \text{tsleep})<=\text{time}
    \text{trigger\_time} = \text{time};
    \text{correction} = 0;
  \text{else}
    \text{correction} = \text{trigger\_time} + \text{tsleep} + \text{n}\times \text{tsleep} - \text{time};
    \text{trigger\_time} = \text{trigger\_time} + \text{tsleep} + \text{n}\times \text{tsleep};
    \text{delay}(j) = \text{delay}(j) + \text{correction};
  \text{if } \text{correction} > \text{delay\_max}(j)
    \text{delay\_max}(j) = \text{correction;}
\text{end}
\text{end}

\text{end}
if prediction(i-1,j)<12*wakeup
    tsleep_1 = 0;
    tsleep_2 = 0;
    n_sleep = 0;
end
if prediction(i-1,j)>=12*wakeup && prediction(i-1,j)<2*tolerance
    tsleep_1 = prediction(i-1,j)/2 - correction;
    tsleep_2 = prediction(i-1,j)/2;
    n_sleep = ceil((distribution(i,j)-tsleep_2)/tsleep_2);
    time_sleep = time_sleep + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
end
if prediction(i-1,j)>2*tolerance
    tsleep_1 = tolerance - correction;
    tsleep_2 = tolerance;
    n_sleep = ceil((distribution(i,j)-tsleep_2)/tsleep_2);
    time_sleep = time_sleep + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
end
ratio(j) = time_sleep/(time + distribution(100,j));
power(j) = pactive - ratio(j)*(pactive - psleep);
delay(j) = delay(j)/100;
end

sleep_ratio_2_sen(:,index) = ratio;
avg_power_2_sen(:,index) = power;
avg_latency_2_sen(:,index) = delay;
worst_latency_2_sen(:,index) = delay_max;

%Delay-insensitive traffic case
ratio = zeros(37,1);
power = zeros(37,1);
delay = zeros(37,1);
delay_max = zeros (37,1);
for j=1:1:37
    time_sleep = 0;
    time = 0;
    tsleep_1 = 0;
    tsleep_2 = 0;
    n_sleep = 0;
    trigger_time = 0;
    correction = 0;
    for i=2:1:100
        time = time + distribution(i-1,j);
        if (trigger_time+tsleep_1+n_sleep*tsleep_2)<=time
            trigger_time = time;
            correction = 0;
        else
            correction = trigger_time + tsleep_1 + n_sleep*tsleep_2 - time;
            trigger_time = trigger_time + tsleep_1 + n_sleep*tsleep_2;
            delay(j) = delay(j) + correction;
            if correction > delay_max(j)
                delay_max(j) = correction;
            end
        end
    end
    if prediction(i-1,j)<12*wakeup
        tsleep_1 = 0;
        tsleep_2 = 0;
        n_sleep = 0;
    end
    if prediction(i-1,j)>=12*wakeup
        tsleep_1 = prediction(i-1,j)/2 - correction;
        tsleep_2 = prediction(i-1,j)/2;
        n_sleep = ceil((distribution(i,j)-tsleep_2)/tsleep_2);
        time_sleep = time_sleep + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
        delay(j) = delay(j) + correction;
        if correction > delay_max(j)
            delay_max(j) = correction;
        end
    end
end
ratio(j) = time_sleep/(time + distribution(100,j));
power(j) = pactive - ratio(j)*(pactive - psleep);
delay(j) = delay(j)/100;
end
sleep_ratio_2_sen(:,index) = ratio;
avg_power_2_sen(:,index) = power;
avg_latency_2_sen(:,index) = delay;
worst_latency_2_sen(:,index) = delay_max;
tsleep_1 = prediction(i-1,j)/2 - correction;
if tsleep_2 > 6*wakeup
    tsleep_2 = 6*wakeup;
end
n_sleep = ceil(((distribution(i,j) - tsleep_1 - correction)/tsleep_2);
time_sleep = time_sleep + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
end

end

ratio(j) = time_sleep/(time + distribution(100,j));
power(j) = pactive - ratio(j)*(pactive - psleep);
delay(j) = delay(j)/100;
end

sleep_ratio_2_ins(:,index) = ratio;
avg_power_2_ins(:,index) = power;
avg_latency_2_ins(:,index) = delay;
worst_latency_2_ins(:,index) = delay_max;
end

for j=1:1:37
    for i=1:1:10
        avg_sleep_ratio_1_sen(j) = avg_sleep_ratio_1_sen(j) + sleep_ratio_1_sen(j,i);
        avg_sleep_ratio_1_ins(j) = avg_sleep_ratio_1_ins(j) + sleep_ratio_1_ins(j,i);
        avg_sleep_ratio_2_sen(j) = avg_sleep_ratio_2_sen(j) + sleep_ratio_2_sen(j,i);
        avg_sleep_ratio_2_ins(j) = avg_sleep_ratio_2_ins(j) + sleep_ratio_2_ins(j,i);
        avg_avg_power_1_sen(j) = avg_avg_power_1_sen(j) + avg_power_1_sen(j,i);
        avg_avg_power_1_ins(j) = avg_avg_power_1_ins(j) + avg_power_1_ins(j,i);
        avg_avg_power_2_sen(j) = avg_avg_power_2_sen(j) + avg_power_2_sen(j,i);
        avg_avg_power_2_ins(j) = avg_avg_power_2_ins(j) + avg_power_2_ins(j,i);
        avg_avg_latency_1_sen(j) = avg_avg_latency_1_sen(j) + avg_latency_1_sen(j,i);
        avg_avg_latency_1_ins(j) = avg_avg_latency_1_ins(j) + avg_latency_1_ins(j,i);
        avg_avg_latency_2_sen(j) = avg_avg_latency_2_sen(j) + avg_latency_2_sen(j,i);
        avg_avg_latency_2_ins(j) = avg_avg_latency_2_ins(j) + avg_latency_2_ins(j,i);
        avg_worst_latency_1_sen(j) = avg_worst_latency_1_sen(j) + worst_latency_1_sen(j,i);
        avg_worst_latency_1_ins(j) = avg_worst_latency_1_ins(j) + worst_latency_1_ins(j,i);
        avg_worst_latency_2_sen(j) = avg_worst_latency_2_sen(j) + worst_latency_2_sen(j,i);
        avg_worst_latency_2_ins(j) = avg_worst_latency_2_ins(j) + worst_latency_2_ins(j,i);
    end
    avg_sleep_ratio_1_sen(j) = avg_sleep_ratio_1_sen(j)/10;
    avg_sleep_ratio_1_ins(j) = avg_sleep_ratio_1_ins(j)/10;
    avg_sleep_ratio_2_sen(j) = avg_sleep_ratio_2_sen(j)/10;
    avg_sleep_ratio_2_ins(j) = avg_sleep_ratio_2_ins(j)/10;
    avg_avg_power_1_sen(j) = avg_avg_power_1_sen(j)/10;
    avg_avg_power_1_ins(j) = avg_avg_power_1_ins(j)/10;
    avg_avg_power_2_sen(j) = avg_avg_power_2_sen(j)/10;
    avg_avg_power_2_ins(j) = avg_avg_power_2_ins(j)/10;
    avg_avg_latency_1_sen(j) = avg_avg_latency_1_sen(j)/10;
    avg_avg_latency_1_ins(j) = avg_avg_latency_1_ins(j)/10;
    avg_avg_latency_2_sen(j) = avg_avg_latency_2_sen(j)/10;
    avg_avg_latency_2_ins(j) = avg_avg_latency_2_ins(j)/10;
    avg_worst_latency_1_sen(j) = avg_worst_latency_1_sen(j)/10;
    avg_worst_latency_1_ins(j) = avg_worst_latency_1_ins(j)/10;
    avg_worst_latency_2_sen(j) = avg_worst_latency_2_sen(j)/10;
    avg_worst_latency_2_ins(j) = avg_worst_latency_2_ins(j)/10;
end

mean = zeros(37,1);
mean(1) = 0.1;
mean(2) = 0.2;
mean(3) = 0.3;
mean(4) = 0.4;
mean(5) = 0.5;
mean(6) = 0.6;
mean(7) = 0.7;
mean(8) = 0.8;
mean(9) = 0.9;
mean(10) = 1;
mean(11) = 2;
mean(12) = 3;
mean(13) = 4;
mean(14) = 5;
mean(15) = 6;
mean(16) = 7;
mean(17) = 8;
mean(18) = 9;
mean(19) = 10;
mean(20) = 20;
mean(21) = 30;
mean(22) = 40;
mean(23) = 50;
mean(24) = 60;
mean(25) = 70;
mean(26) = 80;
mean(27) = 90;
mean(28) = 100;
mean(29) = 200;
mean(30) = 300;
mean(31) = 400;
mean(32) = 500;
mean(33) = 600;
mean(34) = 700;
mean(35) = 800;
mean(36) = 900;
mean(37) = 1000;

figure
semilogx(mean(10:37), avg_sleep_ratio_1_sen(10:37), '-og', mean(10:37),
avg_sleep_ratio_1_ins(10:37), '-sb', mean(10:37), avg_sleep_ratio_2_sen(10:37), '-+k',
mean(10:37), avg_sleep_ratio_2_ins(10:37), '-*r')
xlabel('Average inter-arrival time [ms]')
ylabel('Ratio of time spent sleeping')
legend('Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.')
figure
semilogx(mean(10:37), avg_avg_power_1_sen(10:37), '-og', mean(10:37),
avg_avg_power_1_ins(10:37), '-sb', mean(10:37), avg_avg_power_2_sen(10:37), '-+k',
mean(10:37), avg_avg_power_2_ins(10:37), '-*r')
xlabel('Average inter-arrival time [ms]')
ylabel('Average power consumption [W]')
legend('Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.')
figure
loglog(mean(10:37), avg_avg_latency_1_sen(10:37), '-og', mean(10:37),
avg_avg_latency_1_ins(10:37), '-sb', mean(10:37), avg_avg_latency_2_sen(10:37), '-+k',
mean(10:37), avg_avg_latency_2_ins(10:37), '-*r')
xlabel('Average inter-arrival time [ms]')
ylabel('Average additional latency [ms]')
legend('Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.')
figure
loglog(mean(10:37), avg_worst_latency_1_sen(10:37), '-og', mean(10:37),
avg_worst_latency_1_ins(10:37), '-sb', mean(10:37), avg_worst_latency_2_sen(10:37), '-+k',
mean(10:37), avg_worst_latency_2_ins(10:37), '-*r')
xlabel('Average inter-arrival time [ms]')
A1.2 Matlab code used for the second set of simulations

```
rand('seed',9);
randn('seed',9);
sleep_ratio_1_sen = zeros(7,10);
sleep_ratio_1_ins = zeros(7,10);
sleep_ratio_2_sen = zeros(7,10);
sleep_ratio_2_ins = zeros(7,10);
avg_power_1_sen = zeros(7,10);
avg_power_1_ins = zeros(7,10);
avg_power_2_sen = zeros(7,10);
avg_power_2_ins = zeros(7,10);
avg_latency_1_sen = zeros(7,10);
avg_latency_1_ins = zeros(7,10);
avg_latency_2_sen = zeros(7,10);
avg_latency_2_ins = zeros(7,10);
worst_latency_1_sen = zeros(7,10);
worst_latency_1_ins = zeros(7,10);
worst_latency_2_sen = zeros(7,10);
worst_latency_2_ins = zeros(7,10);

avg_sleep_ratio_1_sen = zeros(7,1);
avg_sleep_ratio_1_ins = zeros(7,1);
avg_sleep_ratio_2_sen = zeros(7,1);
avg_sleep_ratio_2_ins = zeros(7,1);
avg_avg_power_1_sen = zeros(7,1);
avg_avg_power_1_ins = zeros(7,1);
avg_avg_power_2_sen = zeros(7,1);
avg_avg_power_2_ins = zeros(7,1);
avg_avg_latency_1_sen = zeros(7,1);
avg_avg_latency_1_ins = zeros(7,1);
avg_avg_latency_2_sen = zeros(7,1);
avg_avg_latency_2_ins = zeros(7,1);
avg_worst_latency_1_sen = zeros(7,1);
avg_worst_latency_1_ins = zeros(7,1);
avg_worst_latency_2_sen = zeros(7,1);
avg_worst_latency_2_ins = zeros(7,1);

%PARAMETERS IN ms OR W
tolerance = 10;
wakeup = 1;
pactive = 10;
psleep = 2;

for index=1:1:10

%GENERATION OF THE DISTRIBUTION
sum = 0;
i_1 = 0;
while sum<10000 %generation of the distribution during the first minute
    i_1=i_1+1;
    distribution_1=exprnd(3, i_1, 1);
```
sum = 0;
for k=1:1:i_1
    sum = sum + distribution_1(k);
end

i_1=i_1-1;
distribution_1=distribution_1(1:i_1);
sum = 0;
i_2 = 0;
while sum<10000  %generation of the distribution during the second minute
    i_2=i_2+1;
    distribution_2=exprnd(90, i_2, 1);
    sum = 0;
    for k=1:1:i_2
        sum = sum + distribution_2(k);
    end
end
i_2=i_2-1;
distribution_2=distribution_2(1:i_2);
sum = 0;
i_3 = 0;
while sum<10000  %generation of the distribution during the third minute
    i_3=i_3+1;
    distribution_3=exprnd(270, i_3, 1);
    sum = 0;
    for k=1:1:i_3
        sum = sum + distribution_3(k);
    end
end
i_3=i_3-1;
distribution_3=distribution_3(1:i_3);
sum = 0;
i_4 = 0;
while sum<10000  %generation of the distribution during the fourth minute
    i_4=i_4+1;
    distribution_4=exprnd(5, i_4, 1);
    sum = 0;
    for k=1:1:i_4
        sum = sum + distribution_4(k);
    end
end
i_4=i_4-1;
distribution_4=distribution_4(1:i_4);
sum = 0;
i_5 = 0;
while sum<10000  %generation of the distribution during the fifth minute
    i_5=i_5+1;
    distribution_5=exprnd(150, i_5, 1);
    sum = 0;
    for k=1:1:i_5
        sum = sum + distribution_5(k);
    end
end
i_5=i_5-1;
distribution_5=distribution_5(1:i_5);
sum = 0;
i_6 = 0;
while sum<10000  %generation of the distribution during the sixth minute
    i_6=i_6+1;
    distribution_6=exprnd(9, i_6, 1);
    sum = 0;
    for k=1:1:i_6
        sum = sum + distribution_6(k);
    end
end
i_6=i_6-1;
distribution_6=distribution_6(1:i_6);
sum = 0;
i_7 = 0;
while sum<10000 %generation of the distribution during the seventh minute
  i_7=i_7+1;
distribution_7=exprnd(90, i_7, 1);
  sum = 0;
  for k=1:i_7
    sum = sum + distribution_7(k);
  end
end
i_7=i_7-1;
distribution_7=distribution_7(1:i_7);
distribution = [distribution_1; distribution_2; distribution_3; distribution_4;
  distribution_5; distribution_6; distribution_7];
i = i_1 + i_2 + i_3 + i_4 + i_5 + i_6 + i_7;
if index == 1
  mean = [3 3 90 90 270 270 5 5 150 150 9 9 90 90];
t = [0 10 10.001 20 20.001 30 30.001 40 40.001 50 50.001 60 60.001 70]
  figure
  plot (t, mean)
  xlabel ('Time [s]')
  ylabel ('Average inter-arrival time [ms]')
end

%PROPOSED METHOD

%Computation of the arithmetic average on eight samples
average = [1:1:i];
average(1) = distribution(1)/8;
average(2) = (distribution(1)+distribution(2))/8;
average(3) = (distribution(1)+distribution(2)+distribution(3))/8;
average(4) = (distribution(1)+distribution(2)+distribution(3)+distribution(4))/8;
average(5) = (distribution(1)+distribution(2)+distribution(3)+distribution(4)+distribution(5))/8;
average(6) = (distribution(1)+distribution(2)+distribution(3)+distribution(4)+distribution(5)+distribution(6))/8;
average(7) = (distribution(1)+distribution(2)+distribution(3)+distribution(4)+distribution(5)+distribution(6)+distribution(7))/8;
for k=8:1:i
  average(k) = (distribution(k)+distribution(k-1)+distribution(k-2)+distribution(k-3)+distribution(k-4)+distribution(k-5)+distribution(k-6)+distribution(k-7))/8;
end

%Computation of the adaptive exponentially smoothed average
prediction = zeros(i,1);
reset = 0;
for k = 1:i
  if k> 2
    if abs(average(k)-average(k-1))>average(k) || abs(average(k)-average(k-1))>average(k-1)
      prediction(k) = distribution(k);
      reset = k;
    else
      prediction(k) = 2*distribution(k)/(k-reset+2) + (1-2/(k-
        reset+2))*prediction(k-1);
    end
  else
    prediction(k) = distribution(k);
  end
end
%Delay-sensitive traffic case
ratio = zeros(7,1);
power = zeros(7,1);
delay = zeros(7,1);
delay_max = zeros(7,1);
time_sleep = zeros (7,1);
time = zeros (7,1);
tsleep_1 = 0;
tsleep_2 = 0;
n_sleep = 0;
trigger_time = 0;
correction = 0;
flag_1 = 0;
flag_2 = 0;
flag_3 = 0;
flag_4 = 0;
flag_5 = 0;
flag_6 = 0;
for k=2:1:i
    if k<=i_1
        j=1;
    end
    if k<=(i_1+i_2) && k>i_1
        j=2;
    end
    if k<=(i_1+i_2+i_3) && k>(i_1+i_2)
        j=3;
    end
    if k<=(i_1+i_2+i_3+i_4) && k>(i_1+i_2+i_3)
        j=4;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5) && k>(i_1+i_2+i_3+i_4)
        j=5;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5+i_6) && k>(i_1+i_2+i_3+i_4+i_5)
        j=6;
    end
    if k>(i_1+i_2+i_3+i_4) && k>i_1+i_2+i_3+i_4+i_5+i_6)
        j=7;
    end
    if time(j) == 0
        if j == 1
            time(j) = 0;
        else
            time(j) = time(j-1);
        end
    end
    time(j) = time(j) + distribution(k-1);
    if (trigger_time+tsleep_1+n_sleep*tsleep_2)<=time(j)
        trigger_time = time(j);
        correction = 0;
    else
        correction = trigger_time + tsleep_1 + n_sleep*tsleep_2 - time(j);
        trigger_time = trigger_time + tsleep_1 + n_sleep*tsleep_2;
        delay(j) = delay(j) + correction;
        if correction > delay_max(j)
            delay_max(j) = correction;
        end
    end
    if prediction(k-1)<12*wakeup
        tsleep_1 = 0;
        tsleep_2 = 0;
end
n_sleep = 0;
end
if prediction(k-1)> = 12*wakeup && prediction(k-1)< = 2*tolerance
    tsleep_1 = prediction(k-1)/2 - correction;
    tsleep_2 = prediction(k-1)/2;
    n_sleep = ceil((distribution(k)-tsleep_2)/tsleep_2);
    time_sleep(j) = time_sleep(j) + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
end
if prediction(k-1)> 2*tolerance
    tsleep_1 = tolerance - correction;
    tsleep_2 = tolerance;
    n_sleep = ceil((distribution(k)-tsleep_2)/tsleep_2);
    time_sleep(j) = time_sleep(j) + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
end
if j == 2
    ratio(1) = time_sleep(1)/time(1);
    power(1) = pactive - ratio(1)*(pactive-psleep);
    if flag_1 == 0
        delay(1) = delay(1)/i_1;
        flag_1 = 1;
    end
end
if j>2
    ratio(j-1) = time_sleep(j-1)/(time(j-1)-time(j-2));
    power(j-1) = pactive - ratio(j-1)*(pactive-psleep);
    if j == 3 && flag_2 == 0
        delay(2) = delay(2)/i_2;
        flag_2 = 1;
    end
    if j == 4 && flag_3 == 0
        delay(3) = delay(3)/i_3;
        flag_3 = 1;
    end
    if j == 5 && flag_4 == 0
        delay(4) = delay(4)/i_4;
        flag_4 = 1;
    end
    if j == 6 && flag_5 == 0
        delay(5) = delay(5)/i_5;
        flag_5 = 1;
    end
    if j == 7 && flag_6 == 0
        delay(6) = delay(6) / i_6;
        flag_6 = 1;
end
delay(7) = delay(7) / i_7;
sleep_ratio_1_sen(:,index) = ratio;
avg_power_1_sen(:,index) = power;
avg_latency_1_sen(:,index) = delay;
worst_latency_1_sen(:,index) = delay_max;

%Delay-insensitive traffic case
ratio = zeros(7,1);
power = zeros(7,1);
delay = zeros(7,1);
delay_max = zeros (7,1);
time_sleep = zeros (7,1);
time = zeros (7,1);
tsleep_1 = 0;
tsleep_2 = 0;
n_sleep = 0;
trigger_time = 0;
correction = 0;
flag_1 = 0;
flag_2 = 0;
flag_3 = 0;
flag_4 = 0;
flag_5 = 0;
flag_6 = 0;
for k=2:1:i
    if k<=i_1
        j=1;
    end
    if k<=(i_1+i_2) && k>i_1
        j=2;
    end
    if k<=(i_1+i_2+i_3) && k>(i_1+i_2)
        j=3;
    end
    if k<=(i_1+i_2+i_3+i_4) && k>(i_1+i_2+i_3)
        j=4;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5) && k>(i_1+i_2+i_3+i_4)
        j=5;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5+i_6) && k>(i_1+i_2+i_3+i_4+i_5)
        j=6;
    end
    if k>(i_1+i_2+i_3+i_4+i_5+i_6)
        j=7;
    end
    if time(j) == 0
        if j == 1
            time(j) = 0;
        else
            time(j) = time(j-1);
        end
        time(j) = time(j) + distribution(k-1);
        if (trigger_time+tsleep_1+n_sleep*tsleep_2)<=time(j)
            trigger_time = time(j);
            correction = 0;
        else
            correction = trigger_time + tsleep_1 + n_sleep*tsleep_2 - time(j);
            trigger_time = trigger_time + tsleep_1 + n_sleep*tsleep_2;
            delay(j) = delay(j) + correction;
            if correction > delay_max(j)
                delay_max(j) = correction;
            end
        end
        if prediction(k-1)<12*wakeup
            tsleep_1 = 0;
            tsleep_2 = 0;
            n_sleep = 0;
        end
    end
    if prediction(k-1)>=12*wakeup
        tsleep_1 = prediction(k-1)/2 - correction;
        tsleep_2 = prediction(k-1)/16;
        if tsleep_2<6*wakeup
            tsleep_2 = 6*wakeup;
        end
n_sleep = ceil((distribution(k)-tsleep_1-correction)/tsleep_2);
time_sleep(j) = time_sleep(j) + tsleep_1 - wakeup + n_sleep*(tsleep_2 -
wakeup);

if j == 2
    ratio(1) = time_sleep(1)/time(1);
power(1) = pactive - ratio(1)*(pactive-psleep);
if flag_1 == 0
    delay(1) = delay(1)/i_1;
    flag_1 = 1;
end
end
if j>2
    ratio(j-1) = time_sleep(j-1)/(time(j)-time(j-2));
power(j-1) = pactive - ratio(j-1)*(pactive-psleep);
if j == 3 && flag_2 == 0
    delay(2) = delay(2)/i_2;
    flag_2 = 1;
end
if j == 4 && flag_3 == 0
    delay(3) = delay(3)/i_3;
    flag_3 = 1;
end
if j == 5 && flag_4 == 0
    delay(4) = delay(4)/i_4;
    flag_4 = 1;
end
if j == 6 && flag_5 == 0
    delay(5) = delay(5)/i_5;
    flag_5 = 1;
end
if j == 7 && flag_6 == 0
    delay(6) = delay(6)/ i_6;
    flag_6 = 1;
end
end
ratio(7) = time_sleep(7)/(time(7)-time(6));
power(7) = pactive - ratio(7)*(pactive-psleep);
delay(7) = delay(7) / i_7;
sleep_ratio_1_ins(:,index) = ratio;
avg_power_1_ins(:,index) = power;
avg_latency_1_ins(:,index) = delay;
worst_latency_1_ins(:,index) = delay_max;

%SIMPLIFIED METHOD
%Computation of the arithmetic average on eight samples
average = [1:1:i];
average(1) = distribution(1)/8;
average(2) = (distribution(1)+distribution(2))/8;
average(3) = (distribution(1)+distribution(2)+distribution(3))/8;
average(4) = (distribution(1)+distribution(2)+distribution(3)+distribution(4))/8;
average(5) = (distribution(1)+distribution(2)+distribution(3)+distribution(4)+distribution(5))/8;
average(6) = (distribution(1)+distribution(2)+distribution(3)+distribution(4)+distribution(5)+dist ribution(6))/8;
average(7) = (distribution(1)+distribution(2)+distribution(3)+distribution(4)+distribution(5)+dist ribution(6)+distribution(7))/8;
for k=8:1:i
average(k) = (distribution(k)+distribution(k-1)+distribution(k-2)+distribution(k-3)+distribution(k-4)+distribution(k-5)+distribution(k-6)+distribution(k-7))/8;
end

prediction = zeros(i,1);
prediction = average;

% Delay-sensitive traffic case
ratio = zeros(7,1);
power = zeros(7,1);
delay = zeros(7,1);
delay_max = zeros(7,1);
time_sleep = zeros(7,1);
time = zeros(7,1);
tsleep_1 = 0;
tsleep_2 = 0;
n_sleep = 0;
trigger_time = 0;
correction = 0;
flag_1 = 0;
flag_2 = 0;
flag_3 = 0;
flag_4 = 0;
flag_5 = 0;
flag_6 = 0;
for k=2:1:i
    if k<=i_1
        j=1;
    end
    if k<=(i_1+i_2) && k>i_1
        j=2;
    end
    if k<=(i_1+i_2+i_3) && k>(i_1+i_2)
        j=3;
    end
    if k<=(i_1+i_2+i_3+i_4) && k>(i_1+i_2+i_3)
        j=4;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5) && k>(i_1+i_2+i_3+i_4)
        j=5;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5+i_6) && k>(i_1+i_2+i_3+i_4+i_5)
        j=6;
    end
    if k>(i_1+i_2+i_3+i_4+i_5+i_6)
        j=7;
    end
    if time(j) == 0
        if j == 1
            time(j) = 0;
        else
            time(j) = time(j-1);
        end
    end
    time(j) = time(j) + distribution(k-1);
    if (trigger_time+tsleep_1+n_sleep*tsleep_2)<=time(j)
        trigger_time = time(j);
        correction = 0;
    else
        correction = trigger_time + tsleep_1 + n_sleep*tsleep_2 - time(j);
        trigger_time = trigger_time + tsleep_1 + n_sleep*tsleep_2;
        delay(j) = delay(j) + correction;
        if correction > delay_max(j)
            delay_max(j) = correction;
        end
    end
end
end
end
if prediction(k-1)<12*wakeup
    tsleep_1 = 0;
    tsleep_2 = 0;
    n_sleep = 0;
end
if prediction(k-1)>=12*wakeup && prediction(k-1)<=2*tolerance
    tsleep_1 = prediction(k-1)/2 - correction;
    tsleep_2 = prediction(k-1)/2;
    n_sleep = ceil((distribution(k)-tsleep_2)/tsleep_2);
    time_sleep(j) = time_sleep(j) + tsleep_1 - wakeup + n_sleep*(tsleep_2 -
                  wakeup);
end
if prediction(k-1)>2*tolerance
    tsleep_1 = tolerance - correction;
    tsleep_2 = tolerance;
    n_sleep = ceil((distribution(k)-tsleep_2)/tsleep_2);
    time_sleep(j) = time_sleep(j) + tsleep_1 - wakeup + n_sleep*(tsleep_2 -
                  wakeup);
end
if j == 2
    ratio(1) = time_sleep(1)/time(1);
    power(1) = pactive - ratio(1)*(pactive-psleep);
    if flag_1 == 0
        delay(1) = delay(1)/i_1;
        flag_1 = 1;
    end
end
if j>2
    ratio(j-1) = time_sleep(j-1)/(time(j-1)-time(j-2));
    power(j-1) = pactive - ratio(j-1)*(pactive-psleep);
    if j == 3 && flag_2 == 0
        delay(2) = delay(2)/i_2;
        flag_2 = 1;
    end
    if j == 4 && flag_3 == 0
        delay(3) = delay(3)/i_3;
        flag_3 = 1;
    end
    if j == 5 && flag_4 == 0
        delay(4) = delay(4)/i_4;
        flag_4 = 1;
    end
    if j == 6 && flag_5 == 0
        delay(5) = delay(5)/i_5;
        flag_5 = 1;
    end
    if j == 7 && flag_6 == 0
        delay(6) = delay(6) / i_6;
        flag_6 = 1;
    end
end
ratio(7) = time_sleep(7)/(time(7)-time(6));
power(7) = pactive - ratio(7)*(pactive-psleep);
delay(7) = delay(7) / i_7;
sleep_ratio_2_sen(:,index) = ratio;
avg_power_2_sen(:,index) = power;
avg_latency_2_sen(:,index) = delay;
worst_latency_2_sen(:,index) = delay_max;

%Delay-insensitive traffic case
ratio = zeros(7,1);
power = zeros(7,1);
delay = zeros(7,1);
delay_max = zeros (7,1);
time_sleep = zeros (7,1);
time = zeros (7,1);
tsleep_1 = 0;
tsleep_2 = 0;
n_sleep = 0;
trigger_time = 0;
correction = 0;
flag_1 = 0;
flag_2 = 0;
flag_3 = 0;
flag_4 = 0;
flag_5 = 0;
flag_6 = 0;

for k=2:1:i
    if k<=i_1
        j=1;
    end
    if k<=(i_1+i_2) && k>i_1
        j=2;
    end
    if k<=(i_1+i_2+i_3) && k>(i_1+i_2)
        j=3;
    end
    if k<=(i_1+i_2+i_3+i_4) && k>(i_1+i_2+i_3)
        j=4;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5) && k>(i_1+i_2+i_3+i_4)
        j=5;
    end
    if k<=(i_1+i_2+i_3+i_4+i_5+i_6) && k>(i_1+i_2+i_3+i_4+i_5)
        j=6;
    end
    if k>(i_1+i_2+i_3+i_4+i_5+i_6)
        j=7;
    end

    if time(j) == 0
        if j == 1
            time(j) = 0;
        else
            time(j) = time(j-1);
        end
    end

    time(j) = time(j) + distribution(k-1);
    if (trigger_time+tsleep_1+n_sleep*tsleep_2)<=time(j)
        trigger_time = time(j);
        correction = 0;
    else
        correction = trigger_time + tsleep_1 + n_sleep*tsleep_2 - time(j);
        trigger_time = trigger_time + tsleep_1 + n_sleep*tsleep_2;
        delay(j) = delay(j) + correction;
        if correction > delay_max(j)
            delay_max(j) = correction;
        end
    end

    if prediction(k-1)<12*wakeup
        tsleep_1 = 0;
        tsleep_2 = 0;
        n_sleep = 0;
    end
    if prediction(k-1)>=12*wakeup
tsleep_1 = prediction(k-1)/2 - correction;
if tsleep_2<6*wakeup
    tsleep_2 = 6*wakeup;
end
n_sleep = ceil((distribution(k)-tsleep_1-correction)/tsleep_2);
time_sleep(j) = time_sleep(j) + tsleep_1 - wakeup + n_sleep*(tsleep_2 - wakeup);
end
if j == 2
    ratio(1) = time_sleep(1)/time(1);
power(1) = pactive - ratio(1)*(pactive-psleep);
    if flag_1 == 0
        delay(1) = delay(1)/i_1;
        flag_1 = 1;
    end
end
if j>2
    ratio(j-1) = time_sleep(j-1)/(time(j-1)-time(j-2));
power(j-1) = pactive - ratio(j-1)*(pactive-psleep);
    if j == 3 && flag_2 == 0
        delay(2) = delay(2)/i_2;
        flag_2 = 1;
    end
    if j == 4 && flag_3 == 0
        delay(3) = delay(3)/i_3;
        flag_3 = 1;
    end
    if j == 5 && flag_4 == 0
        delay(4) = delay(4)/i_4;
        flag_4 = 1;
    end
    if j == 6 && flag_5 == 0
        delay(5) = delay(5)/i_5;
        flag_5 = 1;
    end
    if j == 7 && flag_6 == 0
        delay(6) = delay(6) / i_6;
        flag_6 = 1;
    end
end
end
ratio(7) = time_sleep(7)/(time(7)-time(6));
power(7) = pactive - ratio(7)*(pactive-psleep);
delay(7) = delay(7) / i_7;
sleep_ratio_2_ins(:,index) = ratio;
avg_power_2_ins(:,index) = power;
avg_latency_2_ins(:,index) = delay;
worst_latency_2_ins(:,index) = delay_max;
end
for j=1:1:7
    for i=1:1:10
        avg_sleep_ratio_1_sen(j) = avg_sleep_ratio_1_sen(j) + sleep_ratio_1_sen(j,i);
        avg_sleep_ratio_1_ins(j) = avg_sleep_ratio_1_ins(j) + sleep_ratio_1_ins(j,i);
        avg_sleep_ratio_2_sen(j) = avg_sleep_ratio_2_sen(j) + sleep_ratio_2_sen(j,i);
        avg_sleep_ratio_2_ins(j) = avg_sleep_ratio_2_ins(j) + sleep_ratio_2_ins(j,i);
        avg_avg_power_1_sen(j) = avg_avg_power_1_sen(j) + avg_power_1_sen(j,i);
        avg_avg_power_1_ins(j) = avg_avg_power_1_ins(j) + avg_power_1_ins(j,i);
        avg_avg_power_2_sen(j) = avg_avg_power_2_sen(j) + avg_power_2_sen(j,i);
        avg_avg_power_2_ins(j) = avg_avg_power_2_ins(j) + avg_power_2_ins(j,i);
        avg_avg_latency_1_sen(j) = avg_avg_latency_1_sen(j) + avg_latency_1_sen(j,i);
    end
end
avg_avg_latency_1_ins(j) = avg_avg_latency_1_ins(j) + avg_latency_1_ins(j,i);
avg_avg_latency_2_sen(j) = avg_avg_latency_2_sen(j) + avg_latency_2_sen(j,i);
avg_avg_latency_2_ins(j) = avg_avg_latency_2_ins(j) + avg_latency_2_ins(j,i);
worst_latency_1_sen(j,i) = worst_latency_1_sen(j,i) + avg_latency_1_sen(j,i);
worst_latency_1_ins(j,i) = worst_latency_1_ins(j,i) + avg_latency_1_ins(j,i);
worst_latency_2_sen(j,i) = worst_latency_2_sen(j,i) + avg_latency_2_sen(j,i);
worst_latency_2_ins(j,i) = worst_latency_2_ins(j,i) + avg_latency_2_ins(j,i);

end
avg_sleep_ratio_1_sen(j) = avg_sleep_ratio_1_sen(j)/10;
sleep_ratio_1_sen = zeros(1, 14);
sleep_ratio_1_sen(1) = avg_sleep_ratio_1_sen(1);
sleep_ratio_1_sen(2) = avg_sleep_ratio_1_sen(2);
sleep_ratio_1_sen(3) = avg_sleep_ratio_1_sen(3);
sleep_ratio_1_sen(4) = avg_sleep_ratio_1_sen(4);
sleep_ratio_1_sen(5) = avg_sleep_ratio_1_sen(5);
sleep_ratio_1_sen(6) = avg_sleep_ratio_1_sen(6);
sleep_ratio_1_sen(7) = avg_sleep_ratio_1_sen(7);
sleep_ratio_1_sen(8) = avg_sleep_ratio_1_sen(8);
sleep_ratio_1_sen(9) = avg_sleep_ratio_1_sen(9);
sleep_ratio_1_sen(10) = avg_sleep_ratio_1_sen(10);
sleep_ratio_1_sen(11) = avg_sleep_ratio_1_sen(11);
sleep_ratio_1_sen(12) = avg_sleep_ratio_1_sen(12);
sleep_ratio_1_sen(13) = avg_sleep_ratio_1_sen(13);
sleep_ratio_1_sen(14) = avg_sleep_ratio_1_sen(14);
sleep_ratio_1_sen(1) = avg_sleep_ratio_1_sen(1);
sleep_ratio_1_sen(2) = avg_sleep_ratio_1_sen(2);
sleep_ratio_1_sen(3) = avg_sleep_ratio_1_sen(3);
sleep_ratio_1_sen(4) = avg_sleep_ratio_1_sen(4);
sleep_ratio_1_sen(5) = avg_sleep_ratio_1_sen(5);
sleep_ratio_1_sen(6) = avg_sleep_ratio_1_sen(6);
sleep_ratio_1_sen(7) = avg_sleep_ratio_1_sen(7);
sleep_ratio_1_sen(8) = avg_sleep_ratio_1_sen(8);
sleep_ratio_1_sen(9) = avg_sleep_ratio_1_sen(9);
sleep_ratio_1_sen(10) = avg_sleep_ratio_1_sen(10);
sleep_ratio_1_sen(11) = avg_sleep_ratio_1_sen(11);
sleep_ratio_1_sen(12) = avg_sleep_ratio_1_sen(12);
sleep_ratio_1_sen(13) = avg_sleep_ratio_1_sen(13);
sleep_ratio_1_sen(14) = avg_sleep_ratio_1_sen(14);
sleep_ratio_1_ins(14) = avg_sleep_ratio_1_ins(7);
sleep_ratio_2_sen(1) = avg_sleep_ratio_2_sen(1);
sleep_ratio_2_sen(2) = avg_sleep_ratio_2_sen(1);
sleep_ratio_2_sen(3) = avg_sleep_ratio_2_sen(2);
sleep_ratio_2_sen(4) = avg_sleep_ratio_2_sen(2);
sleep_ratio_2_sen(5) = avg_sleep_ratio_2_sen(3);
sleep_ratio_2_sen(6) = avg_sleep_ratio_2_sen(3);
sleep_ratio_2_sen(7) = avg_sleep_ratio_2_sen(4);
sleep_ratio_2_sen(8) = avg_sleep_ratio_2_sen(4);
sleep_ratio_2_sen(9) = avg_sleep_ratio_2_sen(5);
sleep_ratio_2_sen(10) = avg_sleep_ratio_2_sen(5);
sleep_ratio_2_sen(11) = avg_sleep_ratio_2_sen(6);
sleep_ratio_2_sen(12) = avg_sleep_ratio_2_sen(6);
sleep_ratio_2_sen(13) = avg_sleep_ratio_2_sen(7);
sleep_ratio_2_sen(14) = avg_sleep_ratio_2_sen(7);
sleep_ratio_2_ins(1) = avg_sleep_ratio_2_ins(1);
sleep_ratio_2_ins(2) = avg_sleep_ratio_2_ins(1);
sleep_ratio_2_ins(3) = avg_sleep_ratio_2_ins(2);
sleep_ratio_2_ins(4) = avg_sleep_ratio_2_ins(2);
sleep_ratio_2_ins(5) = avg_sleep_ratio_2_ins(3);
sleep_ratio_2_ins(6) = avg_sleep_ratio_2_ins(3);
sleep_ratio_2_ins(7) = avg_sleep_ratio_2_ins(4);
sleep_ratio_2_ins(8) = avg_sleep_ratio_2_ins(4);
sleep_ratio_2_ins(9) = avg_sleep_ratio_2_ins(5);
sleep_ratio_2_ins(10) = avg_sleep_ratio_2_ins(5);
sleep_ratio_2_ins(11) = avg_sleep_ratio_2_ins(6);
sleep_ratio_2_ins(12) = avg_sleep_ratio_2_ins(6);
sleep_ratio_2_ins(13) = avg_sleep_ratio_2_ins(7);
sleep_ratio_2_ins(14) = avg_sleep_ratio_2_ins(7);

avg_power_1_ins = zeros(1, 14);
avg_power_1_sen = zeros(1, 14);
avg_power_2_ins = zeros(1, 14);
avg_power_2_sen = zeros(1, 14);
avg_power_1_ins(1) = avg_avg_power_1_ins(1);
avg_power_1_ins(2) = avg_avg_power_1_ins(1);
avg_power_1_ins(3) = avg_avg_power_1_ins(2);
avg_power_1_ins(4) = avg_avg_power_1_ins(2);
avg_power_1_ins(5) = avg_avg_power_1_ins(3);
avg_power_1_ins(6) = avg_avg_power_1_ins(3);
avg_power_1_ins(7) = avg_avg_power_1_ins(4);
avg_power_1_ins(8) = avg_avg_power_1_ins(4);
avg_power_1_ins(9) = avg_avg_power_1_ins(5);
avg_power_1_ins(10) = avg_avg_power_1_ins(5);
avg_power_1_ins(11) = avg_avg_power_1_ins(6);
avg_power_1_ins(12) = avg_avg_power_1_ins(6);
avg_power_1_ins(13) = avg_avg_power_1_ins(7);
avg_power_1_ins(14) = avg_avg_power_1_ins(7);

avg_power_1_sen = zeros(1, 14);
avg_power_1_sen(1) = avg_avg_power_1_sen(1);
avg_power_1_sen(2) = avg_avg_power_1_sen(1);
avg_power_1_sen(3) = avg_avg_power_1_sen(2);
avg_power_1_sen(4) = avg_avg_power_1_sen(2);
avg_power_1_sen(5) = avg_avg_power_1_sen(3);
avg_power_1_sen(6) = avg_avg_power_1_sen(3);
avg_power_1_sen(7) = avg_avg_power_1_sen(4);
avg_power_1_sen(8) = avg_avg_power_1_sen(4);
avg_power_1_sen(9) = avg_avg_power_1_sen(5);
avg_power_1_sen(10) = avg_avg_power_1_sen(5);
avg_power_1_sen(11) = avg_avg_power_1_sen(6);
avg_power_1_sen(12) = avg_avg_power_1_sen(6);
avg_power_1_sen(13) = avg_avg_power_1_sen(7);
avg_power_1_sen(14) = avg_avg_power_1_sen(7);
avg_power_2_sen(3) = avg_avg_power_2_sen(2);
avg_power_2_sen(4) = avg_avg_power_2_sen(2);
avg_power_2_sen(5) = avg_avg_power_2_sen(3);
avg_power_2_sen(6) = avg_avg_power_2_sen(3);
avg_power_2_sen(7) = avg_avg_power_2_sen(4);
avg_power_2_sen(8) = avg_avg_power_2_sen(4);
avg_power_2_sen(9) = avg_avg_power_2_sen(5);
avg_power_2_sen(10) = avg_avg_power_2_sen(5);
avg_power_2_sen(11) = avg_avg_power_2_sen(6);
avg_power_2_sen(12) = avg_avg_power_2_sen(6);
avg_power_2_sen(13) = avg_avg_power_2_sen(7);
avg_power_2_sen(14) = avg_avg_power_2_sen(7);

avg_power_2_ins(1) = avg_avg_power_2_ins(1);
avg_power_2_ins(2) = avg_avg_power_2_ins(1);
avg_power_2_ins(3) = avg_avg_power_2_ins(2);
avg_power_2_ins(4) = avg_avg_power_2_ins(2);
avg_power_2_ins(5) = avg_avg_power_2_ins(3);
avg_power_2_ins(6) = avg_avg_power_2_ins(3);
avg_power_2_ins(7) = avg_avg_power_2_ins(4);
avg_power_2_ins(8) = avg_avg_power_2_ins(4);
avg_power_2_ins(9) = avg_avg_power_2_ins(5);
avg_power_2_ins(10) = avg_avg_power_2_ins(5);
avg_power_2_ins(11) = avg_avg_power_2_ins(6);
avg_power_2_ins(12) = avg_avg_power_2_ins(6);
avg_power_2_ins(13) = avg_avg_power_2_ins(7);
avg_power_2_ins(14) = avg_avg_power_2_ins(7);

avg_latency_1_ins = zeros(1, 14);
avg_latency_1_sen = zeros(1, 14);
avg_latency_2_ins = zeros(1, 14);
avg_latency_2_sen = zeros(1, 14);

avg_latency_1_sen(1) = avg_avg_latency_1_sen(1);
avg_latency_1_sen(2) = avg_avg_latency_1_sen(1);
avg_latency_1_sen(3) = avg_avg_latency_1_sen(2);
avg_latency_1_sen(4) = avg_avg_latency_1_sen(2);
avg_latency_1_sen(5) = avg_avg_latency_1_sen(3);
avg_latency_1_sen(6) = avg_avg_latency_1_sen(3);
avg_latency_1_sen(7) = avg_avg_latency_1_sen(4);
avg_latency_1_sen(8) = avg_avg_latency_1_sen(4);
avg_latency_1_sen(9) = avg_avg_latency_1_sen(5);
avg_latency_1_sen(10) = avg_avg_latency_1_sen(5);
avg_latency_1_sen(11) = avg_avg_latency_1_sen(6);
avg_latency_1_sen(12) = avg_avg_latency_1_sen(6);
avg_latency_1_sen(13) = avg_avg_latency_1_sen(7);
avg_latency_1_sen(14) = avg_avg_latency_1_sen(7);

avg_latency_1_ins(1) = avg_avg_latency_1_ins(1);
avg_latency_1_ins(2) = avg_avg_latency_1_ins(1);
avg_latency_1_ins(3) = avg_avg_latency_1_ins(2);
avg_latency_1_ins(4) = avg_avg_latency_1_ins(2);
avg_latency_1_ins(5) = avg_avg_latency_1_ins(3);
avg_latency_1_ins(6) = avg_avg_latency_1_ins(3);
avg_latency_1_ins(7) = avg_avg_latency_1_ins(4);
avg_latency_1_ins(8) = avg_avg_latency_1_ins(4);
avg_latency_1_ins(9) = avg_avg_latency_1_ins(5);
avg_latency_1_ins(10) = avg_avg_latency_1_ins(5);
avg_latency_1_ins(11) = avg_avg_latency_1_ins(6);
avg_latency_1_ins(12) = avg_avg_latency_1_ins(6);
avg_latency_1_ins(13) = avg_avg_latency_1_ins(7);
avg_latency_1_ins(14) = avg_avg_latency_1_ins(7);

avg_latency_2_sen(1) = avg_avg_latency_2_sen(1);
avg_latency_2_sen(2) = avg_avg_latency_2_sen(1);
avg_latency_2_sen(3) = avg_avg_latency_2_sen(2);
avg_latency_2_sen(4) = avg_avg_latency_2_sen(2);
avg_latency_2_sen(5) = avg_avg_latency_2_sen(3);
avg_latency_2_sen(6) = avg_avg_latency_2_sen(3);
avg_latency_2_sen(7) = avg_avg_latency_2_sen(4);
avg_latency_2_sen(8) = avg_avg_latency_2_sen(4);
avg_latency_2_sen(9) = avg_avg_latency_2_sen(5);
avg_latency_2_sen(10) = avg_avg_latency_2_sen(5);
avg_latency_2_sen(11) = avg_avg_latency_2_sen(6);
avg_latency_2_sen(12) = avg_avg_latency_2_sen(6);
avg_latency_2_sen(13) = avg_avg_latency_2_sen(7);
avg_latency_2_sen(14) = avg_avg_latency_2_sen(7);
avg_latency_2_ins(1) = avg_avg_latency_2_ins(1);
avg_latency_2_ins(2) = avg_avg_latency_2_ins(1);
avg_latency_2_ins(3) = avg_avg_latency_2_ins(2);
avg_latency_2_ins(4) = avg_avg_latency_2_ins(2);
avg_latency_2_ins(5) = avg_avg_latency_2_ins(3);
avg_latency_2_ins(6) = avg_avg_latency_2_ins(3);
avg_latency_2_ins(7) = avg_avg_latency_2_ins(4);
avg_latency_2_ins(8) = avg_avg_latency_2_ins(4);
avg_latency_2_ins(9) = avg_avg_latency_2_ins(5);
avg_latency_2_ins(10) = avg_avg_latency_2_ins(5);
avg_latency_2_ins(11) = avg_avg_latency_2_ins(6);
avg_latency_2_ins(12) = avg_avg_latency_2_ins(6);
avg_latency_2_ins(13) = avg_avg_latency_2_ins(7);
avg_latency_2_ins(14) = avg_avg_latency_2_ins(7);
worst_latency_1_ins = zeros(1, 14);
worst_latency_1_sen = zeros(1, 14);
worst_latency_2_ins = zeros(1, 14);
worst_latency_2_sen = zeros(1, 14);
worst_latency_1_sen(1) = avg_worst_latency_1_sen(1);
worst_latency_1_sen(2) = avg_worst_latency_1_sen(1);
worst_latency_1_sen(3) = avg_worst_latency_1_sen(2);
worst_latency_1_sen(4) = avg_worst_latency_1_sen(2);
worst_latency_1_sen(5) = avg_worst_latency_1_sen(3);
worst_latency_1_sen(6) = avg_worst_latency_1_sen(3);
worst_latency_1_sen(7) = avg_worst_latency_1_sen(4);
worst_latency_1_sen(8) = avg_worst_latency_1_sen(4);
worst_latency_1_sen(9) = avg_worst_latency_1_sen(5);
worst_latency_1_sen(10) = avg_worst_latency_1_sen(5);
worst_latency_1_sen(11) = avg_worst_latency_1_sen(6);
worst_latency_1_sen(12) = avg_worst_latency_1_sen(6);
worst_latency_1_sen(13) = avg_worst_latency_1_sen(7);
worst_latency_1_sen(14) = avg_worst_latency_1_sen(7);
worst_latency_1_ins(1) = avg_worst_latency_1_ins(1);
worst_latency_1_ins(2) = avg_worst_latency_1_ins(1);
worst_latency_1_ins(3) = avg_worst_latency_1_ins(2);
worst_latency_1_ins(4) = avg_worst_latency_1_ins(2);
worst_latency_1_ins(5) = avg_worst_latency_1_ins(3);
worst_latency_1_ins(6) = avg_worst_latency_1_ins(3);
worst_latency_1_ins(7) = avg_worst_latency_1_ins(4);
worst_latency_1_ins(8) = avg_worst_latency_1_ins(4);
worst_latency_1_ins(9) = avg_worst_latency_1_ins(5);
worst_latency_1_ins(10) = avg_worst_latency_1_ins(5);
worst_latency_1_ins(11) = avg_worst_latency_1_ins(6);
worst_latency_1_ins(12) = avg_worst_latency_1_ins(6);
worst_latency_1_ins(13) = avg_worst_latency_1_ins(7);
worst_latency_1_ins(14) = avg_worst_latency_1_ins(7);
worst_latency_2_sen(1) = avg_worst_latency_2_sen(1);
worst_latency_2_sen(2) = avg_worst_latency_2_sen(1);
worst_latency_2_sen(3) = avg_worst_latency_2_sen(2);
worst_latency_2_sen(4) = avg_worst_latency_2_sen(2);
worst_latency_2_sen(5) = avg_worst_latency_2_sen(3);
worst_latency_2_sen(6) = avg_worst_latency_2_sen(3);
worst_latency_2_sen(7) = avg_worst_latency_2_sen(4);
worst_latency_2_sen(8) = avg_worst_latency_2_sen(4);
worst_latency_2_sen(9) = avg_worst_latency_2_sen(5);
worst_latency_2_sen(10) = avg_worst_latency_2_sen(5);
worst_latency_2_sen(11) = avg_worst_latency_2_sen(6);
worst_latency_2_sen(12) = avg_worst_latency_2_sen(6);
worst_latency_2_sen(13) = avg_worst_latency_2_sen(7);
worst_latency_2_sen(14) = avg_worst_latency_2_sen(7);
worse_latency_2_ins(1) = avg_worse_latency_2_ins(1);
worse_latency_2_ins(2) = avg_worse_latency_2_ins(1);
worse_latency_2_ins(3) = avg_worse_latency_2_ins(1);
worse_latency_2_ins(4) = avg_worse_latency_2_ins(2);
worse_latency_2_ins(5) = avg_worse_latency_2_ins(2);
worse_latency_2_ins(6) = avg_worse_latency_2_ins(3);
worse_latency_2_ins(7) = avg_worse_latency_2_ins(3);
worse_latency_2_ins(8) = avg_worse_latency_2_ins(4);
worse_latency_2_ins(9) = avg_worse_latency_2_ins(4);
worse_latency_2_ins(10) = avg_worse_latency_2_ins(5);
worse_latency_2_ins(11) = avg_worse_latency_2_ins(5);
worse_latency_2_ins(12) = avg_worse_latency_2_ins(6);
worse_latency_2_ins(13) = avg_worse_latency_2_ins(7);
worse_latency_2_ins(14) = avg_worse_latency_2_ins(7);

figure
plot(time, sleep_ratio_1_sen, '-og', time, sleep_ratio_1_ins, '-sb', time, sleep_ratio_2_sen, '-+k', time, sleep_ratio_2_ins, '-*r')
xlabel('Time [s]')
ylabel('Ratio of time spent sleeping')
legend(['Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.'])
figure
plot(time, avg_power_1_sen, '-og', time, avg_power_1_ins, '-sb', time, avg_power_2_sen, '-+k', time, avg_power_2_ins, '-*r')
xlabel('Time [s]')
ylabel('Average power consumption [W]')
legend(['Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.'])
figure
semilogy(time, avg_latency_1_sen, '-og', time, avg_latency_1_ins, '-sb', time, avg_latency_2_sen, '-+k', time, avg_latency_2_ins, '-*r')
xlabel('Time [s]')
ylabel('Average additional latency [ms]')
legend(['Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.'])
figure
semilogy(time, worst_latency_1_sen, '-og', time, worst_latency_1_ins, '-sb', time, worst_latency_2_sen, '-+k', time, worst_latency_2_ins, '-*r')
xlabel('Time [s]')
ylabel('Worst case additional latency [ms]')
legend(['Tech.1 d.sens.t.', 'Tech.1 d.ins.t.', 'Tech.2 d.sens.t.', 'Tech.2 d.ins.t.'])
Appendix II

Schematics of the blocks inside the sleep controller

A2.1 Timer
A2.2 Average
A2.3 Detector
A2.4 Exponentially smoothed

The operation of the divider block is shown in the following picture:

where the value $2/(N+1)$ is rounded to the lower nearest power of two.
A2.5 Controller

Notes:

"**" = "00000000001001100010010110100000", corresponding to threshold1 = 12*\text{T}_{\text{wakeup}}

"***" = "000000001001100010010110100000", corresponding to threshold2 = 2*\text{delay}_{\text{max}}

"****" = "000000000010110111000110110000", corresponding to 6*\text{T}_{\text{wakeup}}

"*****" = "00000000001001100010010110100000", corresponding to \text{delay}_{\text{max}}

"******" = "100101010000001011110010000000", corresponding approximately to 5 s

"*******" = "000000000000011110010001001000", corresponding to \text{T}_{\text{wakeup}}
A2.6 Corrector
A2.7 Idle detector
A2.8 Delete detector
## Appendix III

### Experimental results

#### A3.1 Constant rate traffic

<table>
<thead>
<tr>
<th>CONFIGURATION 1.1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rate</strong>: 93.00595 frames/ms</td>
<td></td>
</tr>
<tr>
<td><strong>Support loop</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Min latency</strong></td>
<td>3,38 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>3,5 µs</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,47 µs</td>
</tr>
</tbody>
</table>

**Technique 1**

<table>
<thead>
<tr>
<th><strong>Delay-sensitive traffic</strong></th>
<th><strong>Delay-insensitive traffic</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,38 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>3,5 µs</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,47 µs</td>
</tr>
<tr>
<td>**% sleep time *****</td>
<td>0%</td>
</tr>
<tr>
<td>**% actual sleep time ******</td>
<td>0%</td>
</tr>
<tr>
<td>**% avg saved power *******</td>
<td>0%</td>
</tr>
</tbody>
</table>

**Technique 2**

<table>
<thead>
<tr>
<th><strong>Delay-sensitive traffic</strong></th>
<th><strong>Delay-insensitive traffic</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,38 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>3,52 µs</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,48 µs</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>0%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>0%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>0%</td>
</tr>
</tbody>
</table>

* The rate value expresses the rate experienced by each single emulated ONU.
** The term technique 1 refers to the design implementing the sleep mode technique proposed in chapter 3.
*** The percentages of sleep time are sampled assuming a zero wakeup time for the ONU transceiver.
**** The percentages of actual sleep time take into account the number of triggered sleep requests and are computed assuming a wakeup time of the ONU transceiver equal to 1 ms.
The percentages of average saved power are computed assuming that the power consumed by each ONU in the active state is equal to 10 W, while in the sleep state the required power is 2 W. This limits the maximum achievable percentage of average saved power to 80%.

The term technique 2 refers to the design employing the arithmetical average as prediction of the next frame inter-arrival time.

<table>
<thead>
<tr>
<th>CONFIGURATION 1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rate: 1/20 frames/ms</strong></td>
</tr>
<tr>
<td><strong>Support loop</strong></td>
</tr>
<tr>
<td>Min latency</td>
</tr>
<tr>
<td>Max latency</td>
</tr>
<tr>
<td>Avg latency</td>
</tr>
<tr>
<td><strong>Technique 1</strong></td>
</tr>
<tr>
<td>Delay-sensitive traffic</td>
</tr>
<tr>
<td>Min latency</td>
</tr>
<tr>
<td>Max latency</td>
</tr>
<tr>
<td>Avg latency</td>
</tr>
<tr>
<td>% sleep time</td>
</tr>
<tr>
<td>% actual sleep time</td>
</tr>
<tr>
<td>% avg saved power</td>
</tr>
<tr>
<td><strong>Technique 2</strong></td>
</tr>
<tr>
<td>Delay-sensitive traffic</td>
</tr>
<tr>
<td>Min latency</td>
</tr>
<tr>
<td>Max latency</td>
</tr>
<tr>
<td>Avg latency</td>
</tr>
<tr>
<td>% sleep time</td>
</tr>
<tr>
<td>% actual sleep time</td>
</tr>
<tr>
<td>% avg saved power</td>
</tr>
</tbody>
</table>
## Configuration 1.3

**Rate:** 1/300 frames/ms

### Support Loop

<table>
<thead>
<tr>
<th>Latency Type</th>
<th>Min Latency</th>
<th>Max Latency</th>
<th>Avg Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.34 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>3.48 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3.41 µs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Technique 1

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Delay-Sensitive Traffic</th>
<th>Delay-Insensitive Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.34 µs</td>
<td>3.34 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td>18.75 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>8.78 ms</td>
<td>8.34 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99.75%</td>
<td>99.57%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>89.44%</td>
<td>96.58%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>71.55%</td>
<td>77.27%</td>
</tr>
</tbody>
</table>

### Technique 2

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Delay-Sensitive Traffic</th>
<th>Delay-Insensitive Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.34 µs</td>
<td>3.34 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td>18.74 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>8.88 ms</td>
<td>8.9 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99.48%</td>
<td>99.76%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>89.2%</td>
<td>96.53%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>71.36%</td>
<td>77.23%</td>
</tr>
</tbody>
</table>
**A3.2 Bursty traffic**

**CONFIGURATION 2.1**

<table>
<thead>
<tr>
<th>Bursts: 10 frames at 1 frames/ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle period: 1 s</td>
</tr>
</tbody>
</table>

**Support loop**

| Min latency | 3,32 µs |
| Max latency | 3,48 µs |
| Avg latency | 3,4 µs |

**Technique 1**

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min latency</td>
<td>3,32 µs</td>
</tr>
<tr>
<td>Max latency</td>
<td>8,99 ms</td>
</tr>
<tr>
<td>Avg latency</td>
<td>4,46 ms</td>
</tr>
<tr>
<td>% sleep time</td>
<td>0,98%</td>
</tr>
<tr>
<td>% actual sleep time</td>
<td>0,89%</td>
</tr>
<tr>
<td>% avg saved power</td>
<td>0,71%</td>
</tr>
</tbody>
</table>

| Min latency | 3,32 µs |
| Max latency | 499,47 ms |
| Avg latency | 467,82 ms |
| % sleep time | 49,01% |
| % actual sleep time | 48,91% |
| % avg saved power | 39,13% |

**Technique 2**

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min latency</td>
<td>3,32 µs</td>
</tr>
<tr>
<td>Max latency</td>
<td>8,99 ms</td>
</tr>
<tr>
<td>Avg latency</td>
<td>4,46 ms</td>
</tr>
<tr>
<td>% sleep time</td>
<td>0,99%</td>
</tr>
<tr>
<td>% actual sleep time</td>
<td>0,89%</td>
</tr>
<tr>
<td>% avg saved power</td>
<td>0,71%</td>
</tr>
</tbody>
</table>

| Min latency | 3,32 µs |
| Max latency | 62 ms |
| Avg latency | 52,16 ms |
| % sleep time | 6,15% |
| % actual sleep time | 6,05% |
| % avg saved power | 4,84% |
## CONFIGURATION 2.2

### Bursts: 25 frames at 1 frames/ms

### Idle period: 1 s

### Support loop

<table>
<thead>
<tr>
<th></th>
<th>Min latency</th>
<th>Max latency</th>
<th>Avg latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td>3,48 µs</td>
<td>3,4 µs</td>
</tr>
</tbody>
</table>

### Technique 1

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Min latency</th>
<th>Max latency</th>
<th>Avg latency</th>
<th>% sleep time</th>
<th>% actual sleep time</th>
<th>% avg saved power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay-sensitive traffic</td>
<td>3,32 µs</td>
<td>8,99 ms</td>
<td>1,79 ms</td>
<td>0,97%</td>
<td>0,87%</td>
<td>0,7%</td>
</tr>
<tr>
<td>Delay-insensitive traffic</td>
<td>Min latency</td>
<td>Max latency</td>
<td>Avg latency</td>
<td>% sleep time</td>
<td>% actual sleep time</td>
<td>% avg saved power</td>
</tr>
<tr>
<td></td>
<td>3,32 µs</td>
<td>499,47 ms</td>
<td>441,71 ms</td>
<td>48,24%</td>
<td>48,14%</td>
<td>38,51%</td>
</tr>
</tbody>
</table>

### Technique 2

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Min latency</th>
<th>Max latency</th>
<th>Avg latency</th>
<th>% sleep time</th>
<th>% actual sleep time</th>
<th>% avg saved power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay-sensitive traffic</td>
<td>3,32 µs</td>
<td>8,99 ms</td>
<td>1,79 ms</td>
<td>0,98%</td>
<td>0,88%</td>
<td>0,71%</td>
</tr>
<tr>
<td>Delay-insensitive traffic</td>
<td>Min latency</td>
<td>Max latency</td>
<td>Avg latency</td>
<td>% sleep time</td>
<td>% actual sleep time</td>
<td>% avg saved power</td>
</tr>
<tr>
<td></td>
<td>3,32 µs</td>
<td>62 ms</td>
<td>48,38 ms</td>
<td>6,11%</td>
<td>6,01%</td>
<td>4,81%</td>
</tr>
</tbody>
</table>
**CONFIGURATION 2.3**

**Bursts:** 100 frames at 1 frames/ms  
**Idle period:** 1 s

**Support loop**
- **Min latency:** 3,32 µs  
- **Max latency:** 3,48 µs  
- **Avg latency:** 3,4 µs

**Technique 1**

<table>
<thead>
<tr>
<th></th>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>8,99 ms</td>
<td>449,47 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>0,45 ms</td>
<td>409,11 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>0,91%</td>
<td>44,83%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>0,82%</td>
<td>44,74%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>0,66%</td>
<td>35,79%</td>
</tr>
</tbody>
</table>

**Technique 2**

<table>
<thead>
<tr>
<th></th>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>8,99 ms</td>
<td>62 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>0,45 ms</td>
<td>19,37 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>0,91%</td>
<td>5,63%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>0,82%</td>
<td>5,54%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>0,65%</td>
<td>4,43%</td>
</tr>
</tbody>
</table>
## A3.3 Double-rate traffic

### CONFIGURATION 3.1

**Rate 1:** 10 frames at 1 frames/ms  
**Rate 2:** 10 frames at 1/500 frames/ms

**Support loop**

<table>
<thead>
<tr>
<th></th>
<th>Min latency</th>
<th>Max latency</th>
<th>Avg latency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td></td>
<td>3.48 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td></td>
<td></td>
<td>3.4 µs</td>
</tr>
</tbody>
</table>

**Technique 1**

<table>
<thead>
<tr>
<th></th>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td>3.32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td>217.81 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>6.11 ms</td>
<td>34.81 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99.11%</td>
<td>99.27%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>89.02%</td>
<td>96.08%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>71.21%</td>
<td>76.86%</td>
</tr>
</tbody>
</table>

**Technique 2**

<table>
<thead>
<tr>
<th></th>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td>3.32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td>249 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>6.04 ms</td>
<td>43.18 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>89.21%</td>
<td>89.47%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>80.13%</td>
<td>84.21%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>64.1%</td>
<td>67.37%</td>
</tr>
</tbody>
</table>
### CONFIGURATION 3.2

**Rate 1:** 20 frames at 1/10 frames/ms  
**Rate 2:** 20 frames at 1/30 frames/ms  

**Support loop**

<table>
<thead>
<tr>
<th>Min latency</th>
<th>3,32 µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max latency</td>
<td>3,48 µs</td>
</tr>
<tr>
<td>Avg latency</td>
<td>3,41 µs</td>
</tr>
</tbody>
</table>

**Technique 1**

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>8,41 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,96%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>85,16%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>68,13%</td>
</tr>
</tbody>
</table>

**Technique 2**

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>4,85 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>81,1%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>69,89%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>55,91%</td>
</tr>
</tbody>
</table>
### CONFIGURATION 3.3

Rate 1: 10 frames at 1/20 frames/ms  
Rate 2: 10 frames at 1/25 frames/ms

#### Support loop

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>3,48 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,4 µs</td>
<td></td>
</tr>
</tbody>
</table>

#### Technique 1

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td>5,9 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>6,73 ms</td>
<td>3,42 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,96%</td>
<td>99,96%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>86,63%</td>
<td>84,41%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>69,31%</td>
<td>67,53%</td>
</tr>
</tbody>
</table>

#### Technique 2

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td>5,81 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>7,02 ms</td>
<td>3,65 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,84%</td>
<td>99,83%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>86,53%</td>
<td>84,31%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>69,23%</td>
<td>67,45%</td>
</tr>
</tbody>
</table>
# A3.4 Bursty traffic with superimposed background traffic

## CONFIGURATION 4.1

- **Bursts:** 15 frames at 1/25 frames/ms
- **Idle period:** 2 s
- **Background rate:** 1/150 frames/ms

### Support loop

<table>
<thead>
<tr>
<th>Latency Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min latency</td>
<td>3,32 µs</td>
</tr>
<tr>
<td>Max latency</td>
<td>3,48 µs</td>
</tr>
<tr>
<td>Avg latency</td>
<td>3,41 µs</td>
</tr>
</tbody>
</table>

### Technique 1

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Min latency</th>
<th>Max latency</th>
<th>Avg latency</th>
<th>% sleep time</th>
<th>% actual sleep time</th>
<th>% avg saved power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay-sensitive traffic</td>
<td>3,32 µs</td>
<td>10 ms</td>
<td>4,72 ms</td>
<td>99,93%</td>
<td>87,8%</td>
<td>70,24%</td>
</tr>
<tr>
<td>Delay-insensitive traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min latency</td>
<td>3,32 µs</td>
<td>Max latency</td>
<td>Avg latency</td>
<td>% sleep time</td>
<td>% actual sleep time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% avg saved power</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Technique 2

<table>
<thead>
<tr>
<th>Traffic Type</th>
<th>Min latency</th>
<th>Max latency</th>
<th>Avg latency</th>
<th>% sleep time</th>
<th>% actual sleep time</th>
<th>% avg saved power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay-sensitive traffic</td>
<td>3,32 µs</td>
<td>10 ms</td>
<td>5,09 ms</td>
<td>99,85%</td>
<td>87,71%</td>
<td>70,17%</td>
</tr>
<tr>
<td>Delay-insensitive traffic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min latency</td>
<td>3,32 µs</td>
<td>Max latency</td>
<td>Avg latency</td>
<td>% sleep time</td>
<td>% actual sleep time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>% avg saved power</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>67,2%</td>
</tr>
</tbody>
</table>
## CONFIGURATION 4.2

**Bursts:** 20 frames at 1/20 frames/ms

**Idle period:** 2 s

**Background rate:** 1/100 frames/ms

### Support loop

- **Min latency:** 3.32 µs
- **Max latency:** 3.48 µs
- **Avg latency:** 3.4 µs

### Technique 1

<table>
<thead>
<tr>
<th></th>
<th>Delay-sensitive traffic</th>
<th></th>
<th>Delay-insensitive traffic</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>8.63 ms</td>
<td><strong>Max latency</strong></td>
<td>7.62 ms</td>
<td></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>4.91 ms</td>
<td><strong>Avg latency</strong></td>
<td>1.34 ms</td>
<td></td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99.96%</td>
<td><strong>% sleep time</strong></td>
<td>99.94%</td>
<td></td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>85.06%</td>
<td><strong>% actual sleep time</strong></td>
<td>84.44%</td>
<td></td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>68.05%</td>
<td><strong>% avg saved power</strong></td>
<td>67.55%</td>
<td></td>
</tr>
</tbody>
</table>

### Technique 2

<table>
<thead>
<tr>
<th></th>
<th>Delay-sensitive traffic</th>
<th></th>
<th>Delay-insensitive traffic</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td><strong>Min latency</strong></td>
<td>3.32 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>9.72 ms</td>
<td><strong>Max latency</strong></td>
<td>9 ms</td>
<td></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>5.06 ms</td>
<td><strong>Avg latency</strong></td>
<td>2.31 ms</td>
<td></td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>98.43%</td>
<td><strong>% sleep time</strong></td>
<td>99.67%</td>
<td></td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>83.99%</td>
<td><strong>% actual sleep time</strong></td>
<td>83.36%</td>
<td></td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>67.19%</td>
<td><strong>% avg saved power</strong></td>
<td>66.69%</td>
<td></td>
</tr>
</tbody>
</table>
## CONFIGURATION 4.3

**Bursts:** 25 frames at 1/15 frames/ms  
**Idle period:** 2 s  
**Background rate:** 1/80 frames/ms

### Support loop

| Min latency | 3,32 µs |
| Max latency | 3,48 µs |
| Avg latency | 3,4 µs |

### Technique 1

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>7,3 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,84 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,92%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>80,26%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>64,21%</td>
</tr>
</tbody>
</table>

### Technique 2

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>9,72 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>5,06 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>98,43%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>83,99%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>67,19%</td>
</tr>
</tbody>
</table>
### A3.5 Traffic with superimposed rates

#### CONFIGURATION 5.1

<table>
<thead>
<tr>
<th>Rates: 1/53 frames/ms, 1/73 frames/ms, 1/101 frames/ms, 1/127 frames/ms, 1/151 frames/ms</th>
</tr>
</thead>
</table>

**Support loop**

| Min latency   | 3,32 µs |
| Max latency   | 3,48 µs |
| Avg latency   | 3,4 µs  |

**Technique 1**

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>8,81 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,96 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,92%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>86,45%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>69,16%</td>
</tr>
</tbody>
</table>

**Technique 2**

<table>
<thead>
<tr>
<th>Delay-sensitive traffic</th>
<th>Delay-insensitive traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>9,91 ms</td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>4,04 ms</td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>96,72%</td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>83,4%</td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>66,72%</td>
</tr>
</tbody>
</table>
### CONFIGURATION 5.2

**Rates:** 1/101 frames/ms, 1/127 frames/ms, 1/151 frames/ms, 1/179 frames/ms, 1/199 frames/ms

<table>
<thead>
<tr>
<th>Support loop</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>3,48 µs</td>
<td></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>3,4 µs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Technique 1</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay-sensitive traffic</strong></td>
<td></td>
<td><strong>Delay-insensitive traffic</strong></td>
</tr>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td><strong>Min latency</strong></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td><strong>Max latency</strong></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>4,45 ms</td>
<td><strong>Avg latency</strong></td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,95%</td>
<td><strong>% sleep time</strong></td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>88,56%</td>
<td><strong>% actual sleep time</strong></td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>70,85%</td>
<td><strong>% avg saved power</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Technique 2</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delay-sensitive traffic</strong></td>
<td></td>
<td><strong>Delay-insensitive traffic</strong></td>
</tr>
<tr>
<td><strong>Min latency</strong></td>
<td>3,32 µs</td>
<td><strong>Min latency</strong></td>
</tr>
<tr>
<td><strong>Max latency</strong></td>
<td>10 ms</td>
<td><strong>Max latency</strong></td>
</tr>
<tr>
<td><strong>Avg latency</strong></td>
<td>4,73 ms</td>
<td><strong>Avg latency</strong></td>
</tr>
<tr>
<td><strong>% sleep time</strong></td>
<td>99,85%</td>
<td><strong>% sleep time</strong></td>
</tr>
<tr>
<td><strong>% actual sleep time</strong></td>
<td>88,46%</td>
<td><strong>% actual sleep time</strong></td>
</tr>
<tr>
<td><strong>% avg saved power</strong></td>
<td>70,77%</td>
<td><strong>% avg saved power</strong></td>
</tr>
</tbody>
</table>
**CONFIGURATION 5.3**

Rates: 1/181 frames/ms, 1/193 frames/ms, 1/199 frames/ms, 1/223 frames/ms, 1/229 frames/ms

**Support loop**

<table>
<thead>
<tr>
<th>Latency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min latency</td>
<td>3,32 µs</td>
</tr>
<tr>
<td>Max latency</td>
<td>3,48 µs</td>
</tr>
<tr>
<td>Avg latency</td>
<td>3,41 µs</td>
</tr>
</tbody>
</table>

**Technique 1**

<table>
<thead>
<tr>
<th>Traffic Description</th>
<th>Min latency</th>
<th>Max latency</th>
<th>Average latency</th>
<th>% Sleep Time</th>
<th>% Actual Sleep Time</th>
<th>% Avg Saved Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay-sensitive</td>
<td>3,32 µs</td>
<td>10 ms</td>
<td>4,47 ms</td>
<td>99,94%</td>
<td>88,92%</td>
<td>71,13%</td>
</tr>
<tr>
<td>Delay-insensitive</td>
<td>3,32 µs</td>
<td>42,25 ms</td>
<td>4,38 ms</td>
<td>99,88%</td>
<td>86,96%</td>
<td>69,57%</td>
</tr>
</tbody>
</table>

**Technique 2**

<table>
<thead>
<tr>
<th>Traffic Description</th>
<th>Min latency</th>
<th>Max latency</th>
<th>Average latency</th>
<th>% Sleep Time</th>
<th>% Actual Sleep Time</th>
<th>% Avg Saved Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay-sensitive</td>
<td>3,32 µs</td>
<td>10 ms</td>
<td>4,75 ms</td>
<td>99,81%</td>
<td>88,8%</td>
<td>71,04%</td>
</tr>
<tr>
<td>Delay-insensitive</td>
<td>3,32 µs</td>
<td>24,38 ms</td>
<td>4,96 ms</td>
<td>99,17%</td>
<td>86,24%</td>
<td>68,99%</td>
</tr>
</tbody>
</table>