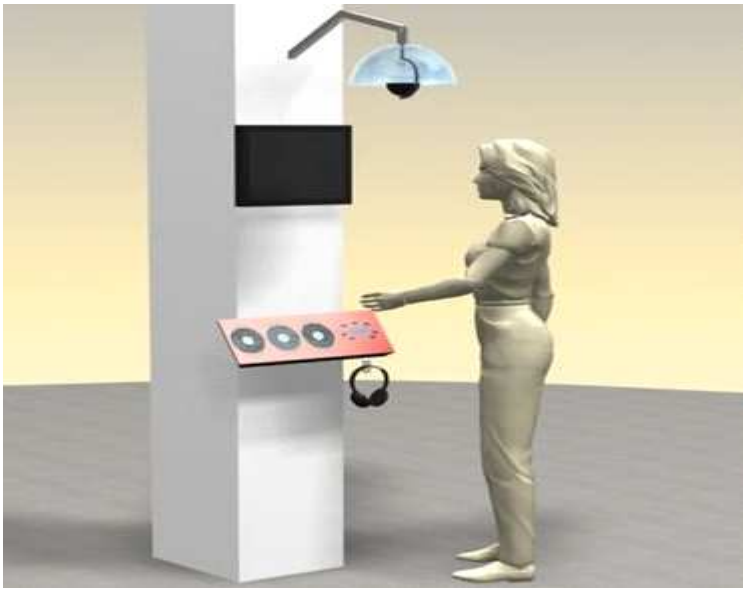
UNIVERSIDADE DA BEIRA INTERIOR
Covilhã | Portugal

Energy-Aware Medium Access Control Protocols for Wireless Sensors Network Applications



Norberto José Gil Barroca

Dissertation submitted for obtaining the degree of
Master in Electrical and Computer Engineering

August 2009

Supervisors: Fernando José da Silva Velez

António Sérgio Portela Fernandes Lebres

August 2009

To my Family

Acknowledgements

To all that believed in me.

This work was only possible due to the encouragement and support of many people, who drove me along this long journey. Among them, I would like to express my deeply gratitude to Prof. Fernando José da Silva Velez and Prof. António Sérgio Portela Fernandes Lebres my thesis mentors, for the friendship and whose technical excellence and unconditional support where the key to achieve the goals for this project. The meetings during the project revealed to be essential in the conception of our formulations, and his vision help me to choose the correct path when sometimes it seems to be sinuous. In the Smart-Clothing project, I'm thankful to all the time they spent with explanations and orientations, for his availability, guidelines, advising and constant support, which were essential to the work.

To all the peoples that make possible the Smart-Clothing project and to Ana Paula our patient during the test made in the project.

To my friends at IT-Covilhã, Luís Borges, João Ferro, João Oliveira, Jorge Tavares and Orlando Cabral, I would like to thank all their understanding and availability, which were decisive for the conclusion of this work. Thank you for your friendship.

I also want to thank all my friends at CFD-Covilhã, especially Helder Miguel, for his particular humour sense, support and encouragement.

Special thank to a special and unique person, Sara Gomes, for her love, attention, support, friendship, encouragement and mainly for her patient along the last months.

At last but not least, I would like to thank my family, especially my parents, and my sister. I am very grateful for their unconditional love, care, understanding, and support, which kept me going in the hardest times.

Norberto José Gil Barroca

Abstract

The main purpose of this thesis was to investigate energy efficient Medium Access Control (MAC) protocols designed to extend the lifetime of a wireless sensor network application, such as tracking, environment monitoring, home security, patient monitoring, e.g., foetal monitoring in the last weeks of pregnancy. From the perspective of communication protocols, energy efficiency is one of the most important issues, and can be addressed at each layer of the protocol stack; however, our research only focuses on the medium access control (MAC) layer. An energy efficient MAC protocol was designed based on modifications and optimisations for a synchronized power saving Sensor MAC (S-MAC) protocol, which has three important components: periodic listen and sleep, collision and overhearing avoidance and message passing. The *Sensor Block Acknowledgement (SBACK)* MAC protocol is proposed, which combines contention-based, scheduling-based and block acknowledgement-based schemes to achieve energy efficiency. In *SBACK*, the use of ACK control packets is reduced since it will not have an ACK packet for every DATA packet sent; instead, one special packet called *Block ACK Response* will be used at the end of the transmission of all data packets. This packet informs the sender of how many packets were received by the receiver, reducing the number of ACK control packets we intended to reduce the power consumption for the nodes. Hence more useful data packets can be transmitted. A comparison study between SBACK and S-MAC protocol is also performed.

Considering 0% of packet losses, SBACK decreases the energy consumption when directly compared with S-MAC, we will have always a decrease of energy consumption.

Three different transceivers will be used and considering a packet loss of 10% we will have a decrease of energy consumption between 10% and 0.1% depending on the transceiver.

When there are no retransmissions of packets, SBACK only achieve worst performance when the number of fragments is less than 12, after that the decrease of average delay increases with the increase of the fragments sent.

When 10% of the packets need retransmission only for the TR1000 transceiver worst results occurs in terms of energy waste, all other transceivers (CC2420 and AT86RF230) achieve better results.

In terms of delay if we need to retransmit more than 10 packets the SBACK protocol always achieves better performance when comparing with the other MAC protocols that uses ACK.

Keywords

Wireless Sensor Networks (WSN), Medium Access Control (MAC), energy efficiency, Sensor Block Acknowledgement (SBACK).

Resumo

O objectivo principal desta tese foi investigar os protocolos de controlo de acesso ao meio eficientes do ponto de vista energético com o objectivo de aumentar o tempo de vida para aplicações de redes de sensores, tais como monitorizar pessoas, monitorizar o meio ambiente, aplicações envolvendo segurança em casa, vigilância de pacientes, ou seja monitorizar o feto nas últimas semanas de gravidez. Na perspectiva dos protocolos de comunicação, a eficiência energética é um dos tópicos mais importantes, e pode ser tratado em cada nível da pilha protocolar. No entanto, a nossa investigação apenas se foca na camada de controlo de acesso ao meio (MAC). Propôs-se um protocolo MAC com base em modificações e optimizações de um protocolo que utiliza sincronização para poupar energia e que tem três componentes: escuta periodicamente o meio indo para o estado de adormecido, evita colisões e "overhearing." Cada no pode ser utilizado como um nó para enviar mensagens para outros nós. O protocolo Sensor Block Acknowledgment (SBACK) combina técnicas baseadas em contenção, sincronização e esquemas de acknowledgment de modo a obter eficiência energética. No SBACK, o uso de pacotes de ACK de controlo é reduzido, dado que não iremos ter um pacote de ACK por cada pacote de dados enviados; em vez disso, um pacote Block ACK Response especial vai ser usado no fim da transmissão de todos os pacotes de dados. Este pacote informa o emissor de quantos pacotes foram recebidos pelo receptor. Reduz-se ainda o número de pacotes de ACK, reduzindo-se em geral o consumo de potência de cada nó. Deste modo, podem ser transmitidos mais pacotes úteis de dados. Efectuou-se um estudo comparativo entre os protocolos SBACK e S-MAC. Considerando 0% de perda de pacotes, o SBACK diminui o consumo de energia quando directamente comparado com o S-MAC. Então, iremos ter sempre uma diminuição do consumo de energia. Podem-se utilizar três diferentes rádio "transceivers". Considerando 10% de perda de pacotes, iremos ter um desperdício do consumo de energia entre 10% e 0.1%, dependendo do "transceiver". Quando não existem retransmissões de pacotes, o SBACK apenas atinge pior desempenho quando o número de fragmentos é menor do que 12, a partir desse valor a diminuição média do atraso aumenta com o aumento de pacotes enviados. Quando a percentagem de pacotes precisam que de retransmissão é 10% apenas no rádio TR1000 ocorrem os piores resultados. Em termos de desperdício de energia, todos os outros rádios (CC2420 e AT86RF230) têm melhores desempenhos. Em termos de atraso, se precisamos de retransmitir mais de 10 pacotes o protocolo SBACK atinge sempre um melhor desempenho quando comparado com outros protocolos de controlo de acesso ao meio que usam ACK.

Palavras-chave

Redes de sensores sem fios, protocolos de controlo de acesso ao meio, eficiência energética, Sensor Block Acknowledgement (SBACK).

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACK	Acknowledgment
ADC	Analog-to-digital Converter
ADDBA	Add Block Acknowledgment
APS	Application Sub Layer
BE	Backoff Exponent
BI	Beacon Interval
BO	Beacon Order
BP	Backoff Period
BPSK	Binary Phase shift Keying
BSN	Beacon Sequence Number
CAP	Contention Access Period
CCA	Clear Channel Assessment
CFP	Contention-Free Period
CMR	Centralized Management of Resources
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSS	Chirp Spread Spectrum
CTS	Clear-to-Send
CW	Contention Window
DCF	Distribute Coordination Function
DELBA	Delete Block Acknowledgment
DIFS	Distributed Interframe Space
DLL	Data Link Layer
DLL	Data Link Layer
DS	Data-Send
DSN	Data Sequence Number
DSN	Data Sequence Number
DSSS	Direct Sequence Spread Spectrum
ECC	Error Correcting Codes
ED	Energy Detection
FFD	Full Function Device
FRTS	Future Request-to-Send
GTSs	Guaranteed Time Slots
IC	Integrated Circuits
IEEE	Electrical and Electronic Engineers
ISM	Industrial, Scientific and medical
IT	Information Technology
LIFS	Long IFS
LLC	Logical Link Control
LPL	Low Power Listening
LQI	Link Quality Indication
LR-WBANs	Low-rate Wireless Personal Area Networks

MCU	Micro-Controller Unit
MEMS	Micro-electro-mechanical systems
MFR	MAC Footer
MHR	MAC Header
MSB	Most Significant Bit
MSDU	MAC Service Data
NAV	Network Allocation Vector
NB	Number of Packets
NBC	Nuclear, Biological and Chemical
NWK	Network layer
O-QPSK	Offset Quadrature Shift Keying
OS	Operating systems
OSI	Open Systems Interconnection
PAN	Personal Area Network
PC	Personal Computer
PCF	Point Coordination Function
PHR	PHY Header
PHY	Physical Layer
POS	Personal Operating Space
PPDU	PHY Protocol Data Unit
PSDU	PHY Layer Service Data Unit
QoS	Quality of Service
RAM	Random Access memory
RSSI	Received Signal Strength Indicator
RTS	Request-to-Send
SBACK	Sensor Block Acknowledgment
SD	Superframe Duration
SFD	Start of Frame Delimiter
SHR	Synchronisation Header
SIFS	Short IFS
SNP	Sensor Network Platforms
SNP	Sensor Network Platforms
SO	Superframe Order
SRAM	Static Random Access Memory
SRAM	Static Random Access Memory
UWB	Ultra-wide band
WAN	Wide Area Network
WBANs	Wireless Body Area Networks
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WSAN	Wireless Sensor Networks and Actuator Networks
WSN	Wireless Sensor Network
ZDO	ZigBee® Device Objects
ZDP	ZigBee® Device Profile

List of Symbols

$P_{transmit}$	Power to transmit a data packet
$P_{transmit\ ACK}$	Power to receive ACK packet
T_{cp}	Time to transmit control packet
$T_{receive}$	Time to receive packet
$T_{receive\ ACK}$	Time to receive ACK packet
T_{sleep}	Time to sleep
T_{Tdp}	Time to transmit a data packet
$T_{transmit}$	Time to transmit a data packet

List of Software

OMNeT++ Simutaltor	OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, with an Eclipse-based IDE and a graphical runtime environment. Domain-specific functionality (support for simulation of communication networks, queuing networks, performance evaluation, etc.) is provided by model frameworks, developed as independent projects. There are extensions for real-time simulation, network emulation, alternative programming languages (Java, C#), database integration, SystemC integration, and several other functions.
Microsoft Excel 2007	It is an electronic spreadsheet program. It is useful to record, to analyse, and to show information. It is also helpful in computing formulas.
Microsoft Office Visio 2007	Visio is a tool for creating all manner of business diagrams, network layouts, storyboards and site flows, software entity relationship diagram, etc.
Microsoft Word 2007	Microsoft Word provides powerful tools for creating and sharing professional word processing documents.
Microsoft Visual Studio 2008	Microsoft Visual Studio is an Integrated Development Environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services

Chapter 1

Introduction

The primary objective of this chapter is the brief description of the MSc thesis and describing the motivations for the work. The approach for the work is also presented. At the end of the chapter, the structure of the thesis is presented.

1.1 Overview

The recent advances in the last years in the field of microelectronic circuits caused an increase on the interest in the development of Wireless Sensor Networks and Actuator Networks (WSAN). This type of networks can be deployed in many scenarios such as factory monitoring, healthcare, environment monitoring, logistics, location of persons in commercial buildings, monitoring of buildings structures, precision agriculture and many other applications [VDMC08] [ROWR04].

Usually a Wireless Sensor Network (WSN) is composed by a large number of battery operated nodes that can be deployed in order to create an infrastructure of sensing, computing and which has communication elements, so that all the phenomena's in a specified environment can be observed and measured.

Recent advances in Integrated Circuits (IC) and in low power circuits have reduced the total amount of power needed to operate the sensor nodes. Besides, the recent advances in technology have an important role by making the sensor nodes used in the WSN as collection of compact size and relatively inexpensive elements that work together in order to achieve a common goal. Because this type of networks are battery operated, one of the most important issues is the energy conservation and the possibility of using rechargeable batteries could be considered as a secondary power source. Therefore, in the context of WSN, other primary source must be used to charge them and one option is to connect the device to a power grid such as a solar panel. However, this solution is not the best one, not only because sometimes is impossible to achieve this, but also because this type of solution could increase the size of the wireless sensor nodes and its complexity, resulting in expensive sensor nodes.

One of the biggest power consumers in the sensor nodes is the radio transceiver, and it has a decisive influence in network lifetime, therefore to achieve energy-efficiency the medium access control (MAC) protocol is used to determinate and the change the operation mode of the radio.

The basic operations modes of the transceiver with different power levels are: sleeping, receiving, transmit and listening. Typically the power consumption wasted with the "listening mode" is the same as receiving. However, transmitting is more "expensive" in terms of energy consumption and it is possible to find situations when receive costs have the same order of magnitude as transmitting costs. All these factors have an important role when we need to design a new MAC protocol, so this way we can derive the following energy problems and design goals [YeHE02a]:

Collisions: When designing a MAC protocol one fundamental aspect is the avoidance of packet collisions, as they waste energy. So, the primary objective is to avoid that two or more interfering nodes transmit data or any kind of packets at the same time. If a transmission with sufficient signal strength interferes with a data packet being sent, the data will be corrupted at the receiving node, leading to an inadequate waste of power. One solution to avoid this problem (that may recover corrupted data) is to use error correcting codes (ECC). However, ECCs add transmission overhead,

which may be contrary to the goal of reducing radio transmit time.

Overhearing: Overhearing loss refers to energy wasted by a node that has its radio in receive mode while a packet is being transmitted to another node. Most of the WSN MAC protocols reduce overhearing by trying to ensure that a node is only awake when there is traffic being transmitted to it. One way to prevent overhearing is to ignore packets sent to other nodes after hearing an RTS/CTS exchange. This is accomplished because unicast frames (e.g. RTS e CTS frames) have one source and one destination node. However since the wireless medium is a broadcast one, as described in the Anglo-Saxon literature, it is possible that a node in the receive mode pick up packets that are destined to other nodes. After overhearing the RTS and CTS, nodes will go to sleep during the network allocation vector (NAV) period. As shown by the authors from [YeHE04] overhearing avoidance can reduce significantly the amount of energy waste. However, there are situations that make overhearing desirable, e.g., when collecting neighbourhood information or estimating the current traffic load for management purposes.

Protocol overhead: The wide range of MAC protocols use control packets. This type of packets can be received by all nodes within radio range of the sender, resulting in power drain in a potentially large number of nodes. When nodes are required to stay awake in order to receive control packets, the battery life can be reduced significantly. Sending and receiving control frames like, RTS/CTS or request packets consumes energy. Therefore, less useful data packets can be transmitted.

Idle listening: As described in [YeHE02a] the last source of inefficiency is idle listening. This happens when a node that is in idle state is ready to receive possible traffic but is not currently receiving anything. To avoid this type of problems one solution is switching off the transceiver. However since changing the radio mode also costs energy, their changing frequency should be kept at “reasonable” levels.

The energy problems and design goals shown that MAC protocols focus on minimize energy consumption, support good scalability and be self configurable, while the traditional MAC protocols are design to maximize the packet throughput, minimize latency and provide fairness.

1.2 Motivation and Approach

The diverse application domains of WSANs (e.g., habitat, ambient, home and factory monitoring) have an impact on many parameters (e.g., deployment strategy, node mobility, available resources, topology, sensor coverage, quality of service network size, lifetime and connectivity). These types of networks are composed by a large set of nodes, scattered over the environment and responsible to interact with the physical world, in order to collect the data needed for the monitoring/control of a predefined area/region. The WSN nodes shown in the Figure 1.1 will work in a multi-hop topology,

where the primary objective is to deliver the data to the control station (also referred as sink).

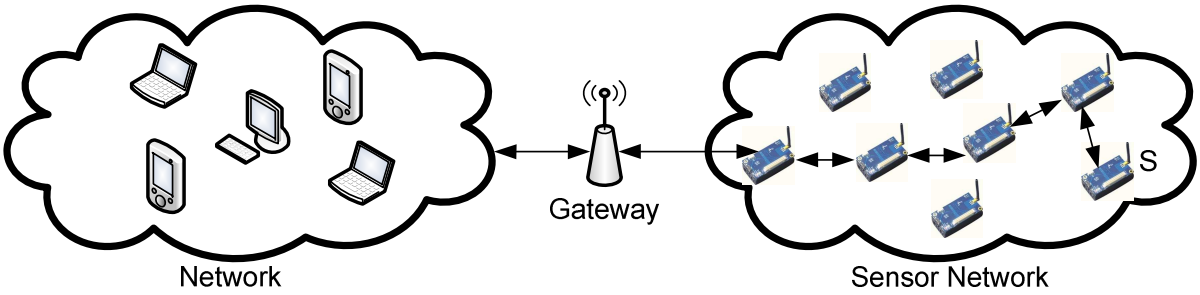


Figure 1.1. Typical topology of a WSN.

As we can see in Figure 1.1, a WSN node acts as:

- A data originator because, it has capabilities such as sensing, data processing and wireless communications, resulting in the production of information related with the sensed phenomenon by interacting with the physical environment and collecting relevant physical parameters. In this case a node will be deployed inside the phenomena or very close to it;
- A data router, that transmits data gathering by neighbour nodes and then forward the information to a control station, which will be responsible to process and analyses the data collected from the different sensors/nodes in the network.

In the Anglo-Saxon literature a rich diversity of node hardware can be found, from small capability nodes, like MICAz [MICA08] or Scatterweb [SCAT08] to more powerful ones like iMote2 [IMOTE08] or Sun Spots [SUNS08]. This hardware heterogeneity enables applications that make a better use of the surrounding resources.

The very low energy consumption is a mandatory characteristic of WSNs because there are strong limitations to power supply in WSN devices. Furthermore, devices must cooperate among themselves in order to efficiently exchange data by using multi-hop in order to save energy (e.g., by appropriately randomising the data while performing multicast through the network towards a given sink). As a consequence, at the level of each individual WSN device, an efficient management of transmission, reception, and sleep modes of operation should be implemented. WSN devices must remain in sleep mode whenever they do not have to receive/process/transmit data packets. Efficient energy-aware routing protocols will also be crucial to ensure error free robust packet delivery while minimizing power consumption.

From the wireless networks applications perspective, the communication needs for wearable technologies comprise two main aspects: one are the WBANs (Wireless Body Area Networks), Figure 1.2, in order to guarantee the communication among sensors placed in different parts of the human body and second are the WSNs (wireless sensor networks) for communication among different patients. In the context of the Smart-Clothing for Health Monitoring and Sport Applications (Smart-

Clothing project), an iCentro project approved by CCDR-C with FEDER funding (that includes IT-Covilhã team, where the authors was involved), an hybrid communication system was created to deliver the data from the WBAN which was installed in a pregnant woman. This research was one of the first steps that motivated the study of new techniques applied to energy efficient MAC protocols used in WSAN.

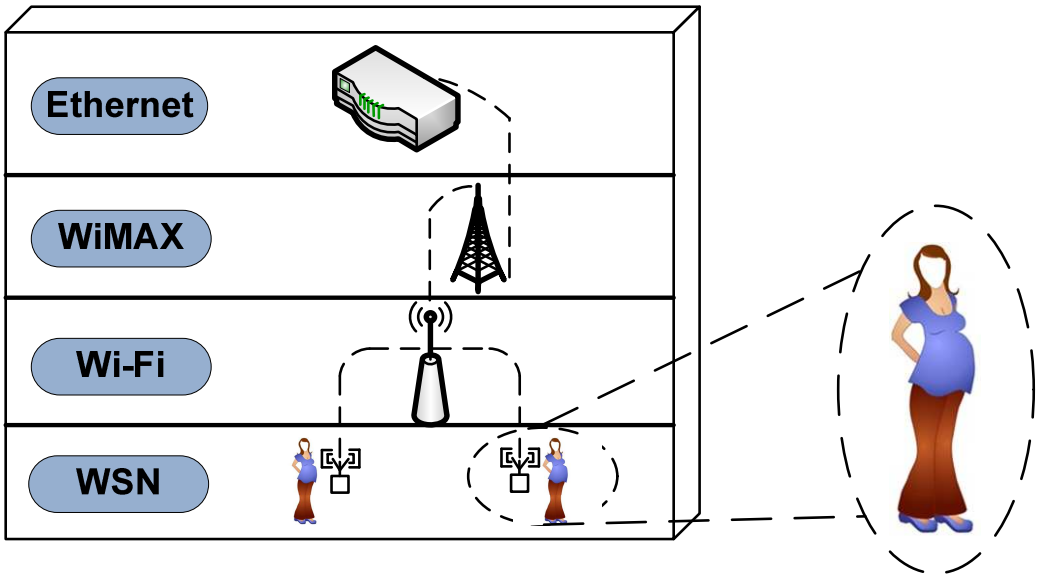


Figure 1.2. Hybrid communications considering the WBAN and other communications networks.

In order to develop and evaluate the performance of a new solution for WSN in terms of energy saving, we propose a solution using an innovative technique called *Block Acknowledgment* and the comparative analysis is based on the S-MAC protocol [YeHE02a].

Simulations were performed in OMNeT++ [OMNE09] is a public-source and component-based simulator engine that is used in this work to simulate the innovative techniques proposed to improve the existing MAC protocol. It has an extensive simulation library that includes support for input/output, statistics, data collection, graphical presentation of simulation data, random number generators and data structures. The simulation kernel uses C++ which makes it possible to be embedded in larger applications. Is simulation IDE is based on the Eclipse platform, and extends it with new editors, views, and wizards. Other functionalities include creating and configuring models (Network Description (NED) and Initialize (INI) files), performing batch executions and analyzing the simulation results, while Eclipse provides C++ editing, Concurrent Version System/Subversion (CVS/SVN) integration and optionally other features (Unified Modeling Language (UML) modelling, bug-tracker integration, database access, etc) via various open-source and commercial plug-ins.

The research for this thesis one given contribution on Smart-Clothing applications and WSN MAC protocols .The publications that arises from the work are the following:

- [BBVL09a] Borges,L.M., Barroca,N., Velez,F.J., and Lebres,A.S., “Smart-Clothing Wireless Flex Sensor Belt Network for Foetal Health Monitoring”, In *Proc. of WiPH 2009 International Workshop on Wireless Pervasive Healthcare (held in conjunction with 3rd International Conference on Pervasive Computing Technologies for Healthcare 2009)*, London, UK, 2009.
- [BBVL09b] Borges,L.M., Barroca,N., Ferro,J.,M. Velez,F.J., and Lebres,A.S., “WIRELESS FLEX SENSOR BELT NETWORKS FOR FOETAL MOVEMENT MONITORING IN LOW RISK PREGNANCIES”, accepted for publication in *XIX IMEKO World Congress Fundamental and Applied Metrology*, Lisbon, Portugal, Sep. 2009.
- [CONF09] Barroca,N., Velez,F.J., Ferro,J.,M., Borges,L.,M. and Lebres,A.S., “Desenho de Protocolos de Controlo de Acesso ao Meio Eficientes do ponto de vista energético em Redes de Sensores Sem Fios”, paper submitted to *Engenharia’2009 – Inovação e Desenvolvimento, Covilhã*, Portugal, Nov. 2009.
- [ISAB09] Barroca,N., Velez,F.J., Ferro,J.,M., Borges,L.,M. and Lebres,A.S., “An Innovative Sensor MAC Protocol for WBAN Applications”, accepted for publication in *ISABEL 2009 - 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies*, Bratislava, Slovak Republic, Nov. 2009.

1.3 Structure of the Thesis

Thesis is structured as follows. Chapter 2 presents an overview of the Smart-Clothing project, together with a description of its applications and scenarios applied to WBAN. Chapter 3 describes the IEEE 802.15.4 standard used in WSN, which is responsible to accommodates lower-end applications by trading-off higher data rates and performance for architectures that benefit from low power consumption and cost. Chapter 4 presents an overview of the most well know MAC protocols for wireless sensor networks. Chapter 5 describes an new innovative Sensor MAC protocol, that improves the performance of the existing one. Chapter 6 presents the results obtained by simulation for the energy-aware MAC protocols. Finally, Chapter 7 presents the conclusions and some suggestions for future work. Some annexes are also presented with useful information, like wireless sensor network platforms available, WSN simulators, and other documents related with the project Smart-Clothing.

Chapter 2

Smart-Clothing Application for Foetal Movement Detection

This chapter provides an overview of the work performed in the Smart-Clothing project. It combines investigation in sensor technologies, functional textiles and wireless communication networks in the context of human body monitoring and communications.

2.1 Scenario Description

The Smart-Clothing Project is an iCentro project approved by CCDR-C with FEDER funding [SMAR08]. The main areas of this project include the development of smart textiles prototypes that combines investigation in functional textiles materials and wireless communications networks in the context of human body monitoring and static methods for the data analysis and treatment as well. Realizing such wireless sensor networks is a crucial step toward a deeply penetrating ambient intelligence concept as they provide, figuratively, the “last 100 meters” of pervasive control. To conceive them, a better understanding of their potential applications and the ensuing requirements is necessary, as it gives an idea of the enabling technologies.

The wireless sensor networks manufacturers and/or designer’s claim that this technological advance will facilitate many existing application areas and bring into existence entirely new ones. This technological advance depends on many factors, but here only some of the envisioned application scenarios will be highlighted. Public and private sector enterprises are always pursuiting new ideas that can lead to more efficient and flexible techniques and also quality policies that have a direct effect on both profitability and performance. As wireless networking moves into the mainstream, many organizations find that the addition of mobile network components offers undeniable direct and indirect business benefits.

The wireless solutions in a direct way can improve the connectedness of a workforce and enhance decision-making by providing faster access to more current information. The wireless solutions can also be easier to maintain and configure, reducing the need for Information Technology (IT) staff. In a indirect way, the wireless and mobile solutions can improve worker satisfaction by providing easier and more flexible access options. They can even improve public perception and introduce new “cutting edge” mechanisms for customer interaction. This chapter considers how the addition of wireless technology can lead to a more efficient workflow, enhance productivity and improve communications in different healthcare scenarios. These brief overviews consider wireless solutions at home or at the hospital in the context of medical diagnostics of foetal health in the pregnant women. The basic idea is to give freedom of movements to the pregnant woman when she needs to do a medical exam.

The chapter examines some of the options available for these interconnections, the hardware needed to implement them and the terminology used to discuss them. There are many options currently available to implement Wide Area Networks (WAN) solutions, which differ in technology, speed and cost. The familiarity with these technologies is also an important part of network design and evaluation. If all the data traffic which comes from the sensor node located in the pregnant woman is within a single building, a small Wireless Local Area Network (WLAN) meets the needs of the data traffic. Besides that, buildings can be interconnected with high-speed data links to form a campus Local Area Network (LAN) if data must flow between buildings on a single campus. However, a WAN is needed to carry data if it must be transferred between geographically separated locations. All the

communications involved in this work aims mainly to aid in the monitoring of the foetal movement essentially in the last four weeks of pregnancy. It is important to optimize the trade-off between energy consumption/ processing and communication capabilities, namely in the MAC layer. Cross-layer design interaction between MAC and physical layers will be addressed in detail and the interactions with network and transport layers will also be considered.

Hybrid communications can be a solution to obtain a network of networks, e.g., by using IP (Internet Protocol). A bottom-up architecture hierarchically formed by: i) WSNs (or WPANs), ii)Wi-Fi, and iii) Ethernet will be explored in order to allow remote healthcare monitoring anywhere and anytime. Since the interface board of the WSN or WBAN can move, the second layer of the networks may use Wi-Fi.

2.2 Objectives

The main objective of the Smart-Clothing Project in the context of Wireless Sensor Networks is to monitor the foetal movement in the last four weeks of pregnancy. Besides the integration of sensors in the garment it will be needed a hierarchical communication system which allows the delivery of the data collected to the doctor from the garment that the pregnant is wearing. The pregnant can be either at home or in the hospital. In the first stage of the project some tests were made using several types of sensors integrated in a belt in order to choose the one that is more reliable for the detection of foetal movement. At the same time a WSN solution based on IEEE 802.15.4 standard was developed, including a wireless hierarchical network solution involving a Wi-Fi layer. Two flex sensor belts were produced to count the foetal movements, one standalone solution and one wireless flex sensor belt network which incorporates a sensor node.

2.3 System Description

In order to accomplish the communications task, instead of using wired connections, the data is transmitted through a WSN composed by Crossbow® IRIS motes. This WSN allows to receive correctly the data from the sensors in a computer and make the results available [BBVL09b]. The simplest device consists of a mote, a small battery and a set of sensor (e.g, flex sensors). Such device will help in the continuous monitoring of the foetal health which traditionally is based on protocols where the foetal movements felt by the mother are counted. So a hybrid communication system is employed in order to deliver the data from the Wireless Body Area Network (WBAN) that is installed in the pregnant woman. All the data from our application is saved, by using a Structured Query Language (SQL) database. SQL helps avoiding redundant and outdated data, and solves security problems related with the malicious or unauthorized access, as the data comes from the belt of the pregnant woman must be protected from corruption [BBVL09b].

The block diagram for the acquisition system is presented in Figure 2.1.

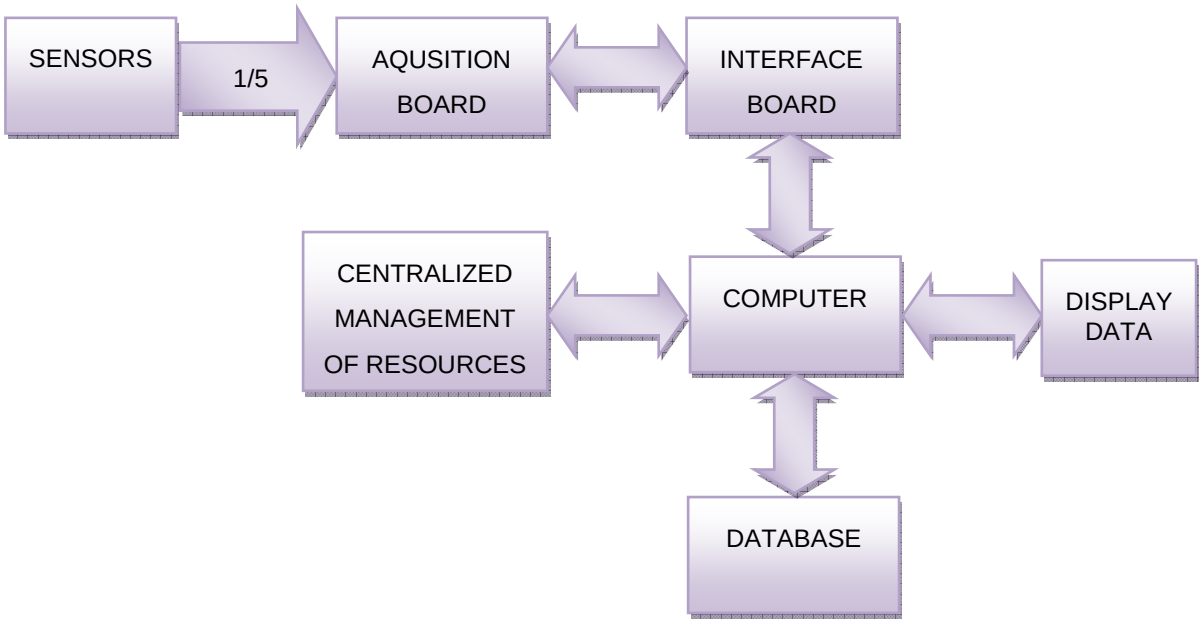


Figure 2.1. Block Diagram for Acquisition System.

The IEEE 802.15.4 sensor network, whose primary function is the gathering of the vital data from the various sensors that are applied in the patient body remotely, allows for the monitoring of the foetal movements from a pregnant woman while transmitting the data to a Mote Interface Board (Gateway) that is directly connected to our Centralized Management of Resources (CMR).

The Centralized Management of Resources entity is formed by a base station, a personal computer, an application that is responsible to show the data and save all the records in a database, and a Wi-Fi module to transmit data across a WLAN, as shown in the Figure 2.2.

In our prototypes, IRIS Motes are used, forming a 2.4 GHz IEEE 802.15.4 tiny wireless measurement system, designed specifically for embedded sensor networks [IRIS08]. The IRIS Mote is one of the components from the MICA family (model XM2110CA). Their primary characteristics are very similar to the MICAz, with the same data transmission rate and radio frequency band as the previous model. The system supports TinyOS and is fully compatible with accessories for the MICA family. It is suitable for applications that require high speed in data transmissions, for example real time reading from a sensor in Indoor Building Monitoring. The main advantages are the following: i) range up to 500 meter (between nodes without the need to amplify the signal), ii) and 8kB of RAM (almost the double than the existing modules in the past) [BBVL09a].

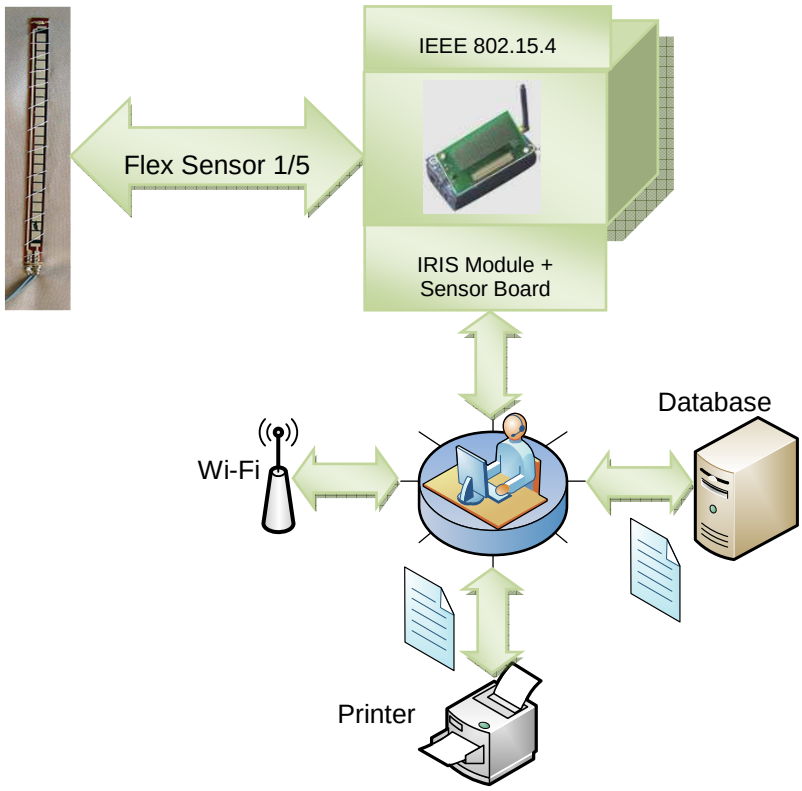


Figure 2.2. Patient Monitoring using IEEE 802.15.4 and a Wi-Fi Wireless networking architecture.

2.3.1 MDA Sensor Board

The sensor board responsible by gathering of the data from the various sensors is the MDA100CB sensor board, as show in Figure 2.3. It has a precision thermistor, a light sensor/photocell and general prototyping area included on it. The prototyping areas support connection to all 51 pins on the expansion connector, and have connection to all eight channels of the Mote's analog-to-digital converter (ADC0–7), both USART serial ports and the I2C digital communications bus. The prototyping area also has 45 unconnected holes that are used for breadboard of circuitry [MDA108], [PFRC08].

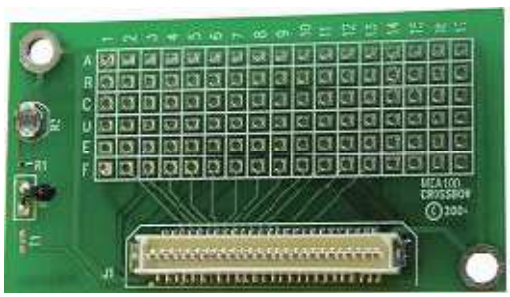


Figure 2.3. MDA100CB Sensor Board.

2.3.2 Thermistor

The thermistor that is connected to the sensor board (YSI 44006) is a sensor with a highly accurate and highly stable sensor element. With proper calibration an accuracy of 0.2 ℃ can be achieved, Table 2.1. The thermistor's resistance varies with temperature,

Table 2.2 The resistance versus temperature graph is presented in Figure 2.4. The sensor is connected to the analog-to-digital converter (ADC) channel number 1 (ADC1) through a basic resistor divider circuit. In order to use the thermistor, the sensor must be enabled by a digital control signal PWM0, as show in the Figure 2.5 [MTSM08], [PFRC08].

Table 2.1. Thermistor Specifications.

Type	**YSI 44006**
Time Constant	10 seconds, still air
Base Resistance	10kΩ at 25º
Repeatability	0.2ºC

Table 2.2. Resistance vs. Temperature.

Temperature (ºC)	**Resistance for the therrmistor (Ω)**	**ADC Reading (% of VCC)**
-40	239.800	4%
-20	78.910	11%
0	29.940	25%
25	10.000	50%
40	5592	64%
60	2760	78%
70	1990	83%

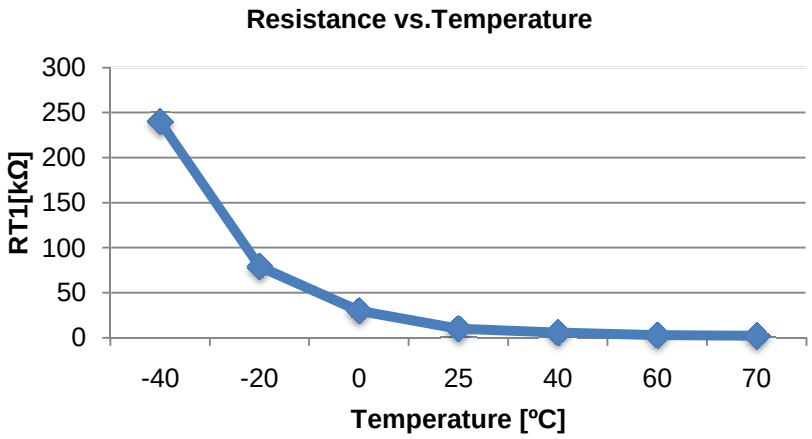


Figure 2.4. Curve of the Resistance vs. Temperature.

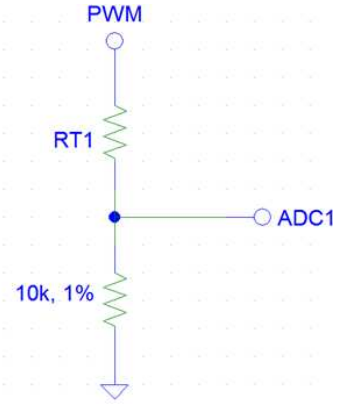


Figure 2.5. Schematic for the Thermistor on MDA100CB.

The Mote's ADC output can be converted from Kelvin to Celsius by using the following approximation over 0 to 50 ℃ range:

$$R_{thr} = R1 \times (ADC\_FS - ADC)/ADC \quad (2.1)$$

where: R1=10kΩ, ADC_FS=1023 (10 bit ADC) and ADC= output value from Mote's ADC measurement.

As presented in Figure 2.4, a thermistor is a temperature dependent resistor. So the resistance of the thermistor changes in a predictable way. The thermistor's resistance depends upon temperature, so the Steinhart-Hart equation gives the reciprocal of absolute temperature as a function of the resistance from a thermistor and using this we can calculate the temperature of the thermistor from the measured resistance.

The Steinhardt-Hart equation is given as follows:

$$\frac{1}{\text{Temperature(K)}} = a + b \times \log(R_{thr}) + c \times [\log(R_{thr})]^3 \quad (2.2)$$

The constants a, b and c can be determined from experimental measurements of resistance, or they can be calculated based on the

Table 2.2.

The values of a, b and c is given as follows:

$$\left(\frac{1}{273}\right) = a + b \times \log(29.294) + c \times [\log(29.940)]^3 \quad (2.3)$$

$$\left(\frac{1}{298}\right) = a + b \times \log(10.000) + c \times [\log(10.000)]^3 \quad (2.4)$$

$$\left(\frac{1}{323}\right) = a + b \times \log(3893) + c \times [\log(3893)]^3 \quad (2.5)$$

The equations above can be solved to obtain the values of a, b and c:

$$a = 0.001010024$$

$$b = 0.000242127$$

$$c = 0.000000146$$

The temperature in (ºC) is given by:

$$\text{Temperature(K)} = \frac{1}{a + b \times log(R_{thr}) + c \times [log(R_{thr})]^3} \quad (2.7)$$

$$\text{Temperature(ºC)} \approx \text{Temperature(K)} - 273.15 \quad (2.8)$$

2.3.3 Light Sensor

The light sensor included in the MDA sensor board is a simple CdSe photocell, Figure 2.6. The maximum sensitivity of the photocell is when the light wavelength is equal to 690 nm. Typical “on” resistance, while exposed to light, is 2 kΩ. Typical “off” resistance, while under dark conditions, is 520 kΩ. In order to use the light sensor, digital control signal PW1 must be turned on. The output of the sensor is connected to the ADC channel 1 (ADC1). When there is light, the nominal circuit output is near VCC or full-scale voltage and when it is dark the nominal output is near GND or zero. The power to the light sensor is controlled by setting the signal INT1 [MTSM08], [PFRC08].

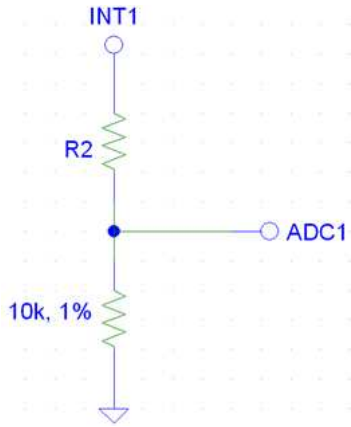


Figure 2.6. Schematic for the Light Sensor.

2.3.4 Prototyping Area

A generous prototyping area has been provided with many solder holes and connection points useful for connecting other sensors and devices to the Mote. The prototyping area layout is shown in the diagram presented in Figure 2.3 and Table 2.3 [MTSM08], [PFRC08].

Table 2.3. Connection Table for MDA100CB.

	A	B	C	D	E	F
1	GND	GND	GND	VCC	VCC	VCC
2	OPEN	OPEN	USART1_CK	INT3	ADC2	PWO
3	OPEN	OPEN	UART0_RX	INT2†	ADC1†	PW1†
4	OPEN	OPEN	UART0_TX	INT1	ADC0†	PW2
5	OPEN	OPEN	SPI_SCK	INT0	THERM_PWR	PW3
6	OPEN	OPEN	USART1_RX	BAT_MON	THRU1	PW4
7	OPEN	OPEN	USART1_TX	LED3	THRU2	PW5
8	OPEN	OPEN	IC2_CLK	LED2	THRU3	PW6
9	OPEN	OPEN	I2C_DATA	LED1	RSTN	ADC7
10	OPEN	OPEN	PWM0	RD	PWM1B	ADC6
11	OPEN	OPEN	PWM1A	WR	OPEN	ADC5
12	OPEN	OPEN	AC+	ALE	OPEN	ADC4
13	OPEN	OPEN	AC-	PW7	OPEN	ADC3
14	GND	GND	GND	VCC	VCC	VCC

15	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
16	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
17	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN

2.4 IRIS Module

2.4.1 Description

The IRIS is a 2.4 GHz Mote module used for enabling low-power wireless sensor networks. The IRIS module has several new capabilities that will enhance the overall functionality of Crossbow's wireless sensor networks [IRIS08], [PFRC08].

The IRIS Mote is one of the modules from the MICAs family with the model XM2110CA. Their primary characteristics are very similar to the MICAz, where they have the same rate of data transmission and radio frequency band, supports TinyOS and is fully compatible with accessories from the MICAs family. Figure 2.7 present the IRIS mote equipment and the respective block diagram.

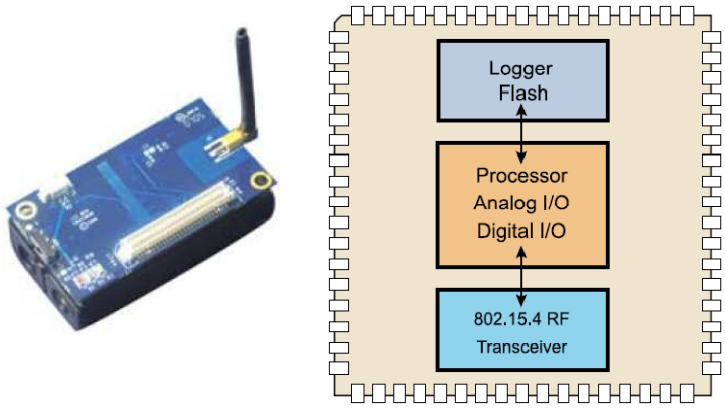


Figure 2.7. IRIS Mote and Block Diagram.

The mote shown in Figure 2.7 is the wireless module that after being attached to the Sensor Board presented in Figure 2.3 will be responsible to transmit all the data from the sensors from the pregnant woman to the Mote Interface Board (Gateway). The Mote Interface Board is directly connected to our CMR.

2.5 Application to read data from IRIS Motes

One of the main goals of the Smart-Clothing for Health Monitoring and Sport Applications project (Smart-Clothing) is the creation of a hybrid communication system, hierarchically organized with

different standards in order to deliver the data from the WBAN which is installed in the pregnant woman. Because there is the need to correctly receive the data from the sensors and make the results available to the user in a way the different signals can be discriminated, a computer program was conceived to accomplish the task of receiving and analyzing the data, using two different methods.

The first method consists of using Cygwin before analyzing the data, Figure 2.8 Cygwin is an UNIX emulator that can be faced as an external application which is used to save the data from the motes into a text file. Then our program can be used to read the data from that text file and properly present the data to the user so that he can understand the results, Figure 2.9.

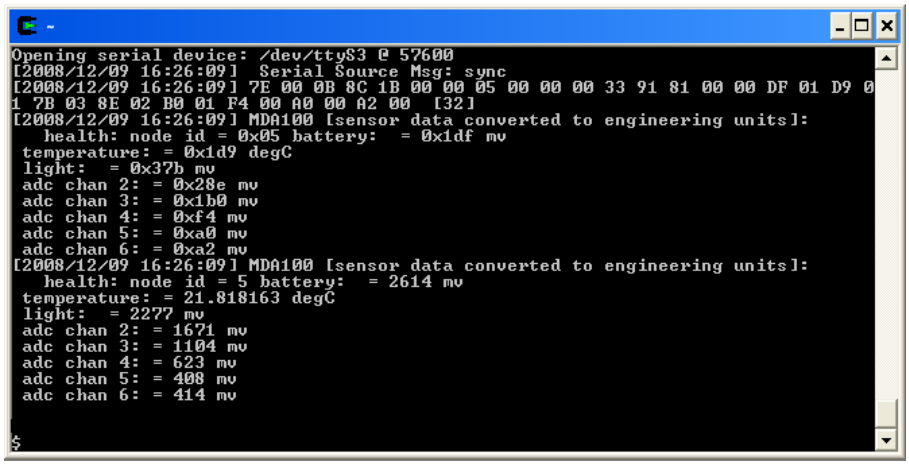


Figure 2.8. Cygwin Data.

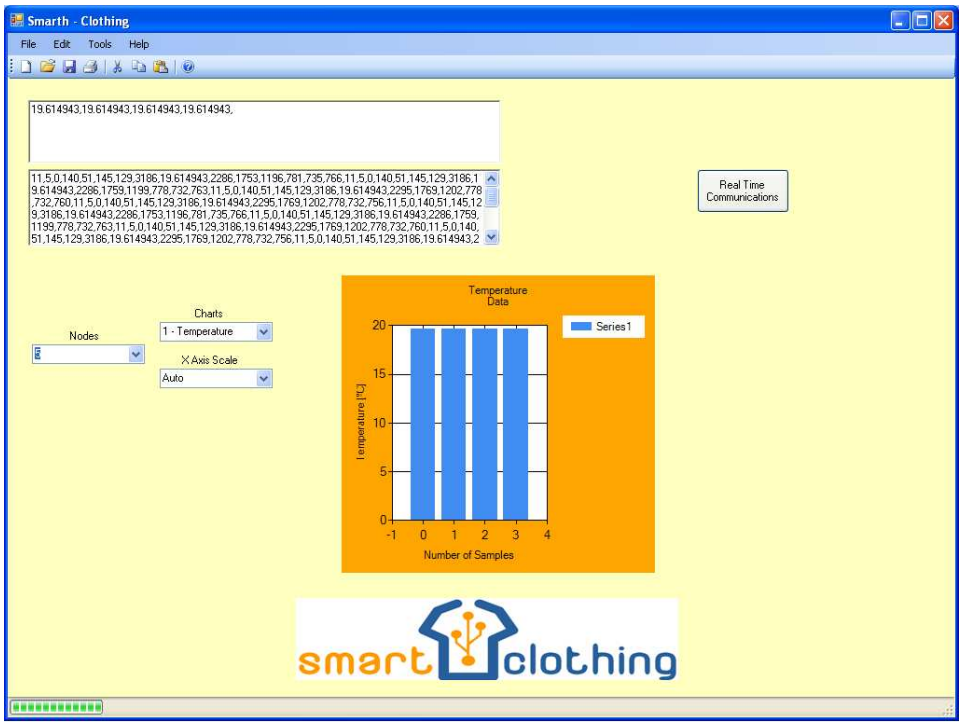


Figure 2.9. Smart Application Main Window (Read Text File).

The second method consists on reading the data directly in real time, by using our application alone. While trying to read the data directly from the “emulated” serial port, the application presents the data to the user after some processing (that distinguish not only the size of received frames, but the order we need to read the data because the data is scrambled), Figure 2.10.

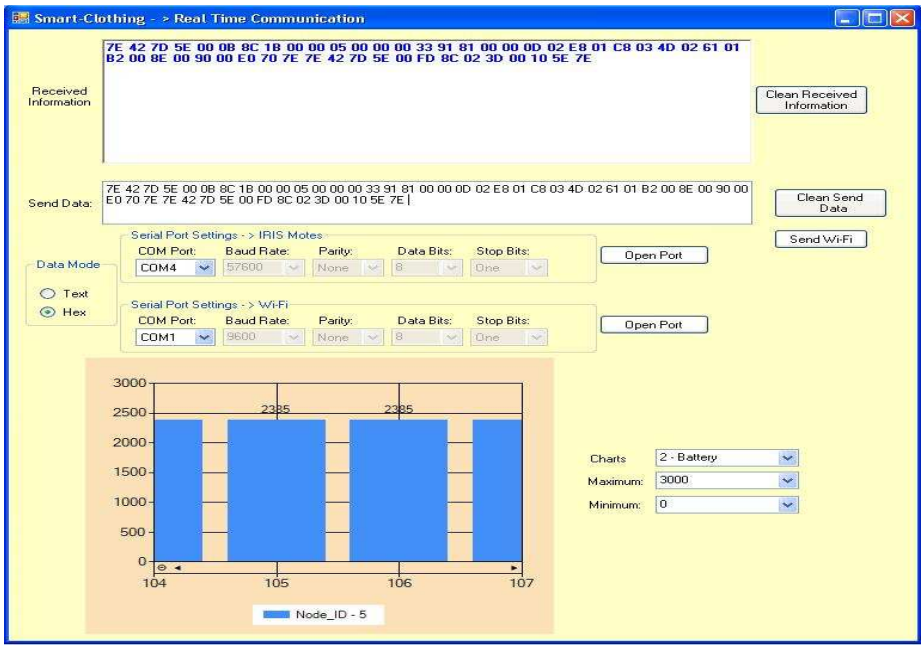


Figure 2.10. Smart Application Real Time Communication.

As presented in Figure 2.8 and 2.9 the data read by the motes was received properly. There are several set of parameters that can be monitorized. In the first set of tests we monitorize the battery voltage of the motes, the ambient temperature, and the light of a room. These parameters are very important because we need to know the state of the batteries and change it if they depleted. Besides, the energy consumption can be monitored and new algorithms based on energy consumption could emerge in order to optimize the trade-off between energy consumption/processing and communication capabilities. The second set of parameters is also important, which are the temperature and the light of the room, respectively. This way we can monitor what changes appear during the pregnancy of the woman if the ambient conditions changes.

In the second set of tests the information that is received from the belt with pressure sensors, that are installed in the belt, considers a scenario where the pregnant woman is waiting to be seen by the doctor and is being monitorized inside the hospital. An IEEE 802.15.4 network can be used to collect the different deformation angles caused by the foetus movements. Figure 2.11 presents a simple scenario of the remote monitoring system.

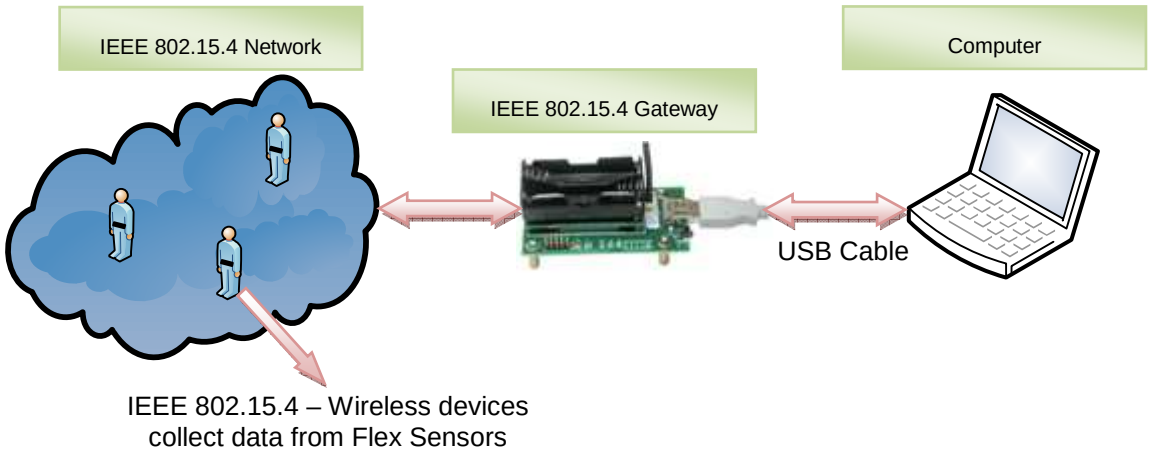


Figure 2.11. Pregnant Monitoring using IEEE 802.15.4 Wireless Monitoring.

After completing the laboratory tests, the next step is to test the flex sensor belt in a real environment. The tests were made in the Hospital Pêro da Covilhã. Figure 2.12 presents a possible scenario on the fifth floor. The CMR is located in a convenient area and the red spots represent the pregnant woman with the belt. This corresponds to the communication system employed to deliver the data from the WBAN that is installed in the pregnant woman.

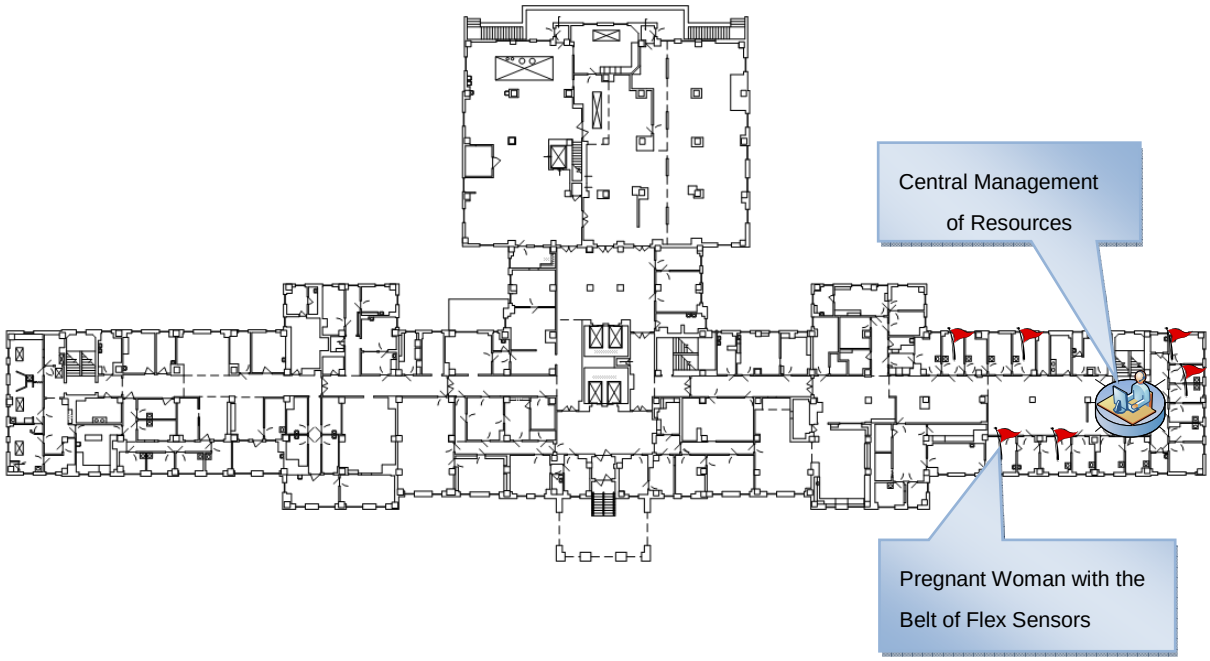


Figure 2.12. Possible Monitoring using IEEE 802.15.4 in a Hospital.

The application presented in Figure 2.9 and 2.10 is responsible for collecting all sensors data collected by sensors available to the nurse/doctor. This will be one additional tool available for diagnostic. Note that we can easily move our CMR to another part or the Hospital, demonstrating that there is no limitation by the geographical position of any components of the system.

Some tests were made in a pregnant woman. The patient during the tests most of the times was sitting in a chair Figure 2.13. A calibration of the flex sensors is made in the belt before any test. We did a test of how the system behaves if a sudden change of the pregnant woman position happens and result is the presented in, Figure 2.14.

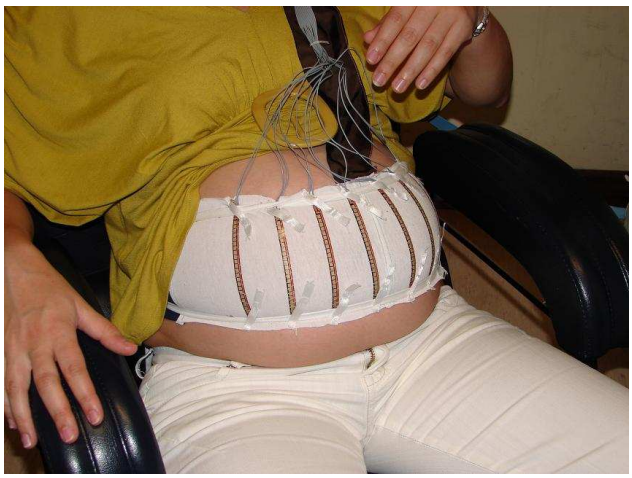


Figure 2.13. Pregnant woman sitting.

Figure 2.14. Results 1 (patient stand-up → patient sited).

The results for the test made when the patient was sitting in a chair as presented in Figure 2.15, a movement was detected by the system, while the patient claimed to have detected two movements. The time instant when the system and the patient detected the foetus movement simultaneously was

at the time ≈910 seconds. The system detected the movement at the flex sensor 1, 2, 3, 4, 5 and 6, but with a stronger intensity in the flex sensor 3. The instant when the patient claim to feel a foetus movement was at time ≈895 but the system didn't detect any movement. At the time≈925 seconds the system claims that detected a foetus movement but it was considered a false positive because the patient did not feel it.

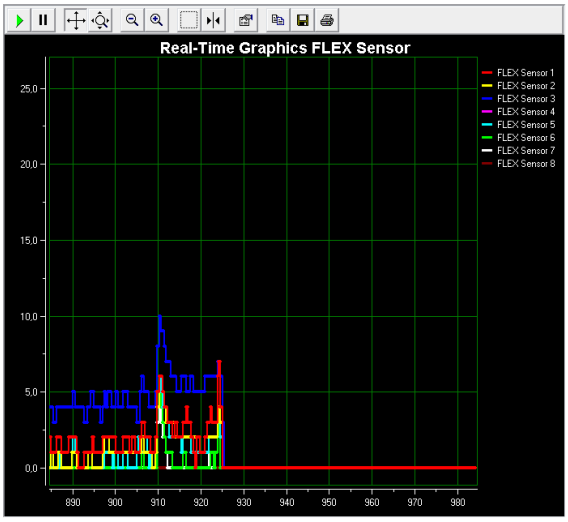


Figure 2.15. Results 4 (patient sited)

2.6 Conclusions

In this part of the work a prototype that allows the continuous monitoring of the foetal health was developed. This was accomplished by using an easy to wear belt and a tele-medicine system that will allow remote monitoring in low risk pregnancies. The belt was tested in a pregnant woman in order to verify their performance and tune the threshold-triggers. The results of these tests shown that sometimes the system detects the movements from the foetus but there was still a lot of motion artifact mixed. Some false positives movements' detection happened during the tests.

To accomplish the remote monitoring, a system was created using a belt with several flex sensors connected to the pregnant woman. It detects the foetal movement based on the bending of the sensors and an IEEE 802.15.4 network was also created in order to deliver all the data collected by the motes to our CMR. An application is responsible to present the data to the user (nurse/doctor) and save all the records in a database. If necessary, it is possible to transmit the data via Wi-Fi, so that the information can be shared or accessed by other people. This solution shows a WBAN application applied to remote patient monitoring. Nowadays, there are increasing demands on remote monitoring and onsite monitoring is becoming more and more important, so in order to have easy access to "up-to-the-minute" information wireless devices can help increase overall efficiency and enable better patient care.

Chapter 3

IEEE 802.15.4

This chapter provides an overview of the IEEE 802.15.4 standard, which specifies the physical and media access control layers for low-rate wireless personal area networks (LR-WPANs). It represents a significant breakthrough from the “bigger and faster” standards that the IEEE 802 organization continues to develop: instead of higher data rates and more functionality, this standard addresses the simple and low-data volume universe in terms of control and sensor networks, which existed without global standardization through a series of proprietary methods and protocols. The lack of a standard approach and a protocol was seen as a major obstacle to large scale manufacture of inexpensive silicon radios that will minimize the cost per node of these networks. Moreover, the conservation mechanism in terms of energy allows the implementation of new energy-efficient protocols, such as those discussed in the next chapters and that can be layered on top of IEEE 802.15.4 MAC to provide power management in these scenarios.

3.1 Context and Motivation

In a world which is constantly changing and several wireless equipments became part of our life, the technology is constantly in change and people claim for new products availability and for improvements in the existing ones. Hence, it is important to create new innovative products to respond to the new needs of the society. Wireless Sensor and Actuator Networks (WSAN) are becoming one of the most important technologies in the 21st century. One of the main reasons is the fact that this type of network can be applied in several applications with different scenarios.

The recent advance in micro-electromechanical systems (MEMS) is responsible for the appearance of a new class of networks: the so-called Wireless Sensor Networks (WSN), in the Anglo-Saxon literature, by a designation proposal [PoKa00] [ASSC02]. This type of networks consists of individual devices called nodes that are deployed in a geographical area with the primary objective to measure and communicate the information related with physical phenomena. To accomplish this task, this type of device has four basic components: a sensing unit responsible for the data acquisition, a processing unit responsible for storage and local data processing, a radio transceiver responsible for wireless communication and finally an embedded battery unit which will be the power source of the entire system. Sometimes in the Anglo-Saxon literature, the term WSAN is also used and the difference of terms is because these networks often includes actuators and control functionalities.

In many scenarios, nodes will have a limited power supply, because typically this type of devices is battery operated. So, it is important to extend battery life time, not only because replacing the batteries has a cost but because sometimes changing batteries is not so practical especially if the nodes are deployed in areas with difficult access.

In this type of networks, minimum bandwidth may be irrelevant when application are tolerant to latency [ADLN98] or when the bit rate requirements used to transmit the data from the motes is small, another important issue is network lifetime, which is directly related with the networks trade-offs. Therefore increasing the use of energy may increase the quality of the network however this will decrease the lifetime and it is important to find an equilibrium between network traffic and energy consumption.

The Institute of Electrical and Electronic Engineers (IEEE) has launched the first release of the IEEE 802.15.4 standard in October 2003 [WPAN03]. The IEEE 802.15.4 specification complements the IEEE 802 set of wireless standards to enable sensor-rich environments. It accommodates lower-end applications by trading off higher speed and performance, for architectures that benefit from low power consumption and cost. In a parallel activity, the former IEEE 802.15.4b Task Group focused on refining IEEE 802.15.4, by removing ambiguities and addressing issues raised during early implementation efforts of IEEE 802.15.4 devices. The 4b Task Group completed its activities in 2006 by releasing an updated version of the original standard [NaBe07]. Finally, in August 2007 an amendment was made to the IEEE 802.15.4 of 2006 adding new alternate Physical Layers (PHYs),

such an Ultra-wide band (UWB) PHY at frequencies of 3 to 5 GHz, 6 to 10 GHz, and less than 1 GHz and a Chirp Spread Spectrum (CSS) PHY at 2450 MHz. The UWB PHY supports an over-the-air mandatory data rate of 851 kb/s, with optional data rates of 110kb/s, 6.81 Mb/s and 27.24 Mb/s. The CSS PHY supports an over-the-air data rate of 1000 kb/s and optionally 250kb/s. The chosen PHY depends on local regulations, application and user preference [WPAN07].

The IEEE 802.15.4, as described before covers the physical layer and the MAC layer of low-rate in the context or the Wireless Personal Area Network (WPAN). It is very common to confuse IEEE 802.15.4 with ZigBee® [ZIGB09] which is a specification created by the ZigBee® Alliance. The ZigBee® standard is responsible by the definition of the networking, the application, and the security layer of the protocol while adapting the IEEE 802.15.4 PHY and MAC layers as part of the ZigBee® networking protocol. Hence it is possible to create short-range wireless networks using only the IEEE 802.15.4 standard without implement the ZigBee® layers. This way the user can develop their own applications, on top of the IEEE 802.15.4 PHY and MAC layers.

As presented in Figure 3.1, IEEE 802.15.4 2006 has only simple layers. The advantage is to implement applications with a small memory size device.

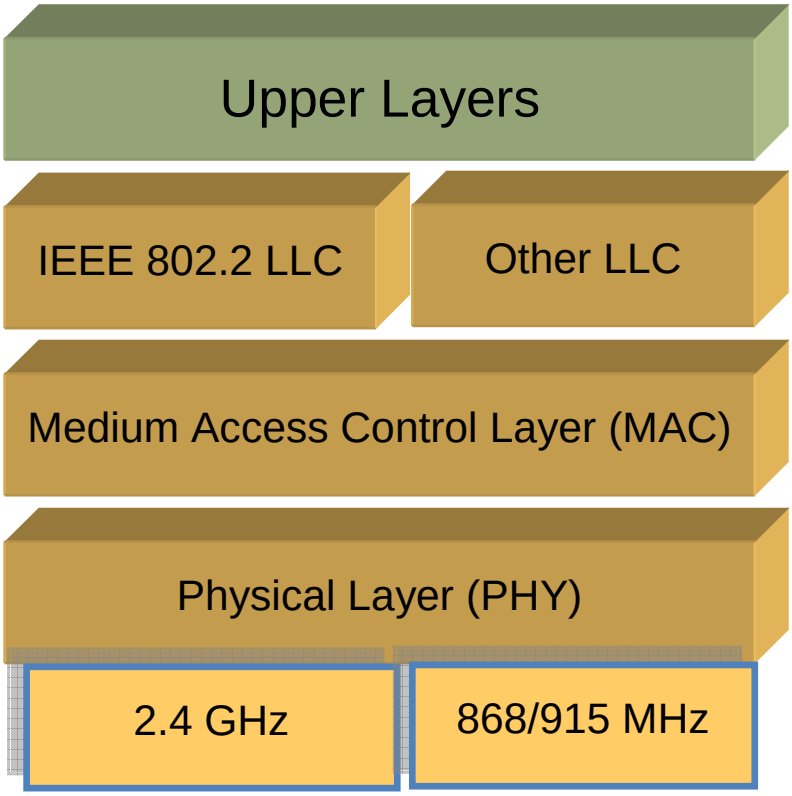


Figure 3.1. IEEE 802.15.4 Protocol Arquitecture.

The ZigBee® Alliance adopted IEEE 802.15.4 2006 for Wireless Personal Area Networks (WPANs). The Stack Architecture of the ZigBee® is represented in the Figure 3.2. The use of the IEEE 802.15.4

specification made it possible to focus the task force of the alliance on design issues for the related network, security and application layers.

The network layer (NWK) is responsible for the organization and provides the routing capability in a multihop network. Therefore the NWK will handle network addressing and routing by invoking some actions in the IEEE 802.15.4 MAC layer, such as start the network (by the co-ordinator), assignment of the network address, add/remove devices from the network, route messages to the destination, apply security to outgoing messages and implement route discovery in mesh topologies (while updating the routing table). The application sub-layer (APS) is responsible for the communication with the application. For example to blink a LED, the APS relays this instruction to the application using the endpoint information in the message. The ZigBee® Device Objects (ZDO) represents the node type of the device (end device, co-ordinator or router), the ZDO Management Plane will be responsible to span the NWK and the APS layers, allowing at the same time the ZDO to communicate with these two layers when performing its internal tasks. Other characteristic is the permission of the ZDO to deal with request that came from the application, related with network access and security functions by using ZigBee® Device Profile (ZDP) messages to achieve it.

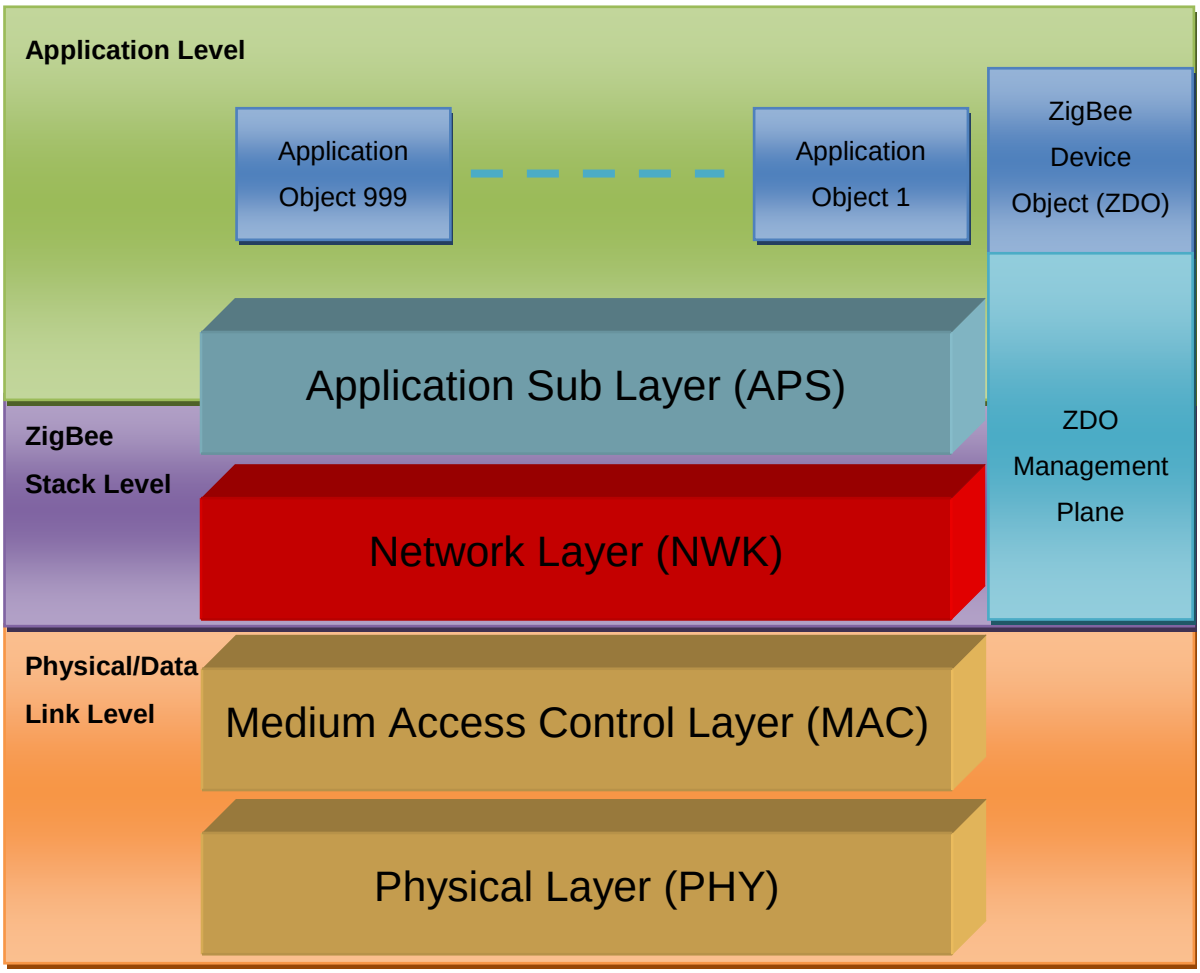


Figure 3.2. ZigBee® functional layer arquitecture and protocol stack.

3.2 IEEE 802.15.4 Physical Layer

The IEEE 802.15.4 physical layer is responsible for the activation and deactivation of the radio transceiver, energy detection (ED), link quality indication (LQI), channel selection and clear channel assessment (CCA), while transmitting/receiving packets across the physical medium. The optional UWB PHY also has the optional feature of precision ranging. The radio transceiver operates at one or more of the following unlicensed bands:

- 868–868.6 MHz (868 MHz Band);
- 902–928 MHz (915 MHz Band);
- 2400–2483.5 MHz (2.4 GHz Band).

In the IEEE 802.15.4 standard, the physical layer offers 20 kb/s bitrates, using a single channel in the frequency range 868–868.6 MHz band. This range of frequencies is used in Europe for applications, such as short-range wireless networking. The other two bands (915 MHz and 2.4 GHz) are part of the industrial, scientific and medical (ISM) frequency bands. The 2.4 GHz band is used worldwide and the 915 MHz band is used mainly in North America.

The IEEE 802.15.4 transceiver requires simultaneous and joint support of the 868 MHz band, and 915 MHz frequency band.The 868/915 MHz BPSK PHY, originally specified in 2003, offers a trade-off between complexity and data rate. The optional PHYs offers a data rate much higher than the one given by the 868/915 MHz BPSK PHY, which provides a data rate of 20 kb/s in the 868 MHz band and 40 kb/s in the 915 MHz band. The ASK PHY offers data rates of 250 kb/s in both the 868 MHz and 915 MHz bands, the same data rate of the 2.4 GHz band PHY. In the 915 MHz band the O-QPSK PHY offers a signalling scheme identical to the one of the 2.4 GHz band PHY and a data rate equal to the one of the 2.4 GHz PHY band. In terms of data rate the O-QPSK PHY in the 868 MHz band supports a data rate of 100 kb/s..

As presented in the

Table 3.1, the frequencies from the IEEE 802.15.4 standard are divided by three different bands with a total of 27 channels. The central frequencies are given by:

- $f_c = 868.3$, in MHz, for $k = 0$
- $f_c = 906 + 2 (k - 1)$, in MHz, for $k = 1, 2, \ldots, 10$
- $f_c = 2405 + 5 (k - 11)$, in MHz, for $k = 11, 12, \ldots, 26$

  where k it is the channel number

The channelization of 27 half-duplex channels specified by the IEEE 802.15.4 group is shown in Figure 3.3 is organized as follows:

- The 868 MHz band has a frequency between 868.0 MHz and 868.6 MHz and it's used in

Europe. It adopts a binary phase shift keying (BPSK) modulation format, with a direct sequence spread spectrum (DSSS) at a chip-rate 300 kchip/s. Only a single channel with data rate 20 kb/s is available and de devices shall be capable of achieving a sensitivity of –92 dBm or better. A pseudo-random sequence of 15 chips is transmitted in a 50 µs symbol period.

- In the 915 MHz band the ranging is between 902 MHz and 928 MHz. It is used in the North American and Pacific area and adopts a BPSK modulation format, with DSSS at a chip-rate of 600 kchip/s. Ten channels with data rate 40 kb/s are available and the devices will be capable of achieving a sensitivity of –92 dBm or better. A pseudo-random sequence of 15 chips is transmitted in a 25 µs symbol period.
- The unlicensed 2.4 GHz ISM band, which extends from 2400 to 2483.5 MHz and is used worldwide, adopts a offset quadrature shift keying (O-QPSK) modulation format, with a DSSS at 2 Mchip/s. Sixteen channels with data rate 250 kb/s are available and devices shall be capable of achieving a sensitivity of –85 dBm or better.

Table 3.1. Frequency bands and data rates.

		Spreading Parameters		**Data Parameters**			
PHY (MHz)	**Frequency Band (MHz)**	**Chip Rate (kchip/s)**	**Modulation**	**Bit Rate (kb/s)**	**Symbol Rate (ksymbol/s)**	**Number of Channels**	**Spreading Method**
868/915	868–868.6	300	BPSK	20	20	1	Binary DSSS
	902–928	600	BPSK	40	40	10	Binary DSSS
868/915 (optional)	868–868.6	400	ASK	250	12.5	1	20-bit PSSS
	902–928	1600	ASK	250	50	10	5-bit PSSS
868/915 (optional)	868–868.6	400	O-QPSK	100	25	1	16-array Orthogonal
	902–928	1000	O-QPSK	250	62.5	10	16-array Orthogonal
2450	2400–2483.5	2000	O-QPSK	250	62.5	16	16-array Orthogonal

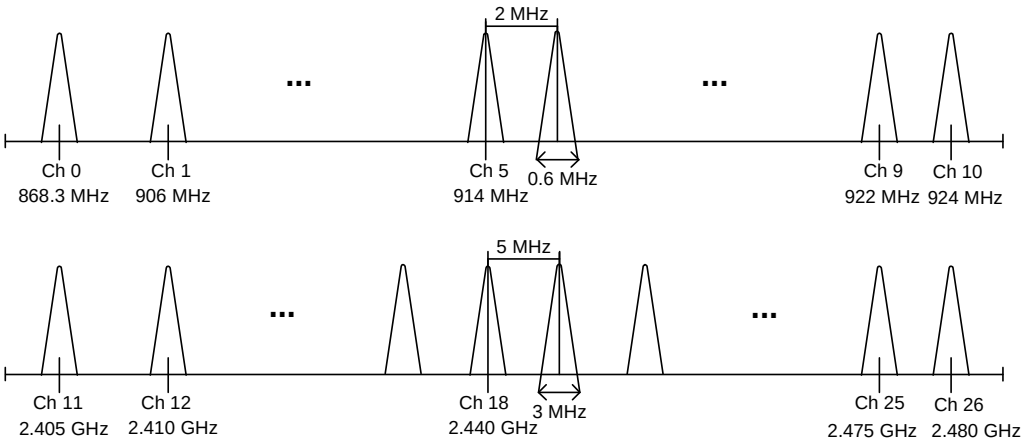


Figure 3.3. IEEE 802.15.4 channelization at the 868/915 MHz and 2.4 GHz bands.

The operation band shows up are based on the Direct Sequence Spread Spectrum (DSSS) spreading technique. Table 3.1 provides further details regarding the ways these three bands are used in the IEEE 802.15.4 standard. If an IEEE 802.15.4 transceiver supports the 868 MHz band it must support 915 MHz band as well, and vice versa. Therefore, these two bands are always bundled together as the 868/915 MHz frequency bands of operation (in the Anglo-Saxon literature). Note that one 'symbol' is equivalent to four 'bits'.

As described before the physical layer of the IEEE 802.15.4 is responsible by the control of several parameters which is described with more detail as follows:

- **Activation and deactivation of the radio transceiver**-The radio transceiver may operate in three different states: transmitting, receiving or sleeping. Upon a request of the MAC sub-layer, the radio is turned ON or OFF. The standard recommends that the turnaround time from transmitting to receiving states and vice versa should be at least 12 symbol periods.
- **Receiver Energy Detection (ED)**-This parameter gives an estimation of the signal received power within the bandwidth of an IEEE 802.15.4 channel. The standard recommends that the energy detection duration should be equal to 8 symbol periods. This measurement is typically used to determine if the channel is busy or idle in the Clear Channel Assessment (CCA) procedure or by the Channel Selection algorithm of the Network Layer.
- **Link Quality Indication (LQI)**-The LQI characterizes the strength/quality of a received signal on a particular link and can be implemented using the receiver energy detection technique, a signal to noise estimation or a combination of both techniques. The results can be used by the higher layers however this procedure is not specified in the standard.
- **Clear Channel Assessment (CCA)**-The CCA operation is responsible by the report of the medium activity state: busy or idle. Basically there are three operational modes:

  - *Energy Detection mode*-In this mode the CCA shall reports a busy medium if the received

energy is above a given threshold, referred as energy detection (ED) threshold.

- *Carrier Sense mode*-The CCA reports a busy medium only if it detects a signal with the modulation and the spreading characteristics of IEEE 802.15.4 standard and which the signal may be higher or lower than the ED threshold.

- *Carrier Sense with Energy Detection mode*-This is a combination of the aforementioned techniques. The CCA reports the medium is busy only if it detects a signal with the modulation and the spreading characteristics of IEEE 802.15.4 and with received energy above the ED threshold.

- **Channel Frequency Selection**-The IEEE 802.15.4 defines 27 different wireless channels. A network can choose to operate within a given channel set. Hence, the Physical Layer should be able to tune its transceiver into a specific channel upon the reception of a request from a higher layer.

In order to maintain a simple interface both MAC and PHY layer share a simple packet structure as presented in Figure 3.4 and this type of packet is know in the Anglo-Saxon literature as PHY Protocol Data Unit (PPDU). It is responsible by the encapsulation of all data structures from higher level of the protocol. The packet is divided in three basic components: a synchronisation header (SHR), a PHY header (PHR) and finally a variable length payload that contains the PHY layer service data unit (PSDU) as follows:

- The SHR consist basically in two fields, a preamble and a start of frame delimiter. The preamble consists in 4 bytes, this field allows a sufficient number of bits in order to achieve chip synchronisation and bit synchronisation. The start of frame delimiter (SFD) has a length of 1 byte and allows a receiver to set up the beginning of a packet.

- The PHR is a field with 8 bits where the most significant bit (MSB) is reserved and the other 7 bits are used to specify the frame length information, allowing to have packet with a total length of 127 bytes.

- The PHY payload only has one field called PSDU and it carries the data payload of the PHY protocol data unit PPDU, i.e., the SHR, PHR, and PHY payload together.

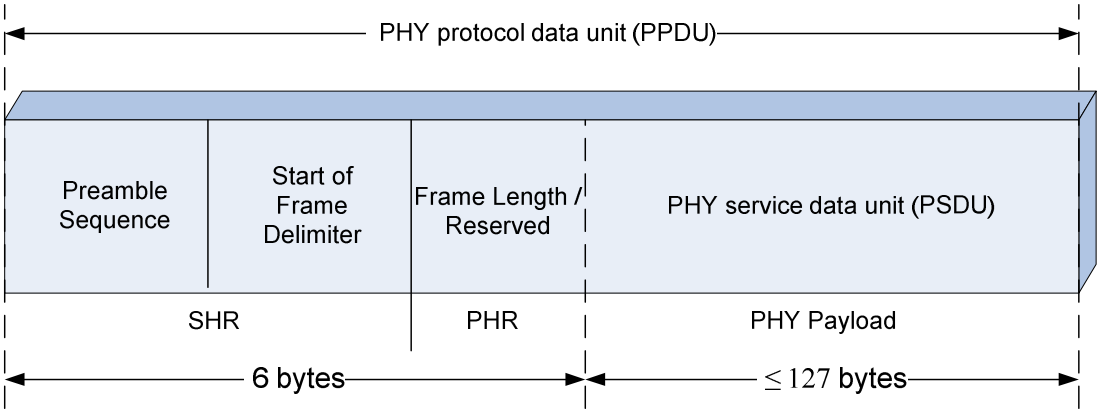


Figure 3.4. IEEE 802.15.4 PHY protocol data unit.

3.3 IEEE 802.15.4 MAC Layer

The IEEE 802.15.4 MAC Layer is designed to support a large number of application such industrial, medical and home applications for control and/or monitoring. The main function performed by the MAC sub layer is the access to the physical radio channel and it is responsible by: the generation of acknowledgment frames, support of personal area network (PAN) association and disassociation and the security control. Also provides optional star network topology function, generation of network beacons if the device is a coordinator and provides application support for the two possible network topologies of the standard (the star topology or the peer-to-peer topology) as presented in Figure 3.5.

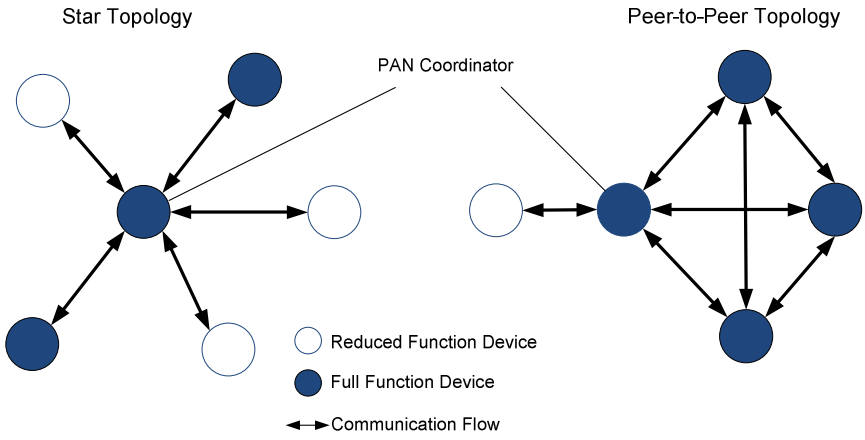


Figure 3.5. Star and peer-to-peer topology examples.

The star topology is formed around a Full Function Device (FFD), so the communication is established

between devices and a single central controller, called the PAN coordinator which is the only node allowed to form links with multiple devices. The function of the PAN coordinator includes not only the run of a specific application but it can also be used to initiate, terminate or route the communication in the network. So the PAN coordinator acts like the primary controller of the personal area network. All devices operating on a network of any type of topology mentioned shall have a unique 64-bit address. This address may be used for direct communication within the PAN, or a short address may be allocated by the PAN coordinator when the device associates and used instead. The PAN coordinator often will be powered by a continuous power supply, while the devices will most likely be battery powered. Applications that benefit from a star topology include: home automation, personal computer (PC) peripherals, toys and games, and personal health care [WPAN06]. The IEEE 802.15.4 MAC protocol supports two operational modes: the *non beacon-enabled mode* and the *beacon-enabled mode* which can be selected by the PAN Coordinator as presented in Figure 3.6. In the *non beacon-enabled mode* the PAN coordinator do not transmit regular beacons and so it transmits a data frame using a non-sloted carrier sense multiple access with collision avoidance CSMA/CA. In the *beacon-enabled mode* the beacons are periodically sent by the PAN Coordinator, so when a device wishes to transfer the data to a coordinator it first listen the medium for a network beacon frame. The beacon frame is responsible by the boundaries establishment for the beginning of a superframe while defining a time interval to exchange packets between different nodes. The medium access is basically a slotted CSMA/CA. This mode is used in applications that require a certain amount of bandwidth and low latency so that the PAN coordinator enables the allocation of some time slots in the superframe. These portions are called guaranteed time slots (GTSs) and they are used in the situation when the node needs to have guaranteed services.

As described before the IEEE 802.15.4 has different types of topologies, so the data transfer model is always related with the network topology. In the peer-to-peer mode a device will communicate with other devices in its vicinity while in the star networks the communication exchange will occurs between a PAN coordinator and a network device. In the presence of a star network two types of data transfer mechanism could exist, depending on whether the PAN coordinator is *beacon-enabled* or *non beacon-enabled*.

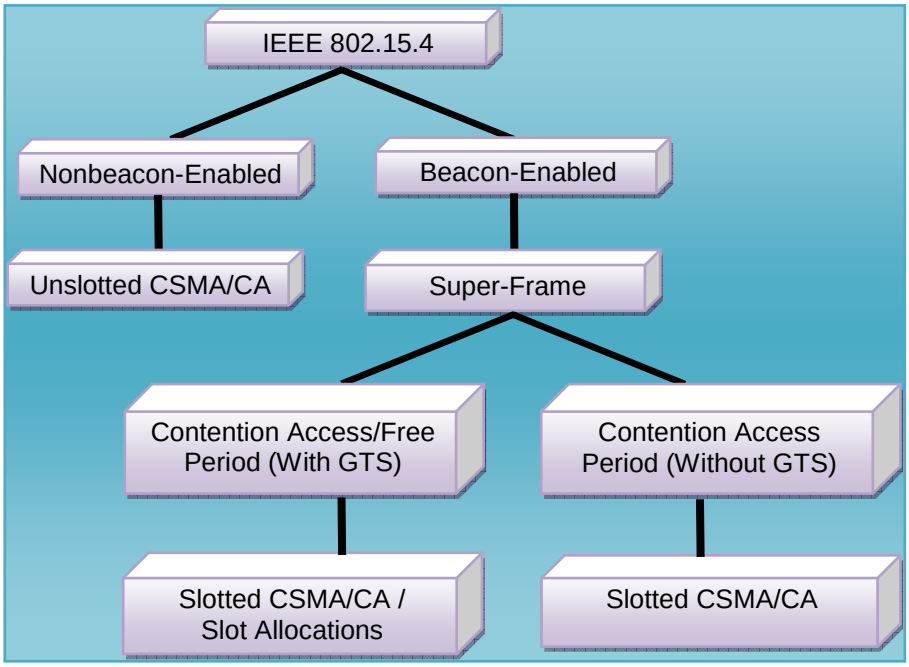


Figure 3.6. IEEE 802.15.4 operational modes.

Beacon Enabled – Star Topology

When we are in the presence of a *beacon-enabled* star topology, a network device that wants to send data to the PAN coordinator needs to listen for a beacon. In this case if a device doesn't have any GTS assigned, it transmits the data frame in the contention access period in accordance with the CSMA/CA procedure. If the device has already a GTS assigned, it needs to wait for the appropriate time within the superframe structure in order to transmit its data frame. After receiving the data frame, the PAN coordinator sends back an acknowledgement to the network device, as presented in Figure 3.7.

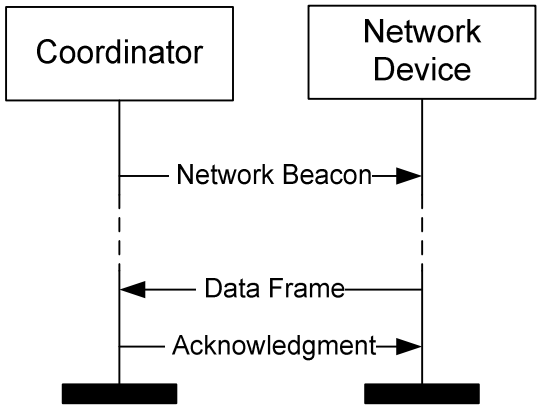


Figure 3.7. Star network - Communication to a coordinator in a beacon-enabled PAN.

In the cases when the PAN coordinator has data to send to a network device, it will set a special flag in its beacon. When the network device detects that the PAN coordinator has pending data for him, it will sends back a “data request” message and the PAN coordinator responds with an acknowledgment followed by the data frame, and finally an acknowledgement is sent from the network device in order to finish the transmission, as presented in Figure 3.8.

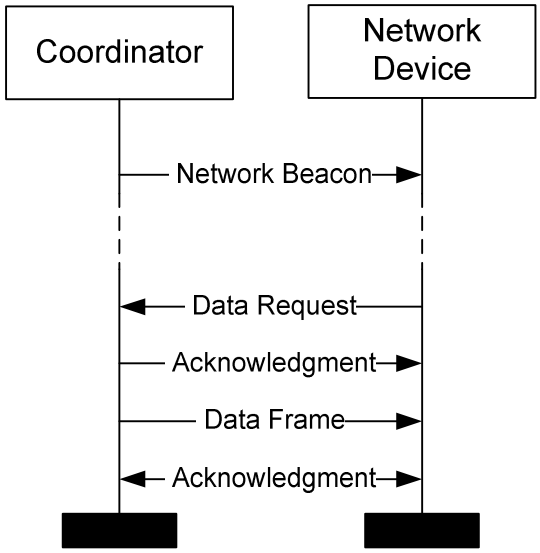


Figure 3.8. Star network - Communication from a coordinator a beacon-enabled PAN.

Nonbeacon Enabled – Star Topology

In the case with a nonbeacon-enabled star network, a network device that wants to transfer data sends a data frame to the PAN coordinator by using the CSMA/CA procedure. After correctly receive the data frame the PAN coordinator will reply to the network device, sending back an acknowledgement message, as presented in Figure 3.9.

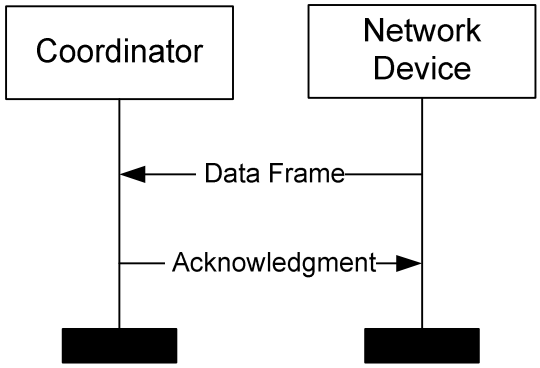


Figure 3.9. Star network - Communication to a coordinator in a nonbeacon-enabled PAN.

In the cases when the PAN coordinator requires a data transfer to a network device, it will keep the data until the network device sends back a data request message. After that the PAN coordinator sends an acknowledgement followed by the data frame. Finally, the network device acknowledges the reception of the data frame, as presented in Figure 3.10.

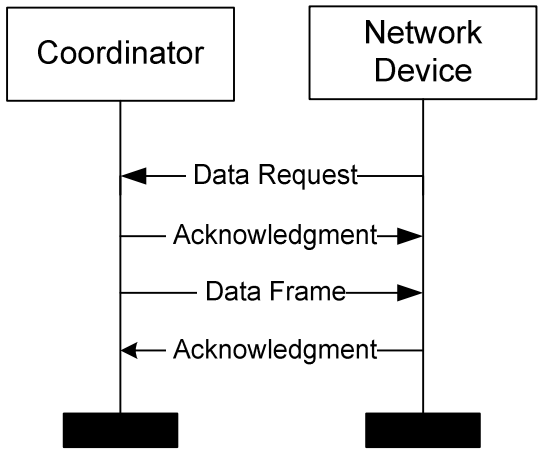


Figure 3.10. Star network - Communication from a coordinator in a nonbeacon-enabled PAN.

Peer-to-Peer Data Transfer

As shown before there are different types of data transfer transactions. In the first type, the data is transferred to a coordinator and the network device transmits the data. In the second one the exchanging of the data is transmitted from a coordinator in which the device receives the data. Finally, in the third one, the exchanging of the data is between two peer devices. When a star topology is used, only two first types of these transactions are used because data may be exchanged only between the coordinator and a device. In a peer-to-peer topology, data may be exchanged between any two devices on the network and consequently all three transactions may be used in this topology. So in the peer-to-peer topology, the strategy is ruled by the specific network layer that is managing the wireless network. A given network device may stay in reception mode, while scanning the radio channel for ongoing communications or it can send periodic “SYNCH” messages with other potential listening devices in order to achieve synchronisation.

3.3.1 The Superframe Structure

From the *beacon-enable mode* as described before, beacon frames are periodically sent by the PAN Coordinator in order to identify the PAN and to synchronize all nodes that are associated to it. The IEEE 802.15.4 standard allows an implementation of an optional superframe structure. The *Beacon Interval* (BI) is responsible by the time definition between two consecutive beacon frames and it includes an active period and an inactive period as shown in Figure 3.11. The active part is called

superframe and is divided in 16 contiguous time slots. If the inactive period exists, nodes can enter in the sleep mode and save energy while they are inactive.

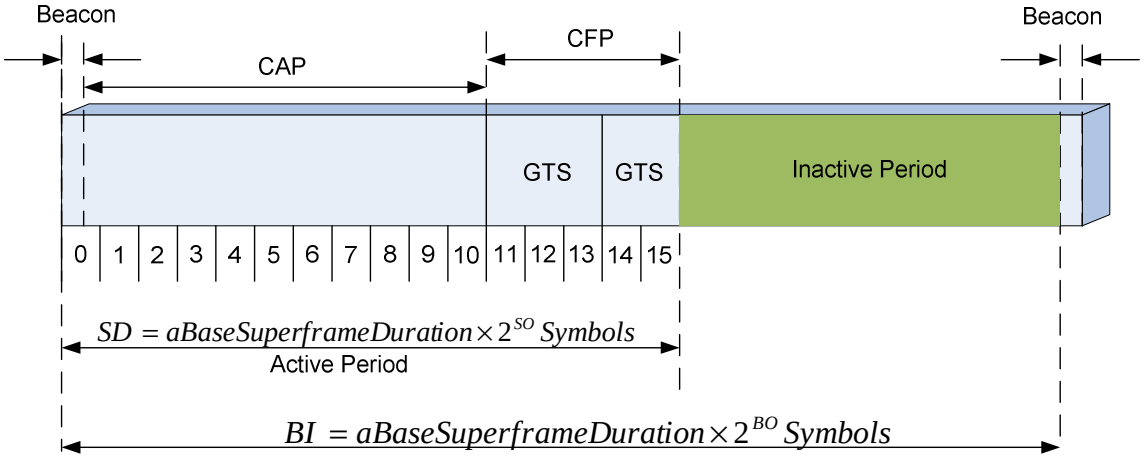


Figure 3.11. Beacon Interval and Superframe Structure.

As presented in Figure 3.11 the *Superframe Duration* (SD) and the *Beacon Interval* (BI) are calculated by using two parameters: the *Superframe Order* (SO) and the *Beacon Order* (BO). The *Beacon Interval* is given by:

$$BI = aBaseSuperframeDuration \cdot 2^{BO} \text{ , for } 0 \leq BO \leq 14$$

The SuperframeDuration, wich correspond to the active period is given by:

$$SD = aBaseSuperframeDuration \cdot 2^{SO} \text{ , for } 0 \leq SO \leq BO \leq 14$$

The *aBaseSuperframeDuration* can be defined as follows:

$$aBaseSuperframeDuration = aBaseSlotDuration \times 16 = 960 \text{ symbols}$$

Where *aBaseSlotDuration it* is the number of symbols forming a superframe slot when the superframe order is equal to 0 and has a value of 60 symbols.

As shown before the *aBaseSuperframeDuration* is equal to 960 radio symbols (where a symbol is equal to 4 bits), corresponding to 15.36 ms (assuming a bit rate of 250 kb/s in the 2.4GHz frequency band) and each time slot as a duration of 15.36/16=0.96 ms.

In Figure 3.11 the active part of a superframe is divided into 16 contiguous time slots that are divided in to three parts: the beacon, the Contention Access Period (CAP) and optionally the Contention-Free Period (CFP). The beacons are used to synchronise the attached devices, to identify the PAN and to

describe the structure of the superframes. The CAP is the period of time immediately following a beacon frame during which devices wish to transmit will compete for channel access using a slotted carrier sense multiple access with collision avoidance (CSMA-CA) mechanism, and the CFP is an optional feature in the IEEE 802.15.4 MAC and it follows immediately after the CAP, extending it to the end of the active portion of the superframe as shown in Figure 3.11. If we want to allocate any guaranteed time slot GTS they must be located within the CFP and the primary objective is the use of these time slots in applications that require bandwidth for delay critical applications.

3.3.2 MAC Frames

The IEEE 802.15.4 defines four MAC frame structures: the beacon; the data, the acknowledge and the MAC command frames.

As described before the beacon frame is used by the coordinator in order to transmit beacons. This type of frame is used to identify the network and its structure, wake up devices from the sleep mode to the listening mode and synchronize devices in the network, assuming an important role in the mesh and cluster-tree networks topology, especially because it can reduce energy consumption and extend battery lifetime due to synchronization. The entire MAC frame is used as a payload in a PHY packet. The MAC beacon frame structure is described in Figure 3.12.

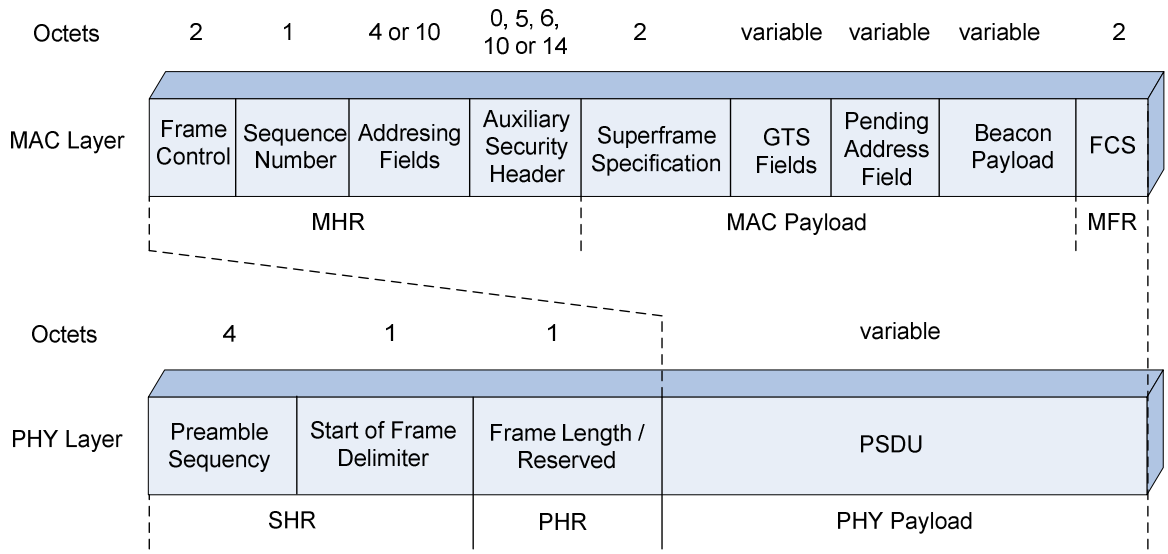


Figure 3.12. Schematic view of the beacon frame and the PHY packet.

In Figure 3.12 the active part of the beacon frame is constituted by three parts: the MAC header (MHR), the MAC payload and the MAC footer (MFR). The MHR contains information about the MAC frame control field, beacon sequence number (BSN), addressing fields, and optionally the auxiliary security header. The MFR contains a field of a 16-bit frame check sequence (FCS).

The data frame shown in Figure 3.13 is provided by NWK layer and the data inside the MAC payload is referred in the Anglo-Saxon literature as MAC Service Data Unit (MSDU). The fields in this frame are similar with the one presented previously in the beacon frame except the Superframe, GTS and pending address field that do not exist in this particularly data frame.

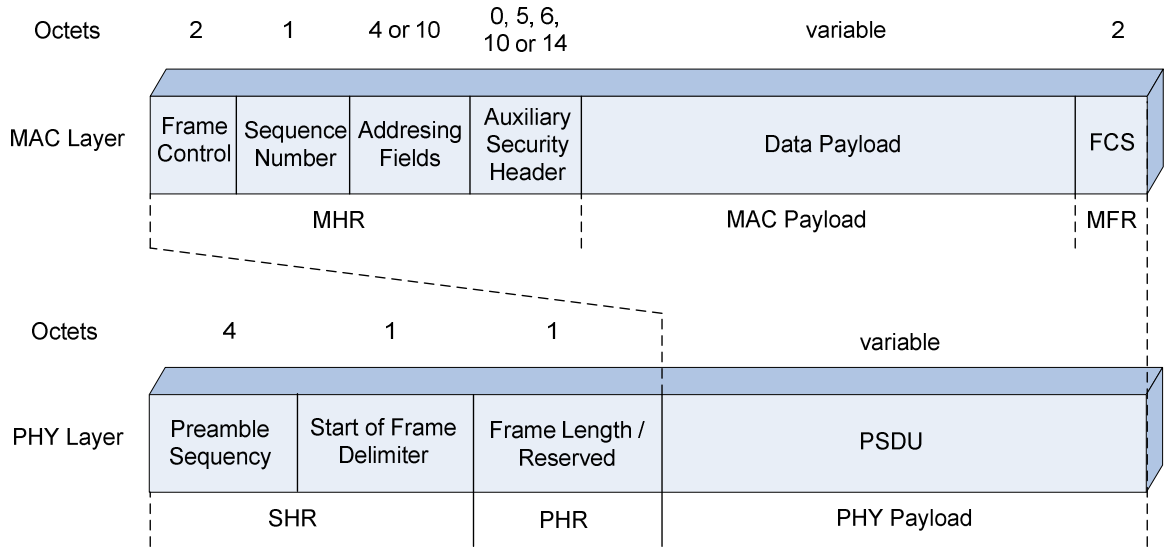


Figure 3.13. Schematic view of the data frame and the PHY packet.

The MAC payload is prefixed with an MHR and appended with an MFR. The MHR contains the Frame Control field, data sequence number (DSN), addressing fields and optionally the auxiliary security header. The MFR is composed by a 16-bit FCS. The MHR, MAC payload, and MFR together form the MAC data frame, (i.e., MPDU).

The MPDU is passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with an SHR, containing the Preamble Sequence and SFD fields, and a PHR containing the length of the PHY payload in octets. The preamble sequence and the data SFD enable the receiver to achieve symbol synchronization. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU) [WPAN06].The acknowledgment frame, presented in Figure 3.14 has a MHR and a MFR field and does not have a MAC payload. The MHR contains the MAC Frame Control field and DSN, and the MFR is composed by a 16-bit FCS. This type of frame is the simplest MAC frame format in the IEEE 802.15.4 standard. The acknowledgment frame is sent whenever a device confirms a correctly reception of a packet that was previously sent to it.

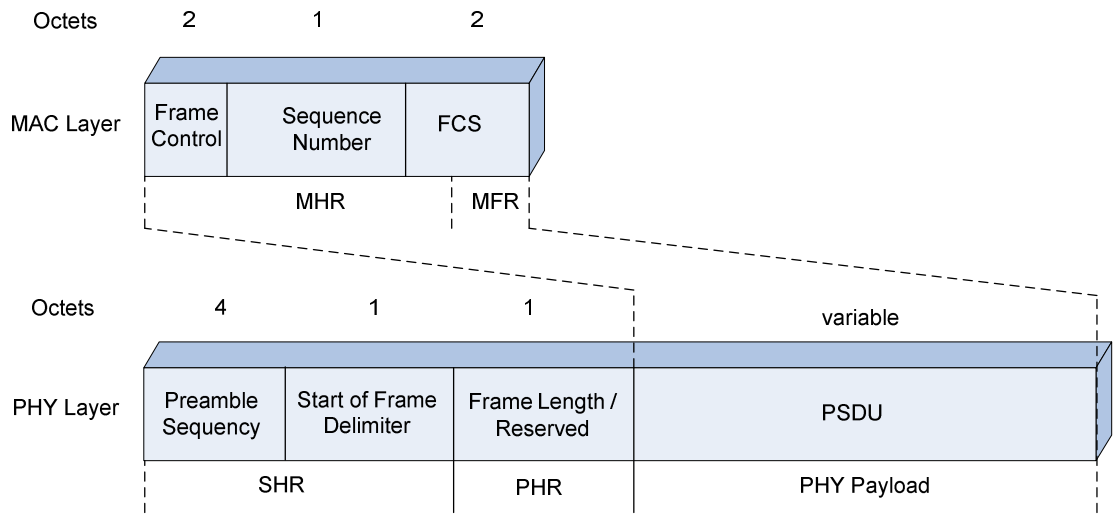


Figure 3.14. Schematic view of the acknowledgment frame and the PHY packet.

The MAC command frame presented in Figure 3.15 is used to request association or disassociation. The MAC payload contains the command type field that is responsible to determinate if the type of the command is a association request or a data request, while the command payload field contains the command itself. In Table 3.2 the different command types are described.

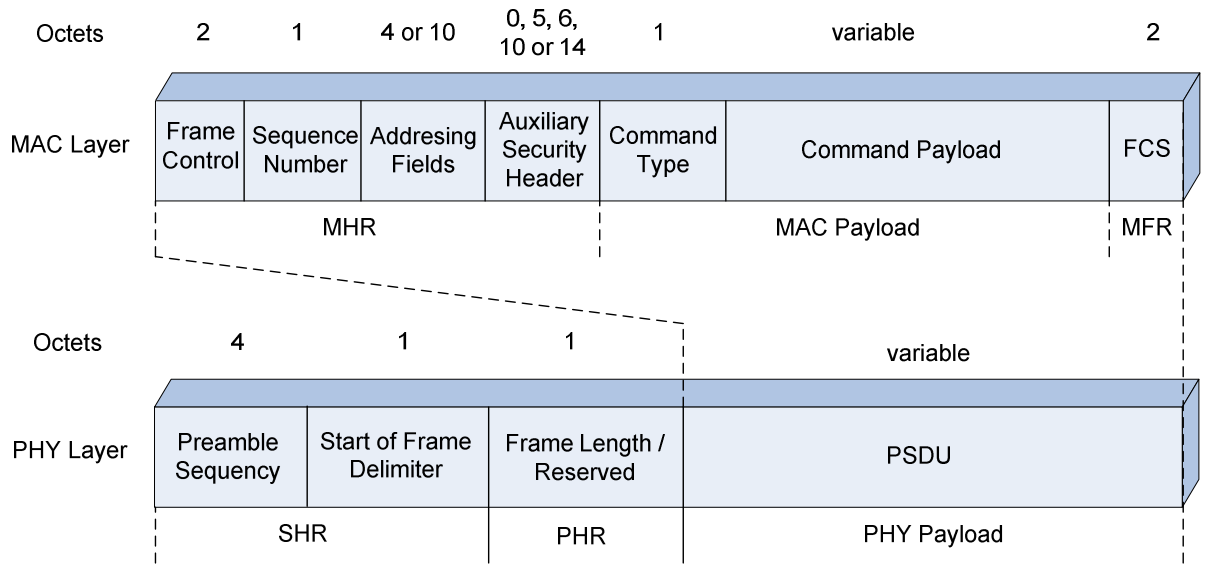


Figure 3.15. Schematic view of the MAC command frame and the PHY packet.

Table 3.2. Command Frame Types.

Command Frame Identifier	Command Name	Reduced Function Device (RFD)	
		Tx	Rx
0x01	Association Request	X	
0x02	Association Response		
0x03	Disassociation Notification	X	X
0x04	Data Request	X	X
0x05	PAN ID Conflict Notification	X	
0x06	Orphan Notification	X	
0x07	Beacon Request		
0x08	Coordinator Realignment		X
0x09	GTS Request		
0x0a - 0xff	Reserved		

All the command frames types defined by the MAC layer that a reduced function device may send or receive are described in Table 3.2, while an FFD device could transmit and receive all the command frame types shown in Table 3.2, with the exception of the GTS request command.

In the IEEE 802.15.4 MAC layer between two successive frames transmitted an IFS period must be inserted. The IFS depends on whether the transmission transaction is acknowledged or unacknowledged. When the acknowledgement is received the IFS follows the acknowledgment frame and when the frame length do not exceeds the *aMaxSIFSFrameSize*, the acknowledgment must be followed by a short IFS (SIFS) period and the duration should be at least *aMinSIFSPeriod*. In the cases that the frame length exceeds *aMaxSIFSFrameSize*, the acknowledgment must be followed by a long IFS (LIFS) period as shown in Figure 3.16.

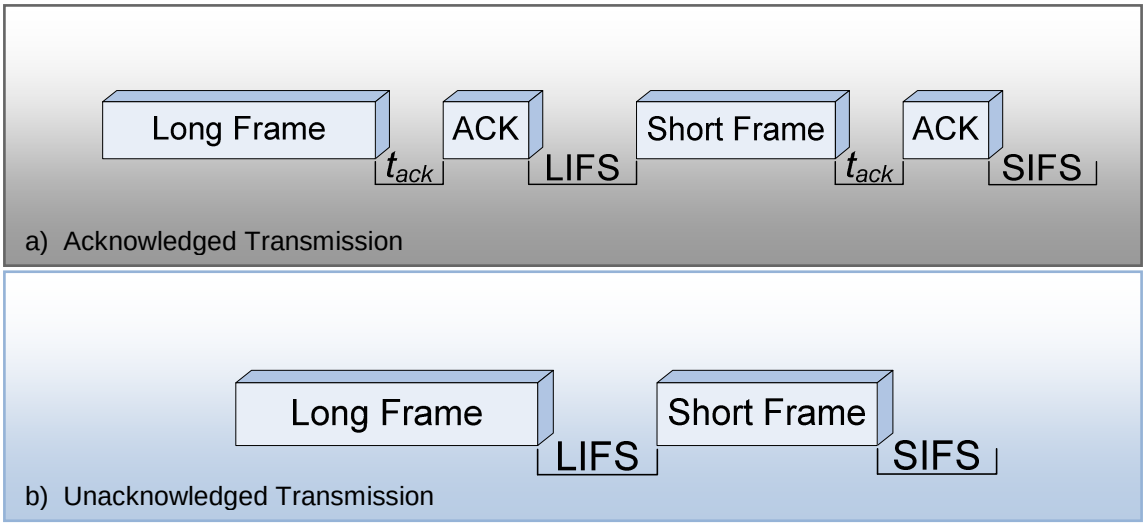


Figure 3.16. The Interframe Spacing (IFS) in a) Acknowledged Transmission and b) Unacknowledged Transmission.

The minimum LIFS and SIFS periods for the three PHY layers are shown in Table 3.3.

Table 3.3. Minimum LIFS and SIFS period.

PHY	macMinLIFSPeriod	macMinSIFSPeriod	Units
868–868.6 MHz BPSK	40	12	Symbols
902–928 MHz BPSK	40	12	Symbols
2400–2483.5 MHz O-QPSK	40	12	Symbols

3.4 CSMA/CA Algorithm

The IEEE 802.15.4 standard defines a CSMA-CA algorithm that should be used before the transmission of data frames or MAC command frames transmitted within the CAP, unless the frame can be quickly transmitted due to an acknowledgment of a data request command. The CSMA-CA algorithm shall not be used for the transmission of beacon frames in a beacon-enabled PAN, acknowledgment frames or data frames transmitted in the CFP [WPAN06]. Two versions of the CSMA/CA mechanism were created: (i) slotted CSMA/CA algorithm – used in the beacon-enabled mode and (ii) non-slotted CSMA/CA algorithm – used in the non beacon-enabled mode. Both

approaches, use a basic time unit called *Backoff Period* (BP), which is equal to *aUnitBackoffPeriod* = *20 Symbols* (0.32 ms). When using slotted CSMA/CA, each operation (channel access, backoff count, CCA) can only occur at the boundary of a BP. Additionally, the BP boundaries must be aligned with the superframe time slot boundaries, Figure 3.11. In non-slotted CSMA/CA the backoff periods of one node are completely independent of the backoff periods of any other node in a PAN. The CSMACA algorithm, represented by the flowchart of the Figure 3.17, is invoked when a packet is ready to be transmitted. This algorithm, maintains three variables for each packet:

1. The *Number of Backoffs* (*NB*) represents the number of times the CSMA/CA algorithm was required to experience backoff due to unavailability. It is initialised to zero before each new transmission attempt.
2. The *Backoff Exponent* (*BE*) enables the computation of the backoff delay, and represents the number of backoff periods that need to be clear of channel activity before a transmission can occur. The backoff delay is a random variable between 0 and (2^{BE}-1).
3. The *Contention Window* (*CW*) represents the number of backoff periods during which the channel must be sensed idle before accessing the channel. This type of variable is only used with slotted CSMA/CA. This value shall be initialized with the value *CW* = 2 before each transmission attempt and reset to two each time the channel is assessed to be busy (corresponding to two CCAs). Each backoff period channel sensing, is performed during the 8 first symbols of the BP.

The slotted CSMA/CA can be summarised in five steps as follows.

Step 1. First, the number of backoffs and the contention window are initialised (*NB* = 0 and *CW* = 2). The backoff exponent is also initialised to *BE* = 2 or *BE* = min (2, *macMin*BE) depending on the value of the *Battery Life Extension* MAC attribute. *macMin*BE is a constant defined in the standard (by default equal to 3). After the initialisation, the algorithm locates the boundary of the next backoff period.

Step 2. Second, the algorithm performs random waiting delay for collision avoidance. It starts counting down a random number of BPs uniformly generated within [0, 2^{BE}-1]. The countdown must start at the boundary of a BP. In order to disable the CA procedure at the first iteration, *BE* must be set to 0, and thus the waiting delay is null and the algorithm goes directly to Step 3.

Step 3. When the timer expires the algorithm then performs one Clear Channel Assessment (CCA) operation at the BP boundary to assess channel activity. If the channel is busy, the algorithm goes to Step 4, otherwise, e.g. the channel is idle and the algorithm goes to Step 5.

Step 4. If the channel assessed is busy, then the *CW* is re-initialised to 2, NB and BE are incremented, however the BE must not exceed *macMax*BE (by default equal to 5).

Incrementing *BE* increases the probability of having greater backoff delays. If the maximum number of backoffs (NB > *macMaxCSMABackoffs*, where *macMaxCSMABackoffs* = 4) is reached, the algorithm reports a failure to the higher layer, otherwise, it goes back to (Step 2) and the backoff operation is restarted.

Step 5. If the channel assessed is idle, the *CW* value is decremented. The CCA is repeated until $CW \neq 0$ (Step 3). This way the algorithm ensures performing two CCA operations to prevent potential collisions of acknowledgement frames. When channel is again sensed as idle ($CW = 0$), the node attempts to transmit. Collisions may still occur if two or more nodes are transmitting at the same time.

The non-slotted CSMA/CA is very similar with a few exceptions as follows.

Step 1. In the first step the CW variable is not used, since the non-slotted has no need to iterate the CCA procedure after detecting an idle channel. Hence, in Step 3, if the channel is assessed to be idle, the MAC protocol immediately starts the transmission of the current frame. Second, the non-slotted CSMA/CA does not support the battery life extension mode and BE is always initialised to the *macMinBE* value.

Step 2, 3 and 4. These steps are very similar to the slotted CSMA/CA algorithm. The only difference is that the CCA starts immediately after the expiration of the random backoff delay generated in Step 2.

Step 5. The MAC sub-layer starts immediately transmitting its current frame just after a channel is assessed to be idle by the CCA procedure.

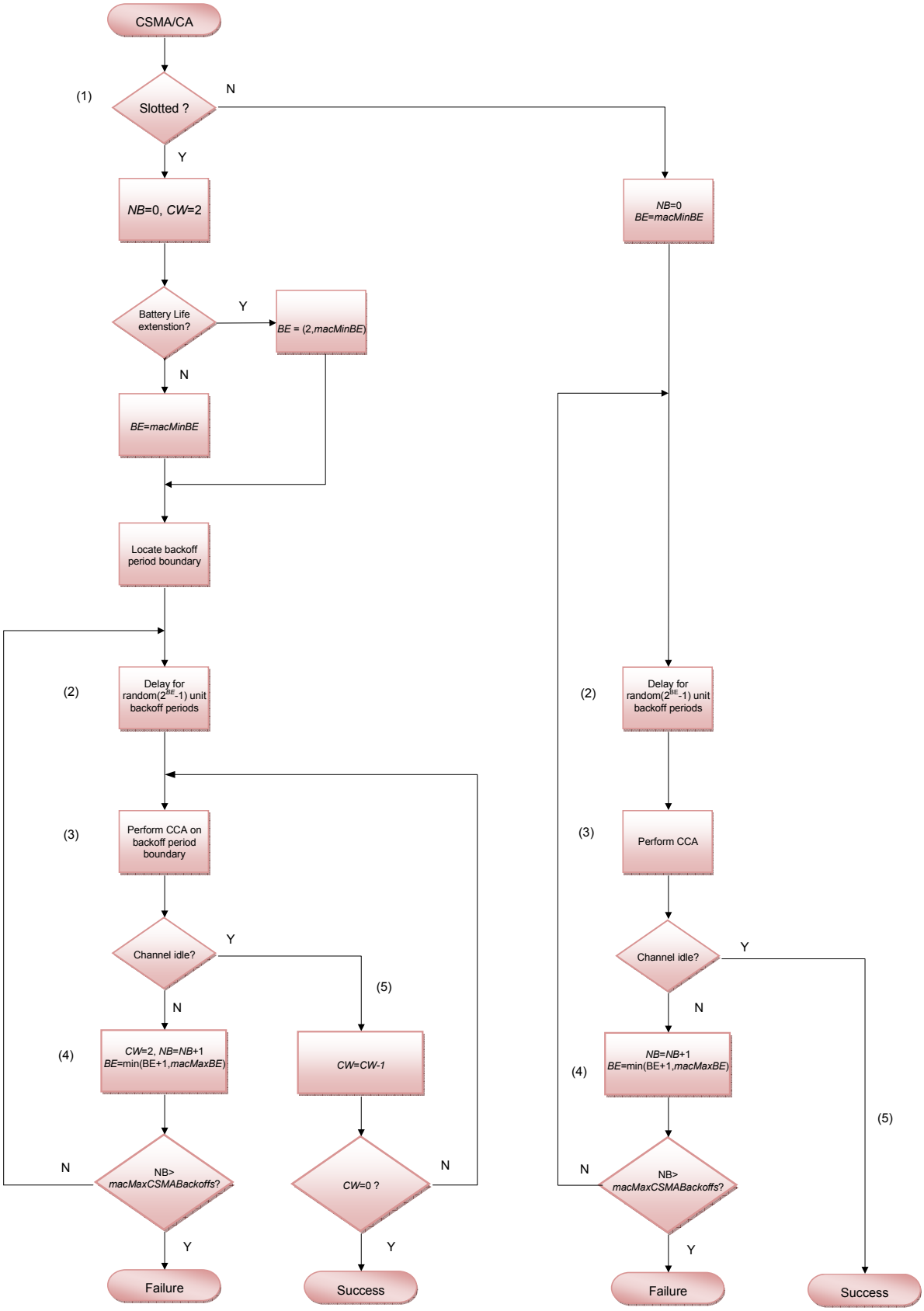


Figure 3.17. CSMA-CA algorithm [WPAN06].

3.5 Conclusions

The IEEE 802.15.4 protocol was designed to become a communication standard for low-cost, low-rate wireless local area network (for LR-WPANs). It was optimized for monitoring and control applications and it is responsible to define the physical layer (PHY) and medium access control (MAC) sublayer specifications for low-data-rate wireless connectivity in fixed, portable, and moving devices with no battery or very limited battery consumption requirements, while operating in a personal operating space (POS) of 10 m. Similar to other low power wireless standards such as low-rate, low-power consumption, low-cost wireless networking and mesh networking, it helped to differentiate it from the field of other existing IEEE technologies and these key features are the ones which typically fit the requirements of WSNs.

The ZigBee® is another alliance that defines a set of communication protocols for low-data-rate short-range wireless networking and adopted the IEEE 802.15.4 Physical and Data Link Layers, building up of IEEE 802.15.4 Physical and Data Link Layers the Network and Application Layer, defining a full protocol stack for LR-WPANs.

The IEEE 802.15.4 standard has the ability to provide very low duty cycles, which is particularly interesting for WSN applications, where energy consumption and network lifetime are the main concerns. Besides it has a very simple frame structure that provides a maximum payload of around 130 bytes that typical is the small amount of date required for these applications. It was also developed to provide CSMA/CA, message acknowledgment to ensure packet delivery and energy detection to analyse channels and links. It also supports multiple network types, including peer-to-peer, star, cluster tree and mesh networks. For networks that require a more deterministic delivery, it can support beaconed networks in order to provide a minimum service guarantee for the corresponding nodes.

Chapter 4

Overview of MAC Protocols for Wireless Sensor Networks

This chapter provides an overview of the sensor platforms used in WSN and how the approach of designing a protocol based in a WSN protocol layer could reduce the design complexity. Then a survey of the most well know Medium Access Control (MAC) protocols is presented and the solutions proposed to solve the simple task of sharing the wireless communication medium between several nodes are discussed. Finally, some techniques will be presented based on others IEEE networking standards that bring new approaches when trying to solve one of the main problems within WSN, which is energy-efficiency.

4.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) may be formed by thousands of small, low-power sensor devices designed to sense information about the surrounding environment and then transmit the information to other network nodes or to a base station. Research involving these devices has proposed a wide range of applications, which include atmospheric monitoring, wildlife tracking, physical perimeter intrusion detection, medical monitoring, homeland security, nuclear, biological, and chemical (NBC) monitoring and a wide range of military applications [MPSC02] [YaSi03] [ASSC02].Because of their unique characteristics such as the mobility, the communication capabilities and the large scale deployment, this type of network has gained a remarkable attention in the last years not only in the academic community but also in the industrial community.

While designing a MAC Protocol to this type of networks, it is important to extend the range of potential deployment scenarios, from sparse networks, where the inter-node communication is difficult to maintain to extremely dense networks, where access to the wireless channel must be carefully arbitrated to avoid collisions and overhearing. Another important characteristic of these networks is the fact that these networks share the same characteristics as wireless networks based on the IEEE 802.11 standard, commonly known as Wi-Fi networks. However they are unique in many ways. The following sections describe the characteristics of some sensor node platforms that make them different from other wireless devices and explore the implications of these differences in terms of MAC protocol behaviours.

4.1.1 Sensor Networks Platforms

Sensor Network Platforms (SNP) have become more advanced and relatively inexpensive in the past years; so, they become deployed in large scale. Because these platforms are used in a large number of applications it is important to have small sensor nodes in order to prevent node destruction and tampering.

This topic of research involving small SNP has been studied in the last years and the Smart Dust project at the Department of Electrical Engineering and Computer Sciences in the University of California, Berkeley, sought to develop wireless sensor nodes, incorporating micro fabricated sensors, optical receiver, passive and active optical transmitters, signal processing and control circuitry and power sources that are one cubic millimetre in size [KaKP99]. In many deployment scenarios such as medical applications or military ones, the devices are abandoned once their power supplies are exhausted; so, it is important build small, inexpensive and energy-efficient devices.

Annex 1 presents a list of several sensor platforms currently available. These devices are characterised by 8-bit processors, where the programme plus data memory can go from 2 to 512 kB of random access memory (RAM) in the most common platforms. However the most advanced may have 32 MB of static random access memory (SRAM), because Static RAM is a type of RAM that holds its data without external refresh, for as long as power is supplied to the circuit. In the Annex 1 the more robust platforms have more memory. The amount of memory used will affect the energy consumption of the system because, typically, some of the existing memories need a constant current flow to maintain the stored data and more memory means more energy consumed, therefore reduces the network lifetime.

Half of the 15 most well know platforms in the market use the IEEE 802.15.4 compliant CC2420 radio [CC2409], which is a ZigBee® RF transceiver, one has a AT86RF230 radio transceiver [AT8609] for IEEE 802.15.4 (and ZigBee®, too), three others use the ChipCon CC1000 radio [CC1009] that can be used in the 315/433/868 and 915 MHz frequency bands, and the others use IEEE 802.15.1 Bluetooth radios.

The different sensor networks platforms have numerous operating systems (OS) and the most popular is TinyOS [TINY09] which runs in almost all platforms that are presented in Annex 1. Other examples include SOS [HKSK05], MANTS [BCDD05] and Contiki [DuGV04].

In Table 4.1 power consumptions, memory and other data related are presented for the sensor platforms mentioned in Annex 1, in order to have a conceptual view of which parameters we intended to optimize in the MAC protocol that will be proposed in this research.

Table 4.1. Power consumption of different sensor platforms.

		MICAz	**IRIS**
Power Consumption	**Sleep**	< 45 µW	24 µW
	Receive	59,1mW	48 mW
	Transmit	52,2 mW	51 mW
RF Transceiver		CC2420	AT86RF230
Data Rate [kb/s]		250	250

4.1.2 Wireless Sensor Network Protocol Stack

In order to reduce the design complexity, most of the existing networks are organized as a stack of layers, each one built upon the one below it, depending on the network, on the number of layers, on the name of each layer and on the contents of each layer. Their functions will differ from network to network. The main objective of each layer is to offer certain services to the higher layers, protecting those layers from the details of how the offered services are actually implemented, so each layer behaves like a virtual machine, offering certain services to the layer above it. When combined with the layers bellow, they implement some functionality.

The seven-layer Open Systems Interconnection Reference Model (OSI model) is used to describe the sensor network protocols. However the seven layers in the OSI model are often reduced to the following five layers: Application, Transport, Network, Link and Physical Layers [KuHH05].

The seven layers presented in Figure 4.1 are the same proposed by the authors in [YeHE02b] to describe the radio communication stack on the Mica Motes developed at USC/ISI and UCLA, which gives an example of a layered abstraction for sensor network protocols. Five layers of the WSN protocol stack are presented on detail as follows.

- **Physical Layer (PHY):** The PHY Layer, commonly referred in the Anglo-Saxon literature as Layer 1 is responsible by the implementation of the functions related with the network communications hardware, such as the transmission and reception of messages, or by others words it is responsible by the moving bits(symbols) between the network nodes. The PHY receives analog symbols from the medium and converts them to digital bits for further processing in the higher layers from the wireless sensor network protocol stack. The PHY functions available in most transceivers are the selection of a frequency channel and the transmit power, the modulation transmitted and demodulation of received data, symbol synchronization and clock generation for received data. The radio transceiver typically has three states: sleep, transmitting and receiving. The PHY layer of the transceiver may also include additional functions, which could reduce the processing requirements of Micro-Controller Unit (MCU). For example, an IEEE 802.15.4 compliant PHY includes: data frame synchronization for perceiving the start of an incoming frame; clear channel assessment for detecting ongoing traffic in a frequency channel; Received Signal Strength Indicator (RSSI) and Link Quality Indication (LQI) for measuring signal strength and estimating link quality to neighbouring nodes; Cyclic Redundancy Check (CRC) calculation for checking bit errors on received frames; data encryption/decryption for improving network security; and automatic acknowledge transmissions after received frames. Since these features are implemented in a more efficient way in the physical layer, it can improve overall network energy efficiently. Yet, the increased complexity increases hardware costs.
- **Data Link Layer (DLL):** The data link layer (DLL) is the layer between the physical and the

network layer. Typically, the DLL have a MAC sublayer and a Logical Link Control (LLC) module. The MAC sublayer provides a fair mechanism to share access to the medium among other nodes and it is responsible of how and when it should use the PHY functions for accessing the shared physical medium by the device. One possible scenario is when multiple nodes intend to transmit the data but they are not allowed to use the same transmission medium and in this case a Medium Access Control must be implemented in order to decide which node gains access to the shared medium. Other function includes the control of the topology, so it must have knowledge of the initial topology and it should have the ability to react to a changing topology. Hence, MAC plays a key role in the maximization of a node's energy efficiency. The LLC operates above MAC and is responsible for encapsulating message segments into frames and adding appropriate header information, with destination and source addresses, control and sequencing information, and CRC calculation. According to this information, a desired destination node can receive a frame, ensure frame integrity and maintain proper sequencing of frames.

- **Network layer:** The network layer is responsible for the function involving network self-configuration and data routing. When configuring the network topology, the network layer selects an appropriate operation mode for the node and determines the more suitable neighbours for association and then form communication links. The network topology is updated after link failures or at regular intervals for assuring network connectivity and optimizing network lifetime by balancing energy consumption among other nodes in the network. A routing protocol executed in the network layer performs end-to-end data routing. The routing protocol decides a suitable next-hop node to forward each data frame in order to the frame eventually reaches the desired destination. Therefore, the source and destination nodes are connected together by a chain of hops.

  For maximizing network lifetime, it is important to evaluate an optimal next-hop node for each data frame according to the link quality and delay requirements, so one of the main goals of the network layer is to provide information to the transport layer on the link quality, because if the data was not correctly received this shows that there is a problem in the transmission. This problem can be due to a topology change when sending data to a node that is no longer active (e.g. because energy failure or it is damaged) or the network is congested and the node receives more data than it can process. The transport layer will find out that the level-of-quality of the link will decrease below a threshold and initiate an improvement process.
- **Transport Layer:** The transport layer is responsible to provide a mechanism to establish a connection between nodes and it will be responsible to perform flow control and to regulate traffic flow through the network according to observed congestion. It must provide also mechanisms to keep the quality of a link based on the information provided by the network layer. The transport layer is also responsible by the upper layer error control that detects missing or corrupted frames not perceived by the DLL. Due to the low transmission power

levels and harsh operation conditions in WSNs, link reliability is much worse than in conventional wired and wireless networks. Thus, it is more feasible to perform flow and error control separately for each hop than from end-to-end, as in conventional networks [KaWi05]. In addition, the transport layer performs fragmentation when dividing upper layer application data into small segments suitable for DLL. On the other hand, the transport layer reorders and reassembles received data segments into data packets applicable for application layer.

- **Application Layer:** The application layer, also know as layer 5 is the last layer of the Wireless Sensor Network Protocol Stack. It offers the network services and the actual functionality for the node by the use of network applications. Some protocols exploit the collaborative nature of the sensors in the application layer in order to reduce the redundancy of the transmitted messages. This can be achieved, by using advanced signal processing techniques, such as the various data aggregation methods and distributed source coding or spatial sampling. In this layer, we may have several processes that can be executed in parallel, e.g. sensing applications for various sensors, actuator and node diagnostics applications and network configuration application. Application layer protocols are responsible for the management of data aggregation and distributed compression schemes, the definition of the format for the exchanged messages and the order of message exchanges between different processes. However, the performance depends largely on the specific MAC protocol and motivates the study of cross-layered protocols all over in the community research.

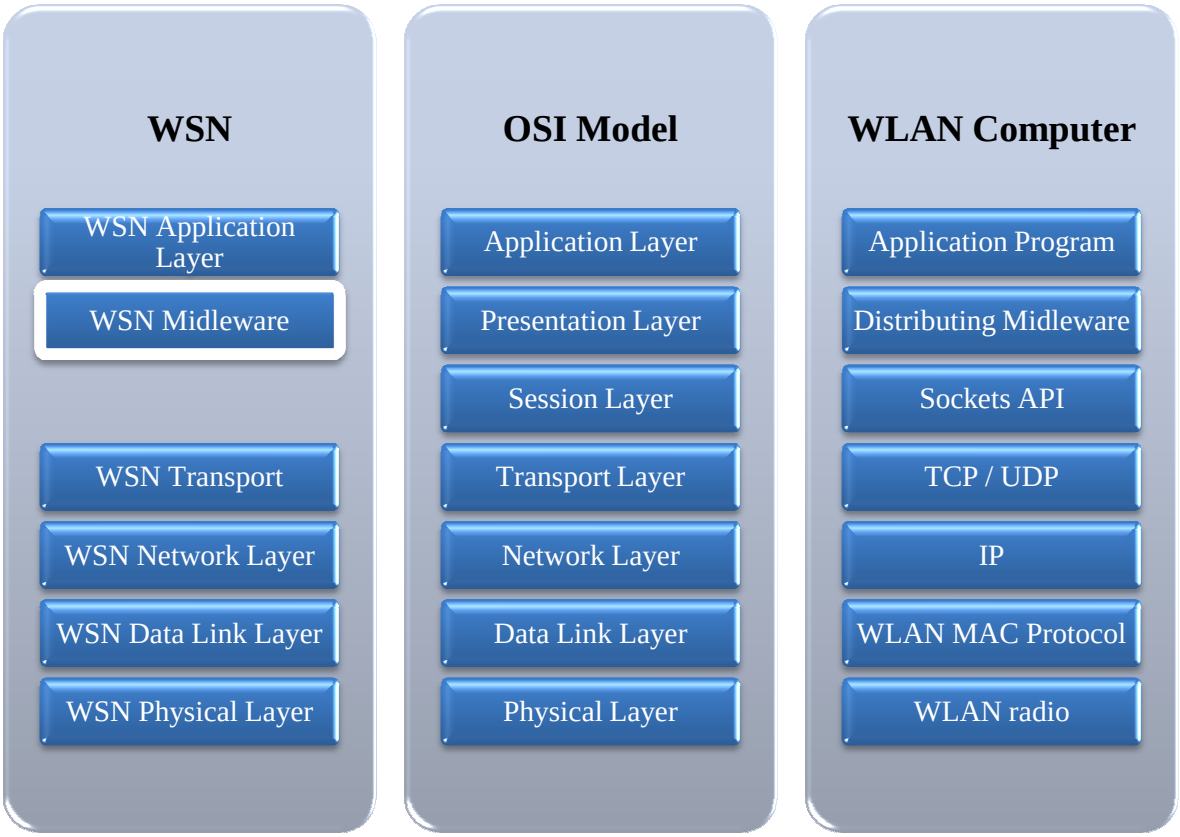


Figure 4.1. OSI model, WSN, and distributed system in WLAN protocol layers.

4.2 Requirements/design constraints for MAC protocols

Energy efficiency can be achieved through various layers of the communication protocol stack. The authors in [HaSm00] have focus their study mainly in the physical layer. However, due to the physical limitations, the WSN research has recently shifted to the higher layers. Besides, the MAC sub layer is part of the data link layer and operates on top of the physical layer. It provides a fair mechanism to share access to the medium among other nodes and is responsible by the determination of how and when the PHY functions should be used.

A MAC protocol is responsible by the management of the radio transmission and reception through the shared wireless medium, and provides connections for overlaying routing protocol. Hence, it has an high effect on network performance and energy consumption. Because of this unique characteristic, a large number of protocols have emerged in the last years intended to optimize energy efficiency. They differ, among others, in the types of media they exchange and in the performance requirements for which they are optimized. However, and despite the differences, the sensor networks protocols use some techniques for collision avoidance and medium access from the current IEEE 802 networking standards. Hence it is important to know some of these techniques in order to understand the following description of features for the WSN protocols described later in next sections. It is

important to understand the power conservation techniques used by these protocols and understand why these protocols are insufficient for WSN deployments.
In the traditional wireless protocols, the most important performance requirements are the throughput, fairness, efficiency, delay, stability and low overhead. In MAC protocols the overhead can result from collisions or can result from the exchange of extra control packets. The collisions will happen at the receiver if the MAC protocol allows two or more nodes to send packets simultaneously, resulting in an impossibility of the receiver to decipher the packet correctly. In this case the transmitters will use the upper layers in order to retransmit the packet. When dealing with a real-time application or time-critical application, it is important to provide deterministic or stochastic guarantees on delivery time or minimal available data rate. In this cases, it is important the concept of priorities in order to handle first the priority packets.

4.2.1 Techniques from IEEE 802.11 networking standards

The IEEE 802.11 standard for wireless local area networks (WLANS) specifies both Medium Access Control (MAC) and Physical Layer (PHY) for wireless network devices. The Wireless LANs based on IEEE 802.11 technologies (also known as wireless Ethernet and WiFi) are currently deployed in university departments, business offices, coffee shops and homes. Many universities install IEEE 802.11 base station across their campus, allowing students to send and receive e-mail or navigate the Web from anywhere on the campus (e.g. library, dorm room, class room, or outdoor campus bench) [KuRo08].

In the IEEE 802.11 wireless LANs the base station (also known as access point) has an important role in the wireless network infrastructure because it is responsible for sending and receiving data to a and from a wireless host that is associated with a base station and coordinates the transmission of multiple associated wireless hosts.

In the cases when the hosts are associated with a base station the operation mode is called **infrastructure mode** since all traditional network services are provided by the network to which a host is connected via base station. In **ad hoc networks** there is no base station and the nodes can only transmit within the link coverage and need to organize themselves, routing among themselves.

The characteristics of the wireless connectivity addressed by IEEE 802.11 are the relative unreliability of wireless links caused by path loss, interference, and multipath propagation, and the hidden terminal problem, described below. Link reliability is achieved via link-layer acknowledgments, a technique also used in most WSN MAC protocols.

In the presence of a wireless medium, the channel is not uniformly shared. As a consequence, the hidden terminal problem may appear. The hidden terminal is an interfering node that is out of the carrier sense range of the sender. However, it is in the interference range of the receiver and in this situations such transmissions cannot be detected by the sender, causing a potential collision at the

receiver. Consider the example presented in Figure 4.2, where there are three nodes A, B and C, and the nodes A and C cannot hear each other due to distance or obstacles. However, A and B are in mutual range. So, A can hear B and, vice versa. The same thing happens with nodes B and C. Assuming that node A wants to transmit a packet to B and, after some time, node C also decides to transmit a packet to B, if C performs a carrier sense the result is an idle medium since C cannot hear A's transmissions. When C starts to transmits the packet a collision will occur at B and both packets are useless. Techniques such as Carrier Sense Multiple Access (CSMA) and Collision Avoidance (CA) reduce collisions but do not eliminate them [FuGa97].

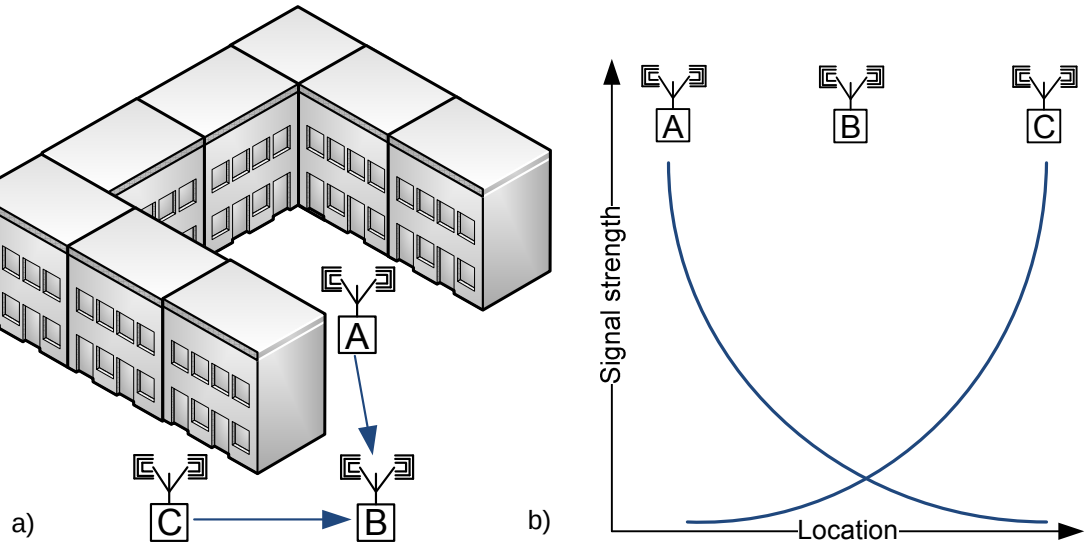


Figure 4.2. Hidden terminal problem caused by obstacle a) and fadding b).

4.2.2 IEEE 802.11 Distributed Coordination Function (DCF)

As referred before one solution to the hidden terminal problem is the use of one scheme involving Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) and the basic IEEE 802.11 MAC layer uses the distributed coordination function (DCF) that dictates distributed medium access rules such as CSMA/CA and random backoffs before packet transmission. It also incorporates a RTS/CTS handshake based on the MACAW protocol [Bhar94]. When a node A wants to transmit a packet to node B, it first sends out a short frame Request-to-Send control frame indicating the length of the data packet and the length of the Acknowledgment (ACK) packet to be transmitted on the channel. If node B do not detects other ongoing transmission it will reply with a Clear-to-Send message, which reserves the medium and then the node A can send the packet to node B. All other nodes in the

proximity of A and B that hear the RTS or CTS packets know that there is a pending data transmission and can avoid the interference with those transmissions. The RTS, CTS, DATA and ACK frame are shown in Figure 4.3.

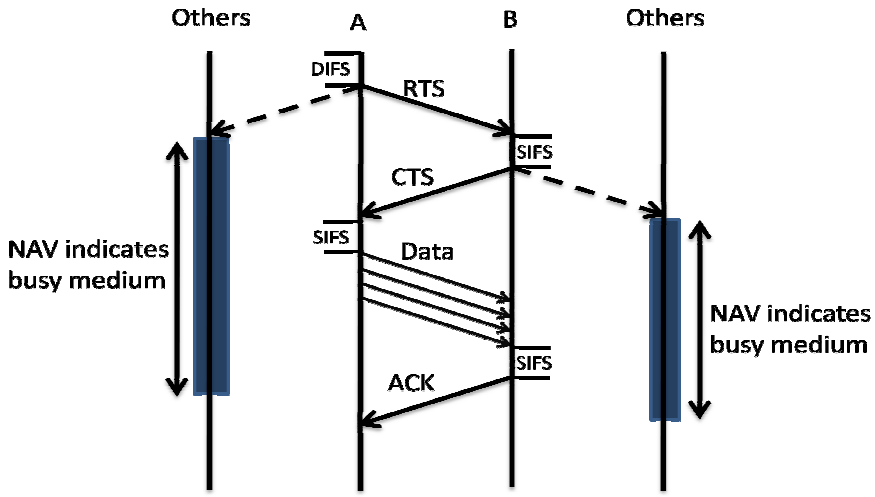


Figure 4.3. RTS/CTS handshake in IEEE 802.11.

The IEEE 802.11 control packets have a duration field which indicates the duration of the data frame to be transmitted. All other nodes that are not part of the ongoing transmission but overhear an RTS or a CTS message, will record this value and set an internal timer called Network Allocation Vector (NAV) to the remaining duration indicated in the respective frame and differs all transmissions for the duration of the NAV. When the timer expires the node could gain access to medium if it has a packet to be sent.

This mechanism is known as virtual carrier-sensing in the Anglo-Saxon literature.

The typical NAV scenario is presented in Figure 4.4 and after a packet is received by node B, it will reply by sending an acknowledgement (ACK), signalling that the packet has been correctly received and informs the other nodes that the exchange of packets with node A is complete.

The short interframe space (SIFS) and distributed interframe space (DIFS) are periods of time where a node waits between the transmissions of different packets and helps to prioritize access to the medium. The SIFS are used when a node has a high priority frame to send (such as a CTS or ACK packet). A node that wants to initiate a unicast exchange or send a broadcast packet. After it finds the channel idle, it waits for the DIFS and then sends the signal. In the described scenario two nodes can send RTS messages at the same time, however because these messages are short, the collision probability is low and the energy consumption is low if a collision does occur.

Despite DCF provides a CSMA/CA algorithm and an optional virtual carrier sense using RTS and CTS control frames, it does not always allow fair access to the wireless medium between network nodes.

So, there are some scenarios that can lead to unacceptable message latency if a node is not capable to access the wireless channel.

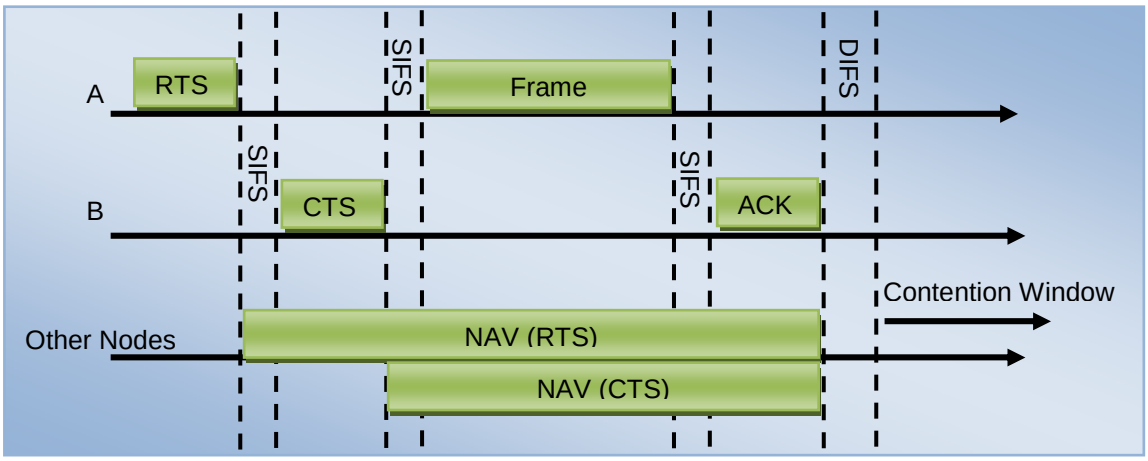


Figure 4.4. Typical NAV scenario [RMBM06]

4.2.3 IEEE 802.11 Point Coordination Function (PCF)

A method called Point Coordination Function (PCF) is only used in an infrastructure network in order to increase fairness and reduce latency. It can be used if it is included in the IEEE 802.11 network devices implementations. Basically, when this method is used the time will be divided into configurable contention-free period (CFP) and contention periods. This access method uses a point coordinator (PC), normally an AP that is responsible to control the medium access during the contention-free period by first transmitting information within beacon management frames in order to synchronise nodes and gain control of the medium, and then by polling each station for traffic.

During the CFP the NAV of all neighbour nodes is established to the maximum expected duration of the CPF. When the contention-free period finishes, the PC will transmit a control packet indicating that the nodes can use the DCF to exchange traffic until the next contention-free period. All neighbour nodes will reset the NAV, allowing each node to send their traffic with a specified maximum delay.

4.3 Wireless Medium Access Control Protocols

Wireless MAC protocols are classified depending on how they access the medium, the number of channels used to send data/control messages, their topology, the degree of organization between nodes, their power constraints and their requirements of quality of service. As described in the previous chapters the best way to conserve energy is to put the radio in sleep mode if there are no packets to send/receive. As presented in Figure 4.5 if there is the need to coordinate the sleep

operation, several approaches can be used, e.g. preambling, wake up frame, duty cycle structure or slotted frame architecture.

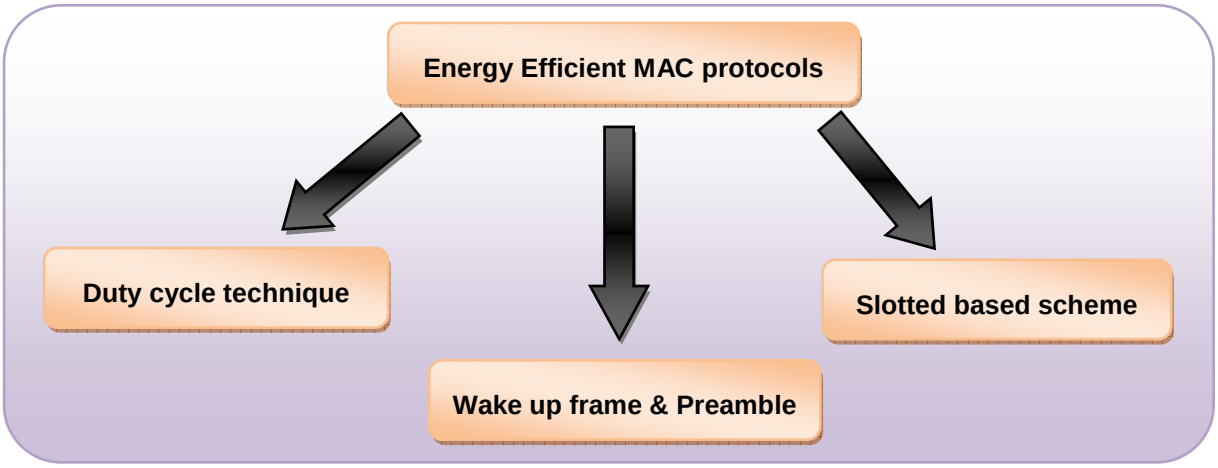


Figure 4.5. Classifications of the energy efficient MAC protocols.

In a *duty cycle technique* nodes periodically wake up to receive incoming packets, in order to avoid spending time in the idle state. The primary objective is always to reduce the communications activities of the nodes and in the best case a node will only leave the sleep state to transmit or receive packets. Some of the energy waste by the nodes is to maintain the neighbour's schedule table. Although a preamble signal is not used, the sender nodes must record receiver's sleep/wakeup and based on the receiver schedule, a sender can successfully transmit a packet to the receiver if no collision occurs. When designing a protocol based on this technique it is important to adjust the length of duty cycle dynamically. If we choose a small duty cycle the node will spend most of the time in sleep mode. Although we avoid idle listening and it conserves energy, the traffic directed from neighbour nodes designated to him concentrates on a small window thus heavy load situations significant competition can occur.

When a *preamble method* is used, a node periodically turns its radio transceiver to receive mode and use a preamble sampling technique. The sender node that wants to transmit the packet will transmit a preamble signal to wake up the receiver at receiver's sampling period, long enough to let the intended receiver pick it up. So it is clear that the sender node will spend more energy to guarantee the wake up operation. In turn, the preamble signal will wake up not only the destination nodes but all the other nodes that overhear the wakeup signal. So if we use a preamble method in is pure form it will become a costly method. Another problem of this method can be the start-up energy needed to switch the transceiver from sleep to idle or receiver mode (during this start-up time, no transmission or reception of data is possible [SHIM01], for example, the µAMPS-1 transceiver needs a start-up time of 466 µs and a power dissipation of 58 mW [MBCI02][MiCh02]).

In order to avoid this type of problems wake up frame schemes are developed to overcome the drawbacks of the preamble method. Using this scheme instead of the preamble signal it can greatly

reduce the energy cost of wake up operation. Thus, only the indicated destination will be wake up and neighbour nodes can go into sleep mode to save more energy. The advantage of these two techniques is that the sensor nodes pay no cost to collect and record the neighbour information.

When a *slotted based scheme* is used, the frame architecture is based on TDMA technique (Time Division Multiple Access) and a precise synchronisation protocol is needed. The time is divided into fixed-length slot times and these slots are assigned to nodes exclusively where they only transmit in the time slot reserved to him. This scheme is suitable to a centralized network architecture or a cluster-based structure. Despite de fact the TDMA protocols are very attractive for low-power applications (since energy is not wasted for collisions and overhearing), the maximum number of nodes in a cluster may be limited and an accurate synchronisation can be a difficult task. Therefore a typically TDMA is very suitable to applications requiring high guaranteed bandwidth and Quality of Service (QoS).

Next we present some of the existing MAC protocols for wireless sensor networks that use the techniques explained before.

4.3.1 MAC protocols based on a duty cycle scheme

Sensor-MAC (**S-MAC**)[YeHE02a] is one of the most well-know protocols for WSN and it performs access control and reduces the energy consumption by synchronising nodes based on scheduling sleep/listen cycles, called frames, between neighbouring nodes, Figure 4.6.

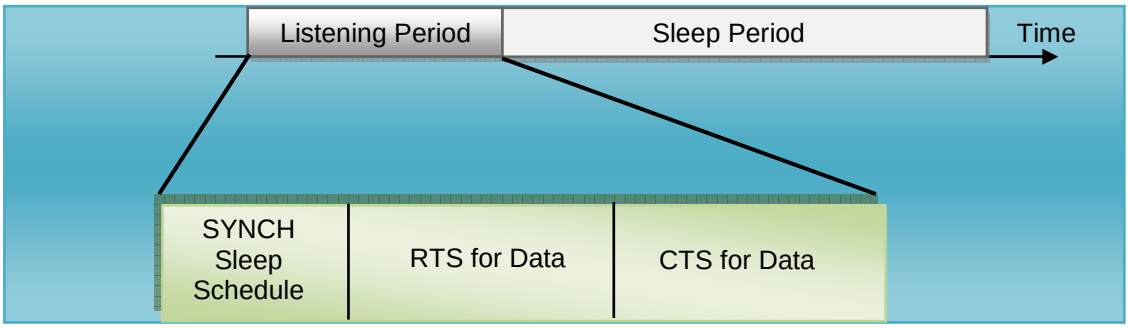


Figure 4.6. SMAC Time Frame Period.

It tries to eliminate the four sources of energy waste: collisions, overhearing, protocol overhead and idle listening. Each node alternates between a fixed-length listening period and a fixed-length sleep period according to its schedule Figure 4.7.

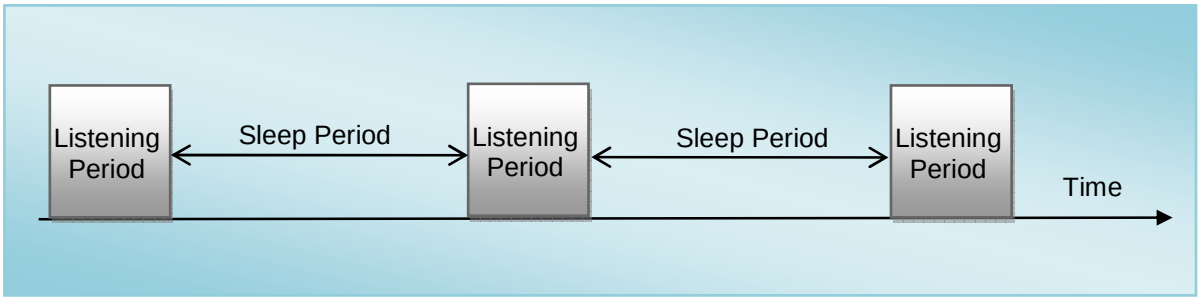


Figure 4.7. S-MAC principle.

Nodes try to organize themselves into virtual clusters using periodic broadcast synchronisation (by sending SYNCH messages). If a neighbour node overhears this message it follows that schedule by setting its schedule to be the same. If it does not hear a SYNCH message before a “timeout time” it broadcasts a SYNCH message announcing its sleep period. If it receives a different schedule after broadcasting he will adopt both schedules. One particularity of S-MAC protocol is the following: it forms nodes into a flat, peer-to-peer topology. Nodes forms virtual clusters around common schedules but communicate directly with his neighbour. As presented in Figure 4.8, node A and B are synchronised and the same happens with nodes C and D, however nodes B and C are not synchronised.

Figure 4.8. S-MAC virtual clusters.

Then if a node wants to transmit it follows an RTS/CTS/DATA/ACK handshake mechanism. This protocol has the problem of sleep latency; however this problem can be reduced by using the adaptive listening scheme proposed by the authors from [YeHE04]. When a node overhears its neighbour’s transmission it wakes up for a short period of time at the end of the transmission. This way, if it is the next-hop of its neighbour it can receive the message without waiting for its scheduled active time.

The S-MAC design is focused on cooperating applications. All the nodes cooperate among themselves in order to achieve a common task. These applications after some working time could have a large amount of information to communicate to its neighbours. To accommodate these requirements while trying to reducing overhead, S-MAC sacrifices channel access fairness and use the concept of message passing in order to reduce control overhead, avoiding overhearing and allowing nodes to send a long message in a burst.

Timeout-MAC (**T-MAC**) [DaLa03] improves S-MAC energy usage by concentrating all load at the beginning of a very short listening time period at the beginning of each active time, Figure 4.9, sacrificing the power conservation by throughput and latency.

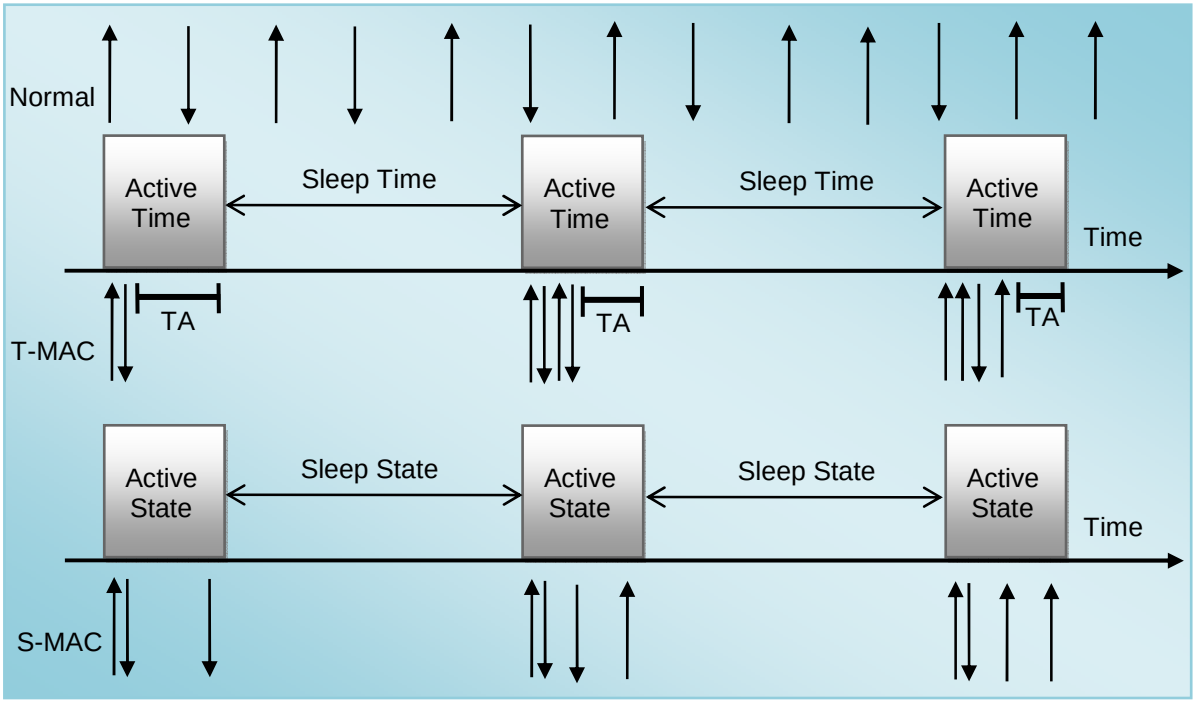


Figure 4.9. T-MAC adaptive timeout.

In Figure 4.9 the arrows represent the transmitted and received messages. T-MAC uses the same synchronisation scheme as S-MAC to form virtual clusters.

When there is no traffic the sensor nodes might go to sleep during a time period called adaptive timeout TA > 1.5 × (T_{CW}+T_{RTS}+T_{SIFS}), where T_{CW} is the duration of the contention window, T_{RTS} is the time to send an RTS packet, and T_{SIFS} is the short inter-frame space (time between the end of the RTS packet and the beginning of the CTS packet).

T-MAC also tries to solve the early sleeping problem that happens when a node goes to sleep when a neighbour still has messages to send to him. The idea is to let other node know that we still have a message for it, but are ourselves prohibited from using the medium. The solution is given by using a scheme called future request-to-send (FRTS). In this scheme if a node overhears a CTS packet destinated to other node and he has data to send, it may immediately send a FRTS packet that contains the length of time that the communication is locked between the sender and the receiver node. The node that receives the FRTS packet knows that it will be the future target of an RTS packet and must be awake by that time. In order to prevent any other node from taking the channel during this time, the node that sends the initial RTS transmits a small data-send (DS) packet. After de DS packet, it must immediately send the normal data packet.

Since the FRTS packet has the same size as a DS packet, it will collide with the DS packet, but not with the following data packet. The DS packet is lost, but that is no problem: it contains no useful information, Figure 4.10.

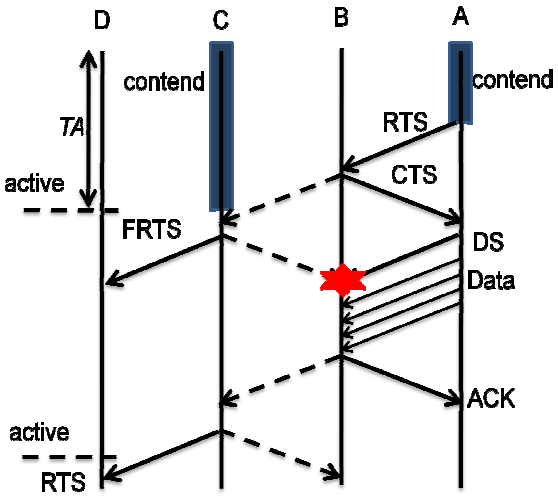


Figure 4.10. Future-request-to-send scheme.

The second approach to the early problem is when the node routing buffer is almost full and it may prefer sending then receiving, Figure 4.11. However, T-MAC uses a threshold: a node may only use this scheme when has failed contention at least twice.

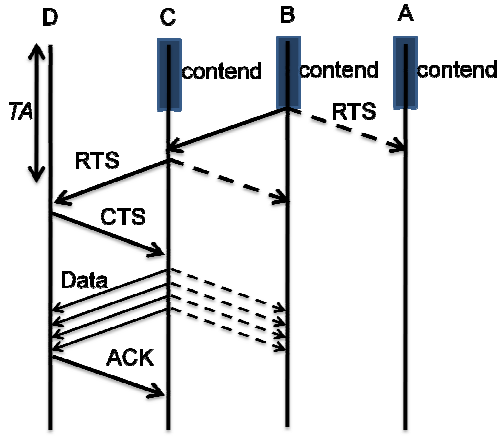


Figure 4.11. Taking priority on full buffers.

Both S-MAC and T-MAC protocols use RTS/CTS/DATA/ACK handshake mechanism to avoid overhearing problems. If the RTS/CTS method is not used, S-MAC and T-MAC protocols will be affected by the overhearing. In addition, the synchronous scheduling of S-MAC and T-MAC make sensor nodes suffer of high contention, reducing the throughput and channel utilization.

4.3.2 MAC protocols based on preamble and wake up frame

Berkeley Media Access Control for low power wireless sensor networks (**B-MAC**) [PoCu04] protocol contains a small core of media access functionality. B-MAC uses clear channel assessment (CCA) and packet backoffs for channel arbitration, link layer acknowledgments for reliability and low power listening (LPL) for low power communication. Although it is a link layer protocol B-MAC has is network services like organization, synchronization, and routing built above its implementation. Unlike other protocols it does not form clusters or attempt to synchronise sleep schedules. Instead, it uses a technique called low-power listening (LPL) with the objective of reducing the power consumption. In low-power listening, nodes periodically wake up at every cycle period and check the wireless channel for preamble signals, Figure 4.12. If a preamble is sensed they keep their radios “on” and they turn “off” the radios after a data packet is received or after a timeout. A node with data to send transmits a long preamble in order to notify the destination node to receive a data packet. The preamble should be long enough so that the periodically wakeup of the receiver detects it and receives the subsequent data packet. Both sender and receiver waste much energy during the communication and the transmission delay may be long.

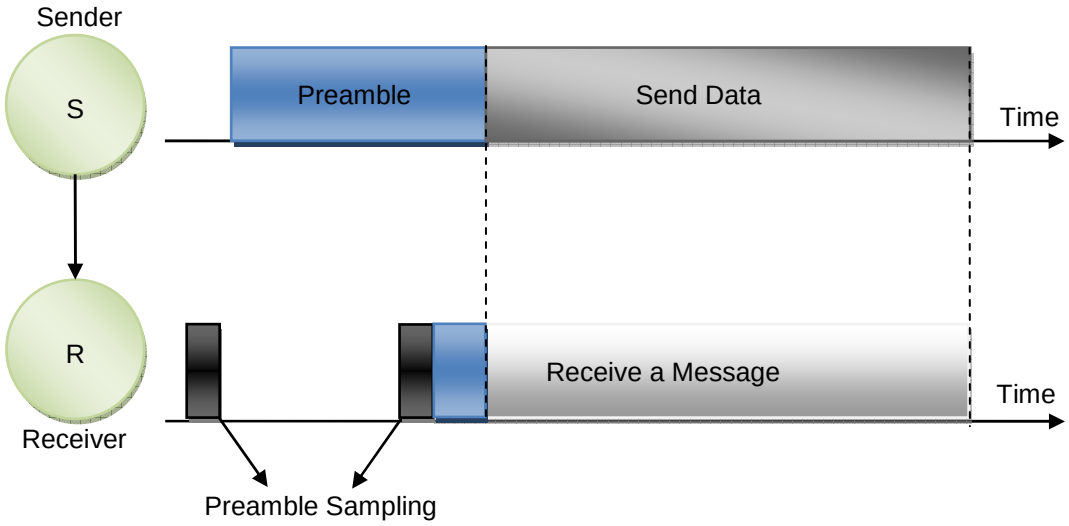


Figure 4.12. Low Power Listening: Preamble Sampling.

The LPL mechanism used in BMAC cannot adapt well to various types of data traffic. When using this method if the nodes in the network transmit frequently, it might cause overlap of preamble that allows the actual packet to be transmitted or to be received correctly.

The advantages of implementing this protocol are the following:

- simplicity to configure the network;
- it does not use explicit synch packets;

- it does not use RTS/CTS/ACK if it there is no need;
- the complexity is low.

Convergent MAC (**CMAC**) [LiFS07] avoids synchronisation overhead while supporting low latency and it uses zero communication when there is no traffic. When there is the presence of traffic it uses anycast to wake up forwarding nodes. Then, it converges from route-suboptimal anycast with unsynchronised duty cycling to route-optimal unicast with synchronised scheduling.

Like B-MAC protocol, CMAC uses a CSMA scheme and the wakeup technique is used. However it uses aggressive RTS to replace the long preamble, which breaks up the long preamble into multiple RTS packets (also called an RTS burst). The RTS control packets do not use long preambles and are separated by fixed short gaps that allow receivers to send back CTS packets.

Once the transmitter receives a CTS packet it sends the data packet immediately. Each gap does not need to accommodate an entire CTS transmission as long as the RTS sender can detect the preamble and cancel the next RTS transmission accordingly. It also introduces a double channel check which works by assessing the channel twice with a fix short separation between them each time a node wakes up, Figure 4.13.

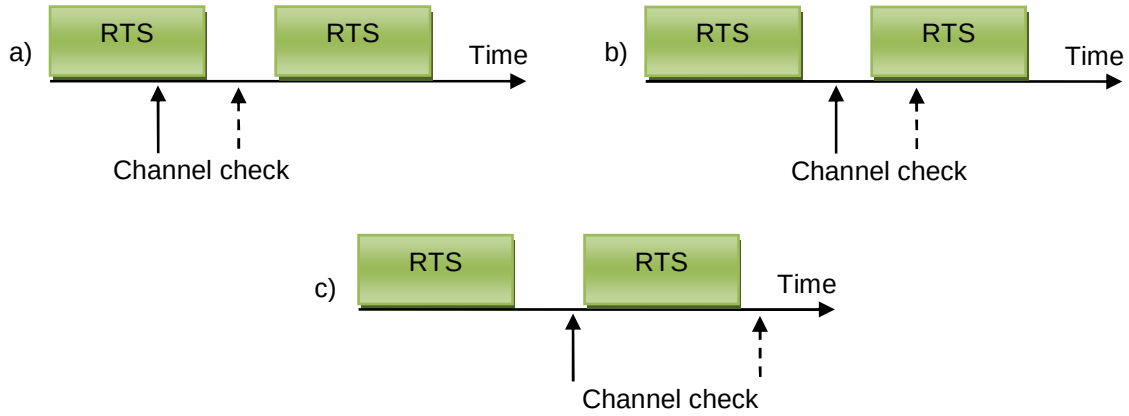


Figure 4.13. Double Channel Check used by CMAC. a) The first check detects the RTS burst. b) The second check detects the RTS burst. c) Impossible if RTS length is chosen carefully.

In C-MAC, the CTS transmissions are prioritized according to the routing metrics of contending nodes. Nodes with better routing metrics can send CTS packets earlier, while other overhearing nodes cancel their CTS transmissions accordingly and nodes that can make little progress could be excluded. The routing metric used is very generalist, such as geographical distance and hop count.

CMAC divides the forwarding region into 3 sub-regions, R_1, R_2, and R_3, such that nodes in R_i are closer to the destination than nodes in R_j for $i < j$, Figure 4.14.

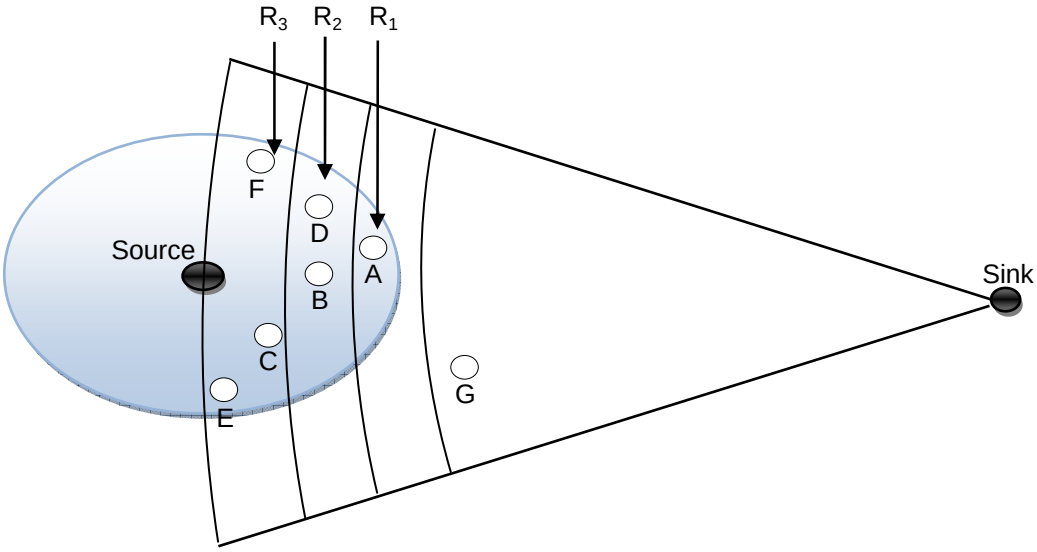


Figure 4.14. Cost region generation in CMAC using geographical distance as routing metric.

In this protocol, each gap between two consecutive RTS packets is divided into 3 sub-intervals called CTS slots. Nodes in region closer to the destination can send CTS packets in earlier CTS slots. Each CTS slot is further divided into several minislots in order to solve the contention within each region and each receiver will randomly choose one minislot to start its CTS transmission, Figure 4.15. When detecting busy channel, pending CTS transmissions will be cancelled assuming the existence of another CTS.

By providing there mechanisms: aggressive RTS, double channel check and convergent packet forwarding, CMAC supports low latency and high throughput as well as low duty cycle operation. So this protocol is highly suitable in scenarios that require low latency while providing a long throughput as well a long network lifetime.

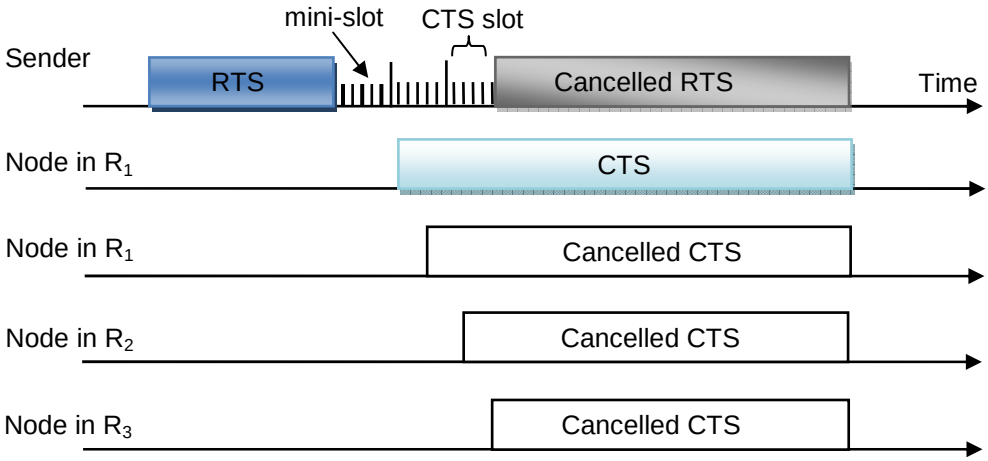


Figure 4.15. CTS contention resolution. The first CTS cancel others.

4.3.3 Slotted based MAC protocols

Traffic-adaptive medium access protocol (**TRAMA**) **[**RaOG06] employs a traffic adaptive distributed election scheme that selects receivers based on schedules announced by transmitters. Nodes using TRAMA, exchange their two-hop neighbourhood information and the transmission schedules specify which nodes are the intended receivers of their traffic in chronological order and then selects the nodes that should transmit and receive during each time slot. Accordingly, TRAMA protocol consists of three components: the Neighbour Protocol (NP), the Schedule Exchange Protocol (SEP), which allow nodes to exchange two-hop neighbour information and their schedules and the Adaptive Election Algorithm (AEA), which uses neighbourhood and schedule information to select the transmitters and receivers for the current time slot, leaving all other nodes in liberty to switch to low-power mode. TRAMA assumes a single, time-slotted channel for both data and signalling transmissions. Figure 4.16 presents the overall time-slot organization of the protocol.

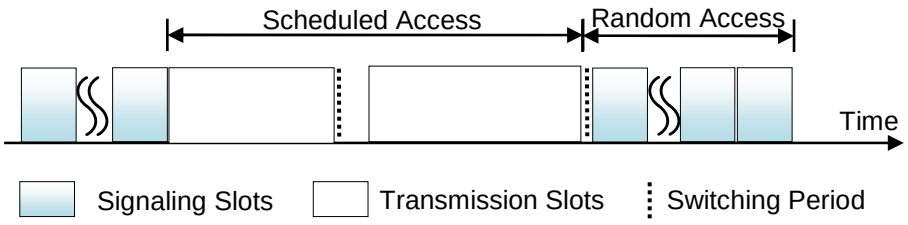


Figure 4.16. Time slot organization.

TRAMA avoids assigning time slots to nodes with no traffic to send and also allows nodes to determine when can switch to idle mode, stopping listening the medium channel. TRAMA is shown to be fair and correct, since no idle node is an intended receiver and no receiver suffers collisions.

Hybrid MAC (**HMAC**) [WaZK06] combines energy efficiency features of the existing contention-based and TDMA-based MAC protocols and adopts a short frame structure compared to S-MAC to execute the packet delivery. HMAC uses a slotted frame structure and a wakeup technique to achieve high energy performance.

Each node randomly selects its own wakeup slot and notifies the slot number to all its neighbours. A node needs to collect one-hop neighbour information constantly. A sender can wake up the receiver by sending a wakeup message at the receiver's wakeup slot. Then, the receiver will wake up at the specified data slot to receive the data packet. Because multiple nodes can try to access the medium, RTS/CTS/DATA/ACK handshake is used to avoid collisions Figure 4.17.

Although, this protocol increased channel utilization and reduce latency, it needs a precise time synchronization which causes a high overhead.

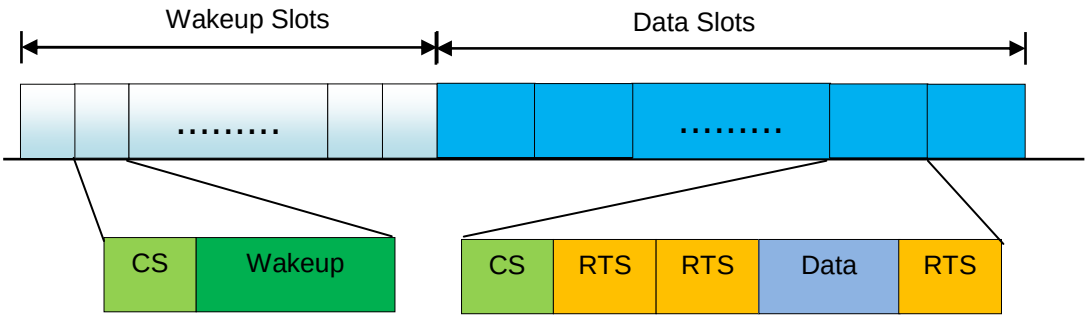


Figure 4.17. Frame structure of HMAC.

4.4 Conclusions

This chapter presents the related research in the area of wireless sensor network energy-efficient MAC protocols. Several different MAC protocols where discussed and the most important characteristics were described.

All were designed with the primary objective of energy conservation, although some other goals like decrease the delay or increase throughput are often traded off for energy conservation. Choosing the "best" MAC protocol is not a easy task and the proper choice is directly related with the application and the specification of the underlying hardware's energy-consumption behaviour. So it is important to choose the correct platform in order to reduce the energy costs of transmitting, receiving and switch between different modes of operation.

Chapter 5

An Innovative Energy-Aware Sensor Mac Protocol

Chapter 4 presented some of the existing “well known” MAC protocols and how they try to achieve energy-efficiency. Based on that, chapter 5 describes the Sensor Block Acknowledgment (SBACK) MAC protocol. And specifies the state transition diagram for S-MAC and SBACK. SBACK is a new innovative protocol that uses a block acknowledgement scheme to achieve energy efficiency. Like other WSN MAC protocols discussed in the previous chapter, it performs access control and reduces the energy consumption by synchronizing nodes based on scheduling sleep/listen cycles.

5.1 SBACK MAC Protocol Overview

The existing research in WSN MAC protocols is based on the desire of satisfying application-specific Quality of Service (QoS) requirements, while extending the sensors battery power by placing nodes into sleep state during the periods the network has no traffic or messages to exchange between nodes. So the primary objective is to maximize sensor network lifetime. As seen in the previous chapter, nodes that are not involved in the transmission/reception any kind of packet gain sleep opportunities after successfully receiving a RTS or a CTS control packet reservation message. Note that the use of a centralized point coordinator, or access point, in homogeneous WSN is generally not employed since normally the clusters are deployed in an ad hoc manner.

Based on Sensor-MAC, an innovative MAC protocol was proposed that uses block acknowledgment, the Sensor Block Acknowledgment (SBACK) protocol.

SBACK MAC protocol is based on a duty cycle scheme and is very similar to the “well known” S-MAC. It tries to reduce the major sources of energy waste such as collisions, overhearing, idle listening and control packet overhead. The main difference relatively to S-MAC is related to the way SBACK treats the acknowledgment (ACK) control packets. A Block ACK mechanism is proposed that intends to improve channel efficiency by aggregating several ACK into one special frame called *Block ACK Response*. This way, energy consumption will be greatly reduced when a series of data messages needs to be transmitted, because it is not needed to transmit and receive several ACK control packets (one for each data packet) which would lead to an extra energy waste, because there extra control packets does not directly results in the communication of information.

5.2 Block Acknowledgment Mechanism

The Block ACK mechanism improves channel efficiency by aggregating several acknowledgments into one special frame called *Block ACK Response*. In our proposal, the Block Ack mechanism is initialized by the exchange of two special packets *RTS ADDBA Request* and *CTS ADDBA Response,* ADDBA is the acronym for Add Block Acknowledgment*.* The structure of these packets is presented in Figure 5.1.

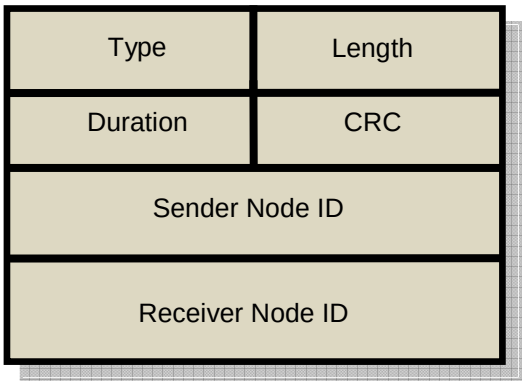


Figure 5.1. *RTS ADDBA ACK Request and CTS ADDBA ACK Response* packet format.

After initialization, the data packets may be transmitted accordingly from the sender to the receiver nodes. The number of data packets sent are limited to 100 (the maximum allowed messages are 10, fragmented into 10 small data packets). When the sender has no more data to transmit, it will send a special packet called *Block ACK Request* in order to verify the amount of packets successfully delivered to the receiver, in response the receiver will send a special packet called *Block ACK Response*. It identifies the packets that were not received properly and require retransmission. The structure of these packets is presented in Figure 5.2.

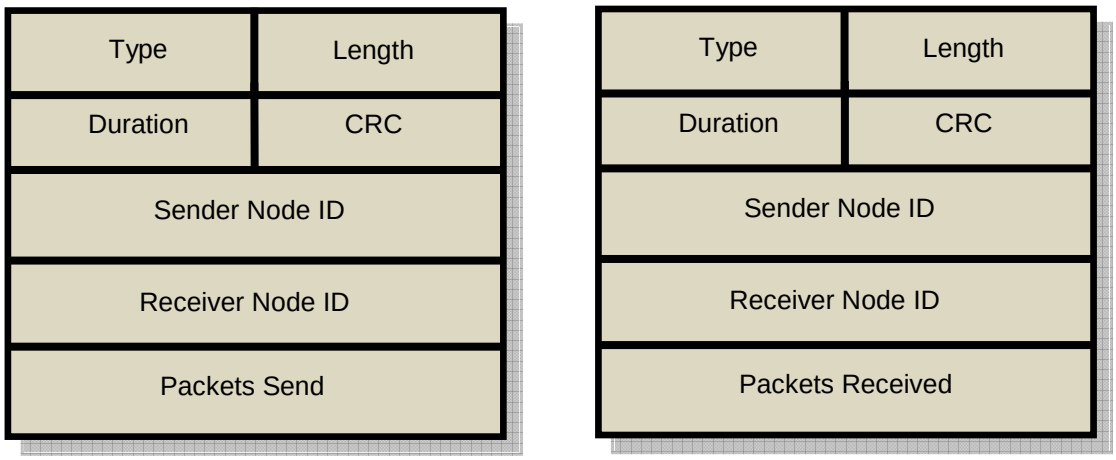


Figure 5.2. a) Block ACK Request b) Block ACK Response packets.

Finally, when the sender receives the *Block ACK Response* it will sends a *RTS DELBA Request* to the receiver that indicating that the *CTS Block ACK Response* was successful received and concludes the Block ACK mechanism. DELBA stands for Delete Block Acknowledgment. Then the receiver will send a *CTS DELBA Response*, and the block ACK sequence is complete. The structure of the DELBA packets is the same presented in Figure 5.1.

Figure 5.3 presents the proposed message sequence chart for the Block ACK sequence.

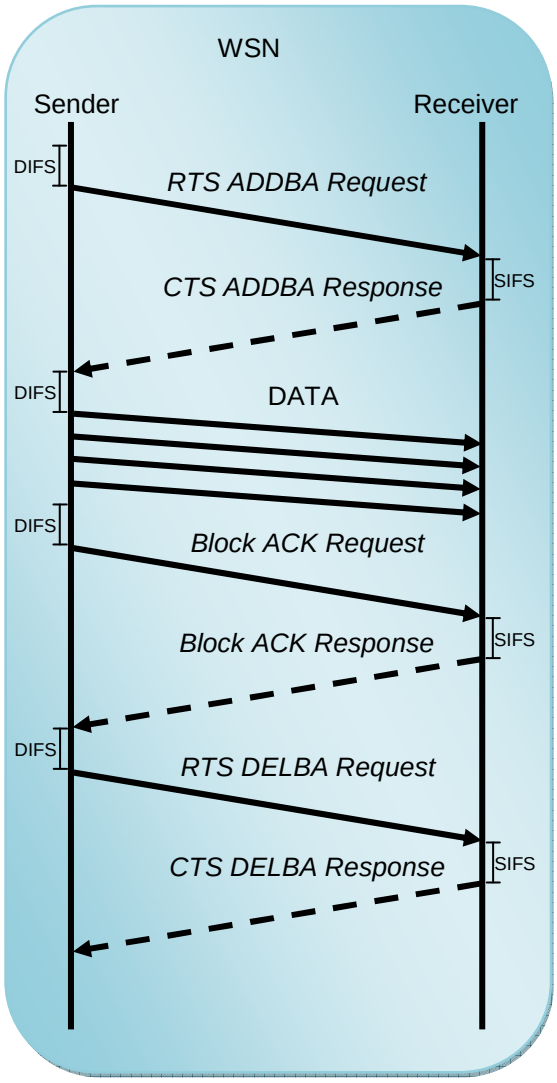


Figure 5.3. SBACK block ACK sequence.

5.3 Periodic Listen and Sleep Operation

SBACK reduces energy consumption by avoiding idle listening. One way to achieve that goal is to use low-duty-cycle operations. Periodically, nodes go to the sleep state so they can switch to OFF state avoiding the waste of energy. They can turn again to ON state when there is traffic in the network. The periodic listen and sleep scheme is very similar to one presented in Figure 4.7. In this scheme, a node goes to sleep for a specific period of time, after this period the node wakes-up again and listens the shared medium to determine if there is another node trying to communicate with it.

Unlike other MAC protocols in which coordination is achieved through a master node, such as a

cluster head, in SBACK nodes form virtual clusters but communicate directly with their peers, to exchange and synchronise their sleep and listen schedules.

5.4 Schedule Selection and Coordination

In order to coordinate their sleeping and listening times, neighbouring nodes coordinate their listen and sleep schedules by exchanging them with their peer during the synchronisation period. All the nodes have a schedule table which stores the schedules of all their known neighbours. In order to select a schedule, a node listens the channel for a certain amount of time. If the node does not hear a schedule from another node it follows its own schedule, and announces its schedule by broadcasting it in a SYNCH packet, after perform physical carrier sensing, Figure 5.4.

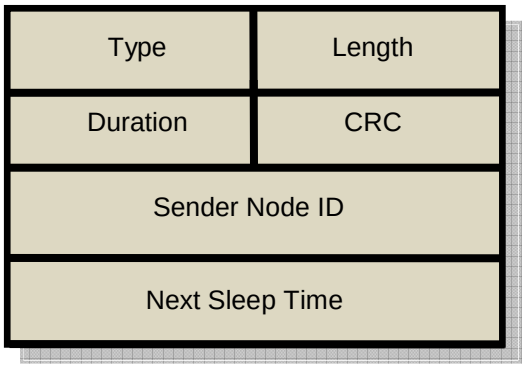


Figure 5.4. SYNCH Packet.

During the synchronisation period if a node receives a schedule from a neighbour, it follows it by setting its own schedule to be the same as the one received

Each node, knows of some neighbouring nodes have already adopted its schedule due to its previous broadcast If the node receives a different schedule from other node it adopts both schedules. This situation is presented in Figure 5.5.

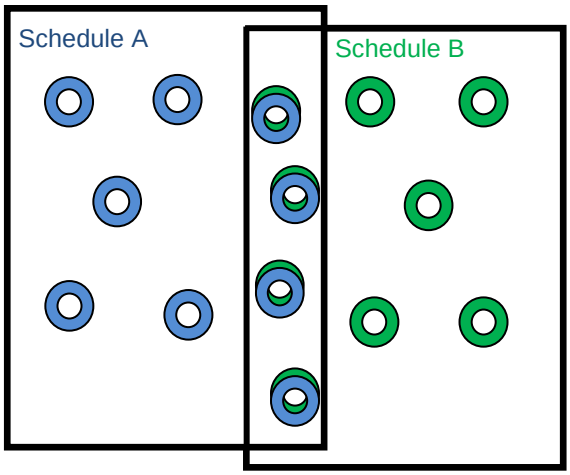


Figure 5.5. Schedule selection and synchronisation near the boundary between two regions.

Like in S-MAC protocol, nodes only rarely carry multiple schedules since every node tries to follow existing schedules before choosing an independent one. However, in this approach the power consumption for the nodes in the boundary region is higher as they will spend less time in sleep mode.

5.5 SBACK Access Control and Data Exchange

In SBACK nodes access the channel based only on its local information. A CSMA/CA scheme is used that includes physical and virtual carrier sensing and the RTS/CTS handshake, to reduce the impact of the hidden terminal problem. We achieve virtual carrier sensing by using a network allocation vector (NAV), the remaining time until the end of the current packet transmission. The NAV is initially set to the value carried in the duration field of the transmitted packet. Then, it is decremented as the time passes until it reaches zero. While the value does not reach zero, the node knows that the medium is busy. Physical carrier sensing is performed by listening to the channel to detect ongoing transmission. The medium is considered free if both virtual and physical carrier sensing indicate that the channel is free. To effectively achieve virtual carrier sensing, nodes may be required to listen to all transmissions from their neighbours. In this case, a node may listen packets that are transmitted to other nodes. This packet overhearing may lead to significant energy waste.

In order to avoid overhearing, SBACK allows nodes to move into sleep mode after they hear the exchange of different types of RTS or CTS packets between two other nodes, as presented in Figure 5.3. The node initializes its NAV with the value contained in the duration field of the RTS/CTS packets and enters the sleep state until the NAV value reaches zero. Because data packets duration is typically larger than the duration of control packets, the overhearing avoidance process may lead to significant energy savings.

The exchange of packets is completed when the node receives a *CTS DELBA Response* from the receiver.

5.6 Message Passing

The SBACK protocol improves application-level performance, by using the message passing scheme. The messages are divided into small fragments. All these fragments are then transmitted in a single burst. Via one *RTS ADDBA Request*/*CTS ADDBA Response* exchange between nodes. When this exchange is finished, the wireless medium is reserved during the time needed to complete the transfer of the entire fragmented message. Furthermore, each fragment carries in its duration field the time needed to transmit all the subsequent fragments plus the *Block ACK Response* that will send after the end of the data messages.

5.7 State Transition Diagram for SMAC and SBACK Protocols

As described before, the SBACK protocol is essentially based in the S-MAC protocol specification. In order to understand all the involved concepts it was decided to describe protocol by *N* state variables. Figure 5.6 presents the S-MAC state transition diagram, while Table 5.1 presents the related event and actions.

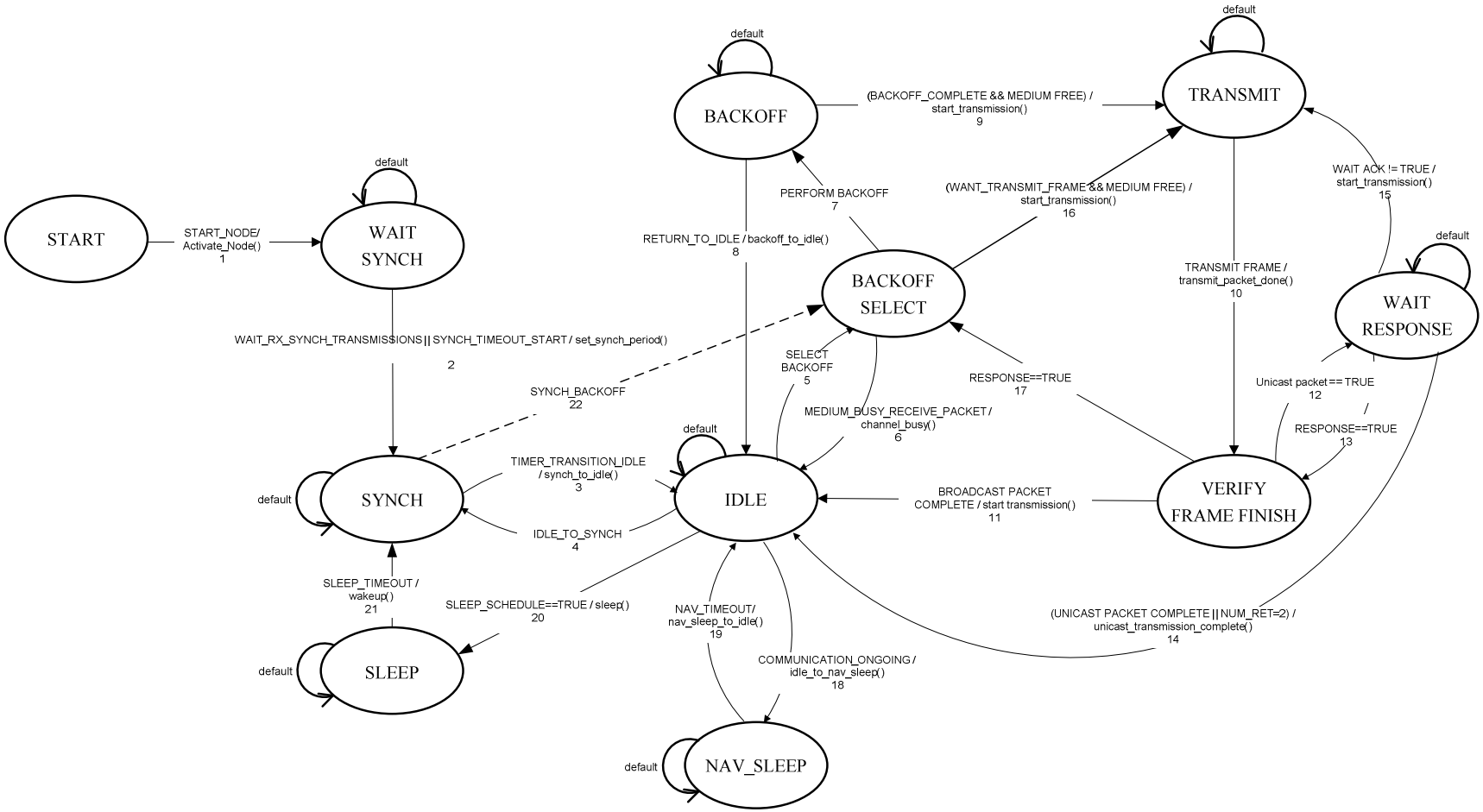


Figure 5.6. S-MAC State Transition Diagram.

Table 5.1. S-MAC events and actions.

EVENTS	ACTIONS
1	**State**: WAIT SYNCH **Schedule:** WAIT_RX_SYNCH_TRANSMISSIONS SYNCH_TIMEOUT_START **Save**: Node_ID Maximum packet length
2	**State**: SYNCH **Schedule:** STOP_SYNCH TIMER_TRANSITION_IDLE SYNCH_BACKOFF See if the synch broadcast packet was successful received
3	**State**: IDLE **Schedule:** SELECT BACKOFF
4	**State**: SYNCH **Update:** Synch table
5	**State**: BACKOFF SELECT **Save**: Type of packet (unicast or broadcast) **Schedule:** PERFORM BACKOFF
6	**State**: IDLE Process received packet
7	**State**: BACKOFF **Schedule:** RETURN_TO_IDLE
8	**State**: IDLE

	Schedule: TRANSMIT FRAME
9	**State**: TRANSMIT **Schedule:** TRANSMIT FRAME
10	**State**: VERIFY FRAME FINISH **If (Unicast packet == TRUE)** Verify if the unicas packet were correctly transmitted **Else** Complete the broadcast packets transmission
11	**State**: IDLE **Schedule:** STOP_ IDLE
12	**State**: WAIT RESPONSE **If (Unicast packet == TRUE)** Wait for CTS due to RTS and ACK due to DATA
13	**State**: VERIFY FRAME FINISH **Schedule:** PERFORM BACKOFF TRANSMIT FRAME
14	**State**: IDLE **Schedule:** IDLE_TO_SYNCH
15	**State**: TRANSMIT **Save**: NUM_RET **Schedule:** TRANSMIT FRAME
16	**State**: TRANSMIT **Schedule:** TRANSMIT FRAME
17	**State**: BACKOFF SELECT **Schedule:** PERFORM_BACKOFF

	TRANSMIT FRAME
18	**State**: NAV_SLEEP **Schedule:** NAV_TIMEOUT
19	**State**: IDLE
20	**State**: SLEEP **Schedule:** SLEEP_TIMEOUT
21	**State**: SYNCH **Schedule:** TIMER_TRANSITION_IDLE
22	**State**: BACKOFF SELECT **Schedule:** PERFORM BACKOFF

The characterization of possible states for the state machine is the following:

Machine

States

- **START**: The node will “turn on”;
- **WAIT SYNCH**: The node is waiting for SYNCH packets;
- **SYNCH**: The node will try to synchronise with neighbour nodes;
- **SLEEP**: The node will “turn off” the radio;
- **BACKOFF SELECT**: The node will select a “backoff time” ;
- **BACKOFF**: The node will activate the “backoff time” ;
- **IDLE**: The node will wait for a task to perform;
- **NAV_SLEEP**: The node goes to sleep until the end of the remaining transmission;
- **TRANSMIT**: The node will transmit the frame;
- **VERIFY FRAME FINISH**: The node will verify if the frame is correctly transmitted;
- **WAIT RESPONSE**: The node will wait for confirmation of a successful transmission.

States of the packet

- **Packet_type**: The type of the packet can be RTS, CTS, DATA, ACK, SYNCH;
- **Number_collision**: number of collisions the packet as suffered;

- **Origin**;
- **Destination**;
- **Time of generation**;
- **Fragmentation**: If fragmentation is used;
- **First_RTS**: Indication of the first RTS sent in case of fragmentation be used;
- **Backoff value**.

Medium States

- **Free**: Medium is free, there is no transmission ongoing;
- **Busy**: Medium not free, there is a transmission ongoing.

Queues

- **Not empty**: *n* packets are waiting for transmission;
- **Empty**: The buffer is empty.

Simulation Variables

- **Event_List**: The list in which all the events are sorted by time;
- **Nodes**: a vector which contains all the nodes;
- **Output**: where the main outputs are saved.

Input Variables

- **NUM_NODES**: total number of nodes;
- **SIMULATION_TIME**: define the time to be simulated;
- **DATA_RATE**: define the data rates;
- **INTER_ARRIVAL_PERIOD**: message inter-arrival period varies from 1 s to 10 s;
- **MAX_PKT_LENGTH**: the maximum allowed packet length is 38 bytes;
- **FRAGMENT_LENGTH**: this value should be smaller than MAX_PKT_LENGTH;
- **DELAY_SEND_PKT**: 20 ms between two consecutive packets;
- **LISTEN_TIME**: 150 ms is the listen state (variable) ;
- **SLEEP_TIME**: 1230 ms is the sleep state (variable) ;
- **RTS_PACKET_SIZE**: 8 bytes;
- **CTS_PACKET_SIZE**: 8 bytes;
- **ACK_PACKET_SIZE**: 8 bytes;
- **SYNCH_PACKET_SIZE**: 2 bytes;
- **RTS_CONTEND_TIME**: 100 ms;

- **CTS_CONTEND_TIME**: 5 ms;
- **DATA_CONTEND_TIME**: 5 ms;
- **ACK_CONTEND_TIME**: 5 ms;
- **NUM_COL**: 2 (collisions limit) ;
- **RTS_THRESHOLD**: turn off RTS/CTS if the length of a data packet is smaller than this value.

The events that change the state of the machine are described next:

- **START NODE**: activate node;
- **WAIT_RX_SYNCH_TRANSMISSIONS**: start the reception of synch packet;
- **SYNCH_TIMEOUT_START**: define time to leave the SYNCH state;
- **SYNCH_BACKOFF**: randomly selects a time slot to finish its carrier sense;
- **TIMER_TRANSITION_IDLE**: synchronism complete or time to periodically wakeup;
- **SLEEP_TIMEOUT**: wakeup from periodically sleep;
- **RETURN_TO_IDLE**: if the node receive packet in the BACKOFF state;
- **COMMUNICATION ONGOING**: shutdown radio if a communication was sensed;
- **NAV_TIMEOUT**: return to idle, the communication ongoing has finished;
- **SELECT_BACKOFF**: select backoff time before synchronise or try to transmit;
- **MEDIUM_BUSY**: return to IDLE state because received a packet while in the BACKOFF SELECT;
- **PERFORM BACKOFF**: start the backoff;
- **BROADCAST PACKET COMPLETE**: the transmission of the synch packet has finished;
- **UNICAST PACKET COMPLETE**: data packet successfully transmitted;
- **BACKOFF COMPLETE**: synch packet successfully transmitted;
- **MEDIUM FREE**: has been no traffic detected in the medium;
- **WANT_TRANSMIT FRAME**: the node has a frame to transmit;
- **TRANSMIT FRAME**: the node starts to transmit;
- **WAIT_ACK**: verify if the ACK was correctly received;
- **RESPONSE**: response successfully received during a unicast transmission;
- **NUM_RET**: number of maximum attempts to transmit the data packet as been reached.

S-MAC divides the time frame into two parts: the listening and the sleep period. The listening period is used to coordinate the nodes that have data to send/receive. The sleep period is used to deactivate some of the functionalities of the nodes, e.g., turn of the transceiver in order to achieve energy efficiency, Figure 5.7.

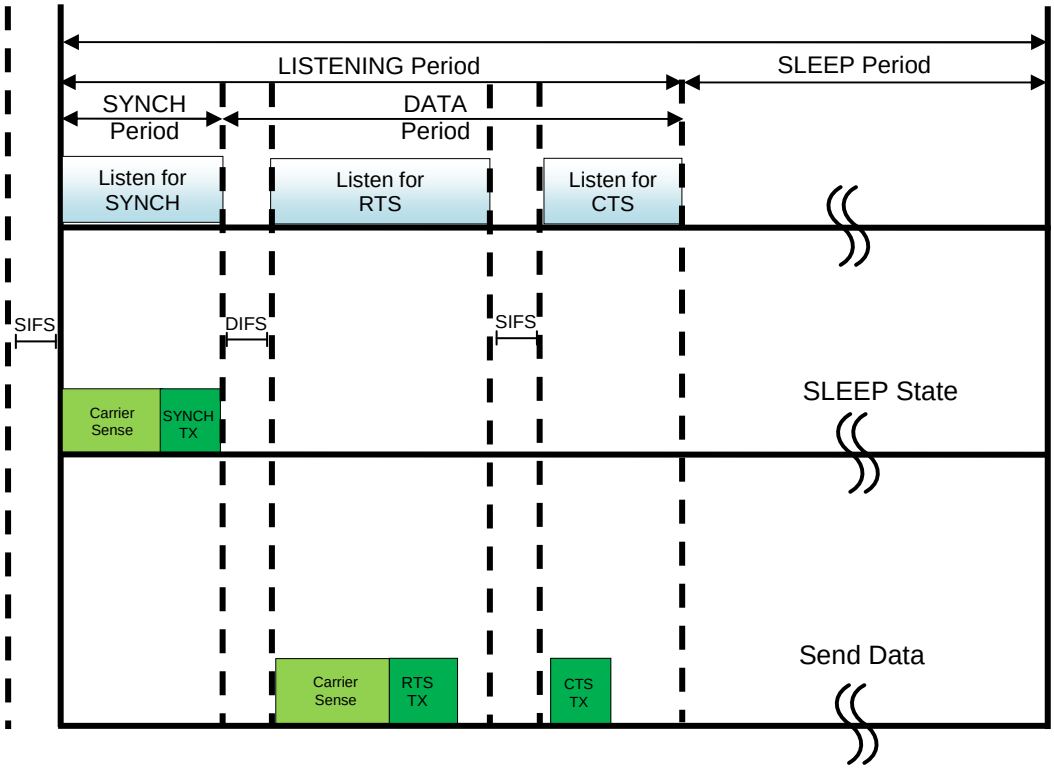


Figure 5.7. Time line of power saving on SMAC protocol.

SBACK uses the state transition diagram of S-MAC presented in Figure 5.6, but introduces some modifications, as it can be seen in Figure 5.8.

In SBACK protocol a TYPE ACK state was introduced. This state, forces the node to follow a procedure based on a *Block ACK Sequence*, as presented in Figure 5.3.

5.8 SBACK Functions

The SBACK MAC protocol was implemented in the OMNeT++ simulator [OMNE09] a public-source, component-based, modular and open-architecture simulation environment, with strong Graphical User Interface (GUI) support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks. Because of its generic and flexible architecture, it has been successfully used in other areas like the simulation of Information Technology (IT). A comparison study of the existing simulation tools for WSN is presented in Annex 2.

In OMNeT++ [USMA09], events occur inside simple modules. Simple modules encapsulate C++ code that generates and reacts to events, implementing the model behaviour. The user creates simple

module types by subclassing the *cSimpleModule* class, which is part of the OMNeT++ class library. *cSimpleModule*, just as *cCompoundModule*, is derived from a common base class, *cModule*, and are modules that communicate with message passing, compound modules are basically simple modes grouped. *cSimpleModule*, although packed with simulation-related functionality, does not do anything useful by itself. So, it needs to be redefining by using some virtual member functions to make it useful for the SBACK MAC protocol.

These member functions are the following:

- void *SBACK::initialize()*
- *void SBACK::handleMessage(cMessage *msg)*

When one initialises OMNeT++ it builds the network by creating the necessary simple and compound modules while connecting them according to the Network Description (NED) definitions. Then, OMNeT++ calls the *initialize()* functions of all modules that will be responsible to initialize the variables of our SBACK MAC protocol.

The *handleMessage()* function is called during event processing. This means that it will implement the model's behaviour in these functions, by implementing different event processing strategies for each simple module. The *handleMessage()* method is called by the simulation kernel when the module receives a message.

The main functions shown in Annex 3 implement actions rising from the events of the SBACK MAC protocol (that cause the change of the state as well) are the following:

- void ***SBACK::send_RTS_ADDBA_Request_Messages****(cMessage *msg)*: this function is responsible to send the special packet "*RTS ADDBA Request*".
- void **SBACK::send_CTS_ADDBA_Response_Messages**(cMessage *msg): this function is responsible to send the special packet "*CTS ADDBA Response*",
- void **SBACK::send_BA_Request_Messages**(cMessage *msg): this function is responsible to send the special packet "*Block ACK Request*".
- void **SBACK::send_BA_Response_Messages**(cMessage *msg): this function is responsible to send the special packet "*Block ACK Response*".
- void **SBACK::send_RTS_DELBA_Request_Messages**(cMessage *msg): this function is responsible to send the special packet "*RTS DELBA Request*".
- void **SBACK::send_CTS_DELBA_Response_Messages**(cMessage *msg)): this function is responsible to send the special packet "*CTS DELBA Response*".

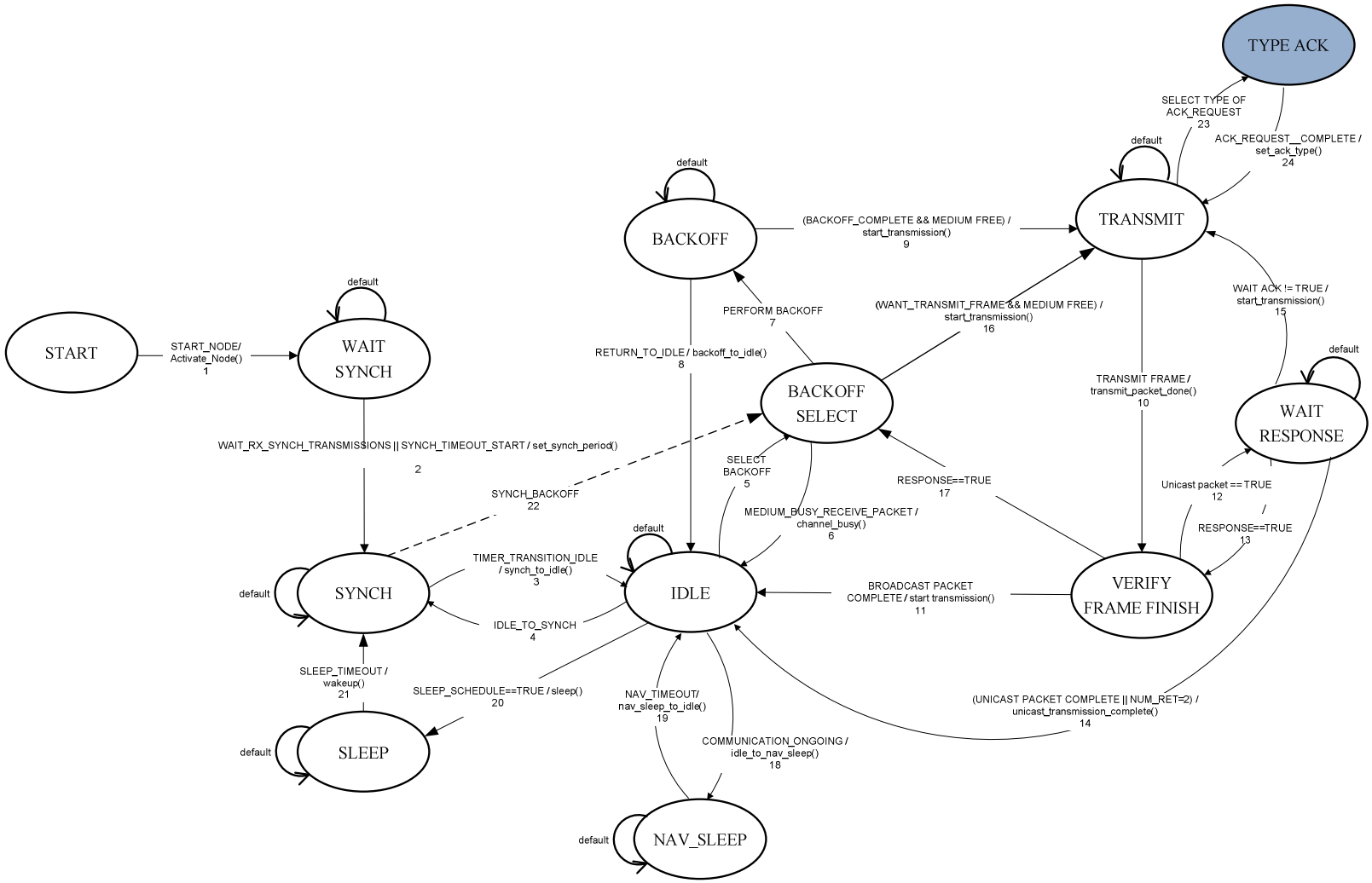


Figure 5.8. SBACK State Transition Diagram.

5.9 Summary and Conclusions

This chapter presented an innovative SBACK MAC protocol for WSN, whose design is based in of S-MAC protocol. The objective of this new version of S-MAC is to achieve good scalability and collision avoidance by using a combined scheduling and contention scheme.

SBACK MAC protocol tries to reduce the protocol overhead by reducing the major sources of energy waste such as collisions, overhearing, idle listening and control packet overhead. It uses a Block ACK mechanism to improve channel efficiency and aggregate several ACK into one special frame called *Block ACK Response*. This aims to reduce the power consumption by transmitting less ACK control packets and by decreasing the times periods the transceivers should switch between different states.

A global perspective of the S-MAC and SBACK protocol was presented, by providing a description of the state machine diagram for these protocols, while presenting all the events and actions, which were implemented in OMNeT++.

Chapter 6

Results for Energy-Aware MAC Protocols

This chapter provides the results for the delay and energy for the S-MAC and SBACK protocol. An analytical model was firstly developed to obtain the energy consumption of S-MAC protocol. This model is also applied to the SBACK in order to compare the energy consumption of the two protocols. The analytical model was implemented in OMNeT++ simulator. A model that gives expected battery life vs system current usage and duty cycle for the IRIS motes is also presented.

6.1 Performance Evaluation

In order to minimize the numbers of the MAC protocol state transitions it is important to access the energy spend on each transceiver state. A two-hop network, with two sources and two sinks, was considered in the OMNeT++ experimental setup, as shown in the topology from Figure 6.1.

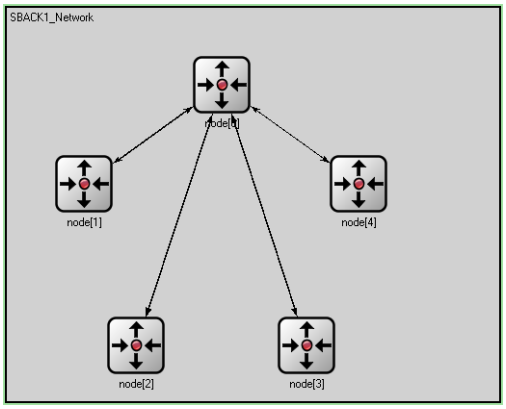


Figure 6.1. Experimental scenario (node 1 and 2 are sources and node 3 and 4 are the sinks).

The packets from source node 1 flow, through node 0, to sink node 3 while the packets originated by source node 2 flow, through node 0, to reach sink node 4. This topology is a simple star topology very common in healthcare applications. The metrics used for a specific evaluation of the number of state transition of the SBACK MAC protocol are the following.

- **Energy to Transmit:** is the amount energy spent to transmit a packet;
- **Energy to Receive/Listen:** is the amount of energy spent to receive a packet or to listen to the medium;
- **Energy to Sleep:** is the amount energy spent by a node during the time of inactivity referred as sleep state;
- **Time spent in a given state:** (Size of a Packet) **/** (Transmission Rate);
- **Total energy consumption:** is the total energy spent per node. It incorporates all previous metrics together into a single one. Such sources of energy waste have not been taken into account separately because the intention is to have a global perspective of all the factors that optimize the total energy.

6.1.1 Models for energy consumption for efficient transceivers

The first step in design SBACK, involves to implements the most important functionalities of S-MAC. In our scenario, there are 4 nodes and one special node (node 0) that will act as a gateway, as presented previously in Figure 6.1. The nodes may only communicate with direct neighbours. In our first set of tests we consider that the radio transceiver to test both S-MAC and SBACK protocols is the TR1000 one, from RF Monolitichs, Inc [TR1009]. Table 6.1 shows the power consumption specifications for this transceiver.

Table 6.1. TR1000 specifications.

		TR1000
Power Consumption [mW]	**Sleep**	0.015
	Receive	13,500
	Transmit	24,750
Data Rate[kb/s]		19.2

Figure 6.2 presents the source nodes energy consumption as a function of the message inter-arrival period for the S-MAC protocol, when using the TR1000 transceiver [YeHE02a].

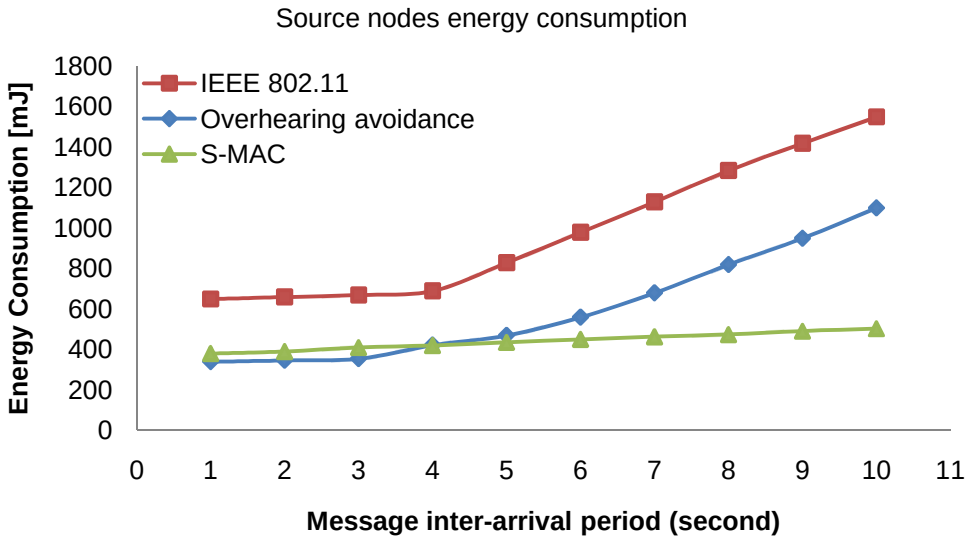


Figure 6.2. S-MAC protocol, energy consumption in the source nodes using the TR1000 transceiver (extracted from [YeHE02a]).

The curves for the source nodes energy consumption in Figure 6.2 were obtained from [YeHE02a].

It is observed that the energy consumption is an increase function of the message inter-arrival period but a perfect linear behaviour only occurs for S-MAC. The curve of S-MAC was also obtained by simulating the energy model from this thesis. The model of energy was created to confirm the practical experimentations. Since authors of the original paper obtain the curve by direct measurement and in our case we need to simulate the MAC protocol,

The following formula was used to calculate the energy consumption in the source nodes:

$$E=(P_{Transmit}\times T_{Transmit})+(P_{Receive}\times T_{Receive})+(P_{Sleep}\times T_{Sleep}) \quad (6.1)$$

The time a radio will be in transmit mode is given by:

$$T_{Transmit}=(\text{Packet Size}) / (\text{Data Rate}) \quad (6.2)$$

where, Packet Size=38 B (6 B Header , 30 B Payload and 2 B CRC)

The time the radio will be in receive or sleep mode is given by:

$$T_{Receive}=T_{Sleep}=\text{Message Inter-Arrival Period} \quad (6.3)$$

Like in S-MAC, SBACK generates 10 messages, which are fragmented into 10 small packets. So, each source node sends a total of 100 data packets. The comparison of the original measured S-MAC results from [YeHE02a] and the ones obtained by simulation is shown in Figure 6.3.

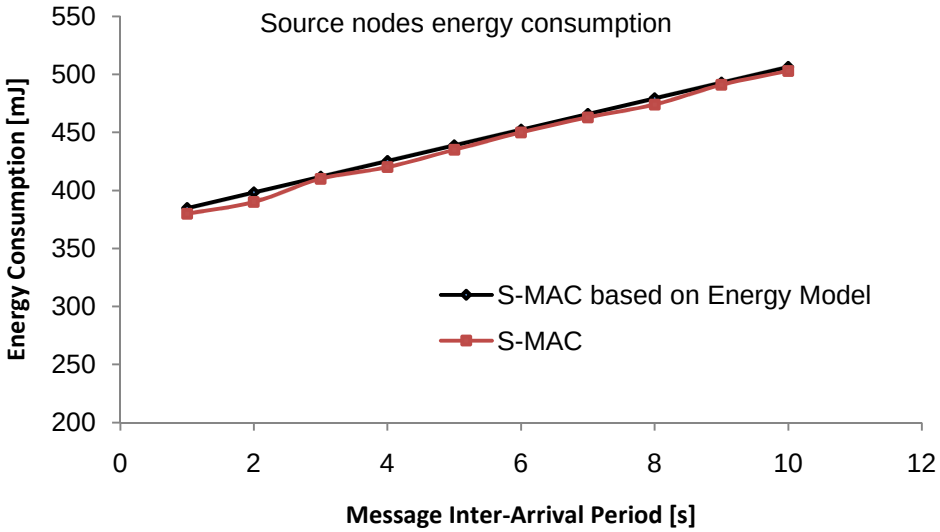


Figure 6.3. S-MAC Original vs S-MAC obtained in OMNeT++.

Very similar results are obtained. Based on the previous results, we have extrapolated this model considering two different radios: CC2420 and AT86RF230.

Table 6.2. CC2420 and AT86RF230 specifications.

		CC2420	AT86RF230
Power Consumption	**Sleep**	0,06 µW	60 nW
	Receive	59,1mW	46,2 mW
	Transmit	52,2 mW	49,5 mW
Data Rate [kb/s]		250	250

The energy consumption in the source nodes obtained for the CC2420 and AT86RF230 transceivers as a function of the message inter-arrival period is presented in Figure 6.4.

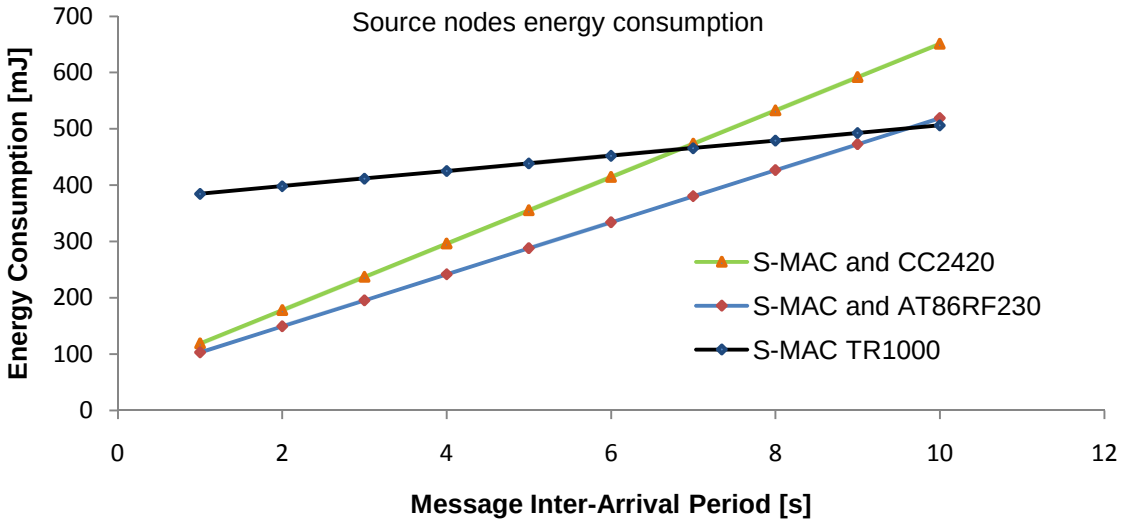


Figure 6.4. S-MAC energy consumption for TR1000, CC2420 and AT86RF230 transceivers.

6.1.2 Performance Evaluation for S-MAC with ACK

Every packet sent by the transmitter and correctly received by the receiver requires an ACK response. The transmitter will waste energy not only by sending the data packet but also receiving the ACK control packets. Therefore the energy model previously presented must take into account the energy spend to receive an ACK packet:

$$E_{ACK}=(P_{Transmit\_ACK}\times T_{Receive\ ACK}) \quad (6.4)$$

The time a radio is in the receiver mode (to receive an ACK packet) is given by:

$$T_{Receive\ ACK}=(Packet\ Size)\ /\ (Data\ Rate) \quad (6.5)$$

where, Packet Size=8 B (6 B Header and 2 B CRC)

When a node receives an ACK packet, it needs to process less 30 B than in the case of a data packet. The source nodes energy consumption for the case when they also need to process the ACK packets is presented in Figure 6.5.

The energy consumption slightly increases relatively to the case where no ACK packets are used, Figure 6.4, mainly for the TR1000 transceiver.

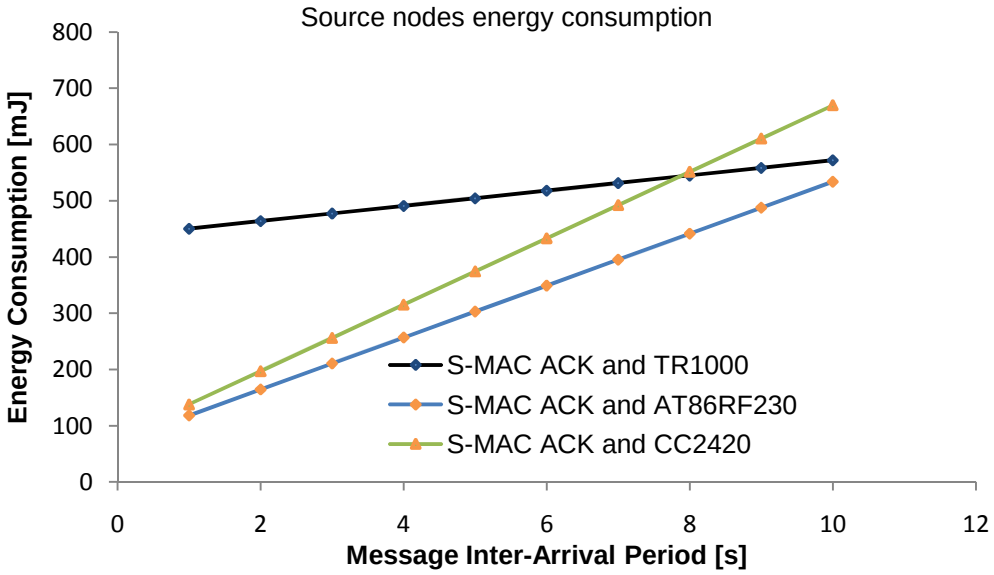


Figure 6.5. S-MAC with ACK - energy consumption in TR1000, CC2420 and AT86RF230 transceivers.

6.1.3 Energy Consumption for SBACK

In the SBACK protocol, before transmitting the nodes first check the value of the Received Signal Strength Indication (RSSI), like explained by the authors of [ReMC07]. The amount of data to be transmitted is 100 packets in total. To guarantee that all the packets reach the destination, the RSSI value must greater than -80 dBm. This is the value requested to achieve a packet reception rate (PRR) of at least 90%. It is assumed that, in order to transmit all the packets (PRR of approximately 100%), the RSSI value must be higher than -85dBm. This value is very close of the sensitivity threshold of the radio transceivers, which is approximately -90dBm. In our simulator we use a random uniform function to generate the value of the RSSI because all intervals of the same length on the distribution's are equally probable, so the value will be randomly and all the values obtained will have the same probability within the interval.The Link Quality Indicator (LQI) is another metric used to

measure the quality of the receiver signal but is not taken into account. The LQI gives an estimate of how easily a received signal can be demodulated by accumulating the magnitude of the error between ideal constellations and the received signal over the 64 symbols immediately following the synch word [RSSI09]. The RSSI does not account for the "quality" or "correctness" of the signal while LQI does not take into account the actual signal strength.

However the signal quality often is linked to signal strength. This is because a strong signal is likely to be less affected by noise and interference and thus it will be seen as "cleaner" or more "correct" by the receiver. As a consequence, the main reasons to choose only the RSSI is the fact that if a weak signal in "total" absence of noise has a low RSSI it also haves low LQI and vice-versa, i.e., when it has a high RSSI it also has a high LQI.

Figure 6.6 presents the source nodes energy consumption as a function of the message inter-arrival period when ACK or BACK mechanisms are used for the TR1000 transceiver. Figure 6.7 presents the percentage of energy saving if the SBACK protocol is used instead of the simple ACK one. The energy savings varies from 12.88% to 10.14% when the message inter-arrival period increments from 1 to 10 s.

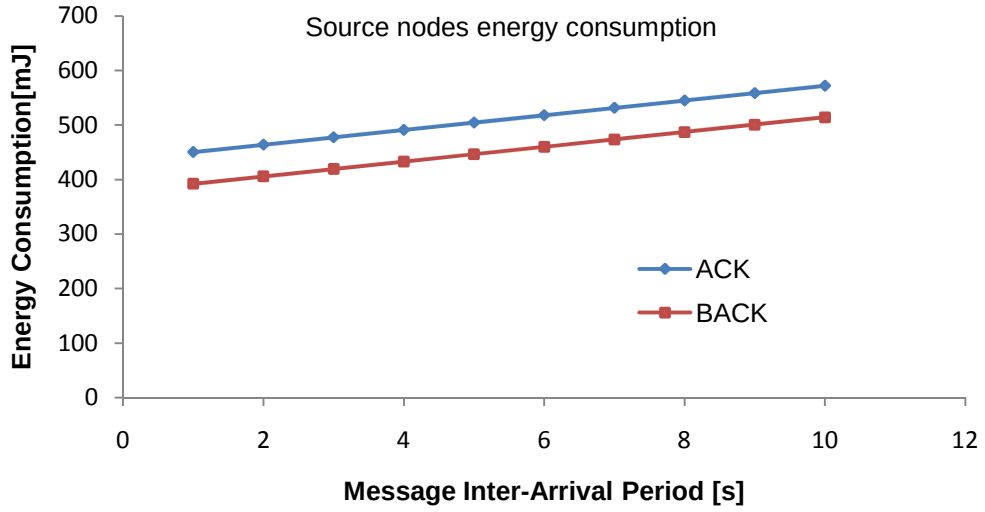


Figure 6.6. Energy consumption in the TR1000 transceiver when using a simple ACK or a Block ACK mechanism.

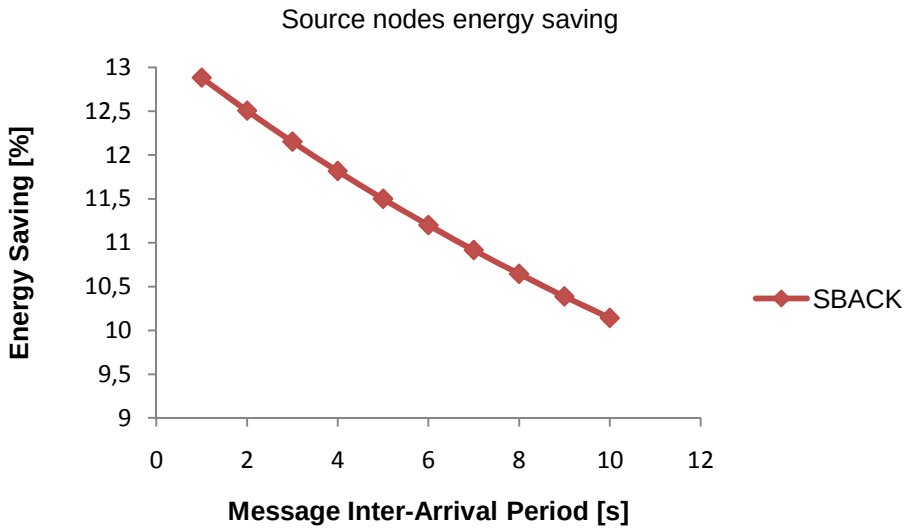


Figure 6.7. Percentage of energy saving in the TR1000 transceiver when using a Block ACK mechanism.

Figure 6.8 presents the source nodes energy consumption as a function of the message inter-arrival period when ACK or BACK mechanisms are used for the CC2420 transceiver. Figure 6.9 presents the percentage of energy saving if the SBACK protocol is used instead of the simple ACK. The energy savings varies from 52.58% to 10.81% when the message inter-arrival period increments from 1 to 10 s.

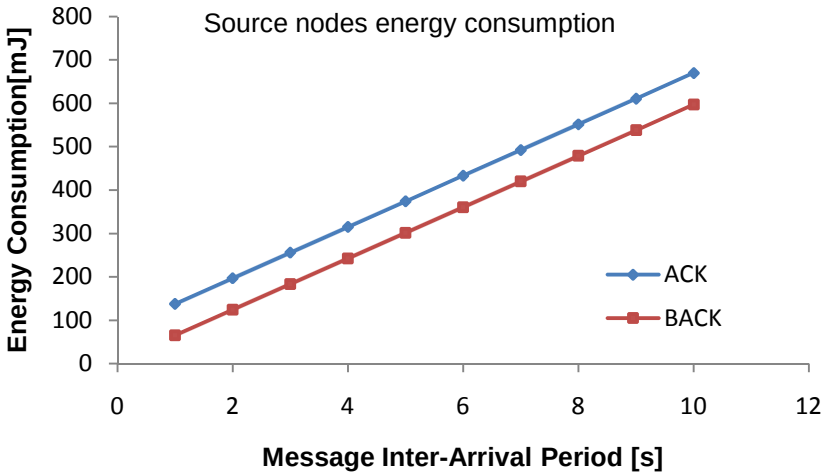


Figure 6.8. Energy consumption in the CC2420 transceiver when using a simple ACK or a Block ACK mechanism.

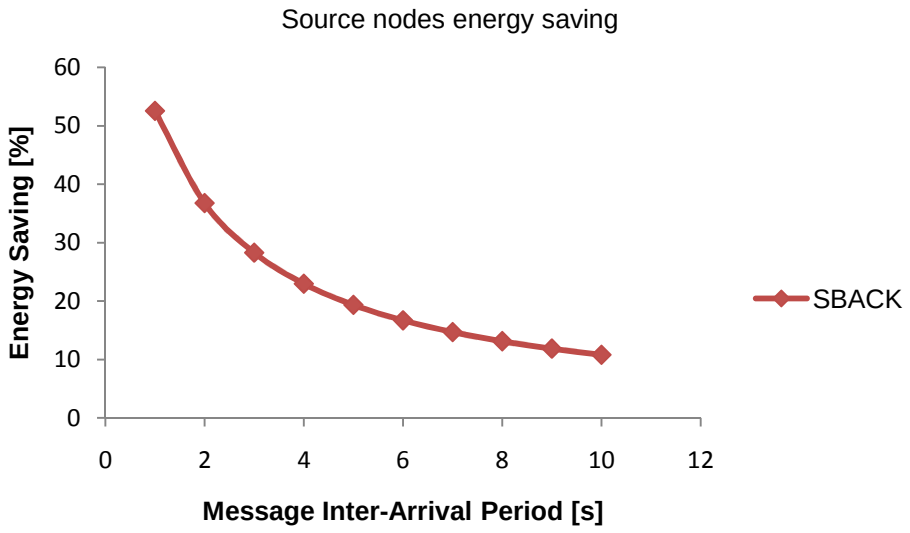


Figure 6.9. Percentage of energy saving in the CC2420 transceiver when using a Block ACK mechanism.

Figure 6.10 presents the source nodes energy consumption as a function of the message inter-arrival period when ACK or BACK mechanisms are used for the AT86RF230 transceiver. Figure 6.11 presents the percentage of energy saving when is used the SBACK protocol.

The energy savings varies from 55.95% to 12.38% when the message inter-arrival period increments from 1 to 10 s.

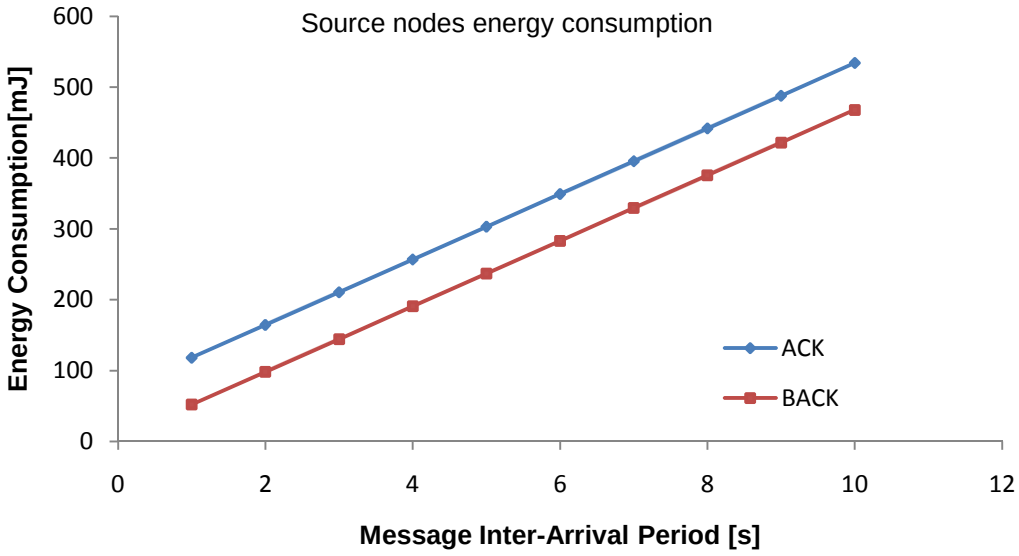


Figure 6.10. Energy consumption in the AT86RF230 transceiver when using a Block ACK mechanism.

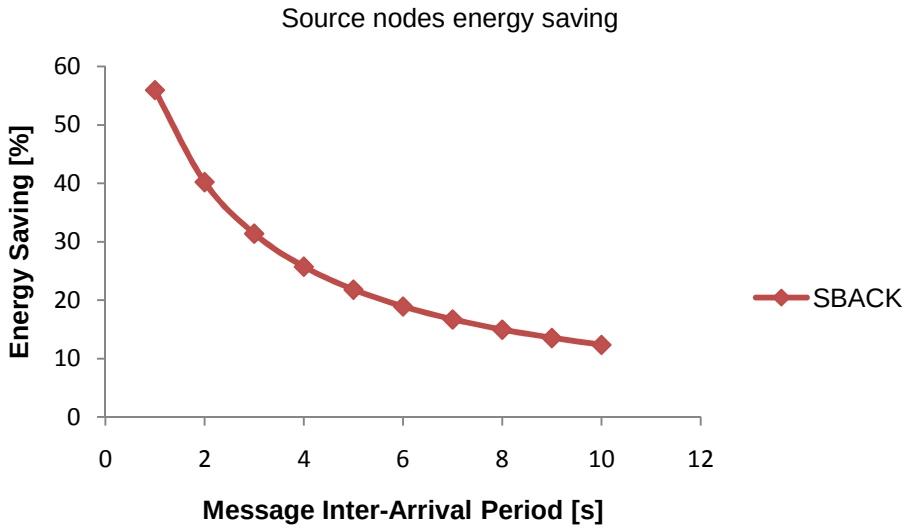


Figure 6.11. Percentage of energy saving in the AT86RF230 transceiver when using a Block ACK mechanism.

Lets considering the case in which the Packet Success Rate/Packet Reception Rate is approximately 90%, i.e. 10% of the total of the packets will be lost. Since the total number of packets is 100, 10 packets will be lost and the energy wasted for the different radio transceivers is Figures Figure 6.12,Figure 6.13 and Figure 6.14.

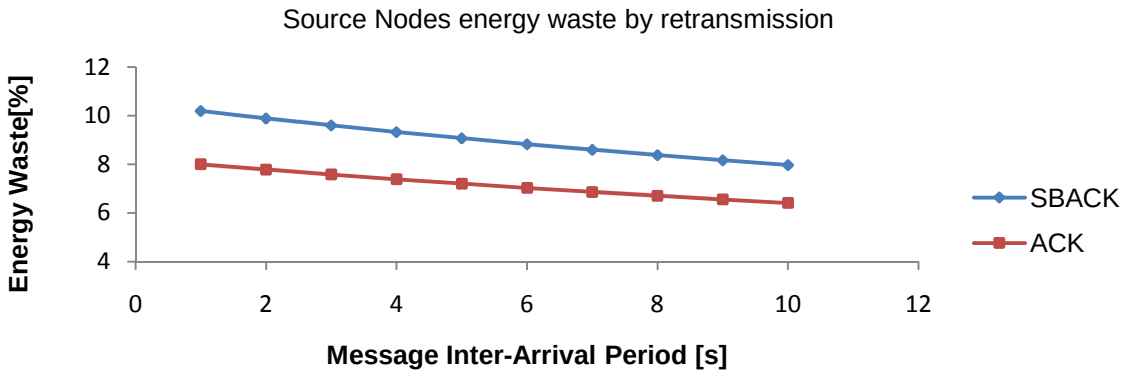


Figure 6.12. Energy waste in the TR1000 transceiver when 10% of the packets need retransmission.

The energy waste using the SBACK and the TR1000 transceiver varies from 10.19% to 7.97% when the message inter-arrival period increments from 1 to 10 s. In the case of S-MAC the energy waste varies from 8.00% to 7.41% when the message inter-arrival period increments from 1 to 10 s.

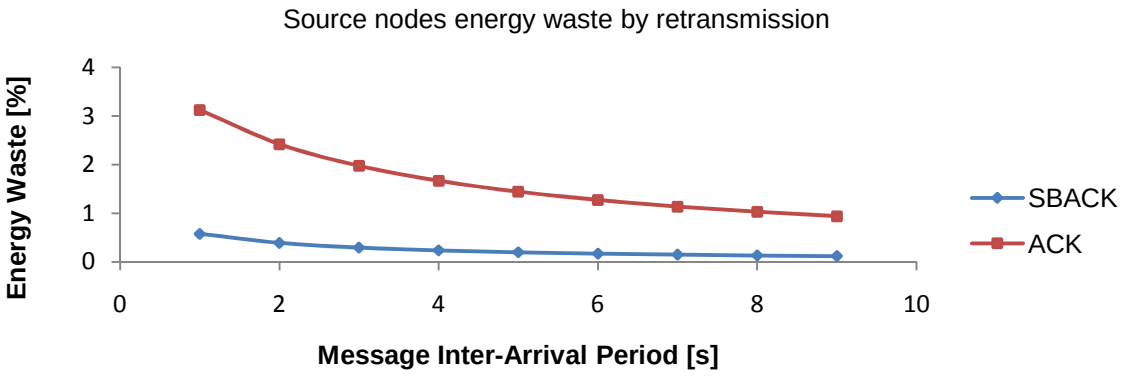


Figure 6.13. Energy waste for the CC2420 transceiver when 10% of the packets need retransmission.

The energy waste using the SBACK and the CC2420 transceiver varies from 0.57% to 0.12% when the message inter-arrival period increments from 1 to 10 s. In the case of S-MAC the energy waste varies from 3.12% to 0.93% when the message inter-arrival period increments from 1 to 10 s.

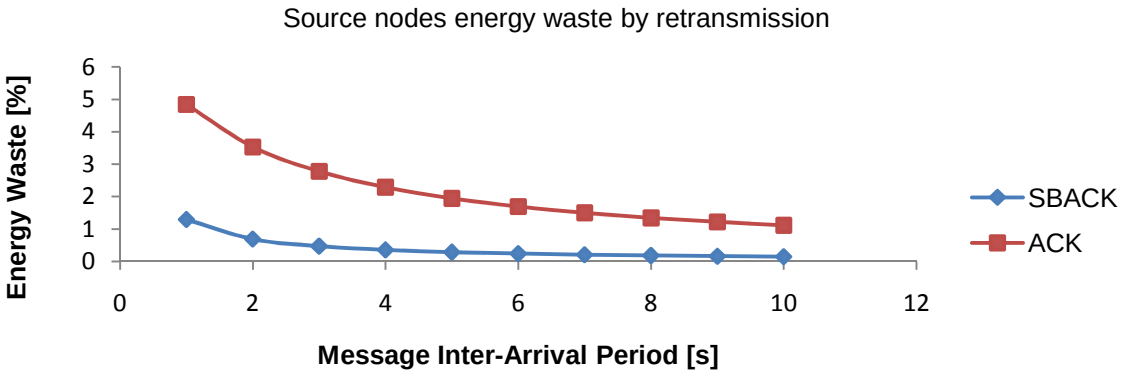


Figure 6.14. Energy waste for the AT86RF230 transceiver 10% of the packets need retransmission.

The energy waste using the SBACK and the AT86RF230 transceiver varies from 1.29% to 0.14% when the message inter-arrival period increments from 1 to 10 s. In the case of S-MAC the energy waste varies from 4.84% to 1.11% when the message inter-arrival period increments from 1 to 10 s.

6.1.4 Average delay for SBACK

The use of SBACK may reduce the delay. Firstly, one considered the case in which all the packets transmitted by the sender nodes are delivered successfully to the receiver's nodes. So, there is no need to retransmitt any packet. Figure 6.15 represents the RTS/CTS/DATA/ACK mechanism without

retransmission in S-MAC while Figure 5.3 presents the Block ACK sequence for the SBACK protocol.

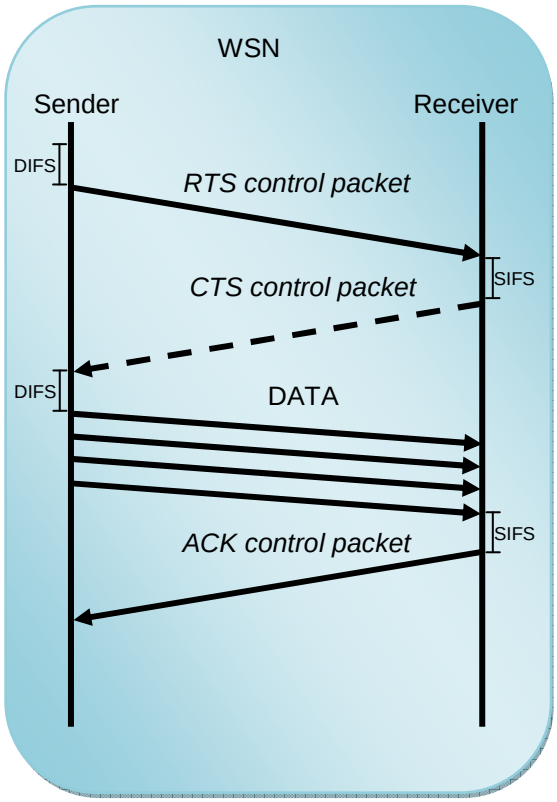


Figure 6.15. RTS/CTS/DATA/ACK sequence without retransmission in S-MAC.

In order to calculate the average delay per packet of S-MAC, one needs to sum all the periods required to transmit the packets plus a sleep delay (the period a sender that as a packet to transmit must wait until the receivers wakes up:

$$\text{Average delay}_{\text{S-MAC}}(\%)=\frac{\text{sleep delay } +2\times T_{Tcp}+(T_{Tdp}+T_{Tcp})\times\text{Number of transmited fragments}}{\text{Total number of transmited fragments}} \quad (6.6)$$

The time spent to transmit a control packet such an RTS, CTS or ACK is given by:

$$T_{Tcp}=(\text{Packet Size})/(\text{Data Rate}) \quad (6.7)$$

where, Packet Size=8 B (6 B Header and 2 B CRC)

The time spent to transmit a data packet is given by:

$$T_{Tdp}=(\text{Packet Size}) / (\text{Data Rate}) \quad (6.8)$$

where, Packet Size=38 B (6 B Header ,30B Payload and 2 B CRC).

In order to calculate the SBACK average delay per packet one needs to sum all the times required to transmit the packets plus a sleep delay (the period that a sender that as a packet to transmit must wait until the receivers wakes up):

$$\text{Average delay}_{\text{S-MAC}}(\%)=\frac{\text{sleep delay } +4\times T_{Tcp}+T_{Tdp}\times(\text{Number of transmited fragments}+2\ )}{\text{Total number of transmited fragments}} \quad (6.9)$$

The time to transmit a control packet such an RTS, CTS or ACK is given by:

$$T_{Tcp}=(\text{Packet Size})/(\text{Data Rate}) \quad (6.10)$$

where, Packet Size=8 B (6 B Header and 2 B CRC)

The time spent to transmit a data packet is given by:

$$T_{Tdp}=(\text{Packet Size}) / (\text{Data Rate}) \quad (6.11)$$

where, Packet Size=38 B (6 B Header ,30 B Payload and 2 B CRC).

The total decrease of the average delay in SBACK (compared with S-MAC) per packet when using the TR1000 transceiver is presented in Figure 6.16.

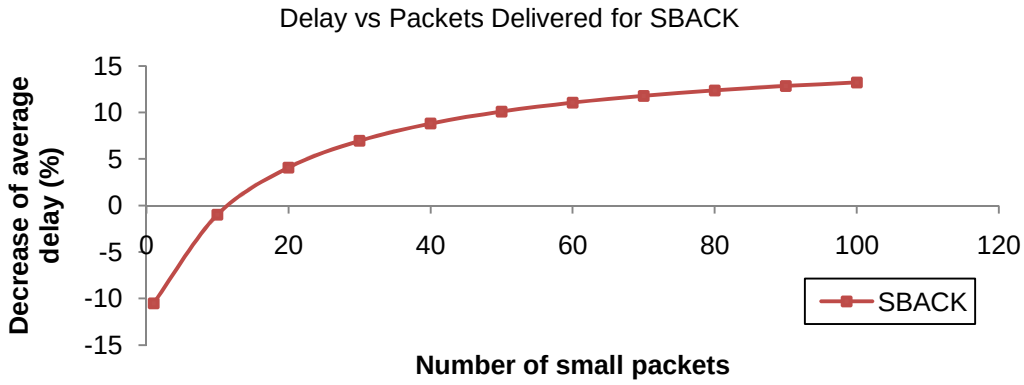


Figure 6.16. Decrease in average delay when using the TR1000 transceiver in SBACK protocol without retransmission (compared with S-MAC).

Considering now that there are no retransmissions of packets, when the TR1000 transceiver is used SBACK only achieves worst performance when the number of fragments is less than 12, after that the decrease of average delay increases with the increase of the fragments sent

The total decrease of the average delay when using the CC2420 and the AT86RF230 transceivers is the same because they have the same data rate, Figure 6.16 shows the values.

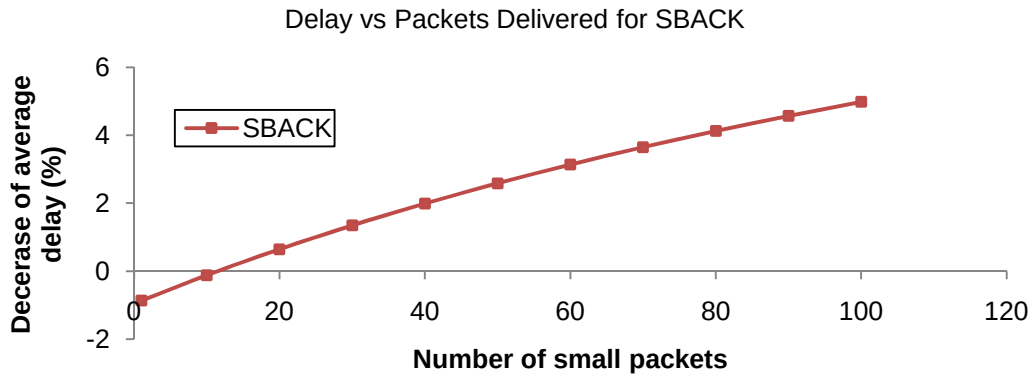


Figure 6.17. Decrease in average delay when using the CC2420 and AT86RF230 transceivers in SBACK protocol without retransmission (compared with S-MAC).

Considering now that there are no retransmissions of packets, when the TR1000 transceiver is used SBACK only achieves worst performance when the number of fragments is less than 11, after that the decrease of average delay increases with the increase of the fragments sent

Secondly, we consider several cases in which some of the packets transmitted by the sender nodes are not delivered to the receiver's nodes, so retransmitting of the packets is needed. Figure 6.18 represents the RTS/CTS/DATA/ACK mechanism with retransmission in S-MAC while Figure 6.19 presents the similar case for the SBACK protocol.

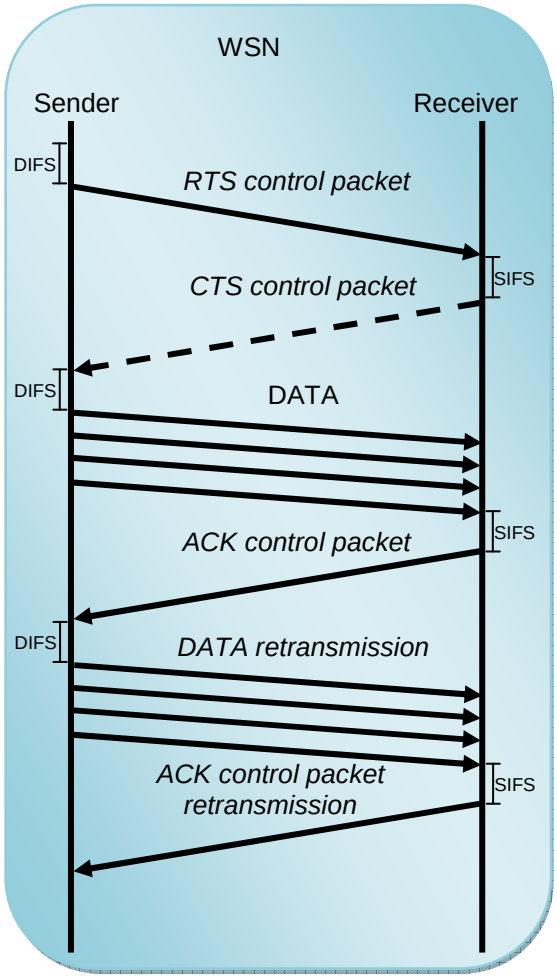


Figure 6.18. RTS/CTS/DATA/ACK sequence with retransmission in S-MAC.

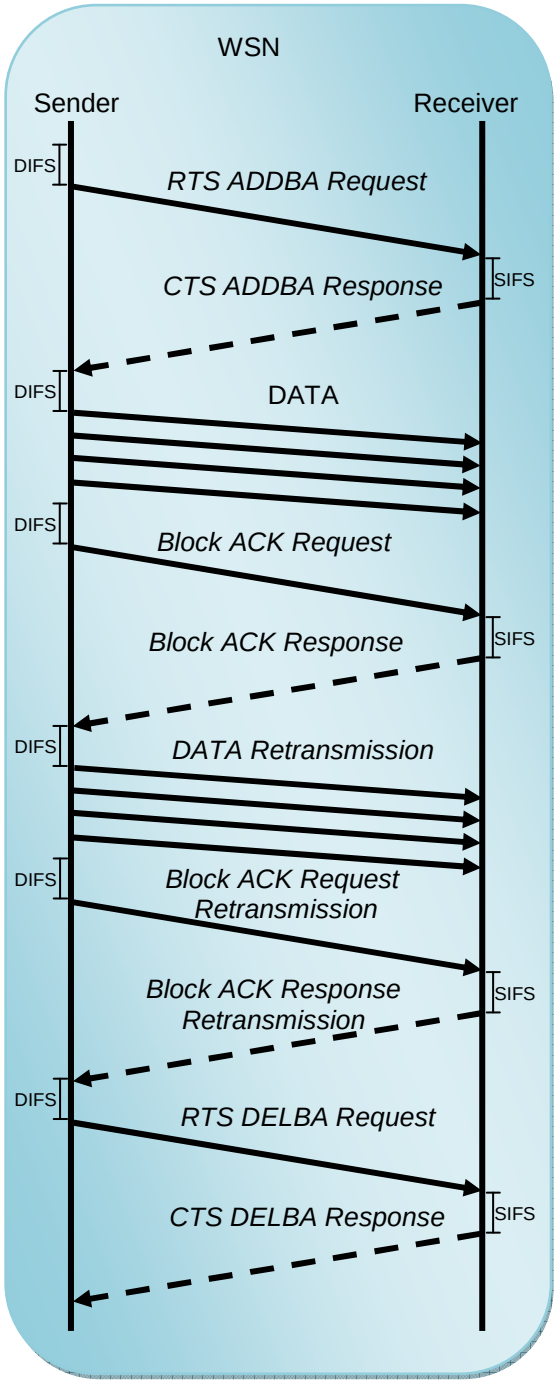


Figure 6.19. SBACK block ACK sequence with retransmission.

In order to determine the average delay per packet for S-MAC when using retransmissions one needs to sum all the time periods required to transmit the packets plus a sleep delay (the period a sender that as a packet to transmit, must wait until the receiver wakes up) plus all the DATA and ACK of retransmitted packets:

$$\text{Average delay}_{\text{S-MAC}}(\%) = \frac{\text{sleep delay} + 2\times T_{Tcp} + (T_{Tdp} + T_{Tcp}) \times (\text{Number of transmited fragments} + \text{Number of retransmited packets})}{\text{Total number of transmited fragments}} \quad (6.12)$$

The time spent to transmit a control packet such, as RTS, CTS or ACK is given by:

$$T_{Tcp} = (\text{Packet Size})/(\text{Data Rate}) \quad (6.13)$$

where, Packet Size=8 B (6 B Header and 2 B CRC)

The time spent to transmit a data packet is given by:

$$T_{Tdp} = (\text{Packet Size})/(\text{Data Rate}) \quad (6.14)$$

where, Packet Size=38 B (6 B Header , 30 B Payload and 2 B CRC).

In order to determine the average delay for SBACK when using retransmissions one needs to sum all the time periods required to transmit the packets plus a sleep delay (the period a sender that as a packet to transmit must wait until the receiver wakes up, plus the DATA and ACK of retransmitted packets plus the special RTS and CTS packets used in the block ACK mechanism.

$$\text{Average delay}_{\text{S-MAC}}(\%) = \frac{\text{sleep delay} + 4\times T_{Tcp} + (T_{Tdp}\times 2) + (T_{Tdp}\times \text{Number of TX fragments}) + T_{Tdp}\times(\text{Number of RET fragments}+2)}{\text{Total number of transmited fragments}} \quad (6.15)$$

The time spent to transmit a control packet such an RTS, CTS or ACK is given by:

$$T_{Tcp} = (\text{Packet Size})/(\text{Data Rate}) \quad (6.16)$$

where, Packet Size=8 B (6 B Header and 2 B CRC)

The time spent to transmit a data packet is given by:

$$T_{Tcp} = (\text{Packet Size})/(\text{Data Rate}) \quad (6.17)$$

where, Packet Size=38 B (6 B Header , 30 B Payload and 2 B CRC).

The total decrease of average delay, when retransmitting 10%, 50% and 70% of the total number of transmitted packets for TR1000 transceiver is presented in Figure 6.20.

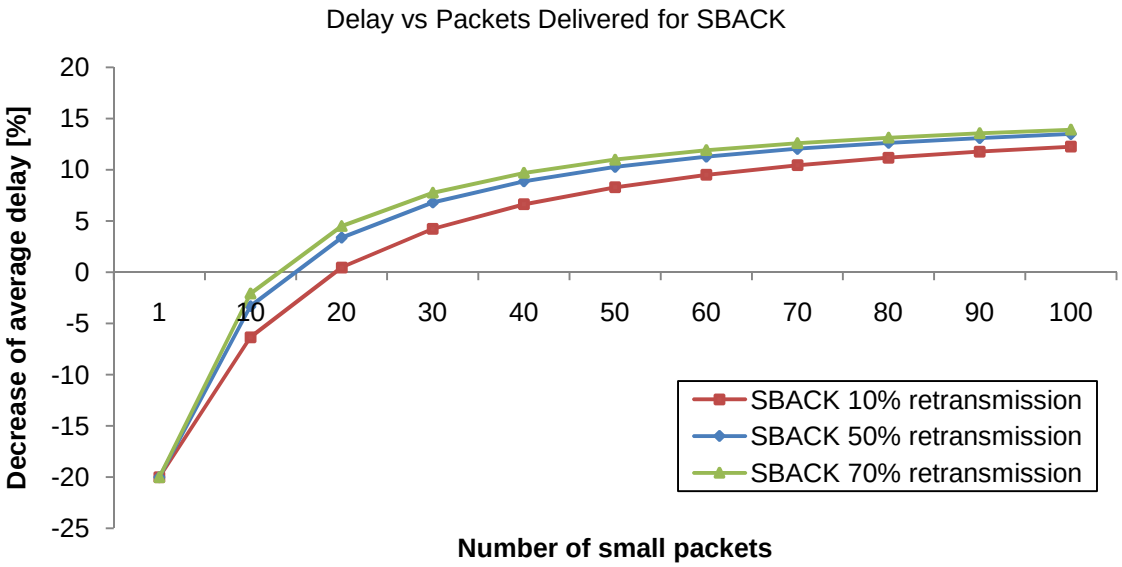


Figure 6.20. Decrease in average delay when using the TR1000 transceiver in SBACK protocol with retransmission of 10%, 50% and 70% of the packets (compared with S-MAC).

Considering there are retransmissions of packets, when the TR1000 transceiver is used SBACK only achieves worst performance when the number of fragments is less than 11 in the case we retransmit 10% of the packets, 15 in the case we retransmit 50% of the packets and 20 in the case retransmit 70% of the packets, after that the decrease of average delay increases with the increase of the fragments sent.

The total decrease of the average delay when retransmitting 10%, 50% and 70% of the total number of packets transmitted using the CC2420 and AT86RF230 transceivers is presented in Figure 6.21.

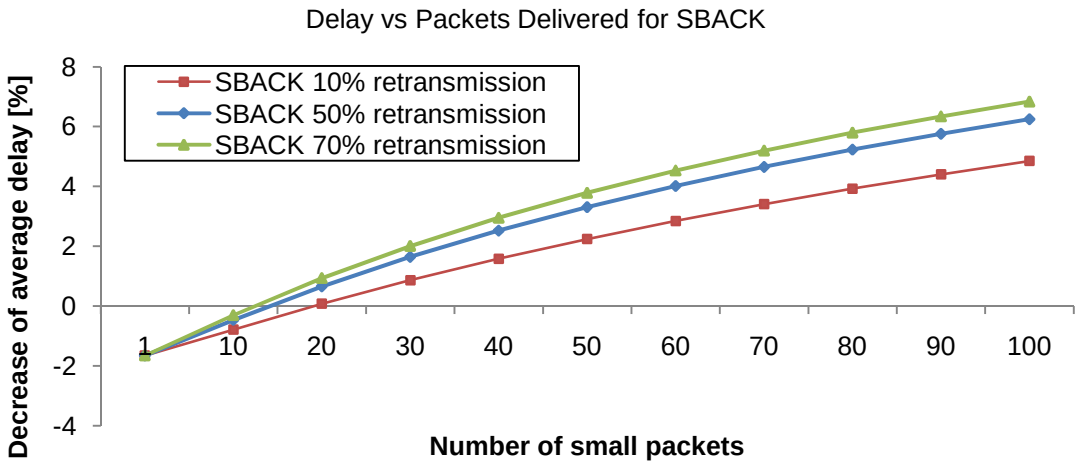


Figure 6.21. Decrease in average delay when using the CC2420 and AT86RF230 transceivers in SBACK protocol with retransmission of 10%, 50% and 70% of packets (compared with S-MAC).

Considering there are retransmissions of packets, when the CC2420 and AT86RF230 transceivers are used SBACK only achieves worst performance when the number of fragments is less than 12 in the case we retransmit 10% of the packets, 13 in the case we retransmit 50% of the packets and 20 in the case retransmit 70% of the packets, after that the decrease of average delay increases with the increase of the fragments sent.

6.2 Battery capacity and lifetime for IRIS mote

As seen in Chapter 2, the IRIS mote presented in Figure 2.7 and attached to the Sensor Board shown in Figure 2.3 is responsible to transmit all the data from the pregnant woman sensors to the Mote Interface Board (Gateway). The Mote Interface Board is directly connected to the Centralized Management of Resources entity. As so, knowing the energy consumption is fundamental, for making estimation of how longer the batteries of the mode that will be available. So, based on the specifications of the motes as presented in [POWE09], two models for energy consumption prediction were studied, involving not only the radio transceiver, but also a microprocessor, a logger and a sensor board. In the first model, during 1% of the time the module will be active and while 99% of the time the module will enter in sleep mode. In the second model, it is assumed that the module will be active during 10% of the time while in the remaining 90% of the time the module will be in the sleep mode. This way it will be possible to know when we need to change the batteries, based on the duration of the transmission.

Table 6.3 presents the expected battery lifetime vs system current usage, based on two duty cycle models.

Table 6.3. System Specifications for the current consumption.

System Specifications					
Current			**Duty Cycles**		
	value	units	Model 1	Model 2	units
Microprocessor (Atmega1281)					
current in full operation mode	8	mA	1	10	%
current in sleep mode	8	µA	99	90	%
Radio					
current in receive mode	15.5	mA	0.75	8	%
current in transmit mode	16.5	mA	0.25	2	%
Current in sleep mode	2	µA	99	90	%
Logger					
Write mode	15	mA	0	5	%
Read mode	4	mA	0	5	%
Sleep mode	2	µA	100	90	%
Sensor Board					
current in full operation mode	5	mA	1	10	%
current sleep mode	5	µA	99	90	%
Battery Specifcations					
Capacity Loss/Yr	3	%			

The computed capacity for the µP is given in mA·h by the following equation:

$$Q_{\mu P} = \frac{\text{current (full operation)} \times \text{Dutty Cycle}}{100} + \frac{0.001 \times \text{current sleep} \times \text{Dutty Cycle}}{100} \quad (6.18)$$

The computed capacity for the Radio is given in mA·h by the following equation:

$$Q_{radio} = \frac{\text{current in receive mode} \times \text{Dutty Cycle}}{100} + \frac{\text{current in transmit mode} \times \text{Dutty Cycle}}{100} + \frac{0.01 \times \text{current in sleep mode} \times \text{Dutty Cycle}}{100} \quad (6.19)$$

The computed capacity for the Flash Memory in mA·h is given by the fllowing equation:

$$Q_{Memory} = \frac{0.001 \times \text{current sleep mode}}{100} \quad (6.20)$$

The computed capacity used for the Sensor Board in mA·h is given by the following equation:

$$Q_{Sensor\ Board} = \frac{\text{current in full operation} \times \text{Dutty Cycle}}{100} + \frac{\text{current in slepp mode} \times \text{Dutty Cycle}}{100} \quad (6.21)$$

Table 6.4 presents the capacity computed to each of these components, enables to compute the total requirements for capacity.

Table 6.4. Total capacity requirements for the different mote components.

Q[mA·h]	**Model 1**	**Model 2**
µP	0.0879	0.8072
Radio	0.1595	1.5718
Flash Memory	0.0020	0.0018
Sensor Board	0.0550	0.5045
Total capacity (mA·h) requirements	**0.3044**	**2.8853**

The battery life time is given, in months, by the following equation:

$$\text{Battery Life (months)} = \frac{Q(\text{mA·h})}{\text{Total Current (mA·h)}} \times \frac{12}{24 \times 365} \quad (6.22)$$

Including variation of the battery capacity loss one obtains the following modified equation:

$$Battery\ Life_{\ \text{modified}}(\text{months}) = \text{Battery Life (months)} \times \left(\frac{\text{1-Battery Life (months)} \times \text{Capacity Loss(Yr)} \times 0.01}{12}\right) \quad (6.23)$$

Table 6.5 presents results for the battery life for different values of the battery capacity

Table 6.5. Computed battery lifetime vs battery size.

Computed battery life vs battery size	Model 1 (active time 1%)	Model 2 (active time 10%)
Battery Capacity (mA·h)	**Battery Life (Months)**	**Battery Life (Months)**
250	1.13	0.23
500	2.25	0.24
1000	4.50	0.47
1500	6.75	0.71
2000	9.00	0.95
3000	13.50	1.42
	Including Battery Capacity Loss	**Including Battery Capacity Loss**
250	1.12	0.23
500	2.24	0.24
1000	4.45	0.47
1500	6.64	0.71
2000	8.80	0.95
3000	13.05	1.42

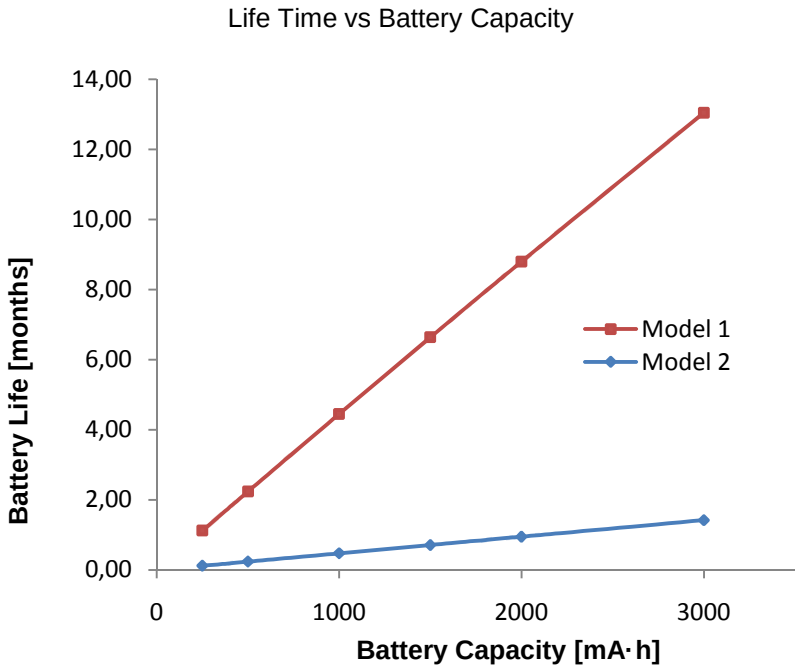


Figure 6.22. Life Time vs Battery Capacity for models 1 and 2 (active times of 1 and 10% respectively).

Figure 6.22 presents the results for the Battery Life Computations for the two models (with different duty cycles). The first model assumes that microprocessor is 1% of the time in full operation and 99% of the time in the sleep mode; the second model assumes that the microprocessor is 10% of the time in full operation and 90% of the time in the sleep mode. In the first model the battery life in months is ten times higher when comparing with the second model. This and other system specifications can be seen in Table 6.3.

As it can be seen in Figure 6.22, to achieve more than one year battery life, the mote must sleep most of the time, as presented in model 1. However, when there is a need for a real time application, this model is not the best choice because, in this scenario, the mote is sleeping most of the time.

6.3 Conclusions

This chapter presents the results for the behaviour of the SBACK MAC protocol compared with S-MAC. An energy model was developed and introduced in the OMNeT++ simulator. The model was firstly implemented to test and validate the S-MAC protocol and the results were compared with the ones presented by the authors of the protocol [YeHE04a., The good match validated this model. The model was applied to both S-MAC and SBACK and to 3 different radios. Considering 0% of packet losses, SBACK decreases the energy consumption when directly compared with S-MAC. For the TR1000 transceiver, if the message inter-arrival varies from 1 to 10 s the energy saving varies from 12 to 10 %. For the CC2420 transceiver if the message inter-arrival varies from 1 to 10 seconds the energy saving varies from 52 to 10 % in the AT86RF230 transceiver, if the message inter-arrival varies from 1 to 10 seconds the energy saving varies from 55 to 12 %.

If 10% of the packets need retransmission, in the case the TR1000 is used the use of the Block ACK mechanism corresponds an higher energy consumption than the DATA/ACK handshake. This is due the fact the TR1000 requires much less energy to receive than to transmit; so, receiving ACK is quite “cheap”. However, considering that the radios that operate at the 2.4GHz band (AT86RF230 and CC2420), which have similar energy consumption for sending and receiving, the SBACK protocol clearly consumes less energy than S-MAC.

Considering now that there are no retransmissions of packets, when the TR1000 transceiver is used, SBACK only achieves worst performance when the number of fragments is less than 12. After that, the decrease of average delay increases with the increase of the fragments sent. The same thing happens in the cases where the CC2420 and the AT86RF230 transceiver are considered.

In the cases we need to retransmit 10%, 50% or 70% of the total amount of packets sent with the TR1000. SBACK only achieves worst performance when the number of fragments is less than 11,15 and 20, respectively. After that the decrease of average delay increases with the increase of the fragments sent.

For the CC2420 and AT86RF230 transceivers, when we need to retransmit 10%, 50% or 70% of the total amount of packets, SBACK only achieves worst performance when the number of fragments is less than 10,13 and 20 respectively. After that the decrease of average delay increases with the increase of the fragments sent.

The life time of an IRIS mote, the one chosen to be used in the Smart-Clothing project, was also studied. The formulas for energy usage were determined for every mote component and the trend for life time for 2 different operating motes was calculated. For the set of considered parameters, using a 3000 mA·h battery and a duty cycle of 1%, one can expect the node to last more than one year without the need for battery replacement.

Chapter 7

Conclusions

In this chapter, the main conclusions of this thesis are pointed out as well as some suggestions for future work.

This thesis gives contributions on Smart-Clothing applications and presents an innovative MAC protocol for WSN. The new protocol, called SBACK MAC protocol tries to reduce the energy wasted by using a Block ACK mechanism to improve channel efficiency and aggregate several ACK into one special frame, called *Block ACK Response*. This method reduces the power consumption by transmitting less ACK control packets and by decreasing the time period the transceiver should switch between different states. An analytical model was proposed for the energy to validate the S-MAC protocol, and the good match between this model and the experimental results from [YeHE02a] enables the validation of the model.

After this validating, the model was applied with 3 different radios with packet loss of 10%, and in all the cases SBACK decreases the energy consumption when directly compared with S-MAC. When we compare the energy consumption between SBACK and S-MAC, considering the transmission of ACK packets for the TR1000 transceiver, if the message inter-arrival varies from 1 to 10 s, the energy saving varies from 12 to 10 %. For the CC2420 transceiver, if the message inter-arrival varies from 1 to 10 s the energy saving varies from 52 to 10%. For the AT86RF230 transceiver, if the message inter-arrival varies from 1 to 10 s the energy saving varies from 55 to 12%.

If 10% of the packets used in the transmission were lost and they need retransmission, in the case the TR1000 is used, the Block ACK mechanism consumes more energy than the DATA/ACK handshake. This is due to the fact the TR1000 requires much less energy to receive than to transmit and receiving ACK is quite “cheap”. However, if we use radios with a high data rate (AT86RF230 and CC2420) which have similar energy consumption for sending and receiving, SBACK consumes less energy than S-MAC.

With TR1000, when there are no retransmissions of packets, SBACK achieves worst performance in terms of delay if the number of fragments is less than 12. After that the improvement increases with the increase of fragments sent. The same thing happens for the CC2420 and the AT86RF230 transceivers.

When we need to retransmit 10%, 50% or 70% of the total amount of packets sent, SBACK only achieves worst performance when the number of fragments in terms of delay is less than 11, 15 and 20, respectively. After that the decrease of average delay increases with the increase of the fragments sent.

With the CC2420 and AT86RF230 transceivers, when we need to retransmit 10%, 50% or 70% of the total amount of packets, SBACK only achieves worst performance when the number of fragments is less than 11,13 and 20, respectively. After that the decrease of the average delay increases with the increase of the fragments sent. For future work, it is suggested an improvement of the simulator, since it does not automatically generates the chart related with energy consumption or other charts with related interest. Another suggestion is to introduce an adaptive sleeping listening to the node who overhears its neighbours’ transmissions in order to wake-up for a short period of time at the end of the transmission, to listen to the others.

Annex 1

Wireless Sensor Network Platforms

The Annex 1 presents a table with a brief description of some Wireless Sensor Network Platforms.

Table A. 1. Simulation Tools

Platform	Program + Data Memory	External Memory	RF Transceiver	Frequency
Cricket	4 KB RAM	128 KB	CC1000	433 MHz
Imote2	32 MB SRAM	32 MB Flash	CC2420	2.4 GHz
Imote2.Net Edition	32 MB SRAM	32 MB Flash	CC2420	2.4 GHz
Intelmote	64 KB RAM	512 KB	Zeevo-BT	2.4 GHz
Intelmote2	256 KB RAM	32 MB	CC2420	2.4 GHz
IRIS	8 KB RAM	128 KB Flash	AT86RF230	2.4 GHz
Mica	128+4KB RAM	512 KB Flash	TR1000	433/916 MHz
mica2	4 KB RAM	128 KB Flash	CC1000	315/433/916 MHz
mica2Dot	4 KB RAM	128 KB Flash	CC1000	315/433/916 MHz
MICAz	4 KB RAM	128 KB Flash	CC2420	2.4 GHz
Rene	512 bytes RAM	8K Flash	TR1000	916 MHz
Sun Spot	512 KB RAM	4 MB Flash	CC2420	2.4 GHz
TelosA	2 KB RAM	60 KB Flash	CC2420	2.4 GHz
TelosB	10 KB RAM	48 KB Flash	CC2420	2.4 GHz
TmoteSky	10 KB RAM	48 KB Flash	CC2420	2.4 GHz

Annex 2

Simulation Tools

The Annex 2 presents a report of the existing Simulation Tools for Wireless Sensor Network.

Introduction

The convergence of the Internet, communications, and information technologies coupled with recent engineering advances, is paving the way for a new generation of inexpensive sensor and actuators, capable of achieving a high order of spatial and temporal resolution and accuracy. The technology for sensing and control includes sensor arrays, electric and magnetic field sensors, seismic sensors, radio-wave frequency sensors, electronic and infrared sensors, laser radars, and location and navigation sensors.

Wireless Sensor and Actuator Networks (WSANs) are among the most addressed research fields in the area of information and communication technologies (ICT) these days, in the US, Europe and Asia. WSANs are composed of possibly a large number of tiny, autonomous sensor devices and actuators equipped with wireless communication capabilities. Theory of control systems is involved, networking, middleware, application layer issues are also relevant. The joint consideration of hardware and software aspects is needed, and their use can range from biomedical to industrial or automotive applications or from military to civil environments.

With this document we try to produce a list of all available simulators that can give us several choices for simulating Medium Access Control (MAC) and Physical Layer (PHY), as well as cross-layer design issues and network aspects (e.g., routing).

Medium Access Control (MAC) protocols solve a seemingly simple task: they coordinate the times and opportunities when a number of nodes access a shared communication medium. An “unoverseeable” number of protocols have emerged in more than thirty years of research in this area. They differ, among others, in types of media they use and in the performance requirements for which they are optimized

.

Objectives

The principal objective of this document is to provide a first approach of the different Simulation Tools for the Wireless Sensor Network and decide what tool we can use to validate the data from the communication among motes, by considering, apart from PHY, MAC, network, and Transport layer issues, e.g., routing among different wireless sensor devices, having in mind the minimization of the overall power consumption, and contributing to the actual achievement of a WSN solution.

Introduction to Simulation Tools

A simulation is the imitation of the operation of a real world process or system over time. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system.

The behaviour of a system as it evolves over time is studied by developing a simulation model. This model usually takes the form of a set of assumptions concerning the operation of the system.

Thus, simulation modelling can be used both as an analysis tool for predicting the effect of changes to existing systems and as a design tool to predict the performance of new systems under varying sets of circumstances.

However, many real-world systems are so complex that models of these systems are virtually impossible to solve mathematically. In these instances, numerical computer-based simulation can be used to imitate the behaviour of the system over time. Data is collected from the simulation as if a real-time system was being observed. This simulation-generated data is used to estimate the measures of performance of the system.

Comparing Simulation Tools

Our aim is to address WSN discrete event driven simulation tools, although time driven approaches can also be used/considered. Different simulation languages and tools are available; hence, a comparison among them is in order.

In this document we will focus in three Wireless Sensor Networks Simulators that can give us a theorical approach of the several paradigms such communication, networking protocols, middleware, security and management.

With the ever-increasing popularity of Wireless Sensor Networks and their tremendous potential to penetrate multiple aspects of our lives, we believe that this document is timely and addressed the needs of a growing community of engineers, networks professionals and managers, and educators. Sensor networks designs must be optimized to extend the network lifetime. The energy and bandwidth constraints and the potential large-scale deployment pose challenges to efficient resource allocation and sensor management.

Next section presents the OMNeT++, NetTopo, and GTNetS simulators. We decided to choose these simulators because by comparing those with the others that are presented in appendix they present the following advantages: they are developed for WSN or including packages for WSN, include the level of detail (nodes or even network details) that as appropriate, already include some of the protocols, e.g., IEEE 802.15.4 PHY and MAC protocol, the source code is public and finally in the major of then the typical interface is user-friendly.

OMNeT++

OMNeT++ is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks and because of its generic and flexible architecture; it has been successfully used in other areas like the simulation of IT systems, queuing networks, hardware architectures and business processes as well. OMNeT++ is rapidly becoming a popular simulation platform in the scientific community as well as in industrial settings. Several Open Source Simulation Models have been published, in the field of the Internet simulations (IP, IPv6, MPLS, etc), mobility and ad-hoc simulations, and other areas.

The goal of this description is to bring together OMNeT++ users and their tools, applications and ideas. It intends to provide a forum for presentations of recent developments and novel ideas in the broad context of network simulation with focus on OMNeT++. It will bring together researchers to focus on the important topics of integrating simulation models, coupling different simulation tools, providing better modelling approaches, and contributing to the active modelling and simulation community with respect to identifying some of the most promising candidate solution methods, architectures and techniques to address the various challenges of network simulation. The benefits are two-fold: On the one hand OMNeT++ users get into direct discussion and on the other hand they can meet with developers. Furthermore, the developers can pick up ideas for the future development.

Topics of interest include, but are not limited to:

- Parallel simulation
- Simulation control
- Result interpretation and analysis
- Debugging
- Simulation in the loop
- Modelling techniques
- Coupling with other simulation/emulation tools
- Integration of hardware-specific code
- Cross-layer protocol design methodologies
- Mobility models
- Simulation of wireless and P2P networks
- Industrial applications
- Use of OMNeT++ in other domains

The OMNeT++ model is a collection of hierarchically nested modules as shown in Figure 1. The top-level module is also called the System Module or Network. This module contains one or more sub-modules each of which could contain other sub-modules. The modules can be nested to any depth and hence it is possible to capture complex system models in OMNeT++. Modules are distinguished as being either simple or compound. A simple module is associated with a C++ file that supplies the desired behaviours that encapsulate algorithms. Simple modules form the lowest level of the module hierarchy. Users implement simple modules in C++ using the OMNeT++ simulation class library. Compound modules are aggregates of simple modules and are not directly associated with a C++ file that supplies behaviours. Modules communicate by exchanging messages. Each message may be a complex data structure. Messages may be exchanged directly between simple modules (based on their unique ID) or via a series of gates and connections. Messages represent frames or packets in a computer network. The local simulation time advances when a module receives messages from another module or from itself. Self-messages are used by a module to schedule events at a later time. The structure and interface of the modules are specified using a network description language. They implement the underlying behaviours of simple modules. Simulation executions are easily configured via initialization files. It tracks the events generated and ensures that messages are delivered to the right modules at the right time [1].

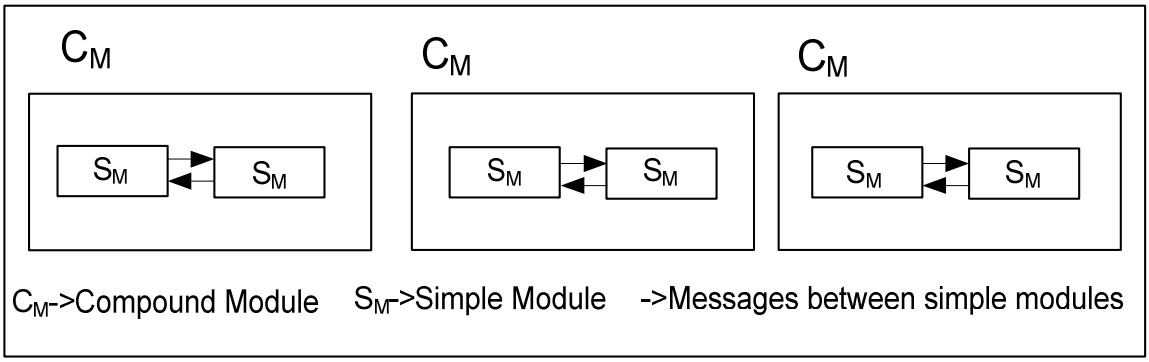


Figure B. 1. Simple and Compound modules in OMNeT++.

2.2.2 NetTopo

NetTopo is an open source research-oriented simulator & visualiser, designed to test and validate algorithms for Wireless Sensor Networks.

The goal of NetTopo is to build a sensor network simulation and visualization tool that gives users extraordinary flexibility to simulate their own algorithms and is a compelling replacement of commercial simulator focusing on visualization of the communication in the real Wireless Sensor Network test bed.

NetTopo will help in the investigation of algorithms in WSNs [2]. With respect to the simulation module, users can easily define a large number of on-demand initial parameters for sensor nodes (we can use more than 390 nodes), e.g. residential energy, transmission bandwidth, radio radius, etc. Users also can define and extend the internal processing behaviour of sensor nodes, such as energy consumption, bandwidth management. It allows users to simulate an extremely large scale heterogeneous WSN. For the visualization module, it works as a plug-in component to visualize test bed's connection status, topology, sensed data, etc.

These two modules paint the virtual sensor nodes and links on the same canvas which is an integration point for centralized visualization. Since the node attributes and internal operations are user definable, it guarantees the simulated virtual nodes to have the same properties with those of real nodes. The sensed data captured from the real sensor nodes can drive the simulation in a pre-deployed virtual WSN. Topology layouts and algorithms of virtual WSN are customizable and work as user defined plug-ins, both of which can easily match the corresponding topology and algorithms of real WSN testbed. As a major contribution of this research work, NetTopo is released as Open Source Software on the SourceForge. Currently, it has more than eight java classes and 11,000 Java lines source codes. Users can freely download the latest version of NetTopo by accessing the NetTopo website.

The friendly GUI makes it easy to use and the modular components enable it to be easily extended. Due to the algorithm-oriented design, NetTopo supports the simulation for an extremely large scale network. It is useful for the rapid prototyping of an algorithm. The visualization function uncovers the real device based WSN topology and displays sensed data. Based on modular components design and common graphical resources, visualization process can drive the simulation.

Generally, such integration takes the first step into the whole vision that applications can run partially in a simulation environment and partially in a physical WSN testbed and interact with each other to create an environment where algorithms can be much more accurately tested and validated.

However, there still exist a few limitations on NetTopo and it needs a lot of future work(s) listed as follows:

- ✓ Integrating with GSN [3] middleware. So far, NetTopo only support visualizing Crossbow WSN, although the framework can be easily extended to integrate new visualisers. GSN is a sensor network middleware. It provides a large number of wrappers (currently more than 25 wrappers) for extracting data from heterogeneous sensor devices. This can help to reduce the workload to implement new wrappers for some GSN supported sensor devices.
- ✓ Simulation process controls the sensor device communication. As mentioned above, currently in NetTopo, visualization process can drive the simulation. However, if the driver of the sensor hardware provides API for controlling the sensors' actions such as route choosing, then our simulation result can be easily applied in testbed for performance comparison.
- ✓ 3D visualization and localization. The basic 3D visualization model for the smart home/office

scenario in NetTopo as Figure 2 shows. As the future implementation work, we could further implement this 3D visualization model and integrate NetTopo with GPS to provide sensor nodes localization functions.

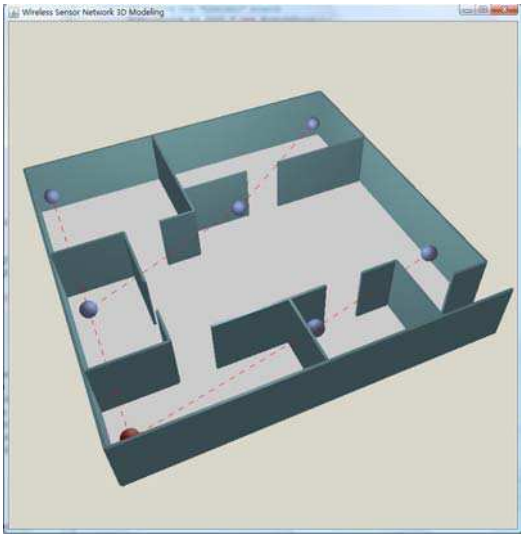


Figure B. 2. A 3D Visualization Model.

NetTopo consists of both simulation and visualization functions. These two functions need to interact with each other and access/manipulate some common resources. For focusing on the integration issues of them, we use component based NetTopo architecture, which is flexible enough for adding new components in the future. The basic architecture is illustrated in Figure 3. Main Control and Utility are two components involved in all layers. Main Control is the core component working as a coordinator in charge of the interactions of other components. It can be regarded as adaptor between input and output interfaces of other components and enables them to work smoothly.

Utility provides some basic services, e.g., defined application exceptions, format verification, number transforms, and dialogue wrappers. File Manager is for the purpose of data persistence, e.g. logging runtime information, recording statistical results, keeping references of virtual sensor nodes, etc. Log information and statistical results are recorded as character streams into human readable format, Figure 3.

References of virtual sensor nodes are stored as serialized format for easy recovery and reuse. All these references are encapsulated in Virtual WSN, which works like a runtime sensor nodes repository and also declares interface to allow other components to add new virtual nodes, delete particular nodes, retrieve the same type of nodes and their derived children, etc. Figure 4. NetTopo Architecture Node, Topology and Algorithm components are designed as highly extensible modules that can be regarded as plug-ins. Node represents a virtual sensor node. Virtual sensor nodes do not have fixed properties or structures. For example, sensor nodes can have very different sensing attributes: temperature, humidity, vibration, pressure, etc. To allow users to create their own virtual

sensor nodes, an abstract interface named VNode is declared to define several basic methods representing actions of a real sensor node.

Any user desired node that wishes to run on the simulator must implement the VNode interface. Topology stands for the topology to be deployed in Virtual WSN. Network topology can be various shapes, e.g., line, circle, triangle and tree. Users can flexibly implement any needed network topology. Algorithm represents an algorithm to be applied in the Virtual WSN. The algorithm can be any routing, clustering, scheduling, controlling algorithm, etc.

Users can freely implement their needed algorithms for their specific studies. The graphical user interface (GUI) in Figure 4 consists of three major components: a display canvas (on the upper left), which can be dragged in case of viewing a large scale WSN, a property tab for displaying node properties (on the upper right), and a display console for logging and debugging information. Painter is separated from the main GUI due to the frequent paining tasks. The painter is also designed as an abstract interface for various painting requirements, e.g., 2D or 3D. The specific painter used in Figure 4 is Painter_2D. Additionally, the painter encapsulates the lower painting API, interacts with the Virtual WSN and main GUI and provides advanced painting methods, e.g. it can paint a link between any two nodes by just using their ID information.

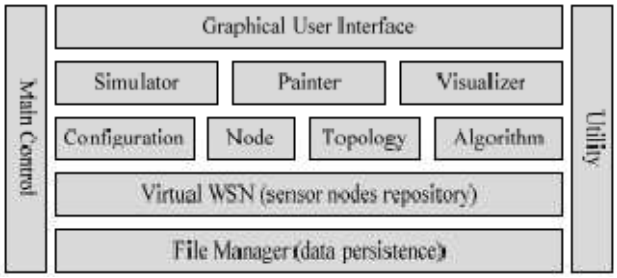


Figure B. 3. NetTopo Architecture.

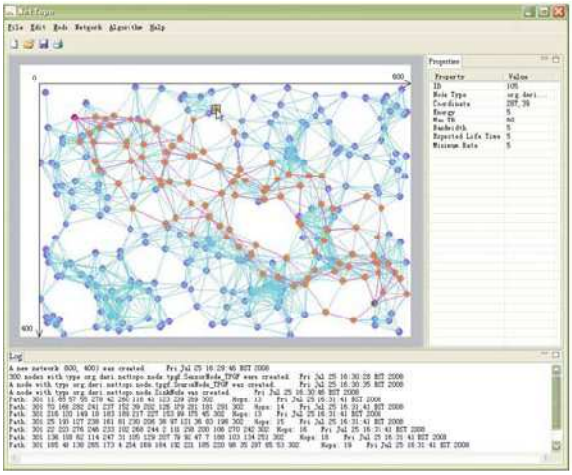


Figure B. 4. NetTopo main GUI (the TPGF [20] multipath routing algorithm is executed in the WSN).

GTNetS

The Georgia Tech Network Simulator (GTNetS) is a C++ free open-source event-driven simulator developed by George Riley at Georgia Tech, Atlanta, USA. It enables researchers world-wide to easily model and simulate computer networks both wired and wire-less. GTNetS has been developed aiming at scalability (for more details, see papers published by G. Riley et al.), it is a full-featured network simulation environment that allows researchers in computer networks to study the behavior of moderate to large scale networks, under a variety of conditions. The design philosophy of GTNetS is to create a simulation environment that is structured much like actual networks are structured. For example, in GTNetS, there is clear and distinct separation of protocol stack layers.

Packets in GTNetS consist of a list of protocol data units (PDUs) that are appended and removed from the packet as it moves down and up the protocol stack. Simulation objects representing network nodes have one or more Interfaces, each of which can have an associated IP address and an associated link. Layer 4 protocol objects in GTNetS are bound to ports, in a fashion nearly identical to the binding to ports in real network protocols. Connections between protocol objects at the transport layer are specified using a source IP, source port, destination IP, and destination port tuple just like actual TCP connections. The interface between applications and transport protocols uses the familiar connect, listen, send, and sendto calls much like the ubiquitous sockets API in Unix environments.

Applications in GTNetS can have one or more associated protocol objects, and can simulate the flow of data (including actual data contents) between applications.

To Simulate Wireless Sensor Networks we need to use GTSNetS [4] that is an extension of GTNetS. GTSNetS has extended GTNetS with wireless sensor network capabilities. It has been developed by George Riley and his students, especially by El-Moustapha Ould Ahmed Vall. Main contributions are the battery and mobility models. When added to the scalability, we consider that GTSNetS is particularly suited for modelling and simulating large scale energy constrained and possibly mobile wireless sensor networks.

Other simulation Tools for WSN

Table B. 1. TOSSIM.

Name:	TOSSIM
Source:	http://www.cs.berkeley.edu/~pal/research/tossim.html
License:	Freeware
Description:	TOSSIM provides a scalable, high fidelity simulation of a complete TinyOS sensor network. The initial design of TOSSIM was focused on this work: it simulates every bit of the mica platform radio stack. As this work was matured, more and more effort has been spent on higher layers, such as complex applications.
Platform:	Windows XP, Windows 2000, Linux
Advantages:	Currently, TOSSIM provides a scalable, high fidelity simulation of a complete TinyOS sensor network
Disadvantages:	Its principal limitation resides in introducing phenomena into the simulation. TOSSIM developers are currently developing a scripting language, as yet unnamed, which will allow users to interact with running and paused simulations. Not support applications that modify low level radio implementations. High fidelity reduces scalability. Bit level simulation degrades performance

Table B. 2. The Network Simulator - ns-2

Name:	The Network Simulator - ns-2
Source:	http://www.isi.edu/nsnam/ns/
License:	Freeware
Description:	Ns-2 is a software package developed for network simulation. It covers a very large number of applications, protocols, network types, network elements, and traffic models.
Platform:	Windows XP(Emulation using Cygwin) , Linux
Advantages:	Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.
Disadvantages:	The original ns-2 is not capable enough of simulating the sensor networks.

Table B. 3. GloMoSim

Name:	GloMoSim http://pcl.cs.ucla.edu/projects/glomosim/
License:	Freeware
Description:	GloMoSim permits building a scalable simulation environment for wireless network systems. It is being designed using the parallel discrete-event simulation capability provided by Parsec.
Platform:	Windows NT, Linux
Advantages:	Most network systems are currently built using a layered approach that is similar to the OSI seven layer network architecture.
Disadvantages:	Not have the standard IEEE 802.15.4

Table B. 4. OMNeT++

Name:	OMNeT++
Source:	http://www.omnetpp.org/
License:	Freeware
Description:	OMNeT++ is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks and because of its generic and flexible architecture; it has been successfully used in other areas like the simulation of IT systems, queuing networks, hardware architectures and business processes as well
Platform:	Windows XP, Linux
Advantages:	One of OMNeT++'s biggest advantages is the ability to write in C++. OMNeT++ is free for academic and non-profit use. Powerful GUI for tracing, debugging and animating your simulations. The documentation tool generates high-quality documentation from commented model source code, with diagrams, tables and cross-references. Integrates well with the Doxygen C++ documentation tool. Wide range of applicability.
Disadvantages:	-

Table B. 5. OPNET

Name:	OPNET
Source:	http://www.opnet.com/
License:	Paid
Description:	OPNET solutions incorporate a high fidelity software model that accurately simulates the behaviour of a real-world network
Platform:	Windows XP
Advantages:	Visualizing TCP/IP mechanisms and variations Designing reliable wireless networks
Disadvantages:	Need to pay a license.

Table B. 6. Avrora

Name:	Avrora
Source:	http://compilers.cs.ucla.edu/avrora/
License:	Freeware
Description:	Avrora, a research project of the UCLA Compilers Group, is a set of simulation and analysis tools for programs written for the AVR microcontroller produced by Atmel and the Mica2 sensor nodes
Platform:	Windows XP, Linux (Java Virtual Machine)
Advantages:	Allow more detailed inspection of the dynamic execution of microcontroller programs and diagnosis of software problems before the software is deployed onto the target hardware. Avrora is a clean and open implementation motivated by this need.
Disadvantages:	Preliminary support for MicaZ has been added in the 1.7.x development branch.

Table B. 7. EmStar

Name:	EmStar
Source:	http://cvs.cens.ucla.edu/emstar/
License:	Freeware
Description:	EmStar is a programming model and software framework for creating Linux-based sensor network applications that are self configuring, reactive to dynamics, and can either be interactively debugged or operate without user interaction.
Platform:	Linux
Advantages:	EmStar incorporates many tools services germane to the creation of WSN.
Disadvantages:	The reliance on FUSD (Framework for User Space Device) introduce several difficulties It has not been easy to export to a broader community, and since the beginning of the project the basis for some of some original assumptions have change

Table B. 8. SENS

Name:	SENS
Source:	http://osl.cs.uiuc.edu/sens/
License:	Freeware
Description:	SENS is a customizable sensor network simulator for WSN applications, consisting of interchangeable and extensible components for applications, network communication, and the physical environment.
Platform:	Linux
Advantages:	SENS allows users to execute the same source code on simulated sensor nodes as deployed on actual sensor nodes, enabling application portability.

Disadvantages:	Absence of information.

Table B. 9. J-SIM

Name:	J-SIM
Source:	http://www.j-sim.org/
License:	Freeware
Description:	*J-Sim* is an open-source, component based compositional network simulation environment that is developed entirely in Java.
Platform:	Linux, Windows
Advantages:	J-Sim is much more scalable than ns-2 J-Sim has been fully integrated with JAVA implementation.
Disadvantages:	Protocol implementation is quite different from a real one No support for new communication paradigms (data or location centric)

Table B. 10. Modelnet

Name:	Modelnet
Source:	http://modelnet.ucsd.edu/
License:	Freeware
Description:	ModelNet is a large-scale network emulator that allows users to evaluate distributed networked systems in realistic Internet-like environments.
Platform:	Linux
Advantages:	-
Disadvantages:	-

Table B. 11. JiST/SWANS

Name:	JiST/SWANS http://jist.ece.cornell.edu/
License:	LICENSEE is hereby granted permission to download, compile, execute, copy, and modify SOFTWARE for non-commercial academic purposes provided that this notice accompanies all copies of SOFTWARE. Copies of modified SOFTWARE may be distributed only for non-commercial academic purposes (a) if this notice accompanies those copies, (b) if said copies carry prominent notices stating that SOFTWARE has been changed, and (c) the date of any changes are clearly identified in SOFTWARE.
Description:	JiST is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. It is a prototype of a new general-purpose approach to building discrete event simulators, called *virtual machine-based simulation*, that unifies the traditional systems and language-based simulator designs. The resulting simulation platform is surprisingly efficient. It out-performs existing highly optimized simulation runtimes both in time and memory consumption.
Platform:	Java compiler
Advantages:	Simulates larger networks faster Includes 3 routing algorithms, including AODV
Disadvantages:	No newer version since 2005 Investigating JiST / SWANS, I notice the following bugs-disadvantages: • broken inheritance (important) • "infinity for" problem (important) • "continuation" problem • "infinity while" problem • broken friendly classes • "new" problem (broken nested classes) • "build path" problem (strange)

Table B. 12. SwarmNet/Shawn

Name:	SwarmNet/Shawn
Source:	http://www.swarmnet.de/
License:	GPL
Description:	Shawn differs in various ways from the above-mentioned simulation tools, while the most notable difference is its focus. It does not compete with these simulators in the area of network stack simulation. Instead, Shawn emerged from an algorithmic background. Its primary design goals are: • Simulate the effect caused by a phenomenon, not the phenomenon itself. • Scalability and support for extremely large networks. • Free choice of the implementation model.
Platform:	Windows (cygwin)/Linux
Advantages:	Capable of simulating large scale networks
Disadvantages:	Different approach: One central approach of Shawn is to simulate the effect caused by a phenomenon, not the phenomenon itself. For example, instead of simulating a complete MAC layer including the radio propagation model, its effects (i.e., packet loss and corruption) are modelled in Shawn.

Table B. 13. SwarmNet/Shawn

Name:	SwarmNet/Shawn
Source:	http://www.swarmnet.de/
License:	Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions
Description:	AlgoSensim is a framework used to simulate distributed algorithms developed at the University of Geneva. It is not protocol stack oriented but **algorithm oriented**. This framework focuses on network specific algorithms like localization, distributed routing, flooding ... AlgoSenSim is easily modulable: It uses XML configuration file. It is efficiency oriented, but optimizations are hidden to the user. AlgoSenSim's main purpose is to facilitate the implementation and quality analysis of new algorithms.
Platform:	Java
Advantages:	Developed to the study of algorithms
Disadvantages:	Still in Alpha Developed to the study of algorithms Tested for static networks only, although it was built to work with mobility

Table B. 14. ModelNet

Name:	ModelNet
Source:	http://modelnet.ucsd.edu/
License:	Free
Description:	ModelNet is a large-scale network emulator that allows users to evaluate distributed networked systems in realistic Internet-like environments. ModelNet enables the testing of unmodified prototypes running over unmodified operating systems across various networking scenarios. In some sense, it combines the repeatability of simulation with the realism of live deployment. The ModelNet user community has deployed it to aid in the design and testing of novel content distribution networks, peer-to-peer systems, transport-layer protocols, content-based switches, distributed stream processors, distributed file systems, and network measurement tools.
Platform:	Linux
Advantages:	
Disadvantages:	Requires a cluster More oriented to IP networks

Table B. 15. NESLSim

Name:	NESLSim
Source:	http://www.ee.ucla.edu/~saurabh/NESLsim/
License:	Free
Description:	NESLsim is a simulation platform based on PARSEC (PARallel Simulation Environment for Complex systems). In NESLsim, a sensor network is modeled as: (i) a collection of sensor nodes, (ii) a channel, and (iii) a supervising entity to create the nodes
Platform:	-

Advantages:	
Disadvantages:	Lack of documentation PARSEC language No fusion point mechanism

Table B. 16. DiSenS

Name:	DiSenS
Source:	http://www.cs.ucsb.edu/~wenye/disens.html
License:	-
Description:	The complete system is composed by a set of full-system simulators, a distributed simulation framework and debugging/profiling facilities. The basis is the set of full-system simulators which provide device simulation with high accuracy (cycle/power). The following table lists the devices we simulate and the features we currently support. Note that all these simulators can boot the original Linux binary image and application binaries can be directly executed.
Platform:	Linux
Advantages:	-
Disadvantages:	No documentation Version for MicaZ (motesim) not available

Table B. 17. COOJA

Name:	COOJA
Source:	http://www.sics.se/~fros/cooja.php
License:	GPL
Description:	A typical simulated node in COOJA is an actual Contiki system running

	and being analyzed by a Java application. This is performed by compiling an entire Contiki system as a shared library, and then loading that library from the Java application. The Java application may then control and analyze the loaded Contiki system through a few functions. For example, the Java part may inform the Contiki system to continue running applications, or fetch the entire current Contiki memory.
Platform:	Contiki
Advantages:	-
Disadvantages:	There are no "stable" builds of COOJA yet Requires the Contiki OS

Table B. 18. JProwler

Name:	JProwler
Source:	http://w3.isis.vanderbilt.edu/Projects/nest/prowler/Index.html
License:	-
Description:	The JProwler tool is a discrete event simulator for prototyping, verifying and analyzing communication protocols of TinyOS ad-hoc wireless networks. The simulator supports pluggable radio models and MAC protocols and multiple application modules. Currently two radio models are implemented: Gaussian and Rayleigh, and one MAC protocol: MICA2 with no acknowledgment. These components have the same underlying dynamic physical model as in the MATLAB Prowler. The simulator is implemented in Java and optimized for raw speed. It can run a simple network-wide broadcast protocol on a 5000-node network in real time (around 1.3 seconds). However, the startup time, during which the simulator creates static data structures, is 35 seconds for a 5000-node network and 1.5 seconds for a 1000-node network. The simulator can visualize the status of the network and application data.
Platform:	Java
Advantages:	Simulates TinyOS

Disadvantages:	Only simulates PHY and MAC layers

Table B. 19. Prowler

Name:	Prowler
Source:	http://w3.isis.vanderbilt.edu/Projects/nest/prowler/Index.html
License:	-
Description:	Networked Embedded Systems (NEST) are large-scale distributed systems with resource limited processing nodes tightly coupled to physical processes via sensors and actuators. Applications running on this distributed platform are strongly affected by the communication channel. Simulators are capable of simulating the behavior of the devices, but usually don't simulate the effects of the communication channels. However, imperfect wireless communication channels greatly affect the performance of the applications, so it is necessary to incorporate in the simulators to get accurate results. Prowler is a probabilistic wireless network simulator capable of simulating wireless distributed systems, from the application to the physical communication layer.
Platform:	Matlab
Advantages:	Simulates TinyOS
Disadvantages:	Only simulates PHY and MAC layers

Table B. 20. Viptos - Visual interface between Ptolemy and TinyOS

Name:	Viptos - Visual interface between Ptolemy and TinyOS
Source:	http://ptolemy.eecs.berkeley.edu/viptos/
Licence:	Freeware
Description:	• Viptos (Visual Ptolemy and TinyOS) is an integrated graphical development and simulation environment for TinyOS-based

	wireless sensor networks. • In particular, Viptos includes the full capabilities of VisualSense, which can model communication channels, networks, and non-TinyOS nodes. • Viptos is compatible with nesC 1.2 and includes tools to harvest existing TinyOS components and applications and convert them into a format that can be displayed as block (and arrow) diagrams and simulated. • TOSSIM is an interrupt-level simulator for TinyOS programs. It runs actual TinyOS code but provides software replacements for the simulated hardware and models network interaction at the bit or packet level. • Ptolemy II is a graphical software system for modelling, simulation, and design of concurrent, real-time, embedded systems. • VisualSense is a Ptolemy II environment for modelling and simulation of wireless sensor networks at the network level. • Viptos provides a bridge between VisualSense and TOSSIM by providing interrupt-level simulation of actual TinyOS programs, with packet-level simulation of the network, while allowing the developer to use other models of computation available in Ptolemy II for modelling various parts of the system. • Viptos supports simulation of heterogeneous networks where each node may run a different program. • Viptos simulations may also include non-TinyOS-based wireless nodes. • Viptos inherits the actor-oriented modelling environment of Ptolemy II, which allows the developer to use different models of computation at each level of simulation. • The user can also model and simulate other aspects of the physical environment including those detected by the sensors (e.g., light, temperature, etc.), terrain, etc.
Platform:	Viptos 1.0.2 works with both Linux and Windows.
Advantages:	• Provides two-tier simulation environments • Can efficiently model large homogeneous networks where the same nesC code is run on every simulated node • Allows networked embedded systems developers to construct block and arrow diagrams to create TinyOS programs from any

	standard library of TinyOS components written in nesC. • Automatically transforms the diagram into a nesC program that can be compiled and downloaded from within the graphical environment onto any TinyOS-supported target platform • Includes the full capabilities of VisualSense, including modeling of communication channels, networks, and non-TinyOS nodes • Allows developers to refine high-level wireless sensor network simulations down to real-code simulation and deployment adds much-needed capabilities to TOSSIM by allowing simulation of heterogeneous networks • Provides a bridge between Ptolemy II and TOSSIM by providing interrupt-level simulation of actual TinyOS programs, with packet-level simulation of the network, while allowing the developer to use other models of computation available in Ptolemy II for modeling the physical environment and other parts of the system.
Disadvantages:	• Some key limitations when using the nesC/TinyOS/TOSSIM programming tool suite such as Users may choose from a few built-in radio connectivity models in TOSSIM, but it is difficult to use other models. • Does not allow simulation of networks that contain different programs.

Table B. 21. Sunflower - Open Hardware Prototypes and Software Platforms for Failure-Prone and Resource-Constrained Embedded Systems

Name: ***Source:***	Sunflower - Open Hardware Prototypes and Software Platforms for Failure-Prone and Resource-Constrained Embedded Systems http://www.sunflowersim.org/
Licence:	Freeware
Description:	• SFLR is a suite of tools, comprising the Sunflower full-system (embedded micro architecture, networking, power, battery and analog signal) simulator, a miniature energy-scavenging hardware sensor platform, and a handheld interface device. The suite is intended to provide a complementary and comprehensive platform for research in micro- and system-architectures for embedded systems, with attention to energy-efficiency, reliability, and

	ecological impact. • The goal of the Sunflower tool suite is to provide an actively evolving ecosystem of both hardware prototypes and simulation / analysis tools, for low-power embedded systems, with an emphasis on the investigation of issues relating to energy-efficiency, energy acquisition, fault-tolerance, and impact of hardware deployments on the environment. The need for research directions investigating the role of transient faults in computing systems, e.g., via the specification of dependability constraints by users of a system and by programmers, is echoed in the recent HiPEAC roadmap.
Platform:	MacOS 10.4.5 (X11, PowerPC) Windows 2000 Linux x86 Console Linux x86
Advantages:	• Enables the evaluation of micro- and system-architectures for networked embedded systems, modelling many aspects of both the hardware platforms and the environments within which they execute. • The Sunflower sensor platform is one physical realization of components modelled within the Sunflower full-system simulator, enabling the calibration and validation of simulator configurations against real hardware implementations. • Additional hardware platforms with complementary hardware capabilities (e.g., wireless communication interfaces and graphical displays) are planned. • A system architecture description file (ADF) defines the components that make up the system, and the interconnections between them. A simple system might define a single processor, a battery and a voltage regulator, in its system architecture description file. • Multiple processors may be instantiated in a given modelled system.
Disadvantages:	-

Table B. 22. WISENES - Wireless Sensor Network Simulator

Name:	WISENES - Wireless Sensor Network Simulator
Source:	http://www.tkt.cs.tut.fi/research/daci/daci_wsn_wisenes.html
Licence:	Freeware
Description:	• WISENES simulates high level WSN protocol and application designs and provides accurate information about their performance in a real environment. Thus, new protocols, protocol configurations, and interoperability between protocols can be evaluated early in the design phase. • WISENES is implemented in Specification and Description Language (SDL). It is widely used in communication protocol design and specification. • In WISENES, the WSN protocol stack consists of data link, network, and middleware layers that are instances of block types implemented in SDL packages. • Allows a modular implementation of protocols. The interfaces between layers a fixed, but a layer can be bypassed, i.e. a network layer can communicate with the application layer at its upper interface. • WISENES Protocol Implementations MAC Protocols at the Data Link Layer • TUTWSN MAC • IEEE 802.15.4 WPAN MAC • S-MAC Routing Protocols at the Network Layer • TUTWSN routing • Flooding routing • ZigBee network layer • Directed diffusion Middleware Layer • TUTWSN task allocation middleware • Basic data passing middleware

Platform:	MacOS 10.4.5 (X11, PowerPC) Windows 2000 Linux x86 Console Linux x86
Advantages:	• The WISENES framework implements models for transmission medium (for modelling wireless communications), sensing channel (for physical phenomena) and nodes (for physical node platforms). • The designer selects the protocols from the library or implements new ones in SDL and integrates them to the WISENES framework. The framework components and node protocols communicate using SDL signals. A node model can be dynamically instantiated separately for each simulated node. • Virtually any number of nodes can be simulated simultaneously.
Disadvantages:	**-**

Table B. 23. SenSor.

Name:	SenSor
Source:	http://www.coventry.ac.uk/researchnet/d/484/a/2498
Licence:	Freeware
Description:	• This project aims to build a configurable software simulator for cogent microsensor networks • SenSor simulator provides a mechanism for co-simulation. • This in effect acts as a bridge between Sensor and our Gumstix hardware, allowing simulation code to be executed on the Gumstix. • The system consisted of a remote execution environment, that allowed sensor simulations to be executed on the constrained Gumstix hardware, the user is able to select which sensors were to be executed remotely the simulator and at runtime the code is transferred to the remote nodes and executed as apart of the simulation. • Until now simulations based in this simulator have been taken to the network discovery, sound location and multi-trilateration sound

	location.
Platform:	Written in Python

Advantages:	• This allows sensor simulations to be executed in three modes:
	  • Pure Simulation
	  • Execution on Hardware
	  • A 'Hybrid' mode allowing a mixture of real and simulated nodes.
	• The same code is used on simulated and physical nodes, so the development process is quick.
	• Sensors have a fixed API, with customizable internals.
	• This enables us to experiment with different algorithms for managing the network topology, fault management and so on, within the same simulation.
	• Individual sensors are be able to:
	  • Gather and process data from a model environment;
	  • Locate and communicate with their (geographically or otherwise) nearest neighbours and
	  • Determine whether they are operating "correctly" and act accordingly to alter the network topology
	• There is a trade-off between data being processed by nodes (which requires processor time) and data being processed by an external computation device (which requires communication time).

Disadvantages:	-

Table B. 24. NetTopo

Name:	NetTopo
Source:	http://www.semanticreality.org/nettopo/index.htm
Licence:	Freeware- NetTopo source code in Java version
Description:	• NetTopo is an open source research-oriented simulator & visualizer designed to test and validates algorithms for wireless sensor networks. • The goal of NetTopo is to build a sensor network simulation and visualization tool that gives users extraordinary flexibility to

	simulate their own algorithms and is a compelling replacement of commercial simulator focusing on visualization of the communication in the real wireless sensor network test bed. • Used to assist the investigation of routing algorithms for WSNs.
Platform:	Windows and Linux
Advantages:	• Customizable sensor network topology layout. • Configurable sensor nodes with user-defined attributes. • Easy extensibility to simulate user-defined algorithms. • Specific step-by-step simulation comparison between GPSR and TPGF. • File-based function facilitating users to save and retrieve their simulating process. • Graphical user interface with local operating system look and feel. • Support for XML import and export. • Platform independent (written in Java). • Integration and virtualization of the real wireless sensor network test bed. • Support for 2D and 3D • The simulation results are formulated into a unified format that allows users to further import them into Microsoft Office Excel to get the graphical results, e.g. curve. [1] • Users can easily define a large number of on demand initial parameters of sensor nodes, e.g. residential energy, transmission bandwidth, and radio radius.[1] • Users also can define and extend the internal processing behavior of sensor nodes, such as energy consumption, bandwidth management. [1] • It allows users to simulate extremely large scale heterogeneous networks. [1] • The visualization module, works as a plug-in component in charge of visualizing WSN testbed's connection states, topology and sensed data.[1] • The sensed data captured from the real sensor nodes can drive our simulation in a predeployed virtual WSN.[1] • Topology layouts and algorithms of virtual WSN are customizable and work as user-defined plug-ins. [1]

Disadvantages:	-

Table B. 25. SENSE

Name:	SENSE
Source:	http://www.ita.cs.rpi.edu/sense/index.html
Licence:	Freeware- source code in C++

Description:	• SENSE is designed to be an efficient and powerful sensor network simulator that is also easy of use. The three most critical factors are:
	  • Extensibility: A component-port model is introduced that frees simulation models from interdependency usually found in an object-oriented architecture, and then proposed a simulation component classification that naturally solves the problem of handling simulated time.
	  • Reusability: The removal of interdependency between models also promotes reusability. A component developed for one simulation can be used in another if it satisfies the latter's requirements on the interface and semantics.
	  • Scalability: Unlike many parallel network simulators, especially SSFNet and Glomosim, parallelization is provided as an option to the users of SENSE. The reflects our belief that completely automated parallelization of sequential discrete event models, however tempting it may seem, is impossible, just as automated parallelization of sequential programs.
	• Currently Available Components and Simulation Engines
	  • Battery Model:
	  • Linear Battery
	  • Discharge Rate Dependent and/or Relaxation Battery
	  • Application Layer:
	  • Random Neighbor
	  • Constant Bit Rate
	  • Network Layer:

	• Simple Flooding • A simplified version of ADOV without route repairing • A simplified version of DSR without route repairing • Self Selective Routing (SSR) • Self Healing Routing (SHR) • MAC Layer: • NullMAC • IEEE 802.11 with DCF • Physical Layer: Duplex Transceiver • Wireless Channel: • Free Space • Adjacency Matrix • Simulation Engine: CostSimEng (sequential) • COST is a library of several classes that facilitates the development of discrete event simulation using CompC++, a component-oriented extension to C++. • COST takes advantage of component-oriented features that are only available in CompC++. [3]
Platform:	Built up of COST (Component- oriented simulation toolkit)[2] Windows and Linux
Advantages:	-
Disadvantages:	• SENSE does not support sensors, physical phenomena, or environmental effects. Overall, the MAC protocol support and radio propagation make SENSE less than ideal for accurate evaluation of wireless sensor network research.[4] • Even if it is possible the parallelization is doomed to be inefficient. Therefore, parallelizable models require more effort than sequential models, but a good portion of users are not interested in parallel simulation at all. In SENSE, a parallel simulation engine can only execute components of compatible components. If a user is content with the default sequential simulation engine, then every component in the model repository can be reused.

Table B. 26. SensorSim

Name:	SensorSim
Source:	http://nesl.ee.ucla.edu/projects/sensorsim/ http://cs.itd.nrl.navy.mil/work/sensorsim/index.php
Licence:	Freeware- open source

Description:	• SensorSim is a simulation framework for modelling sensor networks.
	• It's builds up on the ns-2 simulator and provides additional features for modelling sensor networks.
	• The main features of this platform are:
	  • Sensing channel and sensor models
	  • Battery models
	  • Lightweight protocol stacks for wireless microsensors
	  • Scenario generation
	  • Hybrid simulation
	  • Real application support
	  • Interaction with real nodes
	• Support Mobile Node simulation

Platform:	Linux & Windows (cygwin)- Based up ns-2
Advantages:	• Enables the use of real traffic from the sensor channel that is currently not well understood and the models are not yet mature • Validate protocols and applications running on the real nodes by being able to test these applications in large networks • Study the behaviour of sensor network protocols and applications at scale
Disadvantages:	• Still at pre-release stage. • No documentation. • The software currently has a very specific application hard-coded. • It caters to only one base station.

Table B. 27. Sidh

Name:	Sidh
Source:	http://www.lib.umd.edu/drum/handle/1903/6565
Licence:	The simulator can only be obtained by contacting the developers at this point.
Description:	• Sidh is a simulator specifically designed for wireless sensor networks. • Sidh is composed of a set of modules. • The use of an interface allows modules to be replaced with different implementations. • All other modules interact, via Simulator, through events • The use of events ensures that the timing of interactions is respected. The use of events also ensures that modules are not directly dependent on each other. • The modules currently supported in Sidh can be organized in to the following categories: Simulator; Events; Medium; Environment; Node; Transceiver; Protocols, Applications.
Platform:	Java
Advantages:	• Sidh is efficient. • It scales to simulate networks with thousands of nodes faster than real-time on a typical desktop computer. • Sidh is component based and easily reconfigurable to adapt to different: levels of simulation detail and accuracy; communication media; sensors and actuators; environmental conditions; protocols; and applications.[1]
Disadvantages:	• There are several places in which different modules appear to store the same information. While this information is dynamically created during runtime and does not create independencies, it does use more memory than is needed, limiting the number of nodes that can be simulated and effectively slowing simulation.[2] • The developers of Sidh have not published any results yet, and the simulator is too new to be included in other comparisons.[2] • It seems to be very few protocols implemented.[2]

Table B. 28. SWAN- Simulator for Wireless Ad-Hoc Networks

Name:	SWAN- Simulator for Wireless Ad-Hoc Networks
Source:	http://www.eg.bucknell.edu/swan/
Licence:	Freeware
Description:	• The primary goal of the project is to create a virtual environment for experiments with wireless ad hoc network models. • In SWAN, models of physical process can be defined to represent environmental effects such as the dispersal of a chemical plume, temperature, or barometric pressure. • These effects interact with models of wireless nodes, which can be constructed to "sense" some metric and respond to its variations by communicating with one another. • Implemented in C++, since it is based on DaSSF which is written in C++. • Uses the Domain Modelling Language (DML) for simulator configuration, in the same • Modular protocol graph structure based on the SSFNet design, which in turn, was inspired by the X-kernel design.
Platform:	The system must have a C++ compiler and the Standard Template Library installed. We currently develop on g++ version 3.2.3 and strongly recommend it. The compatibility of DaSSF or SWAN with other C++ compilers is not guaranteed.
Advantages:	• SWAN contains a detailed model of the IEEE 802.11 wireless LAN protocol and a stochastic radio channel model, both of which we used in this study.[2] • Allow protocol designers to run the same code in the simulator as they do in a real system, making it easier to compare experimental and simulation results.[3] • All collected metrics be sub-classed from a statistics base class rather than defined as basic data types such as integer or floating point.[5] • The user also specifies the length of the warm up period for the mobility sub-model.[5]

Disadvantages:	-

Table B. 29. TOSSF- TinyOS Scalable Simulation Framework

Name:	TOSSF- TinyOS Scalable Simulation Framework
Licence:	Freeware
Description:	• The design of the TinyOS Scalable Simulation Framework (TOSSF) was driven to support simulation of Smart Dust project initiated by UC Berkeley [2]. • TOSSF is a WSN extension that simulates native TinyOS code. • TOSSF was built up on two existing projects including DaSSF (Darthmouth Scalable Simulation Framework) and SWAN (Simulator for Wireless Ad-Hoc Networks). • DaSSF provides a streamlined and optimised simulation kernel whilst SWAN offers a range of models for simulating wireless ad hoc networks. TOSSF provides some set of scripts which could adapt the source code for execution in the simulator.[2]
Platform:	The system must have a C++ compiler and the Standard Template Library installed. We currently develop on g++ version 3.2.3 and strongly recommend it. The compatibility of DaSSF or SWAN with other C++ compilers is not guaranteed.
Advantages:	• Provides an environment to simulate TinyOS applications by allowing direct execution at the source code level. [1]
Disadvantages:	• All interrupts are serviced after a task, command or event finishes executing. • Commands and event handlers execute in zero simulation time units. • No pre-emption • Inability to mix different applications in the same simulation run • Only two approaches provide to model radio signals (perfect or totally broken).

	• No performance stability to run large-scale systems.[2]

Table B. 30. GTSNetS

Nome:	GTSNetS
Source:	http://perso.citi.insa-lyon.fr/twatteyn/documents/doxy_gtsnets/index.html
License	**Free**
Description:	GTSNetS (Georgia Tech Sensor Network Simulator) is a large-scale wireless sensor networks simulator. It is best characterized by its scalability, adaptability, and extensibility. It can be used to simulate a WSN scaling up to several hundred thousand nodes. The adaptability comes from the different methods included in the baseline implementations of energy consumption models, reading accuracy models, routing protocols and applications, and tracing options. The extensibility comes from the modular implementations using the C++ object-oriented programming language.
Platform:	Linux (apenas GTNetS em Windows)
Advantages:	Developed for WSN Simulates also control system This simulator allows the user to choose among different implemented alternatives: different network protocols, different types of applications, different sensors, and different energy and accuracy models. New models, if needed, can be easily added. Can be used to collect detailed statistics about a specific sensor network at the functional unit level, the node level as well as at the network level.
Disadvantages:	High-level functional simulator models the process of radio communication in terms of the interactions of events. As a result, they achieve excellent simulation performance but normally do not provide accurate timing information which is critical for debugging and power

	optimization. Only one paper indexed at IEEE

Conclusions

After analyzing the three simulation languages individually we conclude that the simulator that is more suitable to our simulation needs for Wireless Sensor Networks is OMNeT++, because it is a public-source, component-based, modular and open-architecture simulation with a strong graphical user interface support and a embeddable simulation kernel.

Other advantage of using OMNeT++ (compared with the other two simulators) is that OMNeT++ is rapidly becoming popular in the scientific community as well an in the industrial community, and several models have been published. Hence the fact that he has a strong community and a forum where we can exchange information with other people working with Wireless Sensor Networks can be useful in the future for debugging and solve related problems in the simulation of Wireless Sensor Networks. Besides, by comparing not only OMNeT++ not only with the other two simulators but also it with another well known simulator such as ns-2, we conclude that our simulator is at least an order of magnitude faster than ns-2 and uses memory more efficiently than ns-2.

OMNeT++ has the advantage to support two kinds of simulation modes, the event-based and the process-oriented ones, while GTNetS only support discrete event processing.

NetTopo is developed using the JavaTM platform and OMNeT++ uses the C++ platform so the NetTopo does not always provide full access to the features and performance of the platform that the software runs on, C++ is more powerful than JavaTM, and C++ often outperforms Java in arithmetic and trigonometric operations.

Finally, we conclude that OMNeT++ is fully programmable and modular, and it was designed to support modeling for large networks build from reusable model components. All these features make OMNeT++ a good candidate for both simulation and research purposes.

Annex 3

Source Code

The Annex 3 presents the source code of our WSN simulator that was developed in OMNeT++.

SBACK NED FILE

```
simple SBACK
{
    int limit = default(5);   // another parameter with a default value
    volatile double delayTime @unit(s);   // delay before sending back
message
    @display("i=block/routing");
    gates:
        inout gate[];


}



network SBACK1_Network
{
    @display("bgb=511,409;bgl=12");
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 0ms;//100ms
        }

    submodules:
        node[5]: SBACK {
            @display("is=l;p=,,ri");

        }



    connections:
        node[1].gate++ <--> Channel <--> node[0].gate++;
        node[2].gate++ <--> Channel <--> node[0].gate++;
        node[0].gate++ <--> Channel <--> node[3].gate++;
        node[0].gate++ <--> Channel <--> node[4].gate++;

}
```

SBACK msg FILE

```
message SBACK1Msg
{
    int source;
    int destination;
    int hopCount = 0;

}
```

SBACK SYNCH Frame FILE

```
message SBACK_synch_frame {

        int type;
        int length;
        int sender_NodeID;
        int receiver_NodeID;
        int receiver_GateID;
        int syncNode;
        double next_sleepTime;  // my next sleep time from now */
        double duration;
        int crc;


}
```

SBACK DATA Frame FILE

```
message SBACK_data_frame
{
        int type;
        int length;
        int sender_NodeID;
        int receiver_NodeID;
        int message_Number;
        int fragment_Number;
        double duration;
        int crc;

}
```

SBACK Control Frame FILE

```
message SBACK_control_frame
{
        int type;
        int length;
        int sender_NodeID;
        int receiver_NodeID;
        int syncNode;
        double duration;
        int crc;

}
```

SBACK .h FILE

```
#ifndef SBACK_H_
#define SBACK_H_

#include "SBACK_m.h"
#include "SBACK_synch_frame_m.h"
#include "SBACK_control_frame_m.h"
```

```
#include "SBACK_data_frame_m.h"
#include <string.h>
#include <omnetpp.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <iostream>
#include <cmath>


//Total number of nodes
#define total_number_nodes 5

//Total number of messages send by the source nodes (in practice only 100
will be sent)
#define total_number_messages 101


/* Internal SBACK parameters
 *---------------------------
  * DIFS: DCF interframe space (from 802.11), in s. It is used at the
beginning
 *   of each contention window. It's the minimum time to wait to start a
new
 *   transmission.
 * SIFS: short interframe space (from 802.11), in s. It is used before
sending
 *   an CTS or ACK packet. It takes care of the processing delay of each
pkt.
 * SYNC_CW: number of slots in the sync contention window, must be 2^n - 1
 * DATA_CW: number of slots in the data contention window, must be 2^n - 1
 * SYNC_PERIOD: period to send a sync pkt, in s.
 * UPDATE_NEIGHB_PERIOD: period to update neighbor list, is n times of
 *    SYNC_PERIOD. It is used in low duty cycle mode. If there is no SYNC
pkts
 *    from a node within this period, it will be removed from neighbor
list.
 *
 * DATA_ACTIVE_PERIOD (s): This is only used in fully active mode to update
 *    neighbor list. Since there is no SYNC pkts, data pkts are used to
measure
 *    if a neighbor is active recently.
 * TX_PKT_DONE_TIME: max time to wait for Tx done signal from PHY, in s.
 */
#define DIFS 0.01
#define SIFS 0.005
//#define EIFS 0.05
//#define SLOTTIME 0.001

#define Frame_Time 2.3  //sleep/active period = 2.2
#define SYNC_CW 0.015
#define RTS_CW 0.021     //#define DATA_CW 0.031
#define CTS_CW 0.010
#define MAX_TX_SYNC_TIME 0.003
#define MAX_TX_RTS_TIME 0.003
#define MAX_TX_CTS_TIME 0.003
```

```
#define SYNC_PERIOD 12
#define UPDATE_NEIGHB_PERIOD 0.012



// STATE MACHINE
#define START 0      //START Nodes
#define WAIT_SYNCH 1 //WAIT SYNCH Transmissions
#define SYNCH       2  //SYNCH Packet
#define BACKOFF_SELECT 3
#define BACKOFF 4
#define IDLE 5
#define TRANSMIT 6
#define WAIT_RESPONSE 7
#define TYPE_ACK 8
#define VERIFY_FRAME_FINISH 9
#define NAV_SlEEP 10
#define SLEEP 11


//Structures of data

struct Network_Information
{
      double rssi[total_number_nodes];
      double next_sleep_time[total_number_nodes];
      int number_synch_sent;
      int number_synch_received; //Total number_synch_received

};


//Structure with the list of all neighbor nodes
struct Neighbour_List
{
      int nodeID[total_number_nodes];
      int neighbour_nodes[total_number_nodes];
      double message_arrival_time[total_number_nodes];

};



double randdouble(double min, double max);

// initialize an object of type C with an initializer-list
Network_Information network_info[total_number_nodes] = {  };
Neighbour_List neighbour_list[total_number_nodes]={  };



class SBACK  : public cSimpleModule{

private:
```

```
      simtime_t time_between_DATAMessages,time_between_ACKMessages;//Time
between different DATA/ACK Messages
      cMessage *timeoutEvent_SYNCMessages;// holds a pointer to the initial
timeout self-message
      int frame_time;//Frame time

      double backoff_time;
      double NAV_vector[total_number_nodes];


      double
aux_next_sleep_time[total_number_nodes][total_number_nodes];//Matrix with
number of lines == number of nodes, and columns==next_sleep_time of each
node
      double synch_carrier_sense_vector[total_number_nodes];//Information
about synch_carrier_sense of every node
      double rts_carrier_sense_vector[total_number_nodes];//Information
about rts_carrier_sense of every node
      double cts_carrier_sense_vector[total_number_nodes];//Information
about cts_carrier_sense of every node
      int Number_Synch_Sent[total_number_nodes]; //Total Number_Synch_Sent
by every node
      int timeout_Event_ACK_DATA_Messages_count; //Total Number of ACK and
DATA Sent by every node
      int number_timeout_SYNCH_sent[total_number_nodes];//Total number of
timeout_SYNCH Messages sent
      int number_timeout_RTS_sent[total_number_nodes];//Total number of
timeout_RTS Messages sent
      int number_timeout_CTS_sent[total_number_nodes];//Total number of
timeout_CTS Messages sent
      int type_ack;//if 1 use ACK else use Block ACK




      int timeout_Event_RTS_ADDBA_Request_count;
      int send_RTS_BA_Request_Messages_count;
      int BACK_Send_Data_count;//Number of DATA Messages Sent

      int sent_synch;
      int sent_rts;

      int FSM_SYNCH_Global_Variable;
      int FSM_RTS_Global_Variable;
      int numretriesSYNCH;

      int listen_time;
      int sleep_time;
      int synch_contend_time;
      int rssii;
      int number_nodes;



public:
      SBACK();// constructor takes no arguments
      virtual ~SBACK();
```

```
protected:
	//The Following redefined virtual functions holds the algorithm
	virtual void initialize();
	virtual void handleMessage(cMessage *msg);
	virtual void timeout_Event_SYNCMessages(cMessage *msg);
	virtual void send_SYNCMessages(cMessage *msg);
	virtual void timeout_Event_RTSMessages(cMessage *msg);
	virtual void send_RTSMessages(cMessage *msg);
	virtual void timeout_Event_CTSMessages(cMessage *msg);
	virtual void send_CTSMessages(cMessage *msg);
	virtual void timeout_Event_DATAMessages(cMessage *msg);
	virtual void send_DATA_Messages(cMessage *msg);
	virtual void passing_DATA_Messages(cMessage *msg);
	virtual void timeout_Event_ACKMessages(cMessage *msg);
	virtual void send_ACK_Messages(cMessage *msg);
	virtual void timeout_Event_RTS_ADDBA_Request_Messages(cMessage *msg);
	virtual void send_RTS_ADDBA_Request_Messages(cMessage *msg);
	virtual void timeout_Event_CTS_ADDBA_Response_Messages(cMessage
*msg);
	virtual void send_CTS_ADDBA_Response_Messages(cMessage *msg);
	virtual void timeout_Event_RTS_BA_Request_Messages(cMessage *msg);
	virtual void send_RTS_BA_Request_Messages(cMessage *msg);
	virtual void send_CTS_BA_Response_Messages(cMessage *msg);
	virtual void send_RTS_DELBA_Request_Messages(cMessage *msg);
	virtual void send_CTS_DELBA_Response_Messages(cMessage *msg);
	virtual void bubble_sort_next_sleep_time(cMessage *msg);
	virtual void Backoff_Select(cMessage *msg);
	virtual void print_network_info(void);

};

SBACK::SBACK() {

	time_between_DATAMessages=0;
	time_between_ACKMessages=0;

	backoff_time=1;


	timeoutEvent_SYNCMessages=NULL;

	frame_time= Frame_Time;
	listen_time=  DIFS+SYNC_CW+DIFS+RTS_CW+SIFS+CTS_CW+SIFS;

	sleep_time = frame_time-listen_time;
	sent_synch=0;
	type_ack=2;//if 1 use ACK else use Block ACK
	timeout_Event_ACK_DATA_Messages_count=0;
	timeout_Event_RTS_ADDBA_Request_count=0;
	send_RTS_BA_Request_Messages_count=0;
	BACK_Send_Data_count=0;

}

SBACK::~SBACK() {
	cancelAndDelete(timeoutEvent_SYNCMessages);
```

```
}
#endif /* SBACK_H_ */
```

SBACK .CC FILE

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published
by
// Norberto Barroca MCs student of IT-CovilhÃ
// (at your option) any later version.

#include "SBACK.h"

// The module class needs to be registered with OMNeT++
Define_Module(SBACK);


//##########################################################################
############################### SBACK::initialize() //
//Initialize is called at the beginning of the simulation
void SBACK::initialize()
{
      // Initialize variables.
      int i;



      for (i=0;i<total_number_nodes;i++)
            Number_Synch_Sent[i] = 0;

      for (i=0;i<total_number_nodes;i++)
            network_info[i].number_synch_received=0;



      //Creation of a self-message to the in order to send all the
SYNCMessages at the same time
      timeoutEvent_SYNCMessages = new
cMessage("timeoutEvent_SYNCMessages");
      scheduleAt(simTime()+DIFS, timeoutEvent_SYNCMessages);

          double x_synch,x_rts,x_cts;

      x_synch=randdouble(SIMTIME_DBL(simTime()+DIFS),SIMTIME_DBL(simTime()+
DIFS+SYNC_CW-MAX_TX_SYNC_TIME));


      x_rts=randdouble(SIMTIME_DBL(simTime()+DIFS+SYNC_CW+DIFS),SIMTIME_DBL
(simTime()+DIFS+SYNC_CW+DIFS+RTS_CW-MAX_TX_RTS_TIME));


      x_cts=randdouble(SIMTIME_DBL(simTime()+DIFS+SYNC_CW+DIFS+RTS_CW+SIFS)
,SIMTIME_DBL(simTime()+DIFS+SYNC_CW+DIFS+RTS_CW+SIFS+CTS_CW-
MAX_TX_CTS_TIME));
```

```
            if(getIndex()==0)
            {
                  synch_carrier_sense_vector[getIndex()]=x_synch;
                  EV<<"Get_Index: "<<getIndex()<< "
synch_carrier_sense_vector:
"<<synch_carrier_sense_vector[getIndex()]<<endl;

                  EV<<endl;
                  rts_carrier_sense_vector[getIndex()]=x_rts;
                  EV<<"Get_Index: "<<getIndex()<< "
rts_carrier_sense_vector: "<<rts_carrier_sense_vector[getIndex()]<<endl;

                  EV<<endl;
                  cts_carrier_sense_vector[getIndex()]=x_cts;
                  EV<<"Get_Index: "<<getIndex()<< "
cts_carrier_sense_vector:
"<<cts_carrier_sense_vector[getIndex()]<<endl<<endl;

            }
            else
            {
                  synch_carrier_sense_vector[getIndex()]=x_synch;
                  EV<<"Get_Index: "<<getIndex()<<"
synch_carrier_sense_vector:
"<<synch_carrier_sense_vector[getIndex()]<<endl;

                  EV<<endl;
                  rts_carrier_sense_vector[getIndex()]=x_rts;
                  EV<<"Get_Index: "<<getIndex()<< "
rts_carrier_sense_vector: "<<rts_carrier_sense_vector[getIndex()]<<endl;

                  EV<<endl;
                  cts_carrier_sense_vector[getIndex()]=x_cts;
                  EV<<"Get_Index: "<<getIndex()<< "
cts_carrier_sense_vector:
"<<cts_carrier_sense_vector[getIndex()]<<endl<<endl;


            }

}

//#########################################################################
############### SBACK::handleMessage(cMessage *msg) //
//The handleMessage() method is called whenever a message arrives at the
module
void SBACK::handleMessage(cMessage *msg)
{
      EV<<"\n##############  SBACK::handleMessage - BEGIN
####################\n\n"<<endl;

      print_network_info();
      EV<<"SWITCH msg: "<<msg<<"  KIND: "<<msg->getKind()<<endl<<endl;
      EV<<"type ACK: "<<type_ack<<endl<<endl;
      EV<<"simTime: "<<simTime()<<endl<<endl;
```

```
	SBACK_synch_frame *new_msg=static_cast<SBACK_synch_frame*>(msg);
	assert(msg);

	switch (msg->getKind())
	{
	case START://setKind(0)

	case WAIT_SYNCH://setKind(1)

		if(type_ack==1 ||type_ack==2)
		{
			//Schedule Event SYNCMessages
			timeout_Event_SYNCMessages(msg);
		}
		break;

	case SYNCH://setKind(2)

		//###ACK - BEGIN
		if(type_ack==1)
		{

			if(msg->getKind()==2)//If message == timeout_SYNCH
			{
				//Schedule Event RTSMessages
				timeout_Event_RTSMessages(msg);
				goto goto_BACKOFF_SELECT;
			}
		}

		//###BACK - BEGIN
		else
		{

			if(msg->getKind()==2)//If message == timeout_SYNCH
			{
				//Schedule Event RTS_ADDBA_Request_Messages
				timeout_Event_RTS_ADDBA_Request_Messages(msg);
				goto goto_BACKOFF_SELECT;//Go to state
BACKOFF_SELECT
			}
		}

		goto_SYNCH:
		rssii=uniform(80, 100);

		//See if the SYNCH Messages could be sent
		if(rssii<120)
		{
			if (strcmp(msg->getClassName(), "SBACK_synch_frame") ==
0)
				delete msg;
		}
		else
		{
			if (strcmp(msg->getClassName(), "SBACK_synch_frame") ==
0)
```

```
                {

                        send_SYNCMessages(msg);
                        sent_synch++;
                }
        }

        break;

    case BACKOFF_SELECT://setKind(3)
        goto_BACKOFF_SELECT:

        goto goto_BACKOFF;

    break;
    case BACKOFF://setKind(4)
        goto_BACKOFF:
        goto goto_SYNCH;

    break;

    case IDLE://setKind(5)

        if(msg->getKind()==5) //If message == timeout_RTS || message ==
timeout_RTS_ADDBA_Request
        {
            goto goto_TRANSMIT; //Go to state TRANSMIT
        }
        break;

    case TRANSMIT: //setKind(6)
        goto_TRANSMIT:

        //###ACK - BEGIN
        if(type_ack==1)
        {
            rssii=uniform(70, 180);

            if (msg->getKind() == 5 && new_msg->getType()==2) //If
message == timeout_RTS
            {
                //Schedule Event CTSMessages
                timeout_Event_CTSMessages(msg);

                //See if the RTS Messages could be sent
                if(rssii<100)
                {

                }
                else
                {
                    if (strcmp(msg->getClassName(),
"SBACK_control_frame") == 0)
                    {
                        if(getIndex()!=0)
                            send_RTSMessages(msg);
                    }
                    sent_rts++;
```

```
                        }
                  }

                  goto_TRANSMIT_DATA_ACK:

                  //See if the DATA Messages could be sent
                  if(new_msg->getType()==6)//If message == timeout_DATA
                  {
                        send_DATA_Messages(msg);
                  }

                  //See if the ACK Messages could be sent
                  if(new_msg->getType()==9)//If message == timeout_ACK
                  {
                        send_ACK_Messages(msg);
                  }
            }

            //###BACK - BEGIN
            else
            {
                  if (new_msg->getType()==11 && msg->getKind() == 5)//If
message ==timeout_RTS_ADDBA_Request
                  {
                        //Schedule Event CTS_ADDBA_Response
                        timeout_Event_CTS_ADDBA_Response_Messages(msg);

                        //See if the RTS_ADDBA_Request Messages could be
sent
                        if(rssii<100)
                        {

                        }
                        else
                        {
                              if (strcmp(msg->getClassName(),
"SBACK_control_frame") == 0)
                              {
                                    EV<<"Antes de RTS_ADDBA msg:
"<<msg<<endl;

                                    if(getIndex()!=0)

      send_RTS_ADDBA_Request_Messages(msg);
                              }
                              sent_rts++;
                        }
                  }

                  goto_TRANSMIT_DATA_BACK:

                  //See if the DATA Messages could be sent
                  //Schedule Event RTS_BA_Request
                  if(new_msg->getType()==6)//If message == timeout_DATA
                  {
                        if(getIndex()==1)
                        {
                              send_DATA_Messages(msg);
```

```
                              BACK_Send_Data_count++;

                              if(BACK_Send_Data_count==100)
                              {

      timeout_Event_RTS_BA_Request_Messages(msg);
                              }

                        }
                  }

            }
      break;

      case WAIT_RESPONSE://setKind(7)
            //###ACK - BEGIN
            if(type_ack==1)
            {
                  if(msg->getKind()==7)//If message == timeout_CTS
                  {
                        send_CTSMessages(msg);

                        if(new_msg->getType()==4) //If message ==
timeout_CTS
                        {
                              if(timeout_Event_ACK_DATA_Messages_count==0)
//Send only one ACK and one DATA Message per node
                              {
                                    timeout_Event_DATAMessages(msg);
                                    timeout_Event_ACKMessages(msg);

      timeout_Event_ACK_DATA_Messages_count++;
                              }
                        }
                  }

                  if(new_msg->getType()==6 || new_msg->getType()==9)//If
message == timeout_DATAMessages || message ==timeout_Event_ACKMessages
                  {
                        if(type_ack==1)
                              goto goto_TRANSMIT_DATA_ACK;
                  }
            }

            //###BACK - BEGIN
            else
            {
                  if(msg->getKind()==7)//If message ==
timeout_CTS_ADDBA_Response
                  {
                        send_CTS_ADDBA_Response_Messages(msg);

                        if(timeout_Event_RTS_ADDBA_Request_count==0)
                        {
                              timeout_Event_DATAMessages(msg);
                              timeout_Event_RTS_ADDBA_Request_count++;
                        }
                  }
```

```
                  if(new_msg->getType()==6)//If message ==
timeout_Event_DATAMessages
                  {
                        goto goto_TRANSMIT_DATA_BACK;
                  }
            }

            break;

      case TYPE_ACK://setKind(8)
            //###ACK - BEGIN
            if(type_ack==1)
            {
                  if(new_msg->getType()==9)//If message ==
timeout_Event_ACKMessages
                  {
                        send_ACK_Messages(msg);
                  }

                  if(new_msg->getType()==7)//If message == DATA_Messages
                  {
                        passing_DATA_Messages(msg);
                  }

            }

            //###BACK - BEGIN
            if(type_ack==2)
            {
                  if(new_msg->getType()==7)//If message == DATA_Messages
                  {
                        passing_DATA_Messages(msg);
                  }
            }
            break;

      case VERIFY_FRAME_FINISH://setKind(9)

            if(new_msg->getType()==15)//If message ==
timeout_RTS_BA_Request
            {
                  send_RTS_BA_Request_Messages(msg);
            }

            if(new_msg->getType()==16)//If message == RTS_BA_Request
            {
                  send_CTS_BA_Response_Messages(msg);
            }

            if(new_msg->getType()==17)//If message ==  CTS_BA_Response
            {
                  send_RTS_DELBA_Request_Messages(msg);
            }

            if(new_msg->getType()==18)//If message ==  RTS_DELBA_Request
            {
                  send_CTS_DELBA_Response_Messages(msg);
```

```
            }


            break;


      case NAV_SlEEP://setKind(10)
            EV<<"WAIT until NAV expires"<<endl;
            break;
      case SLEEP://setKind(11)
            EV<<"WAIT until SLEEP expires"<<endl;
            break;



      default:

            EV<<"Fim de switch Defaul msg: "<<msg<<endl;

            break;
      }

      EV<<"\n###############  SBACK::handleMessage - END
####################\n\n"<<endl;

}

//#####################################################################
SBACK::timeout_Event_SYNCMessages(SBACK_synch_frame *msg)
void SBACK::timeout_Event_SYNCMessages(cMessage *msg)
{
      if(msg->isSelfMessage())
      {
            // Initialize variables
            int i;
            char msgname[20];// define size of message

            if(getIndex()==0)//If node == 0
            {
                  //The backoff_time to transmit a SYNCH must be between
the end of the carrier sense and the maximum value of the Synch TX window

      backoff_time=synch_carrier_sense_vector[getIndex()]+MAX_TX_SYNC_TIME;
//backoff_time=SIMTIME_DBL(simTime())+synch_carrier_sense_vector[getIndex()
]+SYNCH_TX_TIME;

                  int n=gateSize("gate");
                  for (i = 0; i < n; i++)
                  {

                        //###### timeout_SYNCH FRAME - CREATION ######
                        sprintf(msgname,"timeout_SYNCH %d to
%d",getIndex(),i+1);//+1 because index of node==0
                        SBACK_synch_frame *msg = new SBACK_synch_frame
(msgname);
                        msg->setSender_NodeID(getIndex());   //Set
Sender_NodeID
```

```
                        msg->setReceiver_NodeID(i+1);        //Set
Receiver_NodeID
                        msg->setReceiver_GateID(i);          //Set
Receiver_GateID
                        msg->setNext_sleepTime(sleep_time);  //Set
Next_sleepTime
                        msg->setType(0);                             //Set
Type (timeout_SYNCH)
                        msg->setKind(2);                             //Set
Next FSM (SYNCH)
                        //#####################################

                        //###### NETWORK INFORMATION ######
                        network_info[getIndex()].rssi[i]=rssii; //Save the
last information about RSSI

      network_info[getIndex()].next_sleep_time[i]=sleep_time; //Save the
last information about sleep_time
                        //#####################################

                        //###### NEIGHBOUR LIST ######

      neighbour_list[getIndex()].nodeID[getIndex()]=getIndex();

      neighbour_list[getIndex()].message_arrival_time[i]=SIMTIME_DBL(msg-
>getArrivalTime());
                        //############################



      number_timeout_SYNCH_sent[getIndex()]=number_timeout_SYNCH_sent[getIn
dex()]+1;

                        scheduleAt(backoff_time, msg);//timer to send the
SYNCH Message
                  }
            }

            else //If node != 0
            {
                  //The backoff_time to transmit a synch must be between
the end of the carrier sense and the maximum value of the Synch TX window

      backoff_time=synch_carrier_sense_vector[getIndex()]+MAX_TX_SYNC_TIME;
//backoff_time=SIMTIME_DBL(simTime())+synch_carrier_sense_vector[getIndex()
]+SYNCH_TX_TIME;
                  int n=gateSize("gate");

                  for (i = 0; i < n; ++i)
                  {
                        //###### timeout_SYNCH - CREATION ######
                        sprintf(msgname,"timeout_SYNCH %d to
%d",getIndex(),i);//because index of node==0
                        SBACK_synch_frame *msg = new SBACK_synch_frame
(msgname);
                        msg->setSender_NodeID(getIndex());   //Set
Sender_NodeID
```

```
                        msg->setReceiver_NodeID(i);          //Set
Receiver_NodeID
                        msg->setReceiver_GateID(i);          //Set
Receiver_GateID
                        msg->setNext_sleepTime(100+getIndex());
//Set Next_sleepTime
                        msg->setType(0);                              //Set
Type (timeout_SYNCH)
                        msg->setKind(2);
//Set Next FSM (SYNCH)
                        //#####################################


      number_timeout_SYNCH_sent[getIndex()]=number_timeout_SYNCH_sent[getIn
dex()]+1;

                              //###### NETWORK INFORMATION ######
                              network_info[getIndex()].rssi[i]=rssii;
//Save the last information about RSSI

      network_info[getIndex()].next_sleep_time[i]=100+getIndex();
                              //###################################

                              //###### NEIGHBOUR LIST ######

      neighbour_list[getIndex()].nodeID[getIndex()]=getIndex();

      neighbour_list[getIndex()].message_arrival_time[i]=SIMTIME_DBL(msg-
>getArrivalTime());
                              //#############################

                              scheduleAt(backoff_time, msg);//timer to send
the SYNCH Message
                  }
            }

      }
}

//######################################################################
SBACK::send_SYNCMessages(cMessage *msg)
void SBACK::send_SYNCMessages(cMessage *msg)
{
      SBACK_synch_frame *new_msg=static_cast<SBACK_synch_frame*>(msg);

      // Initialize variables
      int i;
      char msgname[20];// define size of message

      //The next_sleep_time of Sender_NodeID() and Receiver_NodeID() will
be the same
      if(new_msg->getKind()==2)//If message == timeout_SYNCH
      {
            if(network_info[new_msg-
>getReceiver_NodeID()].number_synch_received==0)
            {
```

```
                  network_info[new_msg-
>getReceiver_NodeID()].next_sleep_time[0]=(network_info[new_msg-
>getSender_NodeID()].next_sleep_time[0]);
            }
            if(network_info[new_msg-
>getReceiver_NodeID()].number_synch_received==1)
            {
                  network_info[new_msg-
>getReceiver_NodeID()].next_sleep_time[1]=(network_info[new_msg-
>getSender_NodeID()].next_sleep_time[0]);
                  //If a node receives a different schedule after selects
and broadcasts its own schedule
                  // it adopts both schedules and broadcasts it own
schedule before go to sleep.
                  for(i=1;i<total_number_nodes+1;i++)

      network_info[i].next_sleep_time[1]=(network_info[0].next_sleep_time[1
]);
            }
            //Network Information about the number of SYNCH's received
            network_info[new_msg-
>getReceiver_NodeID()].number_synch_received=network_info[new_msg-
>getReceiver_NodeID()].number_synch_received+1;
      }


      if(new_msg->isSelfMessage())
      {

            if(new_msg->getReceiver_NodeID()==0) //If Receiver_NodeID()==0
            {
                  //Number_Synch_Sent by the Sender_NodeID()
                  Number_Synch_Sent[new_msg-
>getSender_NodeID()]=Number_Synch_Sent[new_msg->getSender_NodeID()]+1;

                  //network_info[new_msg-
>getSender_NodeID()].number_synch_sent   have the Number_Synch_Sent send by
the node
                  network_info[new_msg-
>getSender_NodeID()].number_synch_sent=Number_Synch_Sent[new_msg-
>getSender_NodeID()];

                  //###### SYNCH FRAME - CREATION ######
                  sprintf(msgname,"SYNCH %d to %d" ,new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());
                  new_msg->setType(1);                              //Set
Type (SYNCH)
                  new_msg->setKind(3);                              //Set
Next FSM (BACKOFF_SELECT)
                  new_msg->setName(msgname);                //Set Message
Name
                  new_msg->setArrivalTime(new_msg->getSendingTime());
//ArrivalTime
                  //##############################

                  sendDelayed(new_msg,0,"gate$o",new_msg-
>getReceiver_GateID()); //Send message based on the Receiver_GateID()
```

```
            }
            else//If Receiver_NodeID()!=0
            {
                  //Number_Synch_Sent by the Sender_NodeID()
                  Number_Synch_Sent[new_msg-
>getSender_NodeID()]=Number_Synch_Sent[new_msg->getSender_NodeID()]+1;

                  //network_info[new_msg-
>getSender_NodeID()].number_synch_sent   have the Number_Synch_Sent send by
the node
                  network_info[new_msg-
>getSender_NodeID()].number_synch_sent=Number_Synch_Sent[new_msg-
>getSender_NodeID()];

                  //###### SYNCH FRAME - CREATION ######
                  sprintf(msgname,"SYNCH %d to %d" ,new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());
                  new_msg->setType(1);                              //Set
Type (SYNCH)
                  new_msg->setKind(3);                              //Set
Next FSM (BACKOFF_SELECT)
                  new_msg->setName(msgname);                  //Set Message
Name
                  new_msg->setArrivalTime(new_msg->getSendingTime());
//ArrivalTime
                  //##############################

                  sendDelayed(new_msg,0,"gate$o",new_msg-
>getReceiver_GateID()); //Send message based on the Receiver_GateID()

            }
      }

}

//#########################################################################
SBACK::timeout_Event_RTSMessages(cMessage *msg)
void SBACK::timeout_Event_RTSMessages(cMessage *msg)
{
      SBACK_control_frame *new_msg=static_cast<SBACK_control_frame*>(msg);

      // Initialize variables
      char msgname[20];// define size of message

      if(getIndex()==0)//If node == 0
      {
            //The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

      backoff_time=rts_carrier_sense_vector[getIndex()]+MAX_TX_RTS_TIME;

            //###### timeout_RTS FRAME - CREATION ######
            sprintf(msgname,"timeout_RTS %d to %d",new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());//+1 because index of
node==0
            SBACK_control_frame *msg = new SBACK_control_frame (msgname);
            msg->setSender_NodeID(getIndex());
//Set Sender_NodeID
```

```
            msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            msg->setType(2);
//Set Type (timeout_RTS)
            msg->setKind(5);
//Set Next FSM (IDLE)
            //#########################################


      number_timeout_RTS_sent[getIndex()]=number_timeout_RTS_sent[getIndex(
)]+1;

            scheduleAt(backoff_time, msg);//Corrigir isto com o backoff
time

      }
      else //If node != 0
      {

            //The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

      backoff_time=rts_carrier_sense_vector[getIndex()]+MAX_TX_RTS_TIME;

            //###### timeout_RTS FRAME - CREATION ######
            sprintf(msgname,"timeout_RTS %d to %d",new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());//because index of
node==0
            SBACK_control_frame *msg = new SBACK_control_frame (msgname);
            msg->setSender_NodeID(getIndex());;
//Set Sender_NodeID
            msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            msg->setType(2);
//Set Type (timeout_RTS)
            msg->setKind(5);
                        //Set Next FSM (IDLE)
            //#########################################


      number_timeout_RTS_sent[getIndex()]=number_timeout_RTS_sent[getIndex(
)]+1;

            scheduleAt(backoff_time, msg); //Corrigir isto com o backoff
time

      }

}
//#####################################################################
SBACK::send_RTSMessages(cMessage *msg)
void SBACK::send_RTSMessages(cMessage *msg)
{
      SBACK_synch_frame *new_msg=static_cast<SBACK_synch_frame*>(msg);

      // Initialize variables
      char msgname[20];// define size of message
```

```
	if(new_msg->isSelfMessage())
	{
		if(new_msg->getReceiver_NodeID()==0)  //If Receiver_NodeID()==0
		{
			//###### RTS FRAME - CREATION ######
			sprintf(msgname,"RTS %d to %d" ,new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());
			new_msg->setType(3);                                //Set
Type (RTS)
			new_msg->setKind(20);                               //Set
Next FSM
			new_msg->setName(msgname);                          //Set
Message Name
			//#################################

			sendDelayed(new_msg,0,"gate$o",new_msg-
>getReceiver_GateID()); //getReceiver_DateID
		}
		else//If Receiver_NodeID()!=0
		{
			//###### RTS FRAME - CREATION ######
			sprintf(msgname,"RTS %d to %d" ,new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());
			new_msg->setName(msgname);
			new_msg->setType(3);    //este campo indica que se trata
de uma mensagem rts
			new_msg->setKind(20);   //este campo indica que se trata
de uma mensagem rts
			//#################################

			sendDelayed(new_msg,0,"gate$o",new_msg-
>getReceiver_GateID()); //getReceiver_DateID
		}
	}
}

//#####################################################################
SBACK::timeout_Event_CTSMessages(cMessage *msg)
void SBACK::timeout_Event_CTSMessages(cMessage *msg)
{

	SBACK_synch_frame *new_msg=static_cast<SBACK_synch_frame*>(msg);

	// Initialize variables
	char msgname[20];// define size of message

	if(getIndex()==0 && new_msg->isSelfMessage()) //If node == 0
	{
		//The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

	backoff_time=cts_carrier_sense_vector[getIndex()]+MAX_TX_CTS_TIME;

		//###### timeout_CTS FRAME - CREATION ######
		sprintf(msgname,"timeout_CTS %d to %d",new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());
		SBACK_control_frame *msg = new SBACK_control_frame (msgname);
```

```
            msg->setSender_NodeID(getIndex());
//Set Sender_NodeID
            msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            msg->setType(4);
                        //Set Type (timeout_CTS)
            msg->setKind(7);
                        //Set Next FSM (WAIT_RESPONSE)
            //#########################################


      number_timeout_CTS_sent[getIndex()]=number_timeout_CTS_sent[getIndex(
)]+1;

            scheduleAt(backoff_time, msg);//Corrigir isto com o backoff
time

      }
      else//If node != 0
      {
            //The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

      backoff_time=cts_carrier_sense_vector[getIndex()]+MAX_TX_CTS_TIME;

            //###### timeout_CTS FRAME - CREATION ######
            sprintf(msgname,"timeout_CTS %d to %d",new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());
            SBACK_control_frame *msg = new SBACK_control_frame (msgname);
            msg->setSender_NodeID(getIndex());
//Set Sender_NodeID
            msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            msg->setType(4);
                        //Set Type (timeout_CTS)
            msg->setKind(7);
                        //Set Next FSM (WAIT_RESPONSE)
            //#########################################


      number_timeout_CTS_sent[getIndex()]=number_timeout_CTS_sent[getIndex(
)]+1;

            scheduleAt(backoff_time, msg); //Corrigir isto com o backoff
time

      }

}

//#########################################################################
SBACK::send_CTSMessages(cMessage *msg)
void SBACK::send_CTSMessages(cMessage *msg)
{

      SBACK_control_frame *new_msg=static_cast<SBACK_control_frame*>(msg);

      // Initialize variables
```

```
	int m;
	int i;
	char msgname[20];// define size of message


	if(new_msg->isSelfMessage() && (strcmp(new_msg->getName(),
"timeout_CTS 0 to 1") == 0)) //If Receiver_NodeID()==0
	{
		m=gateSize("gate");
		for (i = 0; i < m; ++i)
		{
			//###### CTS FRAME - CREATION ######
			sprintf(msgname,"CTS 1 to %d",0);
			SBACK_control_frame *new_msg = new SBACK_control_frame
(msgname);
			new_msg->setSender_NodeID(0);
//Set Sender_NodeID
			new_msg->setReceiver_NodeID(i+1);
			//Set Receiver_NodeID
			new_msg->setType(5);
				//Set Type (CTS)
			new_msg->setKind(6);
				//Set Next FSM (TRANSMIT)
			//#################################

			sendDelayed(new_msg,0,"gate$o",i);
		}
	}
}

//########################################################################
SBACK::timeout_Event_DATAMessages(cMessage *msg)
void SBACK::timeout_Event_DATAMessages(cMessage *msg)
{
	SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);//Tem
de ser control frame para ver correctamente a mensagem que estÃ¡ a chegar

	EV<<"timeout_Event_DATAMessages msg: "<<new_msg->getName()<<endl;
	EV<<endl;
	EV<<"getSender_NodeID: "<<new_msg->getSender_NodeID()<<endl;
	EV<<"getReceiver_NodeID: "<<new_msg->getReceiver_NodeID()<<endl;

	// Initialize variables
	char msgname[40];// define size of message

	if(getIndex()==1 && new_msg->isSelfMessage()) //Se o nÃ³ que vai
receber Ã© o nÃ³ 0
	{
		double i=0,j=0,x=1;
		simtime_t i_aux=0,j_aux=0,x_aux;

		for(i=1;i<11;i++)//i<11
		{
			for(j=1;j<11;j++)//j<11
			{
				//###### CTS FRAME - CREATION ######
				sprintf(msgname,"timeout_DATA 1 to %d - Message NÂº
%d Fragment NÂº %d",0,(int)i,(int)j);
```

```
                        SBACK_data_frame *msg = new SBACK_data_frame
(msgname);
                        msg->setSender_NodeID(1);
//Set Sender_NodeID
                        msg->setReceiver_NodeID(0);
                  //Set Receiver_NodeID
                        msg->setMessage_Number(i);
                  //Set Message_Number
                        msg->setFragment_Number(j);
                  //Set Fragment_Number
                        msg->setType(6);
                        //Set Type (timeout_DATA)
                        msg->setKind(7);
                        //Set Next FSM (WAIT RESPONSE)
                        //###################################

                        EV<<"############"<<endl;
                        EV<<"i: "<<i<<endl;
                        EV<<"j: "<<j<<endl;
                        EV<<"############"<<endl;

                        i_aux=(i/100);
                        j_aux=(j/100);
                        x_aux=(x/100);

                        EV<<"i_aux: "<<i_aux<<endl;
                        EV<<"j_aux: "<<j_aux<<endl;
                        EV<<"x_aux: "<<x_aux<<endl;

                        EV<<"Teste: "<<time_between_DATAMessages<<endl;

      time_between_DATAMessages=time_between_DATAMessages+0.000001;


      scheduleAt(DIFS+SYNC_CW+DIFS+RTS_CW+SIFS+CTS_CW+SIFS+time_between_DAT
AMessages, msg);//scheduleAt(9+i+j+x, msg)
                  }
                  x++;
            }
      }
}

//#####################################################################
SBACK::send_DATA_Messages(cMessage *msg)
void SBACK::send_DATA_Messages(cMessage *msg)
{
      SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

      if(new_msg->getType()==6)
      {
            // Initialize variables
            char msgname[40];// define size of message
            int i,j;

            i=new_msg->getMessage_Number();//Get Message_Number()
            j=new_msg->getFragment_Number();//Get Fragment_Number()

            if(getIndex()==1)
```

```
            {
                  //###### DATA FRAME - CREATION ######
                  sprintf(msgname,"DATA 1 to 0 - Message NÂº %d Fragment
NÂº %d",i,j);
                  SBACK_data_frame *new_msg = new SBACK_data_frame
(msgname);
                  new_msg->setSender_NodeID(1);
//Set Sender_NodeID
                  new_msg->setReceiver_NodeID(new_msg-
>getReceiver_NodeID()); //Set Receiver_NodeID
                  new_msg->setMessage_Number(i);
            //Set Message_Number
                  new_msg->setFragment_Number(j);
            //Set Fragment_Number
                  new_msg->setType(7);
                        //Set Type (DATA)
                  new_msg->setKind(8);
                        //Set Next FSM (TYPE_ACK)
                  //#################################

                  sendDelayed(new_msg,0,"gate$o",0); //getReceiver_DateID

            }
      }
}

//#####################################################################
SBACK::passing_DATA_Messages(cMessage *msg)
void SBACK::passing_DATA_Messages(cMessage *msg)
{
      SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

      // Initialize variables
      char msgname[40];// define size of message

      if(getIndex()==0)
      {
            if(new_msg->getSender_NodeID()==1)
            {
                  //###### DATA_PASSING FRAME - CREATION ######
                  sprintf(msgname,"DATA 0 to %d - Message NÂº %d Fragment
NÂº %d",3,new_msg->getMessage_Number(),new_msg-
>getFragment_Number());//aleatoriamente o programa escolheu um pacote RTS 1
to 0 para ser enviada a informaÃ§ao do CTS indica qual o nÃ³ que vai ganhar
o meio
                  SBACK_data_frame *new_msg = new SBACK_data_frame
(msgname);
                  new_msg->setSender_NodeID(0);
//Set Sender_NodeID
                  new_msg->setReceiver_NodeID(3);
//Set Receiver_NodeID
                  new_msg->setType(8);
                        //Set Type (DATA_PASSING)
                  new_msg->setKind(8);
                        //Set Next FSM (TYPE_ACK)
                  //###### DATA_PASSING FRAME - CREATION ######

                  sendDelayed(new_msg,0,"gate$o",2);
```

```
			}
		}
}

//#####################################################################
SBACK::timeout_Event_ACKMessages(cMessage *msg)
void SBACK::timeout_Event_ACKMessages(cMessage *msg)
{
		SBACK_control_frame *new_msg=static_cast<SBACK_control_frame*>(msg);

		EV<<"getSender_NodeID: "<<new_msg->getSender_NodeID()<<endl;
		EV<<"getReceiver_NodeID: "<<new_msg->getReceiver_NodeID()<<endl;

		// Initialize variables
		char msgname[40];// define size of message

		if(getIndex()==0 && new_msg->isSelfMessage())
		{
			double i=0,j=0,x=1;
			simtime_t i_aux=0,j_aux=0,x_aux;

			for(i=1;i<11;i++)//i<11
			{
				for(j=1;j<11;j++)//j<11
				{
					//###### timeout_ACK FRAME - CREATION ######
					sprintf(msgname,"timeout_ACK 0 to %d - Message NÂº
%d Fragment NÂº %d",1,(int)i,(int)j);//aleatoriamente o programa escolheu
um pacote RTS 1 to 0 para ser enviada a informaÃ§ao do CTS indica qual o
nÃ³ que vai ganhar o meio
					SBACK_data_frame *msg = new SBACK_data_frame
(msgname);
					msg->setSender_NodeID(0);
//Set Sender_NodeID
					msg->setReceiver_NodeID(1);
//Set Receiver_NodeID
					msg->setMessage_Number(i);
//Set Message_Number
					msg->setFragment_Number(j);
//Set Fragment_Number
					msg->setType(9);
					//Set Type (timeout_ACK)
					msg->setKind(8);
					//Set Next FSM (TYPE_ACK)
					//#########################################

					EV<<"############"<<endl;
					EV<<"i: "<<i<<endl;
					EV<<"j: "<<j<<endl;
					EV<<"############"<<endl;

					i_aux=(i/100);
					j_aux=(j/100);
					x_aux=(x/100);

					EV<<"i_aux: "<<i_aux<<endl;
					EV<<"j_aux: "<<j_aux<<endl;
```

```
                        EV<<"x_aux: "<<x_aux<<endl;

                        EV<<"Teste2: "<<time_between_ACKMessages<<endl;


      time_between_ACKMessages=time_between_ACKMessages+0.0000010001;


      scheduleAt(DIFS+SYNC_CW+DIFS+RTS_CW+SIFS+CTS_CW+SIFS+time_between_ACK
Messages, msg);//scheduleAt(9+i+j+x+0.1, msg)
                  }
                  x++;
            }
      }
}

//#####################################################################
SBACK::send_ACK_Messages(cMessage *msg)
void SBACK::send_ACK_Messages(cMessage *msg)
{
      SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

      // Initialize variables
      char msgname[40];// define size of message

      if(getIndex()==0)
      {
            //###### ACK FRAME - CREATION ######
            sprintf(msgname,"ACK 0 to %d - Message NÂº %d Fragment NÂº
%d",1,new_msg->getMessage_Number(),new_msg-
>getFragment_Number());//aleatoriamente o programa escolheu um pacote RTS 1
to 0 para ser enviada a informaÃ§ao do CTS indica qual o nÃ³ que vai ganhar
o meio
            SBACK_control_frame *new_msg = new SBACK_control_frame
(msgname);
            new_msg->setSender_NodeID(0);
//Set Sender_NodeID
            new_msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            new_msg->setType(10);
                  //Set Type (ACK)
            new_msg->setKind(8);
                  //Set Next FSM (TYPE_ACK)
            //#################################

            sendDelayed(new_msg,0,"gate$o",0); //simTime()

      }
}

//#####################################################################
SBACK::timeout_Event_RTS_ADDBA_Request_Messages(cMessage *msg)
void SBACK::timeout_Event_RTS_ADDBA_Request_Messages(cMessage *msg)
{
      SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

      // Initialize variables
      char msgname[40];// define size of message
```

```
	if(getIndex()==0)
	{
		//The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

	backoff_time=rts_carrier_sense_vector[getIndex()]+MAX_TX_RTS_TIME;

		//###### timeout_RTS_ADDBA_Request FRAME - CREATION ######
		sprintf(msgname,"timeout_RTS_ADDBA_Request %d to %d",new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());//+1 because index of
node==0
		SBACK_control_frame *msg = new SBACK_control_frame (msgname);
		msg->setSender_NodeID(getIndex());
//Set Sender_NodeID
		msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
		msg->setType(11);
			//Set Type (timeout_RTS_ADDBA_Request)
		msg->setKind(5);
			//Set Next FSM (IDLE)
		scheduleAt(backoff_time, msg);//Corrigir isto com o backoff
time
		//#############################################################

	}
	else
	{
		//The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

	backoff_time=rts_carrier_sense_vector[getIndex()]+MAX_TX_RTS_TIME;

		//###### timeout_RTS_ADDBA_Request FRAME - CREATION ######
		sprintf(msgname,"timeout_RTS_ADDBA_Request %d to %d",new_msg-
>getSender_NodeID(),new_msg->getReceiver_NodeID());//because index of
node==0
		SBACK_control_frame *msg = new SBACK_control_frame (msgname);
		msg->setSender_NodeID(getIndex());
//Set Sender_NodeID
		msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
		msg->setType(11);
			//Set Type (timeout_RTS_ADDBA_Request)
		msg->setKind(5);
			//Set Next FSM (IDLE)
		scheduleAt(backoff_time, msg); //Corrigir isto com o backoff
time
		//#############################################################

	}
}

//#####################################################################
SBACK::send_RTS_ADDBA_Request_Messages(cMessage *msg)
void SBACK::send_RTS_ADDBA_Request_Messages(cMessage *msg)
{
	SBACK_synch_frame *new_msg=static_cast<SBACK_synch_frame*>(msg);
```

```
	// Initialize variables
	char msgname[40];// define size of message

	if(new_msg->isSelfMessage())
	{
		if(new_msg->getSender_NodeID()==1) //Corrigir isto porque
apenas o nÃ³ 1 envia
		{
			//###### RTS_ADDBA_Request_Messages FRAME - CREATION
######
			sprintf(msgname,"RTS_ADDBA_Request_Messages %d to %d"
,new_msg->getSender_NodeID(),new_msg->getReceiver_NodeID());
			new_msg->setName(msgname);
//Set Name
			new_msg->setType(12);
		//Set Type (RTS_ADDBA_Request_Messages)
			new_msg->setKind(20);
		//Set Next FSM
			sendDelayed(new_msg,0,"gate$o",new_msg-
>getReceiver_GateID()); //getReceiver_DateID
			//#############################################################
		}
	}
}


//#####################################################################
SBACK::timeout_Event_CTS_ADDBA_Response_Messages(cMessage *msg)
void SBACK::timeout_Event_CTS_ADDBA_Response_Messages(cMessage *msg)
{
	SBACK_synch_frame *new_msg=static_cast<SBACK_synch_frame*>(msg);

	// Initialize variables
	char msgname[40];// define size of message

	if(getIndex()==0 && msg->isSelfMessage()) //Se o nÃ³ que vai receber
Ã© o nÃ³ 0
	{

		//The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

	backoff_time=cts_carrier_sense_vector[getIndex()]+MAX_TX_CTS_TIME;

		//###### timeout_CTS_ADDBA_Response_Messages FRAME - CREATION
######
		sprintf(msgname,"timeout_CTS_ADDBA_Response_Messages %d to
%d",new_msg->getSender_NodeID(),new_msg->getReceiver_NodeID());//+1 because
index of node==0
		SBACK_control_frame *msg = new SBACK_control_frame (msgname);
		msg->setSender_NodeID(getIndex());
		  //Set Sender_NodeID
		msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
		msg->setType(13);
//Set Type (timeout_CTS_ADDBA_Response_Messages)
```

```
            msg->setKind(7);
                        //Set Next FSM (WAIT RESPONSE)

      //#####################################################################

            scheduleAt(backoff_time, msg);//Corrigir isto com o backoff
time

      }
      else
      {
            //The backoff_time to transmit a SYNCH must be between the end
of the carrier sense and the maximum value of the Synch TX window

      backoff_time=cts_carrier_sense_vector[getIndex()]+MAX_TX_CTS_TIME;

            //###### timeout_CTS_ADDBA_Response_Messages FRAME - CREATION
######
            sprintf(msgname,"timeout_CTS_ADDBA_Response_Messages %d to
%d",new_msg->getSender_NodeID(),new_msg->getReceiver_NodeID());//because
index of node==0
            SBACK_control_frame *msg = new SBACK_control_frame (msgname);
            msg->setSender_NodeID(getIndex());
                  //Set Sender_NodeID
            msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            msg->setType(13);
//Set Type (timeout_CTS_ADDBA_Response_Messages)
            msg->setKind(7);
                        //Set Next FSM (WAIT RESPONSE)

      //#####################################################################

            scheduleAt(backoff_time, msg); //Corrigir isto com o backoff
time

      }
}

//#####################################################################
SBACK::send_CTS_ADDBA_Response_Messages(cMessage *msg)
void SBACK::send_CTS_ADDBA_Response_Messages(cMessage *msg)
{
      SBACK_control_frame *new_msg=static_cast<SBACK_control_frame*>(msg);

      // Initialize variables
      int m;
      int i;
      char msgname[40];// define size of message


      if(new_msg->isSelfMessage() && (strcmp(new_msg->getName(),
"timeout_CTS_ADDBA_Response_Messages 0 to 1") == 0))
      {
            m=gateSize("gate");
            for (i = 0; i < m; ++i)
            {
```

```
                  //###### CTS_ADDBA_Response_Messages FRAME - CREATION
######
                  sprintf(msgname,"CTS_ADDBA_Response_Messages 1 to %d",0);
                  SBACK_control_frame *new_msg = new SBACK_control_frame
(msgname);
                  new_msg->setSender_NodeID(0);
                   //Set Sender_NodeID
                  new_msg->setReceiver_NodeID(i+1);
//Set Receiver_NodeID
                  new_msg->setType(14);
                   //Set Type (CTS_ADDBA_Response_Messages)
                  new_msg->setKind(6);
                         //Set Next FSM (TRANSMIT)

      //#############################################################

                  sendDelayed(new_msg,0,"gate$o",i);
            }
      }
}

//#####################################################################
SBACK::timeout_Event_RTS_BA_Request_Messages(cMessage *msg)
void SBACK::timeout_Event_RTS_BA_Request_Messages(cMessage *msg)
{
      SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

      EV<<" SBACK::send_RTS_BA_Request_Messages"<<new_msg->getName()<<endl;
      EV<<"getSender_NodeID: "<<new_msg->getSender_NodeID()<<endl;
      EV<<"getReceiver_NodeID: "<<new_msg->getReceiver_NodeID()<<endl;
      EV<<"getMessage_Number: "<<new_msg->getMessage_Number()<<"
getFragment_Number"<<new_msg->getFragment_Number()<<endl;
      EV<<"getIndex: "<<getIndex()<<endl;

      // Initialize variables
      char msgname[40];// define size of message
      int i,j;

      i=new_msg->getMessage_Number();
      j=new_msg->getFragment_Number();

      if(getIndex()==0 || getIndex()==1)
      {
            //###### RTS_BA_Request_Messages FRAME - CREATION ######
            sprintf(msgname,"timeout_RTS_BA_Request_Messages 1 to 0 -
Message NÂº %d Fragment NÂº %d",i,j);
            SBACK_data_frame *new_msg = new SBACK_data_frame (msgname);
            new_msg->setSender_NodeID(1);
             //Set Sender_NodeID
            new_msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            new_msg->setMessage_Number(i);
                  //Set Message_Number
            new_msg->setFragment_Number(j);
                  //Set Fragment_Number
            new_msg->setType(15);
//Set Type (timeout_RTS_BA_Request_Messages)
```

```
            new_msg->setKind(9);
//Set Next FSM (VERIFY FRAME FINISH)
            //#########################################################

            scheduleAt(msg-
>getArrivalTime()+simTime(),new_msg);//scheduleAt(20,new_msg)  0.136
      }
}

//#####################################################################
SBACK::send_RTS_BA_Request_Messages(cMessage *msg)
void SBACK::send_RTS_BA_Request_Messages(cMessage *msg)
{
            SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);
            EV<<" SBACK::send_RTS_BA_Request_Messages"<<new_msg-
>getName()<<endl;
            EV<<"getSender_NodeID: "<<new_msg->getSender_NodeID()<<endl;
            EV<<"getReceiver_NodeID: "<<new_msg-
>getReceiver_NodeID()<<endl;
            EV<<"getMessage_Number: "<<new_msg->getMessage_Number()<<"
getFragment_Number"<<new_msg->getFragment_Number()<<endl;
            EV<<"getIndex: "<<getIndex()<<endl;

            // Initialize variables
            char msgname[60];// define size of message
            int i,j;

            i=new_msg->getMessage_Number();
            j=new_msg->getFragment_Number();

            if(getIndex()==1)
            {
                  //###### RTS_BA_Request_Messages FRAME - CREATION ######
                  sprintf(msgname,"RTS_BA_Request_Messages 1 to 0 - Message
NÂº %d Fragment NÂº %d",i,j);//aleatoriamente o programa escolheu um pacote
RTS 1 to 0 para ser enviada a informaÃ§ao do CTS indica qual o nÃ³ que vai
ganhar o meio
                  SBACK_data_frame *new_msg = new SBACK_data_frame
(msgname);
                  new_msg->setSender_NodeID(1);
                   //Set Sender_NodeID
                  new_msg->setReceiver_NodeID(new_msg-
>getReceiver_NodeID());  //Set Receiver_NodeID
                  new_msg->setMessage_Number(i);
                        //Set Message_Number
                  new_msg->setFragment_Number(j);
                        //Set Fragment_Number
                  new_msg->setType(16);
                        //Set Type (RTS_BA_Request_Messages)
                  new_msg->setKind(9);
                   //Set Next FSM (VERIFY FRAME FINISH)
                  //#########################################################

                  sendDelayed(new_msg,0,"gate$o",0); //getReceiver_DateID
            }
}

void SBACK::send_CTS_BA_Response_Messages(cMessage *msg)
```

```
{
	SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

	// Initialize variables
	char msgname[60];// define size of message
	int i,j;

	i=new_msg->getMessage_Number();
	j=new_msg->getFragment_Number();

	if(getIndex()==0)
	{
		//###### CTS_BA_Response_Messages FRAME - CREATION ######
		sprintf(msgname,"CTS_BA_Response_Messages 1 to 0 - Message NÂº
%d Fragment NÂº %d",i,j);//aleatoriamente o programa escolheu um pacote RTS
1 to 0 para ser enviada a informaÃ§ao do CTS indica qual o nÃ³ que vai
ganhar o meio
		SBACK_data_frame *new_msg = new SBACK_data_frame (msgname);
		new_msg->setSender_NodeID(1);
		 //Set Sender_NodeID
		new_msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
		new_msg->setMessage_Number(i);
			//Set Message_Number
		new_msg->setFragment_Number(j);
			//Set Fragment_Number
		new_msg->setType(17);
			//Set Type (CTS_BA_Response_Messages)
		new_msg->setKind(9);
//Set Next FSM (VERIFY FRAME FINISH)
		//#########################################################

		sendDelayed(new_msg,0,"gate$o",0); //getReceiver_DateID
	}
}
void SBACK::send_RTS_DELBA_Request_Messages(cMessage *msg)
{
	SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

	// Initialize variables
	char msgname[60];// define size of message
	int i,j;

	i=new_msg->getMessage_Number();
	j=new_msg->getFragment_Number();

	if(getIndex()==0 || getIndex()==1)
	{
		//###### RTS_DELBA_Request_Messages FRAME - CREATION ######
		sprintf(msgname,"RTS_DELBA_Request_Messages 1 to 0 - Message
NÂº %d Fragment NÂº %d",i,j);//aleatoriamente o programa escolheu um pacote
RTS 1 to 0 para ser enviada a informaÃ§ao do CTS indica qual o nÃ³ que vai
ganhar o meio
		SBACK_data_frame *new_msg = new SBACK_data_frame (msgname);
		new_msg->setSender_NodeID(1);
		 //Set Sender_NodeID
		new_msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
```

```
            new_msg->setMessage_Number(i);
                  //Set Message_Number
            new_msg->setFragment_Number(j);
                  //Set Fragment_Number
            new_msg->setType(18);
                  //Set Type (RTS_DELBA_Request_Messages)
            new_msg->setKind(9);
                  //Set Next FSM (VERIFY FRAME FINISH)
            //#############################################################

            sendDelayed(new_msg,0,"gate$o",0); //getReceiver_DateID
      }
}

void SBACK::send_CTS_DELBA_Response_Messages(cMessage *msg)
{
      SBACK_data_frame *new_msg=static_cast<SBACK_data_frame*>(msg);

      // Initialize variables
      char msgname[60];// define size of message
      int i,j;

      i=new_msg->getMessage_Number();
      j=new_msg->getFragment_Number();

      if(getIndex()==0)
      {
            //###### CTS_DELBA_Response_Messages FRAME - CREATION ######
            sprintf(msgname,"CTS_DELBA_Response_Messages 1 to 0 - Message
NÂº %d Fragment NÂº %d",i,j);
            SBACK_data_frame *new_msg = new SBACK_data_frame (msgname);
            new_msg->setSender_NodeID(1);
             //Set Sender_NodeID
            new_msg->setReceiver_NodeID(new_msg->getReceiver_NodeID());
//Set Receiver_NodeID
            new_msg->setMessage_Number(i);
                  //Set Message_Number
            new_msg->setFragment_Number(j);
                  //Set Fragment_Number
            new_msg->setType(19);
                  //Set Type (CTS_DELBA_Response_Messages);
            //#############################################################

            sendDelayed(new_msg,0,"gate$o",0);
      }
}
void SBACK::bubble_sort_next_sleep_time(cMessage *msg)
{
      int i,j;

      for (i = 0; i < total_number_nodes; i++)
      {
            for(j=0;j<total_number_nodes;j++)
            {

      aux_next_sleep_time[i][j]=network_info[i].next_sleep_time[j];

            }
```

```
	}
}

//generates a psuedo-random double between min and max
double randdouble(double min, double max)
{
	if (min>max)
	{
		return dblrand()*(min-max)+max;
	}
	else
	{
		return dblrand()*(max-min)+min;
	}
}

void SBACK::print_network_info(void)
{
//EV<<"network_info[0].next_sleep_time[1]:"<<network_info[0].next_sleep_tim
e[0]<<endl;
//EV<<"network_info[0].next_sleep_time[2]:"<<network_info[0].next_sleep_tim
e[1]<<endl;
//EV<<"network_info[0].next_sleep_time[3]:"<<network_info[0].next_sleep_tim
e[2]<<endl;
//EV<<"network_info[0].next_sleep_time[4]:"<<network_info[0].next_sleep_tim
e[3]<<endl;
//EV<<endl;
//EV<<"network_info[1].next_sleep_time[0]:"<<network_info[1].next_sleep_tim
e[0]<<endl;
//EV<<"network_info[2].next_sleep_time[0]:"<<network_info[2].next_sleep_tim
e[0]<<endl;
//EV<<"network_info[3].next_sleep_time[0]:"<<network_info[3].next_sleep_tim
e[0]<<endl;
//EV<<"network_info[4].next_sleep_time[0]:"<<network_info[4].next_sleep_tim
e[0]<<endl;
//EV<<endl;
//EV<<"network_info[1].next_sleep_time[1]:"<<network_info[1].next_sleep_tim
e[1]<<endl;
//EV<<"network_info[2].next_sleep_time[1]:"<<network_info[2].next_sleep_tim
e[1]<<endl;
//EV<<"network_info[3].next_sleep_time[1]:"<<network_info[3].next_sleep_tim
e[1]<<endl;
//EV<<"network_info[4].next_sleep_time[1]:"<<network_info[4].next_sleep_tim
e[1]<<endl;
//EV<<endl;
//EV<<"network_info[0].number_synch_received:"<<network_info[0].number_sync
h_received<<endl;
//EV<<"network_info[1].number_synch_received:"<<network_info[1].number_sync
h_received<<endl;
//EV<<"network_info[2].number_synch_received:"<<network_info[2].number_sync
h_received<<endl;
//EV<<"network_info[3].number_synch_received:"<<network_info[3].number_sync
h_received<<endl;//EV<<"network_info[4].number_synch_received:"<<network_in
fo[4].number_synch_received<<endl;

}
```

References

[ADLN98] Asada,G., Dong,M., Lin,T.S., Newberg,F., Pottie,G., and Kaiser W.J., "Wireless Integrated NetworkSensors: Low Power Systems on a chip", In *Proc*. *of the 1998 European Solid State Circuits Conference*, The Hague, Netherlands, 1998.

[ASSC02] Akyildiz,I.,F., Su,W., Sankarasubramaniam,Y. and Cayirci,E., "A survey on sensor networks," *IEEE Communications*, Vol. 3, No. 8, Aug. 2002, pp. 102–114.

[AT8609] http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf, Aug. 2009.

[BBVL09a] Borges,L.M., Barroca,N., Velez,F.J., and Lebres,A.S., "Smart-Clothing Wireless Flex Sensor Belt Network for Foetal Health Monitoring", In *Proc*. *of WiPH 2009 International Workshop on Wireless Pervasive Healthcare (held in conjunction with 3rd International Conference on Pervasive Computing Technologies for Healthcare 2009)*, London, UK, 2009.

[BBVL09b] Borges,L.M., Barroca,N., Velez,F.J., and Lebres,A.S., "WIRELESS FLEX SENSOR BELT NETWORKS FOR FOETAL MOVEMENT MONITORING IN LOW RISK PREGNANCIES", accepted for publication In *XIX IMEKO World Congress Fundamental and Applied Metrology*, Lisbon, Portugal, Sep. 2009.

[BCDD05] Bhatti,S., Carlson,J., Dai,H., Deng,J., Rose,J., Sheth,A., Shucker,B., Gruenwald,C., Torgerson,A., and Han,R., "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *Mobile Networks and Applications (MONET) Journal, Special Issue on Wireless Sensor Networks*, Vol. 10, No. 4, Aug. 2005, pp. 563–579.

[Bhar94] Bharghavan,V., "MACAW: A Media Access Protocol for Wireless LANs ". in *Proc of the ACM SIGCOMM'94 Conference*, London, UK, Aug., 1994

[CC1009] http://www.cse.ohio-state.edu/siefast/nest/nest_webpage/datasheet/Chipcon%20-%20CC1000%20Data %20Sheet%20v2.1.pdf, Aug. 2008.

[CC2409] http://focus.ti.com/lit/ds/symlink/cc2420.pdf, Aug. 2009.

[CONF09] Barroca,N., Velez,F.J., Ferro,J.,M., Borges,L.M. and Lebres,A.S., "Desenho de Protocolos de Controlo de Acesso ao Meio Eficientes do ponto de vista energético em Redes de Sensores Sem Fios", submitted to *Engenharia'2009 – Inovação e Desenvolvimento*, Covilhã, Portugal, Nov. 2009.

[DaLa03] Dam,T.V. and Langendoen,K., “An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks,” In *Proc. of the 1st International Conference on Embedded Networked Sensor Systems*, California, USA, Nov. 2003.

[DuGV04] Dunkels,A., Gronvall,B. and Voigt,T., “Contiki - a lightweight and flexible operating system for tiny networked sensors,” In *Proc. of 29th Annual IEEE International Conference on Local Computer Networks*, Florida, USA, Nov. 2004.

[FuGa97] Fullmer,C.L. and Garcia-Luna-Aceves,J.J., “Solutions to hidden terminal problems in wireless networks,” in *Proc. of Association for Computing Machinery's Special Interest Group on Data Communication 97*, Cannes, FRANCE, Sep. 1997.

[HaSm00] Havinga,P.J.M. and Smit,G.J, “Design techniques for low power systems,” *Journal of Systems Architecture*, Vol. 46, No. 1, Jan. 2000, pp. 1–21.

[HKSK05] Han,C., Kumar,R., Shea,R., Kohler,E. and Srivastava,M., “A dynamic operating system for sensor nodes,” In *Proc. of Third International Conference on Mobile Systems, Applications and Services*, Seattle, USA, June 2005.

[IRIS08] http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/IRIS_Datasheet.pdf, Dec. 2008.

[ISAB09] Barroca,N., Velez,F.J., Ferro,J.,M., Borges,L.M. and Lebres,A.S., “An Innovative Sensor MAC Protocol for WBAN Applications”, accepted for publication in ISABEL 2009 - 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies, Bratislava, Slovak Republic, Nov. 2009.

[KaKP99] Kahn,J.M., Katz,R.H. and Pister,K.S, “Next century challenges: mobile networking for Smart Dust”Wesley, *In Proc. of 5th Annual ACM/IEEE* International Conference on Mobile Computing and Networking, Seattle, USA, Aug. 1999.

[KaWi05] Karl,H. and Willig,A., *Protocols and Architectures for Wireless Sensor Networks*, Wiley, Chichester, UK, 2005.

[KuHH05] Kuorilehto,M., Hannikainen,M., and Hamalainen,T.D, “A Survey of Application Distribution in Wireless Sensor Networks”, *Journal on Wireless Communication and Networking*, VOL.2005, No 5, Oct. 2005, pp 774-788.

[KuRo08] Kurose,J.F. and Ross,K.,W., *Computer Networking: A Top-Down Approach* , Addison-Wesley, Boston, USA 2008.

[LiFS07] Liu,S, Fan,K.,W., and Sinha,P., “CMAC: An Energy Efficient MAC Layer Protocol Using Convergent Packet Forwarding for Wireless Sensor,” *in Proc. of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks(SECON '07)*, California, USA, Jun. 2007.

[MBCI02] Min,R., Bhardwai,M., Cho,S.,H., Ikes,N., Shih,E, Sisha,A., Wang,A. and Chandrakasa,A. ,“Energy – Centric Enabling Technologies for Wireless Sensor Networks,” *IEEE Wireless Communication*, VOL. 9, No 4, Aug. 2002, pp 28-39.

[MDA108] http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS_MDA_Datasheet. pdf, Dec. 2008.

[MICA08] http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf, Dec. 2008.

[MiCh02] Yang,H. and Sikdar,B., “A framework for energy-scalable communication in high-density wireless networks,” In *Proc. of ISLPED’ 02*, Monterey, California, USA, Aug. 2002.

[MPSC02] Mainwaring,A., Polastre,J., Szewczyk,R., Culler,D. and Anderson,J., “Wireless sensor networks for habitat monitoring”, In *Proc. of Second ACM International Workshop on Wireless Sensor Networks and Application*, Atlanta, USA, Sept. 2002.

[MTSM08] http://www.xbow.com/Support/Support_pdf_files/MTS-MDA_Series_Users_Manual.pdf, y. 2008.

[NaBe07] Nardis,L.D. and Benedetto, M.-G., “Overview of the IEEE 802.15.4/4a standards for low data rate Wireless Personal Data Networks”, In *Proc*. *of Positioning, Navigation and Communication, 2007 WPNC '07, 4th Workshop*, Hannover, Germany, 2007.

[OMNE09] http://www.omnetpp.org/, Aug. 2009.

[OMNE09] http://www.omnetpp.org/, Aug. 2009.

[PoCu04] Polastre,J., Hill ,J. and Culler,D., ”Versatile Low Power Media Access for Wireless Sensor Networks,” In *Proc. of* the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, USA, Nov. 2004.

[PoKa00] Pottie,G. J. and Keiser,W.J., “Wireless integrated network sensor”, In *Proc. of First IEEE INFOCOM 2002*, New York, June 2002.

[POWE09] http://www.xbow.com/Support/Support_pdf_files/PowerManagement.xls, Aug. 2009.

[PR8009] http://standards.ieee.org/announcements/pr_802154.html, Jan. 2009.

[PRFC08] http://www.xbow.com/Support/Support_pdf_files/Product_Feature_Reference_Chart.pdf, Dec. 2008.

[RaOG06] Rajendran,V., Obraczka,K., and Garcia-Luna-Aceves,J.,J., “Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks,” In *Journal of Wireless Networks*, VOL. 12, Feb. 2006, pp. 63-78.

[ReMC07] Ren,H., Meng,M. and Cheung,C., “Experimental Evaluation of On-body Transmission Characteristics for Wireless Biosensors”, In *Proc. of International Conference on*

Integration Technology, Shenzhen, China, Mar. 2007.

[RMBM06] Raymond,D., Marchany,R., Brownfield,M. and Midkiff,S., "Effects of Denial of Sleep Attacks on Wireless Sensor Network MAC Protocols", In *Proc. of Workshop on Information Assurance*, West Point, NY, June. 2006.

[ROWR04] Roundy,S., Wrigth,P.K. and Rabaey,J.M., *Energy Scavenging for Wireless Sensor Networks with Special Focus on Vibrations*, Kluwer Academic Publishers, Massachusetts, USA, 2004.

[RSSI09] https://community.ti.com/forums/p/362/1046.aspx, Aug. 2009.

[SHIM01] Shih,E., Hwan,S., Ickes,N., Min,R., Sinha,A., Wang,A. and Shandrakasan,A. "Physical Layer Driven Protocol and Algorithm Design for EnergyEfficient Wireless Sensor Networks," In *Proc. of Seventh Annual International Conference on Mobile Computing and Networkin 2001 (MobiCom)*, Rome, Italy, July. 2001.

[SMAR08] http://www.e-projects.ubi.pt/smart-clothing/, Dec. 2008.

[TINY09] http://www.tinyos.net/, May. 2009.

[TR1009] http://www.rfm.com/products/data/tr1000.pdf, Aug. 2009.

[USMA09] http://www.omnetpp.org/doc/omnetpp40/manual/usman.html, Aug. 2009.

[VDMC08] Verdone,R., Dardari,D., Mazzini,G. and Conti,Andrea, *Wireless Sensor and Actuator Networks Technologies, Analysis and Design*, London, UK, 2008.

[WaZK06] Wang,H., Zhang,X., and Khokhar,A., "An Energy-Efficient Low-Latency MAC Protocol for Wireless Sensor Networks," In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM '06)*,California, USA , Nov. 2006.

[WPAN03] LAN MAN Standards Committee of the IEEE Computer Society, Wireless LAN Medium access control (MAC) and physical layer (PHY) specifications for Low-rate Wireless Personal Area Networks (LR-WPANs), IEEE, New York, NY, USA, IEEE Std 802.15.4-2003 edition, 2003.

[WPAN06] LAN MAN Standards Committee of the IEEE Computer Society, Wireless LAN Medium access control (MAC) and physical layer (PHY) specifications for Low-rate Wireless Personal Area Networks (LR-WPANs), IEEE, New York, NY, USA, IEEE Std 802.15.4-2006 edition, 2006.

[WPAN07] LAN MAN Standards Committee of the IEEE Computer Society, Wireless LAN Medium access control (MAC) and physical layer (PHY) specifications for Low-rate Wireless Personal Area Networks (LR-WPANs), IEEE, New York, NY, USA, IEEE Std 802.15.4-2007 edition, 2007.

[YaSi03] Yang,H. and Sikdar,B., "A protocol for tracking mobile targets using sensor networks," *in*

Proc. of First IEEE International Workshop on Sensor Network Protocols and Applications, Anchorage, USA, May 2003.

[YeHE02a] Ye,W., Heidemann, J. and Estrin, D., “An Energy-Efficient MAC Protocol for Wireless Sensor Networks”, *in Proc. of First IEEE INFOCOM 2002*, New York, June 2002.

[YeHE02b] Ye,W., Heidemann, J. and Estrin, D., *Technical Report on A flexible and reliable radio communications stack on motes*, USC/ISI, ISE-TR-565, 2002.

[YeHE04] Ye,W., Heidemann,J. and Estrin, D., “Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks”. In IEEE/ACM on Networking, VOL. 12, No 3, Jun. 2004, pp. 493-506.

[ZIGB09] http://www.zigbee.org/, Aug. 2009.