

GOAT: Development of a Wireless Sensor Network analysis tool

Barrachina Muñoz, Sergio

Curs 2014-2015

Director: BORIS BELLALTA JIMÉNEZ

GRAU EN ENGINYERIA TELEMÀTICA



**Universitat
Pompeu Fabra**
Barcelona

Escola
Superior Politècnica

Treball de Fi de Grau

GOAT: Development of a Wireless Sensor Network analysis tool

Sergio Barrachina Muñoz

TREBALL FI DE GRAU

GRAU EN ENGINYERIA TELEMÀTICA

ESCOLA SUPERIOR POLITÈCNICA UPF

2015

DIRECTOR DEL TREBALL

Boris Bellalta Jiménez

To my beloved grandfather. From him I learned
the difference between doing and not doing.

Special Thanks

To Boris, for supporting me and letting me go a step further.

To my parents Cati and Enrique, for being there.

To Iris, for encouraging me up day after day.

Abstract

Looking at the trend followed by information and communication technologies, we can note a constant evolution towards embedded devices, becoming smaller and more efficient, endowed with greater processing power, storage capacity and ease of communications. Based on these technological advances, network physiognomies have changed from being composed of a limited number of wired connected nodes to a central computer, to be smaller, cheaper and lower power devices capable of processing information locally and transfer it wirelessly.

In that sense, Wireless sensor networks (WSNs) are positioned to be one of the fastest growing fields of study in the next years. WSNs are based on sensor nodes, which are low cost and low consumption devices able to obtain data from their environment, process it locally, and then communicate it via wireless links to a central coordinating node, known as sink. Due to the small size of the nodes, batteries must also be small; therefore saving energy consumption is vital in these networks since it is not always possible to recharge them. Hence, there is a need to meet the goal of energy efficiency, which is to maximize the lifetime of the network while still providing the applications required quality of service (QoS).

This project intends to make possible the analysis of the effect of different Medium Access Control (MAC) and routing protocols on energy consumption in several low node density WSNs scenarios (up to 1,000 nodes). To that end, the GOAT software tool has been developed. GOAT is a graphical network analysis tool that allows designing WSNs and estimating its energy consumption and overall lifetime in thoroughly configurable scenarios. The aim of the tool is to base future WSN designs on the results gathered through the simulations.

Resum

Observant la tendència seguida per les tecnologies de la informació i comunicació, notem una evolució constant cap a dispositius integrats, cada vegada més petits i eficients, dotats d'un major poder de processament, capacitat d'emmagatzematge i facilitat de comunicació. Sobre la base d'aquests avanços, les fisionomies de les xarxes han passat de compondre's d'un nombre limitat de nodes connectats mitjançant cable a un ordinador central, a ser aquests més petits, més barats i de menor consum, capaços de processar la informació a nivell local i transferir-la inalàmbriament.

Les xarxes de sensors sense fils (WSN són les seves sigles en anglès) estan en condicions de ser un dels camps d'estudi de més ràpid creixement en els propers anys. Les WSN estan basades en dispositius de baix cost i consum (nodes de mesura) que són capaços d'obtenir les dades del seu entorn, processar-les a nivell local, i comunicar-les després a través d'enllaços sense fils a un node central de coordinació, conegut com a gateway. A causa de la reduïda mida dels nodes, les bateries han de ser també petites, per tant, l'estalvi de consum energètic és vital en aquest tipus de xarxes, ja que no sempre serà possible accedir a recarregar-les. Hi ha, aleshores, una feient necessitat de

complir l'objectiu de l'eficiència energètica, que tracta de maximitzar la vida útil de la xarxa mentre es mantenen els requisits de qualitat de servei de les aplicacions ofertes.

Aquest projecte té la intenció de permetre l'anàlisi de l'impacte de diversos protocols de control d'accés al medi (MAC) i protocols d'enrutament sobre el consum d'energia en WSNs de baixa densitat (fins a 1.000 nodes). Per a això, l'eina software GOAT ha estat desenvolupada. GOAT és un analitzador gràfic de xarxes que permet dissenyar WSNs i estimar el seu consum d'energia i temps de vida general en escenaris profundament configurables. L'objectiu de l'eina és basar els futurs dissenys d'aquest tipus de xarxes sobre els resultats recollits a través de les simulacions.

Resumen

Observando la tendencia seguida por las tecnologías de la información y comunicación, notamos una evolución constante hacia dispositivos integrados, cada vez más pequeños y eficientes, dotados de un mayor poder de procesamiento, capacidad de almacenamiento y facilidad de comunicación. Sobre la base de estos avances tecnológicos, las fisonomías de las redes han pasado de componerse de un número limitado de nodos conectados por cable a un ordenador central, a ser éstos más pequeños, más baratos y de menor consumo, capaces de procesar la información a nivel local y transferirla inalámbricamente.

Las redes de sensores inalámbricas (WSN son sus siglas en inglés) están en condiciones de ser uno de los campos de estudio de más rápido crecimiento en los próximos años. Las WSN están basadas en dispositivos de bajo coste y consumo (nodos de medida) que son capaces de obtener los datos de su entorno, procesarlos a nivel local, y comunicarlos luego a través de enlaces inalámbricos a un nodo central de coordinación, conocido como gateway. Debido al pequeño tamaño de los nodos, las baterías deben ser también pequeñas, con lo cual, el ahorro de consumo energético es vital en éste tipo de redes, ya que no siempre es posible acceder a recargarlas. Por lo tanto, hay una fehaciente necesidad de cumplir el objetivo de la eficiencia energética, que trata de maximizar la vida útil de la red mientras se mantienen los requisitos de calidad de servicio de las aplicaciones ofrecidas.

Este proyecto tiene la intención de permitir el análisis del impacto de diversos protocolos de control de acceso al medio (MAC) y protocolos de enrutamiento sobre el consumo de energía en WSNs de baja densidad (hasta 1.000 nodos). Para ello, la herramienta de software GOAT ha sido desarrollada. GOAT es una herramienta de análisis gráfico de redes que permite diseñar WSNs y estimar el consumo de energía y tiempo de vida general en escenarios profundamente configurables. El objetivo de la herramienta es basar los futuros diseños WSN sobre los resultados recogidos a través de las simulaciones.

Contents

Abstract	vii
1. INTRODUCTION	1
1.1 Motivation.....	1
1.2 Objectives.....	2
1.3 Document structure.....	2
2. WIRELESS SENSOR NETWORKS	3
2.1 Introduction to WSNs	3
2.2 What is a WSN?	3
2.3 Architecture and components.....	5
a) Sensor node.....	6
b) Gateway	9
c) Software	10
2.4 Challenges	12
2.5 Applications.....	13
3. WIRELESS TECHNOLOGIES FOR WSN	15
3.1 MAC layer impact on energy	15
3.2 IEEE 802.15.4	16
a) Physical layer	16
b) MAC layer	17
c) Upper layers.....	18
3.3 IEEE 802.11ah	18
a) Main technological features	19
b) Physical layer.....	19
c) MAC layer	20
d) Energy savings	21
3.4 Bluetooth Low Energy (BLE)	21
3.5 Other WSN technologies	23
4. GOAT	25
4.1 Introduction.....	25
a) What is GOAT?	25
b) Main features	25
c) System requirements.....	26
4.2. Architecture and design	26
a) Functional design.....	26
b) Technical design	27
4.3 Graphical User Interface (GUI)	29
a) Design	29
b) Structure.....	31
4.3 Topology creation	33
a) Create and modify topology	33
b) Open and save topologies	36

4.4 Graphical information displaying	37
4.5 Modules	39
a) Physical models	39
b) Battery models	40
c) MAC models.....	41
d) Routing models	46
4.6 Configuration.....	49
4.7 Simulation	50
a) Input	50
b) Output	51
4.8 Limitations.....	52
5. EVALUATION	55
5.1 Street parking	55
a) Scenario definition	55
b) Modeling and results.....	56
5.2 Plague tracking	61
a) Scenario definition	61
b) Modeling and results.....	62
6. CONCLUSIONS AND FUTURE WORK	67
ANNEX.....	69
A1. UML Class diagrams.....	69
a) BatteryModel.java	69
b) Goat.java	69
c) MacModel.java	70
d) Map.java.....	70
e) Node.java	71
f) PhyModel.java.....	71
g) Point.java.....	71
f) RoutingModel.java	72
A2. Configuration variables	72
References	75

1. INTRODUCTION

1.1 Motivation

The design of effective and sustainable wireless sensor networks (WSNs) is not trivial. Limited energy resources of sensor nodes are a top constraint in this kind of networks due to the fact that, in most cases, nodes are battery-powered devices and, consequently, energy-constrained. Hence, the main concern is how to reduce the energy consumption in order to extend the overall network lifetime while providing a proper enough performance.

In order to face that issue, one possible solution would be to manually replace each of the node batteries when they are running low. However, energy has a huge impact on both economic and environmental matters. Hence, the mentioned solution would require a big financial investment and would entail a great waste of energy. Another unreasonable option would be to simply plug in each of the nodes to the stream. Obviously, that would suppose losing almost all the mobility of the network, which has a heavy impact on the overall design. Also, in most scenarios where WSNs are currently deployed, these possibilities do not even exist (e.g. high risk and dangerous scenarios). Therefore, studying how WSNs can be optimized and improved is a matter of interest taking into account their potential ability to mix flexibility, mobility and performance.

Regarding energy consumption, the Medium Access Control (MAC) layer protocol is crucial due to its influence on the sensor transceiver, which is the most energy-consuming component of a sensor node. In this project, different types of MAC protocols and routing algorithms are studied aiming to determine which kinds of them are the most suitable with energy saving in low node density scenarios (up to 1,000 nodes). To that end, the GOAT software tool has been developed. GOAT is a graphical network analyzer that allows designing WSNs and estimating its energy consumption and overall lifetime in a huge variety of scenarios. This software aims to be as modular as possible in order to facilitate future open source collaborations to improve the analyzer itself. In that sense, GOAT implements several physical, battery, MAC and routing models that can be selected by the user to build the simulation scenarios. GOAT has several features such as network topology graphical representation, reading and saving of topology data files, packet rates displaying, functional graphical user interface (GUI), energy consumption estimation in a huge number of scenarios, etc.

As a proof of concept, this project presents the analysis of two abstractions of real WSNs scenarios where GOAT has served to determine the optimal MAC protocol and network design: street parking and plague tracking.

1.2 Objectives

The main objective of this project is to develop a WSN analyzer that will make it possible to determine the energy consumption and overall network lifetime in a huge variety of configurable scenarios. This software analyzer, GOAT, will take into account several physical, battery, MAC and routing models. The mentioned tool should serve to design and test future real operating WSNs before actually building them.

Also, as a proof of concept, two abstractions of real scenarios will be analyzed by using GOAT. These analyses will serve to present the simple procedures and useful results of the tool, which should allow us to define the most suitable WSN designs in terms of energy saving.

1.3 Document structure

This document has been divided in 6 sections. At 1. *Introduction*, an overview of the project can be found. Wireless sensor networks and its features and context are depicted at 2. *Wireless Sensor Networks*. At section 3. *WSN Technologies*, the state of the art of WSN technologies is presented. The GOAT analyzer description, design and details are depicted at 4. *GOAT*. At 5. *Evaluation*, two scenarios modeling and results are presented. Finally, at 6. *Conclusion and future work*, the project conclusions and future steps for continuing with the work are presented.

2. WIRELESS SENSOR NETWORKS

2.1 Introduction to WSNs

In recent decades there has been a significant growth of computer networks, particularly of wireless communication kind, prompted by continuous technological advances. As a result, smaller, lower cost and lower energy consumption electronic sensor devices, and both wireless and wired systems capable of processing information locally and communicating with other devices of the same type, have been deployed. These technologies are known as Machine-to-Machine (M2M) and enable to exchange information and operate without human intervention.

There are two categories of M2M technologies: Wireless Sensor Networks (WSNs), which are networks composed by a group of sensor nodes located over a singular area; and cellular networks, composed by radio cells where each cell includes a transceiver providing larger radio coverage. Radio cells could be used to allow the gateway of a particular WSN to reach the Internet [1].

Regarding wireless networks, the smart cities first conceptions irruption have motivated the growth of WSNs and this upward trend will continue over the next years, with over 10 billion mobile-connected devices in 2017 [2], so that the wireless industry could see a \$1.2 trillion revenue opportunity by 2020 [3]. All these factors have promoted the WSN research field, which has been identified as one of the most promising technologies for various technology researchers, institutes and specialized magazines.

Nowadays, several types of sensors can be found in a huge amount of electronic devices and systems. In most applications, these sensors act just as transducers, taking measurements of one or more environment variables and sending this data to a central node that is responsible for processing tasks. However, a new generation of sensors, with their own intelligence and able to organize autonomously and interconnect with other sensors is being deployed. That is where WSN come through, a revolutionary technology field for applications in areas like agriculture, industry, automotive, domotics, medicine, etc.

This chapter provides a general introduction to wireless sensor networks, their components, challenges, applications and simulators.

2.2 What is a WSN?

A wireless sensor network is a network consisting of three main components: sensor nodes, gateways and software. In a WSN, nodes are spatially distributed over an area in ad-hoc manner to monitor the devices themselves or the environmental conditions, such as temperature, sound, light, etc. The data gathered by a node is transmitted among the rest of nodes, depending on the routing model, through virtual (wireless) links until the data reaches the sink (or gateway). The gateway can both operate independently, or be

connected to a server allowing collecting, processing, analyzing and presenting the data measurements through software implementation.

This kind of networks often includes actuators, devices that convert electrical signals into physical actions, however the commonly accepted name in the literature has remained as wireless sensor networks.

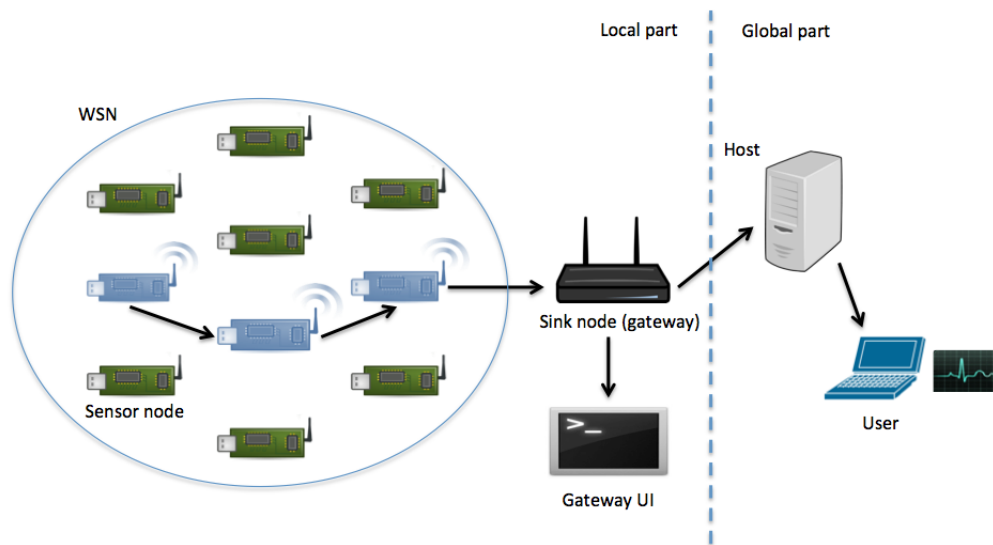


Figure 1: Wireless Sensor Network overall architecture

In the picture above, a WSN diagram is shown. The local part of the system contains the WSN, the gateway and the user interface for configuring the sink node. The global part consists of the server or host where the sink sends the data to, and the terminals or endpoints, where users can access this data. Software allows the user monitoring the network throughout graphical representations, performance indicators, alarms, etc.

Wireless sensor networks enable developing **ad-hoc** networks without pre-established physical infrastructure or central administration [4]. The ad-hoc expression refers to a network where there is no central node, but where all devices are equal. Ad-hoc is one of the easiest ways to create a network; due to they are a type of network composed by a group of mobile nodes forming a temporary network without the aid of any external infrastructure. For this to be put into practice, it is required that nodes collaborate with each other to achieve a common goal: that any packet reaches its destination even if the sink is not accessible directly from the source node. Regarding this goal, the routing protocol is responsible for discovering routes among nodes to make all of them able to both receive and transmit data to the sink.

In most cases, WSNs have the ability to self-restore, that is, if a node fails, the network find new ways to route data packets. Therefore, the network will continue operating as a whole, even if individual nodes deplete their battery and stop working. Self-diagnostic capabilities, self-configuration, self-organization, self-restoration and self-repair, are properties that have been developed for these networks to allow them to be unattended (to work without human intervention).

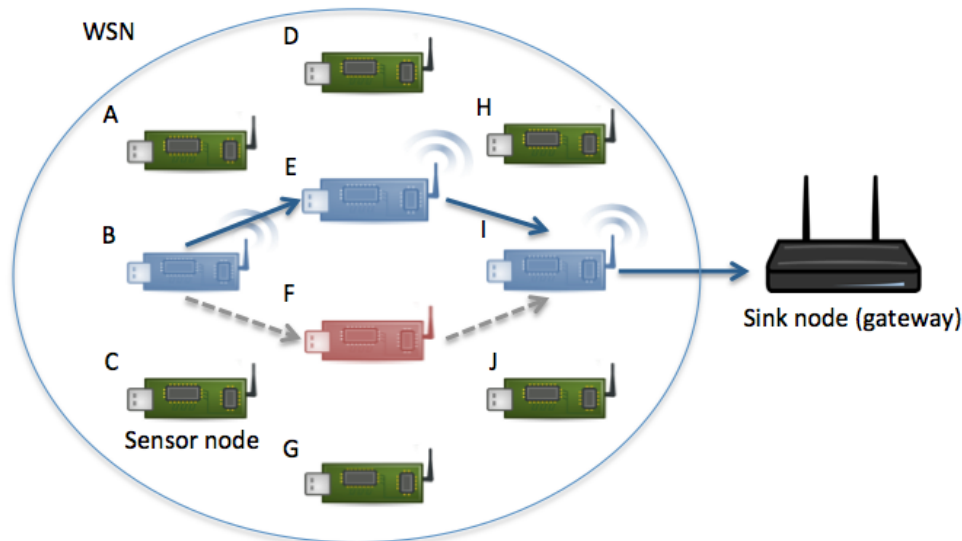


Figure 2: Rerouting after node failure

2.3 Architecture and components

A Wireless sensor network is composed of sensor nodes, gateways and software. This subsection depicts these components and the architecture they compose.

A **sensor** is a device capable of detecting physical or chemical magnitudes, e.g. light, sound, temperature, or pressure, and converting them into electrical signals to be processed in order to get the measures data. Sensors are typically small, battery-powered and low cost devices. A **sensor node**, instead, is the basic unit of a sensor network. It has to provide some computation or processing, wireless communication with the rest of sensor nodes in the network, and sensing functionalities. Usually sensor nodes are not used individually, but are part of a larger and more complex system [5].



Figure 3: Libelium waspmote¹

Once a sensor node takes events or measurement, the data is converted from physical to digital in the node itself. Then, this data is transmitted outside the network, throughout other sensor nodes if needed (when there is no direct communication between the

¹ Retrieved April 20, 2015 from http://www.libelium.com/uploads/2013/02/waspmote_pro-400px.jpg

source node and the sink), via a gateway element to a base station. At the base station, information can be stored and treated temporarily. Also, it can end up on a server with more capacity, allowing composing a historical sequence or making further data analysis.

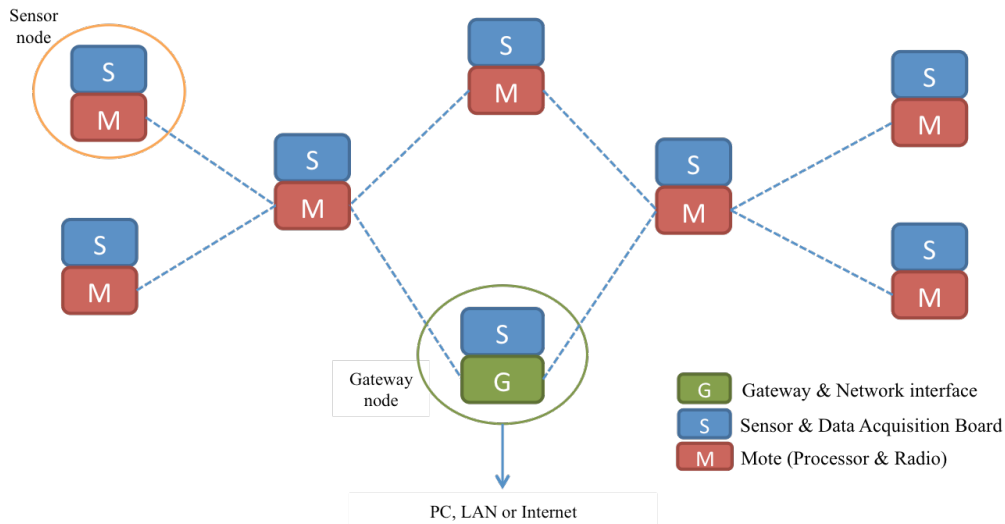


Figure 4: WSN architecture (based on [6])

The components of a WSN are detailed below.

a) Sensor node

As explained above, the sensor node is the basic unit of sensor network. It consists of a mote and a sensor board. The mote is the entity composed of a processor and radio devices, and the sensor board is a data acquisition board connected to the mote through an expansion connector that includes a set of sensors. Some sensor node models include the sensors in the mote itself and that is why in some bibliography sensor nodes and motes are considered to be the same. Examples of magnitudes measured by a sensor are: temperature, humidity, pressure or light.

Below it is shown the general architecture of a sensor node. It consists of four key components: processor (or **controller**) and memory unit, detection unit (or **sensors**), **transceiver**, and **power supply**. This basic design could be extended depending on the need for specialized application or hardware, including modules such as a power generator, a positioning system or some kind of actuators. Sensor nodes are expected to be low cost and small, which implies having high hardware limitations.

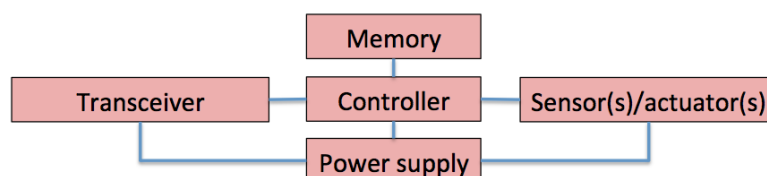


Figure 5: Sensor node architecture

The components of a sensor node are explained below.

- **Controller:** This component includes a microprocessor and memory unit that provide computational and storage logic. Some of its main functionalities are processing and data handling, temporary storage, encryption, forward error correction (FEC), digital modulation and transmission. The computational and storage requirements in a WSN vary depending on the application and can range from the use of an 8-bit microcontroller up to 64 bits. Storage requirements can also range from 0.01 to 2 gigabytes (GB) [7]. In addition to the microcontroller memory, some models include additional external ones such as flash memories. Hence, the processor and memory capabilities of a sensor node are significantly low. This implies that the computational cost of the algorithms should be minimized as much as possible.

Currently two of the most widely used low consumption processors in WSN developments are the ones listed below:

- **ARM7:** Reduced Instruction Set Computing (RISC) microprocessor family with a versatile processor designed for mobile devices and other low power electronics. This processor is capable of up to 130 million instructions per second (MIPS) on a typical 0.13 μ m process [8].

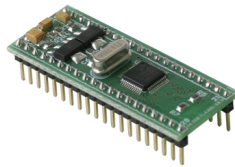


Figure 6: ARM 7 microcontroller²

- **Atmel AVR:** AVR are a family of RISC microcontrollers from Atmel. The AVR is a Harvard architecture CPU with 32 8-bit registers. Some instructions operate only on a subset of these records [9].

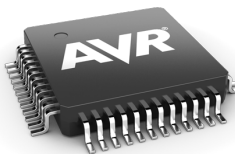


Figure 7: Atmel AVR microcontroller³

Sensors: A sensor is a hardware device that measures a physical or chemical quantity and converts it into a signal. That is, it converts physical phenomena like sound, pressure or light, into electrical signals. It behaves as a nexus between the environment and the sensor network. When the electrical signal is generated, it is sent to the microcontroller in order to be processed. There is a wide range of sensor types:

² Retrieved April 26, 2015, from http://skpang.co.uk/catalog/images/olimex/Philips_LPC/lpc-h40.jpg

³ Retrieved April 26, 2015, from <http://www.atmel.com/products/microcontrollers/avr/>

temperature, pressure, sound, humidity, smoke, accelerometers, magnetic, etc. Sensors can be classified into two big categories:

- **Passive sensors:** Passive sensors sense data without impacting the environment by active probing. They are self-powered (energy is just needed to amplify their analog signal) [10]. There are two types of passive sensors:
 - Omnidirectional sensors: they have no notion of direction involved in their measurements [11].
 - Narrow-beam sensors: Narrow-beam sensors have a well-defined notion of direction of measurement, similar to a camera. That is, they should be properly disposed in order to measure the desired event or phenomena [11].
- **Active sensors:** Active sensors sense probing the environment actively and they must be continuously connected to a power source. The radar is a typical example of active sensor.

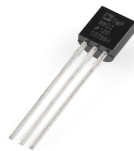


Figure 8: Temperature sensor⁴

- **Actuators:** An actuator is a device that converts electrical control signals to physical action, and constitutes the mechanism by which an agent acts upon the physical environment [12].



Figure 9: Micro motor reducer⁵

- **Transceiver:** A transceiver is a device comprising a transmitter and a receiver. In half-duplex mode, an electronic switch allows the transmitter and receiver to be connected to the same antenna, and prevents the transmitter output from damaging the receiver. With a transceiver of this kind, it is impossible to receive signals while transmitting. Full-duplex transceivers allow reception of signals during transmission periods and require that the transmitter and receiver operate on different frequencies in order to avoid interferences. The transmitted signal is called the uplink, and the received signal is called the downlink [13].

⁴ Retrieved April 26, 2015, from <https://cdn.sparkfun.com/assets/parts/4/1/8/8/10988-01.jpg>

⁵ Retrieved April 26, 2015, from <http://fadisel.com/imgs/c-6064.jpg>

The transceiver is the most energy-consuming component of a sensor node, so there is a need for low-cost and energy-efficient radios.

Regarding the transceiver, there are defined the following node states:

- Transmit: Send data packets to the network.
 - Receive: Receive data packet from the network. A transceiver in this state could also receive packets destined to a different node (overhead).
 - Idle: Waiting for packet reception. Energy consumption is reduced due to some hardware functions are switched off.
 - Sleep: Most of the hardware functionalities are switched off in this state. The transceiver is not ready to immediately start receiving packets. Some kind of wake-up mechanism has to be implemented in order to notify the receiver when to start listening.
- **Power supply:** An appropriate power supply must be able to feed the node for months, or even years, depending on the system requirements. Unattended WSNs typically incorporate a supply via a separate power system (usually batteries), combined with any charging source (e.g solar cells) or auxiliary power. These recharging strategies are needed where it is not possible to manually replace the nodes batteries, due to sensor nodes are normally placed in remote and difficult-to-access areas



Figure 10: Node solar panel⁶

Considering the limited lifetime of the device, it is needed to provide efficient power management (energy efficiency). The limited capacity of this unit requires energy-efficient operation for the tasks of each component. All sensor node components consume energy, but not at the same levels; the most energy consuming state is the data transmission, being lower in the processing and data gathering of the sensors.

b) Gateway

While motes sense any physical magnitudes of their surroundings and relay the information received from neighbor nodes to other nodes in transit to the gateway, the **gateway** collects all the information received from the motes and stores it (for example,

⁶ Retrieved April 27, 2015, from <http://www.voltaicsystems.com/6-watt-kit>

in a database), and makes this information available usually via a wireless network [14]. The gateway also acts as the network coordinator in charge of node authentication and message buffering, and provides the interface between the sensor nodes and the network infrastructure.

A gateway must have enough computing power to be able to run a database, perform local calculation and communicate with an existing network, but should be low power enough to run autonomously in the field [14]. There can be multiple gateways in a WSN, each communicating on a different, non-overlapping software-selectable wireless channel. Gateways main tasks are: transfer WSN measurements to remote users, transmit user commands and code updates to one or more of the network's motes, and alert the network administrators of potential network faults [15].



Figure 11: Wireless sensor network (WSN) gateway WSDA? Base⁷

c) Software

Most of WSNs are programmed based on the event-based model. **Event based**, or event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions sensor outputs, or messages from other programs/threads [16]. Event based programming is useful for WSN applications due to they are centered on performing certain actions in response to an input (motes packet transmissions). Also, the asynchronous mode of communication in WSNs, and faulty nature of sensor nodes makes this paradigm the most elected choice.

The **operating Systems** (OS) for wireless sensor network nodes is typically less complex than general-purpose operating systems. The main tasks performed by a WSN OS are to control and protect the access to resources by managing the nodes allocation to different users, and support for concurrent execution of processes. Some of the most used WSN operating systems are listed below [17].

- **TinyOS** is maybe the first operating system specifically designed for wireless sensor networks. TinyOS is written in the nesC programming language.
- **LiteOS** is an open source UNIX-like abstraction and support for the C programming language.

⁷ Retrieved April 27, 2015, from <http://orcom.com.ua/en/cat/19/9212/>

- **Contiki** is an open source OS developed for the field of the Internet of Things. It uses a simple programming style in C while providing advances such as 6LoWPAN and Protothreads.
- **RIOT** is an open source microkernel OS that provides multithreading with standard API and allows for development in C/C++. RIOT supports common Internet of Things (IoT) protocols such as 6LoWPAN, IPv6, RPL, TCP, and UDP.
- **ERIKA Enterprise** is an open-source and royalty-free OSEK/VDX Kernel offering BCC1, BCC2, ECC1, ECC2, multicore, memory protection and kernel fixed priority-adopting C programming language.

Software in WSNs can also refer to those applications that allow the user or administrator to configure the nodes and filter, manage and visualize the data.

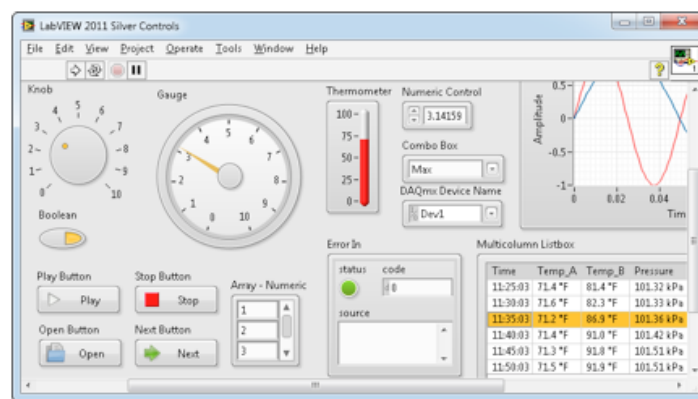


Figure 12: NI LabVIEW 2011 Silver Controls⁸

WSN Simulators

Running real experiments on a WSN testbed is costly (in terms of money and time), and it is also very difficult to isolate one single parameter to be analyzed. Hence, simulating WSNs is almost imperative to design and deploy them.

When it comes to simulating, two concepts arise: simulator and emulator. A **Simulator** is used for developing and testing WSN protocols. Simulations are very low cost and can be executed in relatively short periods of time. An **emulator**, however, is a tool implemented in real nodes that performs the simulation by means of firmware, hardware, and software in some cases [22]. The table below summarizes the most relevant simulators and emulators for wireless sensor networks.

⁸ Retrieved April 27, 2015, from <http://www.ni.com/white-paper/12892/es/>

Table 1: WSN simulators and simulators (based on [22])

Name	Kind	Event or Driven	GUI	Open Source	WSN specific
NS-2	Simulator	Discrete-Event	No	Yes	No
TOSSIM	Emulator	Discrete-Event	Yes	Yes	Yes
EmStar	Emulator	Trace-Driven	Yes	Yes	Yes
OMNeT++	Simulator	Discrete-Event	Yes	Noncommercial license and commercial license	No
J-Sim	Simulator	Discrete-Event	Yes	Yes	No
ATEMU	Emulator	Discrete-Event	Yes	Yes	Yes
Avrora	Simulator	Discrete-Event	No	Yes	Yes

2.4 Challenges

Different systems (ZigBee, 802.15.4, Bluetooth, or even proprietary radio solutions) have been considered for transmitting data in common Machine-to-Machine (M2M) scenarios. However, none of those systems has prevailed because of the current diversity and complexity of applications and environments [18].

Regarding WSNs, **power consumption** is perhaps the central design consideration whether they are powered using batteries or energy harvesters. Several essential issues are key to develop low-power wireless sensor applications such as efficiently harvesting, converting, and storing the energy as well as using available energy in the most efficient way, while keeping a proper performance (range, data rate, latency, and/or standards compliance) [19]. Today this is becoming possible because of the development of ultra-low-power transceiver radio chip working in combination with a microcontroller (MCU), which manages the transceiver; switching it on, making it listen, transmit, wait for or receive acknowledge signals, or re-transmit, all in accordance to the communications protocol being used [19].

Apart from energy consumption, most envisioned sensor network applications also encounter the following main challenges:

- **Architecture:** Ad hoc deployment requires that the system identifies and copes with the resulting distribution and connectivity of nodes [20]. There is no unified system and networking architecture that is stable and mature enough to build different applications on top.

- **Dynamic environmental conditions:** It requires the system to adapt over time to changing connectivity and system stimuli [20].
- **Unattended operation:** Configuration and reconfiguration must be automatic (self-configuration) [20].
- **Hardware cost:** The current cost of each individual sensor unit is still very high. Commercially available platforms cost in the order of €70 per unit with temperature, humidity and light sensors when bought in large quantities [19]. Capable sensors able to track human mobility inside buildings are costing around €200 per unit [19].
- **Wireless Connectivity:** Wireless communication in indoor environments is still quite unpredictable using low-power consumption RF transceivers, in particular in clutter environments common inside buildings, with many interfering electromagnetic fields, such as the one produced by elevators, machinery and computers, among others [19].
- **Programmability:** Some form of network re-programmability is desirable; doing so in energy and communication conservative form remains a challenge [19].
- **Security:** From the system point of view, it is critical that the information provided by the nodes be authenticated and the integrity verified, since this information provides the feedback loop to expensive equipment controlling power consumption [19]. From the users' point of view, it is also critical that this information cannot be easily spoofed and remains protected in the back end processor, since it may affect the privacy of users [19].

In conclusion, The technological obstacles such as the incomplete de facto standardization of a protocol, and the energy constraints the application of wireless sensors are tremendous and make them a fascinating area with great potential. It is important to note that the impact of this area on the world can rival the impact that the Internet has had in the past.

2.5 Applications

WSNs are capable of supporting a lot of very different real-world applications, but they are also a challenging research and engineering problem because of this flexibility. Some of the applications provided by Wireless Sensor Networks are mentioned below [21]:

- **Area monitoring:** Monitor some physical or chemical phenomena in a certain area or region.
- **Earth monitoring:** Earth science research such as sensing volcanoes, tectonic plates, jungles, etc.
- **Air quality and pollution monitoring:** Frequently measurements of air composition in order to safeguard people and the environment.

- **Forest fire detection:** Fast fire alert mechanisms can be deployed with WSNs.
- **Landslide detection:** Detect slight movements of soil and other phenomena that could become avalanche or similar disasters.
- **Natural disaster prevention:** Prevent natural disasters like floods. This kind of network could also be used to deploy fast ways of communicating in damaged scenarios.
- **Medical care:** Real-time patient monitoring, home monitoring for chronic and elderly patients, and collection of long-term databases of clinical data are examples of features of medical care wireless sensor networks.
- **Industrial monitoring:** Based on the data gathered from the nodes, the processes in industry can be optimized.
- **Agriculture:** Using a wireless network frees the farmer from the maintenance of wiring in a difficult environment. Some examples of systems that can be monitored by a WSN are accurate agriculture, irrigation management, or green house.
- **Smart home monitoring:** Biometric authentication for home entrance, energy consumption saving mechanisms, or remote blinds actions can be achieved by deploying wireless sensor network at home.

3. WIRELESS TECHNOLOGIES FOR WSN

There are two categories of communication technologies for Machine-to-Machine (M2M) applications: Wireless Sensor Networks (WSNs), for interconnecting spatially distributed autonomous sensor nodes over a certain region; and cellular networks, for isolated nodes located on cells providing coverage by a fixed transceiver (base station) [1]. Cellular networks could be also used to connect the gateway of a WSN to the Internet.

Regarding WSNs, several systems (ZigBee, 802.15.4, Bluetooth, or even proprietary radio solutions) have been considered for transmitting data among the nodes. However, due to the diversity and complexity of the applications and scenarios none of those systems has prevailed [1]. In fact, unlike WiFi standards, which have been established as de facto, there is not yet a common standard entirely focused on WSNs.

3.1 MAC layer impact on energy

Medium Access Control (MAC) directly controls the transceiver operation, which is the most energy-consuming component of a sensor node. Hence, apart from the transceiver design, the MAC layer is the most important part of the protocol stack when it comes to energy saving.

By identifying the possible states of a node, MAC protocols can intend to reduce the overall energy consumption in order to avoid wasting more energy than the needed one. For instance, if it is known that a node will not receive packets in a period of time, this node could be sleeping and consume just the enough energy to remain operative. The main sources of energy waste should be identified in order to try to reduce them [23]:

- **Idle listening:** Listening to the channel when there is nothing to receive. Identified as the major energy waste in WSNs (low traffic loads).
- **Collisions:** When more than one message is simultaneously transmitted in the same channel. Energy spent receiving and transmitting without any profit.
- **Overhearing:** Receiving messages for another destination. Energy spent receiving without any profit.
- **Overhead:** Headers and other kind of overhead.

As mentioned above, idle listening is the major energy waste source. Thus, making the node sleep as much as possible, that is, switching the radio off (no packets could be received or sent), could be one solution. In sleep mode the energy consumed is much lower than in transmit, receive or sampling modes. In order to make a node know if it is going to receive something and then whether it has be active or not, the common solution is to put the transceiver into sleep mode periodically (what is known as “duty cycle”). Then, a MAC protocol that takes this into account will notify the node when and how to put itself active.

In this section three of the most commonly used standards for WSN are depicted: IEEE 802.15.4, IEEE 802.ah and Bluetooth Low Energy (BLE). Also, a summary comparing the most relevant WSN standards can be found at **3.5 Other WSN technologies**.

3.2 IEEE 802.15.4

IEEE 802.15.4 is a standard that specifies the physical and link layer of (Low-Rate Wireless Personal Area Networks) LR-WPANs. IEEE 802.15.4 is specially designed for use in low-cost devices with low data rate and low power consumption [24].

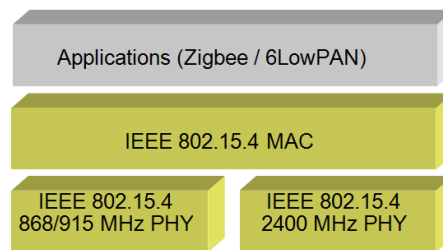


Figure 13: IEEE 802.15.4 Stack architecture [24]

a) Physical layer

A radio frequency (RF) transceiver and the protocol stack constitute the physical layer of this standard. Its frequency bands correspond to the free frequency bands available worldwide [24]. The **Table 2** summarizes this information:

Table 2: Frequency bands of IEEE 802.15.4 (based on [24])

Frequency (MHz)	Available Channels	Data Rate (kbps)	Available Regions
868 - 868.6	1	20	Europe
902 - 928	10 (release 2003) 20 (release 2006)	30	USA
2400	16	250	Europe

Figure 14 shows the common packet structure of IEEE 802.15.4. In these packets, within the 127 bytes of the PHY Service Data Unit (PSDU), 25 bytes are used for physical layer headers, and the remaining 102 bytes are available for network level and application level [24].

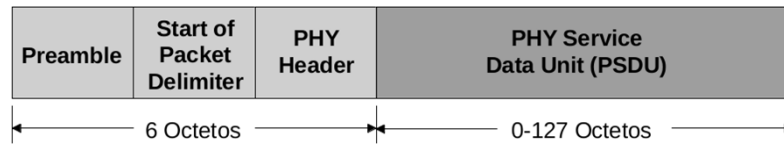


Figure 14: IEEE 802.15.4 Packet structure [24]

b) MAC layer

Regarding the Medium Access Control (MAC) layer, IEEE 802.15.4 uses a protocol based on the CSMA/CA algorithm, which requires listening to the channel before actually transmitting in order to reduce packet collisions probability. Two operational modes are designed, which correspond to two different channel access mechanisms [24]:

- **Non beacon-enabled mode:** Unslotted CSMA/CA protocol is used to access the channel and transmit using units of time called backoff periods. The procedure is the following:
 - Each node delays any activities for a random number of backoff periods.
 - Then, channel sensing is performed for one unit of time.
 - If the channel is free: The transmission is started.
 - If the channel is busy: The node enters again in backoff state.
 - When the maximum number of times a node can try to sense the channel is reached, the transmission is aborted.
- **Beacon-enabled mode:** In this mode, the access to the channel is managed through a superframe. This superframe begins with a beacon that is transmitted by the WPAN coordinator. The superframe may contain an inactive part, allowing nodes to go sleep. The active part is divided into two parts: the Contention Access Period (CAP) and the Contention Free Period (CFP), composed of Guaranteed Time Slots (GTSs), that can be allocated by the sink to specific nodes. The use of GTSs is optional. The figure below shows the superframe structure.

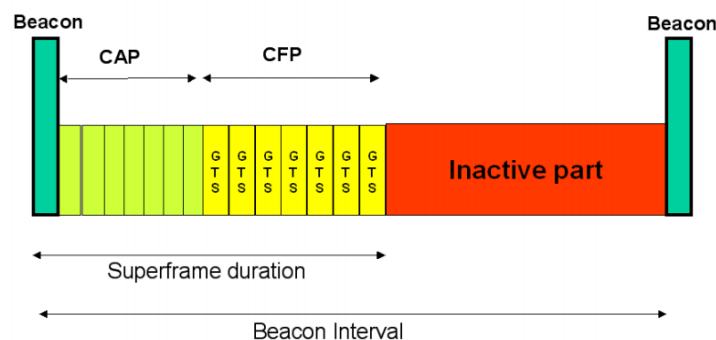


Figure 15: IEEE 802.15.4 Superframe structure [24]

c) Upper layers

IEEE 802.15.4 is the protocol with the simplest. In fact, other protocols such as ZigBee or 6LoWPAN use 802.15.4 MAC as subjacent protocol layer. The physical and Mac layers are the same at all protocols; being the data transmission method used the difference among them [24]. A comparison according the TCP/IP model among the protocols using 802.15.4 as a subjacent protocol layer is shown in **table 3**.

Table 3: Classification of different protocols in the TCP/IP model (based on [25])

TCP/IP Layer	802.15.4 MAC	6LoWPAN	ZigBee	RF4CE
Application	User defined	User defined	Standardized and user defined ZigBee profiles	Public or proprietary application profile
Transport	-	e.g. UDP	-	-
Network	-	IPv6 and 6LoWPan	ZigBee NWK layer	ZigBee RF4CE NWK layer
Data link	802.15.4 MAC			
Physical	802.15.4 PHY			

3.3 IEEE 802.11ah

The IEEE 802.11ah standardization Task Group (TGah)⁹ was created in 2010 and aims to create a WLAN standard for the PHY and MAC layers able to operate at sub 1 GHz frequency bands [24]. Various scenarios have been proposed in which this new protocol could be applied. Some examples of these scenarios would be: smart cities, agriculture monitoring, medical care, etc.

A WLAN protocol for operating below 1 GHz would involve the benefits of low frequency transmission such as wider wireless coverage and lower loss rate. However, it must be noted that there is a lack of interoperability between devices in this frequency range, because of, in most of the cases, the non-shared manufacturers' developments [24].

⁹ IEEE 802.11ah Task Group - http://www.ieee802.org/11/Reports/tgah_update.htm

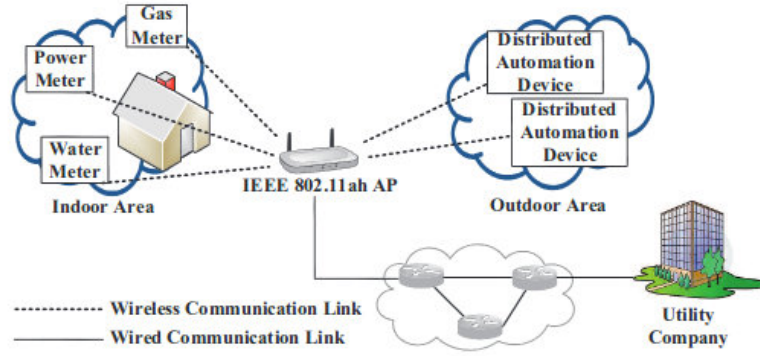


Figure 16: Adopted IEEE 802.11ah use case: smart grid¹⁰

a) Main technological features

The **requirements** defined by IEEE 802.11ah to support M2M communications are as follows [18]:

- Up to 8,191 devices associated with an access point (AP) through a hierarchical identifier structure.
- Carrier frequencies of approximately 900 MHz (license-exempt) that are less congested and guarantee a long range.
- Transmission range up to 1 km in outdoor areas.
- Data rates of at least 100 kbps.
- One-hop network topologies.
- Short and infrequent data transmissions (data packet size approximately 100 bytes and packet inter-arrival time greater than 30 s.).
- Very low energy consumption by adopting power saving strategies. Cost-effective solution for network device manufacturers.

In order to satisfy these requirements, IEEE 802.11ah provides with new PHY and MAC layers including several modifications with respect to consolidated IEEE standards for supporting the special constraints of M2M communications. The IEEE 802.11ah PHY layer can be considered a sub-1GHz version of the PHY layer on the IEEE 802.11ac. Similarly, the IEEE 802.11ah MAC layer incorporates most of the main IEEE 802.11 characteristics, adding some novel power management mechanisms [24].

b) Physical layer

IEEE 802.11ah operates over a set of unlicensed sub-1GHz radio bands that depend on country regulations. For example, the targeted frequency bands are 863-868 MHz in Europe, 902-928 MHz in the US, and 916.5-927.5 MHz in Japan. Regarding channel

¹⁰ Retrieved 18 May 2015, from <http://www.cnx-software.com/2014/02/21/802-11ah-wi-fi-900-mhz-to-provide-low-power-long-range-connectivity-for-the-internet-of-things/>

bandwidths, 1 MHz and 2 MHz have been widely adopted, although in some countries broader configurations using 4, 8, and 16 MHz are also allowed [24].

Physical transmissions are based on Orthogonal Frequency Division Multiplexing (OFDM) waveforms consisting of 32 or 64 tones/sub-carriers with 31.25 kHz spacing. The supported modulations include BPSK, QPSK, and from 16 to 256-QAM. Technologies introduced in IEEE 802.11ac such as single-user beam forming, Multi Input Multi Output (MIMO) and downlink multi-user MIMO are also adopted in IEEE 802.11ah [24].

c) MAC layer

The MAC layer is designed to maximize the number of stations supported by the network while ensuring minimum energy consumption, and it is expected to have a deeper impact than the physical layer on energy saving. Three types of stations, each with different procedures and time periods to access the common channel are defined [1]:

- **Traffic indication map (TIM) stations**

This kind of station needs to listen to AP beacons to send or receive data. Their data transmissions must be performed within a restricted access window (RAW) period with three differentiated segments (multicast, downlink, and uplink). Stations with a high traffic load should use this procedure to access the channel because it combines periodic data transmission segments with energy efficiency mechanisms.

- **Non-TIM Stations**

Non-TIM stations do not need to listen to any beacons to transmit data. During the association process, non-TIM devices directly negotiate with the AP to obtain a transmission time allocated in a periodic restricted access window (PRAW). The following transmissions can be either periodically defined or renegotiated, depending on the requirements set by the station. Although non-TIM stations can transmit data periodically, it is advisable to deploy TIM stations for high-volume data applications to achieve better management of channel resources and benefit from all the improvements developed by IEEE 802.11ah.

- **Unscheduled Stations**

These stations do not need to listen to any beacons, similar to non-TIM stations. Even inside any restricted access window, they can send a poll frame to the AP asking for immediate access to the channel. The response frame indicates an interval (outside both restricted access windows) during which unscheduled stations can access the channel. This procedure is meant for stations that want to sporadically join the network.

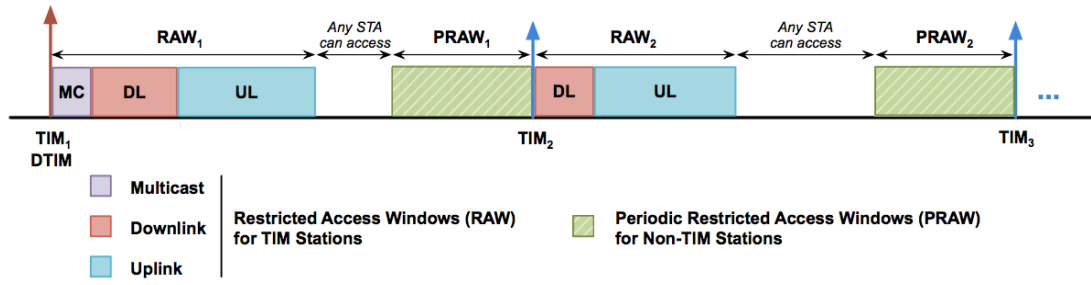


Figure 17: Distribution of channel access restricted windows (RAW and PRAW) among signaling beacons [1]

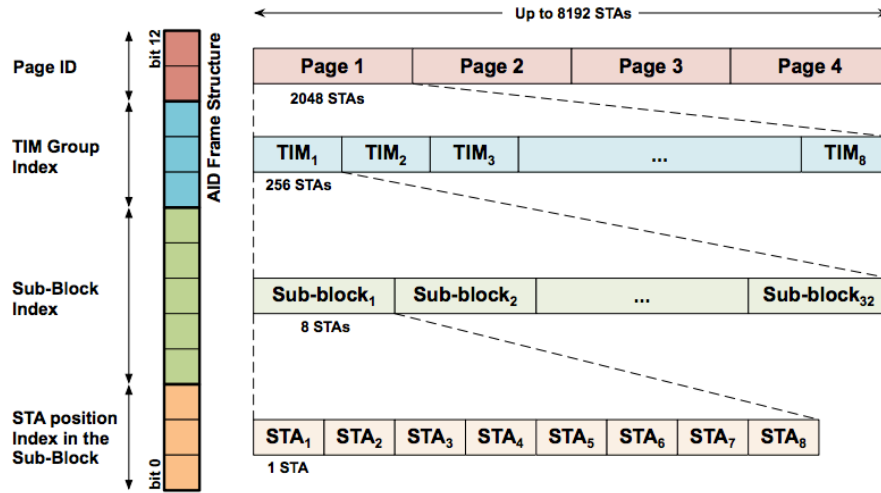


Figure 18: AID frame structure based on the hierarchical association of stations [1]

d) Energy savings

The possibility of keeping nodes in sleep mode for long periods of time without losing their association with the corresponding AP is one of the main features introduced by 802.11ah. These energy saving mechanisms would be essential in smart metering' environments, where transmission periods usually are very large (hours, days or weeks) and are clearly established.

Energy consumption can also be reduced when stations are not sleeping by deactivating the radio module during non-traffic periods. Moreover, a greater reduction of energy consumption is achieved due to IEEE 802.11ah restricts to a particular group of stations to simultaneously contend for the same channel in a specific period, which allows TIM stations from the same group to be in a sleep mode for the rest of the time [1].

3.4 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) is essentially a simplified version of Bluetooth, a wireless technology standard for exchanging data over short distances from fixed and mobile devices, and building personal area networks (PANs). BLE uses the same

physical layer in the ISM 2.4 GHz for easy interoperability with existing devices and allows data rates of up to 1 Mb/s at a maximum range of 10 meters [24].

With respect to the classic Bluetooth technology, BLE optimizes three basic areas of functionality: connection and discovery modes, number of packets transmitted during connections and size of individual packets [24]. These improvements make BLE to be very efficient when transmitting small amounts of data to other devices at very low latencies, becoming 15 times more efficient when compared with the classic Bluetooth.

In classical Bluetooth technology, when two devices want to communicate each other, they have to use the same channel (or frequency) simultaneously becoming synchronized. In order to get synchronized these devices must look for an appropriate channel to start the exchange of information. There are 32 channels defined in Bluetooth technology making the synchronization last up to two seconds, with the corresponding energy consumption [24]. Instead, BLE defines just 3 channels, making the synchronization process much faster.

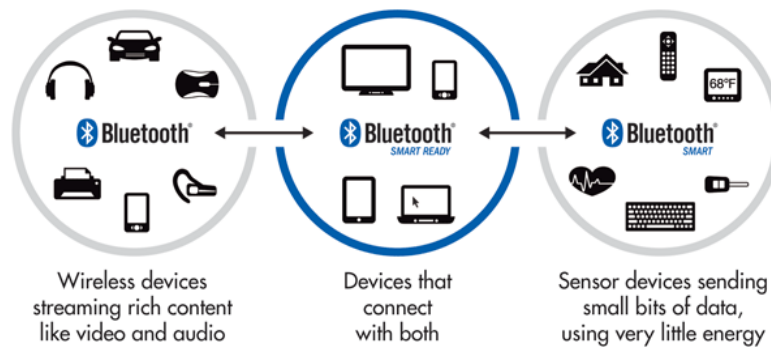


Figure 19: Bluetooth, Bluetooth Smart Ready and Bluetooth Smart¹¹

Another difference regarding the two mentioned protocols is that BLE uses a higher modulation index. The signal, then, occupies more bandwidth and channel filters do not have very demanding requirements. Because of this, the channels are doubly spaced (2 MHz) [24]. In addition, when a slave node does not have data to transmit, it does not have to maintain its radio module on. However, when this device has data to transmit, it can wake up and send it during the next suitable event of communication.

Bluetooth Low Energy does not support mesh networks, which is a clear disadvantage, compared with Zigbee, 6LoWPAN or Z-Wave.

¹¹ Retrieved 14 May, 2015, from <http://www.aislelabs.com/blog/2014/06/06/what-is-bluetooth-low-energy/>

3.5 Other WSN technologies

As mentioned above in this section, it is challenging to find patterns of proper comparison between the most common WSN technologies used nowadays. Nonetheless, the outcomes of some previous research in this area are shown in the tables below.

Table 4: Comparison of different technical features from different technologies (based on [18][26][27])

	IEEE 802.11b/g	Bluetooth	UWB-IR	IEEE 802.15.4	Zigbee
Frequency Band	ISM 2,4 GHz	ISM 2,4 GHz	3,1 GHz – 10,6 GHz	ISM 2,4 GHz, 915 MHz, 868 MHz	ISM 2,4 GHz
Spreading	DSSS	FHSS / TDD	Baseband	DSSS	DSSS
Modulation	BPSK, QPSK, CCK, OFDM	GFSK	Impulse radio, time-domain	O-QPSK	O-QPSK
Range Coverage	100 m.	10 m.	<5 m.	10 m.	10 - 75 m.
Data Rate	<54 Mbps	1 Mbps	20 kbps, 250 kbps, 10 Mbps	250 kbps	250 kbps
Transmission Power	High	High	Ultra high	Very low	Very low
Roaming	Yes	No	Yes	Yes	Yes
Maximum Number of Nodes	32 per AP	8 per Piconet	10 - 1000	<65536	<65536
Energy Consumption	Medium	Low	Ultra low	Very low	Very low
Complexity	Complex	Very complex	Simple transmitter, complex receiver	Simple	Simple
Security	WEP, WPA	64 o 128 bits	128 bits	NULL, 32, 64 or 128 bits	128 bits
Cost	High	Medium	Very low	Low	Low

Table 5: Specific comparison of features among Zigbee, Bluetooth and IEEE 802.11ah [18]

	Zigbee	Bluetooth	IEEE 802.11ah
Standard	IEEE 802.15.4	IEEE 802.15.1	IEEE 802.11ah
Frequency band	EU: 868 MHz NA: 915 MHz Global: 2.4 GHz	2.4 GHz	Sub-1GHz
Data rate	868 MHz band: 20 kb/s 915 MHz band: 40 kb/s; 2.4 GHz band: 250 kb/s	1 Mb/s	if BW = 1 MHz: 0.15–4 Mb/s; If BW = 2 MHz: 0.65–7.8 Mb/s
Typical range	2.4 GHz band: 10–100 m	10–30 m	100–1000m
TX power	1–100 mW	1–10 mW	<10 mW – <1 W (depending on the country's regulations)
Bandwidth per channel	868 MHz band: 0.3 MHz 915 MHz band: 0.6 MHz 2.4 GHz band: 2 MHz	1 MHz	1, 2, 4, 8, or 16 MHz
Modulation	BPSK (+ASK), OQPSK	GFSK	BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM
Transmission technique	DSSS	FHSS	OFDM
Topology	Multihop	Star	Single-hop
Battery operation	From months to years	From days to weeks	From months to years
Power saving mechanisms	Only in ZigBee RF4CE	Only in Bluetooth Low Energy (BLE)	Native
Packet length	≈ 100 bytes	From kbytes to Mbytes	≈ 100 bytes
Typical scenarios	Multihop networks with few nodes	Multimedia data exchange between nearby nodes	One-hop networks with many nodes

4. GOAT

4.1 Introduction

a) What is GOAT?

GOAT is a graphical Wireless Sensor Network analyzer running on Windows, Mac OS and Linux kind operative systems that allow designing WSNs and estimating its energy consumption. With GOAT, we provide a tool that allows WSN managers to compare the performance of different MAC and routing protocols in terms of power consumption.

As mentioned before, GOAT is a graphical analyzer. It displays the topology of the network, that is, nodes and links, and also different parameters of interest such as rates, number of nodes, map size, etc. All the mentioned graphical items are included in a Graphical User Interface (GUI) that enables interacting with the analyzer throughout a series of panels, buttons, combo boxes and checkboxes.

GOAT is aimed to be an open source project. To that end, it has been designed taking into account code modularity in order to promote future collaborations, making clear how the program is structured and how it can be expanded and improved. The main goal is that GOAT could be used in college and research fields, facilitating learning WSN aspects in the first case, and allowing deeper analysis on the second one.

b) Main features

GOAT implements several functions. The main ones are mentioned below:

- Create and modify Wireless Sensor Network topologies.
- Save and open topologies.
- Set the sending rate of each of the nodes of the WSN created.
- Set and modify the sink (or gateway) of the WSN.
- Add and delete one or all nodes.
- Display distance, power and coverage matrices.
- Open and save the WSN topology.
- Display coverage links.
- Establish and display routing links automatically depending on the physical and routing models selected.
- Display sending (λ), aggregated sending, receiving and overhearing rates of each node.
- Estimate the energy consumption of each node and its lifetime depending on the MAC model selected.
- Analyze the selected battery model impact on the WSN energy consumption.
- Estimate the proportion of the entered observation time a node stays in each of the possible states.
- Save simulation results.

c) System requirements

GOAT simulator can be executed on Windows, Mac OS and Linux kind operative systems. Just Java JDK 7.0 or above is required to be installed. Java 7 has the following system requirements¹²:

- Windows
 - Windows 8 (Desktop)
 - Windows 7
 - Windows Vista SP2
 - RAM: 128 MB; 64 MB for Windows XP (32-bit)
 - Disk space: 124 MB
- Mac OS X
 - Intel-based Mac running Mac OS X 10.7.3 (Lion) or later.
 - Administrator privileges for installation
- Linux
 - Oracle Linux 5.5+
 - Oracle Linux 6.x (32-bit), 6.x (64-bit)
 - Oracle Linux 7.x (64-bit)
 - Red Hat Enterprise Linux 5.5+, 6.x (32-bit), 6.x (64-bit)
 - Ubuntu Linux 10.04 and above

4.2. Architecture and design

a) Functional design

GOAT allows creating and modifying WSNs topologies through its GUI. Also, from the GUI, a user has the possibility to select different physical, battery, MAC, and routing models for modeling the desired scenario. In order to open stored topologies and save new ones, the GUI counts with two buttons for reading input topologies files in txt format and writing new ones.

It is needed to modify the code of the application to change some of the analyzer parameters. However, the code is structured in such way that the user can easily find the variables corresponding to the parameter he wants to modify. By compiling the code again, the new user's configuration can be executed.

The main functionality of GOAT is simulating and estimating the consumed energy by the network. To do so, the user can run the simulation from the GUI and analyze the results on the console included in the application.

¹² What are the system requirements for Java? - <http://java.com/en/download/help/sysreq.xml>

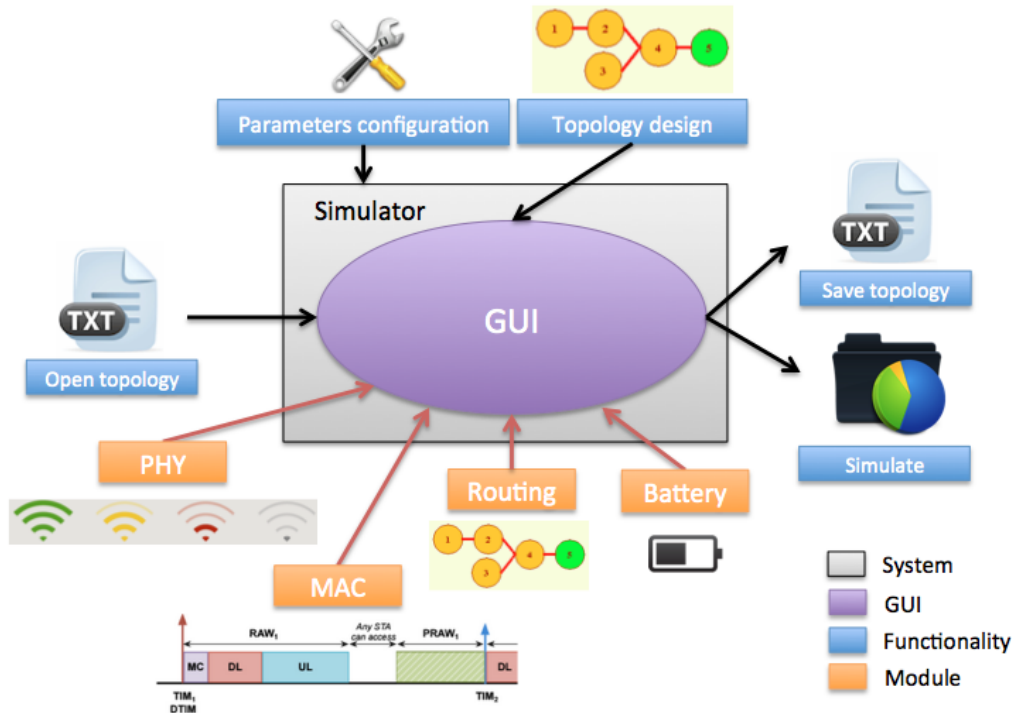


Figure 20: Functional design diagram

b) Technical design

GOAT simulator has been fully written in Java programming language and using Netbeans 8.0.2¹³ Integrated Development Environment (IDE).

The most relevant available **Java packages** and classes used are listed below:

- Java.awt¹⁴: Contains all of the classes for creating user interfaces and for painting graphics and images. Desktop - Open pdf.
- Java.io Provides for system input and output through data streams, serialization and the file system.
- Java.util: Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).
- Java.text: Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.
- Java.lang.math: The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- Javax.swing: Swing library is an official Java GUI toolkit released by Sun Microsystems. It is used to create Graphical user interfaces with Java. Swing is an advanced GUI toolkit. It has a rich set of widgets: from basic widgets like

¹³ NetBeans IDE 8.0.2 Information - <https://netbeans.org/community/releases/80/>

¹⁴ Package java.awt - <http://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html>

buttons, labels, scrollbars to advanced widgets like trees and tables. Some of the Swing package classes used in this project are listed below:

- JFrame
- JButton
- JLabel
- JComboBox
- JCheckBox
- JTextField

UML class diagram

The Unified Modeling Language (UML) class diagram purpose is to depict the classes within a model. In Object-oriented programming (OOP), classes have attributes (items), operations (methods and functions) and relationships with other classes. The **Figure 22** shows the GOAT's UML diagram. UML classes' items and methods can be found at annex **A1.UML class diagram**.

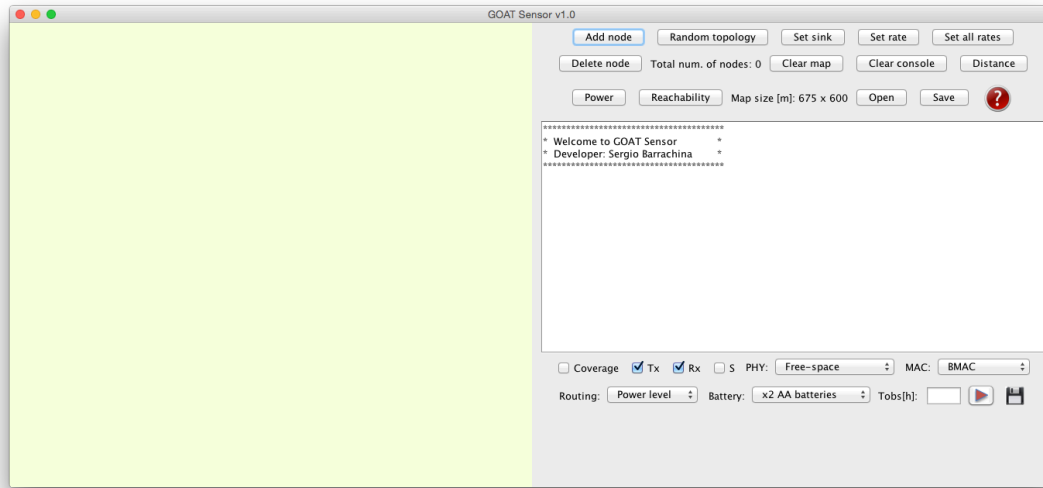


Figure 22: GOAT Graphical User Interface (GUI)

The ease of remembering is another concept that has been taken into account. Indeed, practically no remembering will be needed due to the simplicity of use, so we expect that an expert (e.g. wireless fields researchers), and novice users (e.g. undergraduate students) will get their desired WSN designs and simulation results in similar times. To achieve that, we have developed a minimalist design, with few text and buttons placed in a recognizable interface, matching the usual patterns, commands and metaphors (e.g. play icon for simulating) of the most common simulators. Also, monochromatic backgrounds are used in order to avoid distracting the user while keeping a nice environment.

Regarding this, GOAT counts with simple interfaces that help to maintain a low user error rate. As an example, when deleting a node, the user is only allowed to do that if a node is previously properly selected. Another example is the simulation process. When a user wants to run a simulation, the error rate has been decreased as minimum too; each user has to introduce and select few really understandable parameters such as observation time, physical model, MAC model, routing model, etc. That implies that any user will be able to easily get into the program. Also, if the user makes a mistake introducing wrong the asked parameters, GOAT will provide clear warnings notifying it.

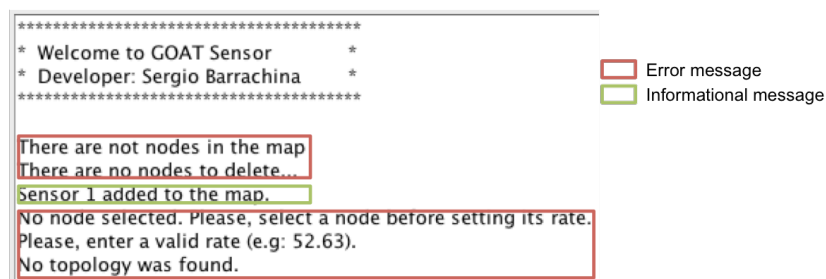


Figure 23: Informational and error messages displaying

Another aspect to take into account is the enjoyable/satisfaction perceived by the user. By applying User Centered Design (UCD), we have received some feedback and proposals from researchers and undergraduate students for improving the GUI in different ways. It is important to note that this user feedback, even not affecting the core code of the program, must be received and applied the sooner the better, in the first prototypes in order to avoid fixing more effort problems in the future. As an example, most of the users asked for buttons with icons instead of text and for a larger console. Then, based on these comments, we decided to hone the GUI with some icons and images and increase the size of the console.

We do not want to forget about the efficiency of task performance. Thus, we have tried to optimize the topology displaying in order to provide the user a faster experience. However, that it is not a priority of the service GOAT provides. Making the user to place the nodes in 2 minutes instead of 30 second until the full topology is created is not the point. Also, simulations are expected to be run, even in highly dense scenarios, in relatively short period of times. That is why, as said before, we prefer to focus on the user ease of learning and, overall, the exactitude of the presented simulation results.

b) Structure

The GUI is structured in two differentiated parts: The map and the components.

- **Map:** The WSN topology is displayed in this section. Nodes shapes, identifiers, links, rates, control graphics, etc. will compose the map.

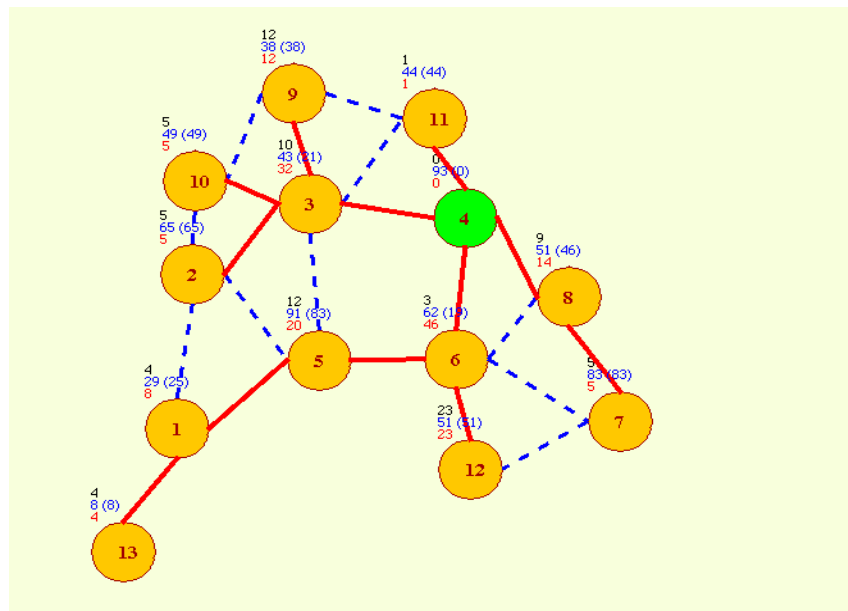


Figure 24: GOAT Map - WSN example design

- **Components:** The components section consists of the buttons, checkboxes, text fields, combo boxes, labels and also, the console. These components implement action listeners that allow the user to interact with the analyzer by adding or deleting nodes, selecting models, running a simulation, etc.

- **Console:** The console is a special component. It serves to show the results of the simulation and the events performed by the user such as deleting a node, adding a new one, open a topology file, etc. Thus, the main goal of the console is to inform the user and display the simulations results in an understandable manner. For convenience, the console implements an auto scroll that automatically shows the most recently generated messages. This way, GOAT avoids making the user to drag the scroll bar anytime he wants to check the last message.

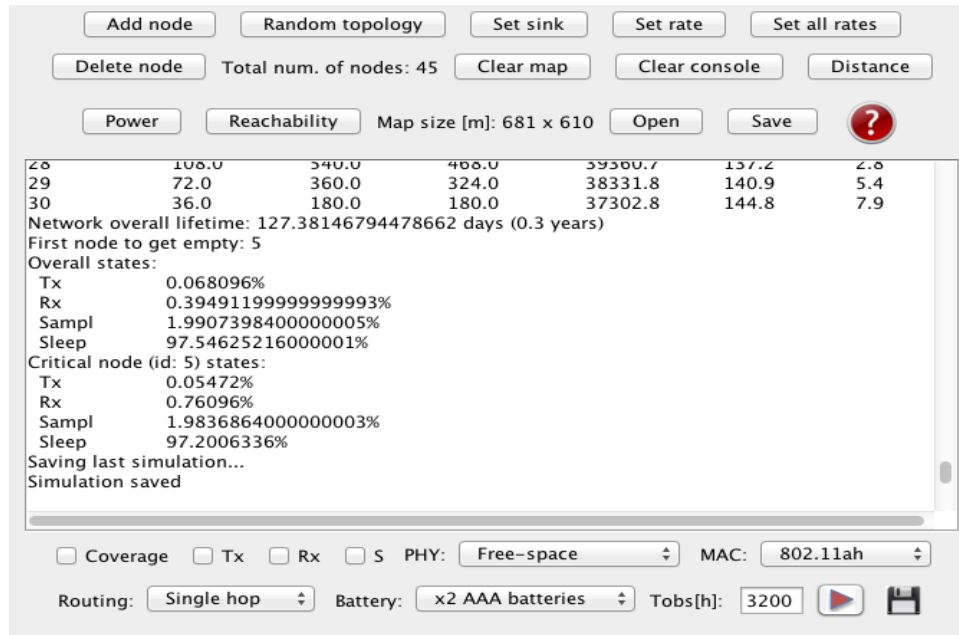


Figure 25: GOAT components

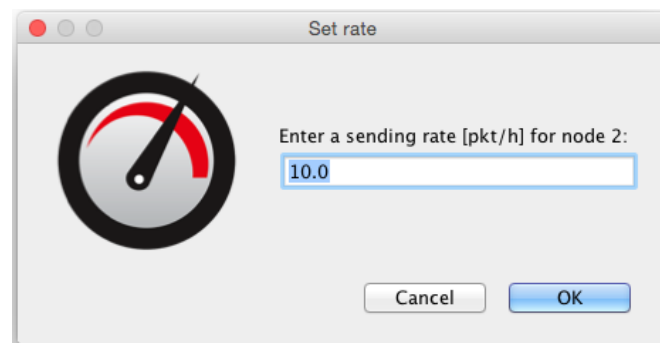


Figure 26: Sending rate (λ) pop-up window

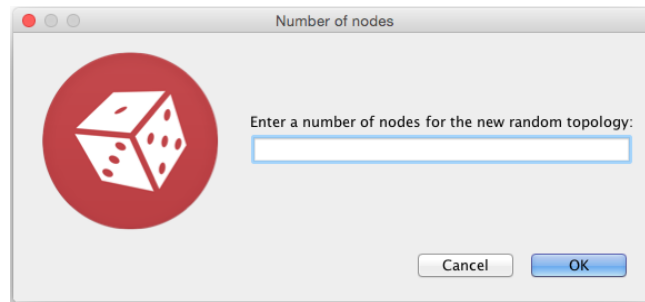







Figure 27: Random topology pop-up window

The GUI implements some **metaphors** that are listed in the table below.

Table 6: Metaphors

Concept	Description	Image
Simulate	Icon placed in the button for running a simulation.	
Speed ¹⁵	Icon placed in the pop-up window shown when “Set rate” or “Set all rates” buttons are clicked	
Help ¹⁶	Icon placed on the button for redirecting the user to the user’s manual.	
Random	Icon placed on the button for generating a new randomly distributed topology.	
Save	Icon placed on the button for saving the last simulation results.	

4.3 Topology creation

a) Create and modify topology

Topologies can be both created and modified in GOAT. For designing a new topology from the scratch, the user can add new nodes to the WSN manually by clicking “Add node”, generate a random topology by clicking “Random topology”, or open a topology txt file (which is covered in the next subsection).

¹⁵ Retrieved 18 May, from https://lh6.ggpht.com/0t6_XvWIL1aOf_9uF6sq3nXxhtluj1zPsa3vuS_BOrEMwdfhty9GDC82vbVSULMfI4o=w300

¹⁶ Retrieved 18 May, from http://online.lbcc.edu/pluginfile.php/34778/block_html/content/helpicon.png



Figure 28: *Add node* and *Random topology* buttons

Add node adds a single node to the network whether the network has already nodes or not. The sending rate of the new node will be set to the default rate configured in the code (see **4.6 Configuration**). If the user adds a node on a WSN with motes and a sink defined, it automatically will be included in the routing model, generating the links corresponding to the new topology.

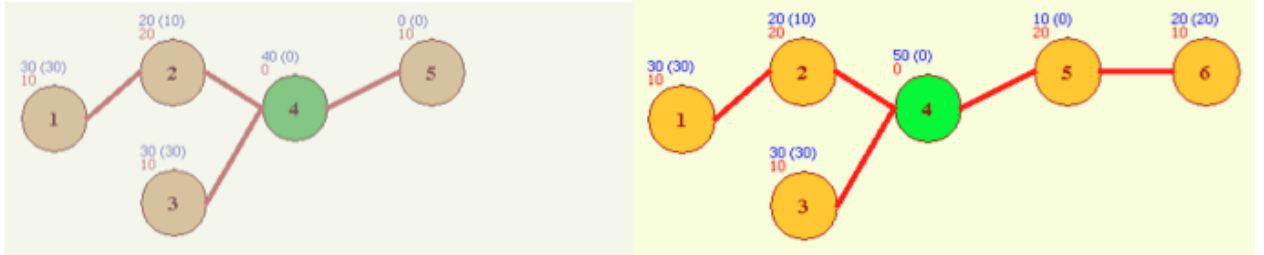


Figure 29: Rerouting after adding node 6 to the WSN

In the figure above, “add sensor” was clicked from the topology of the image on the left (with 5 nodes) resulting on the topology of the image on the right (with six nodes). To define the sink or gateway, the user has to select the node that is going to be **set as sink** and then click “Set sink”. As shown in the figure below, defining a sink will automatically generate the routing links corresponding to the topology and the selected physical and routing models on the GUI.

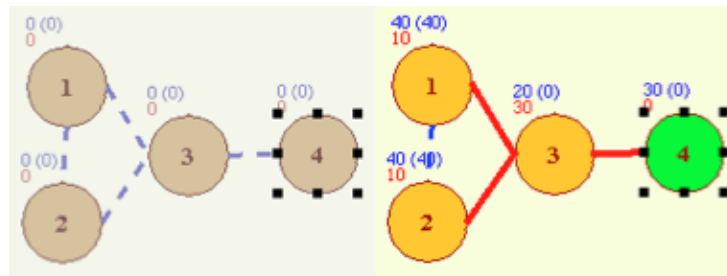


Figure 30: Routing after setting a sink node

Random topology generates a new randomly distributed topology with a number of nodes entered by the user. The algorithm used for achieving this, places each node following a uniform distribution over the map through the java class `Random.java`. Below it is shown a piece of code for generating the locations of the nodes to be added to the WSN.

```
Random rn = new Random();
double randomValueX = rn.nextDouble();
double randomValueY = rn.nextDouble();
Point p_sensor = new Point(width * randomValueX, height * randomValueY);
```

The size of the nodes will vary depending on the total number of nodes of the WSN in order to allow displaying properly the topology on the GUI's map. Four levels have been defined:

Table 7: Node size levels

Number of nodes	Node radius [m]
Until 15	25
From 16 to 45	15
From 46 to 70 nodes	10
More than 70 nodes	5

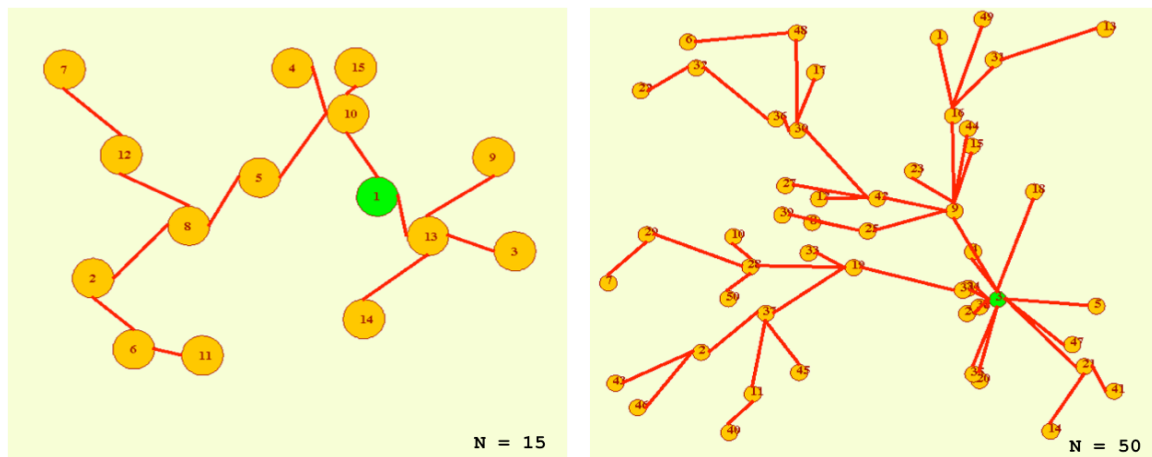


Figure 31: Size level 1 (left) and size level 3 (right) WSN examples

Set rate allows setting the sending rate of a selected node. The sending rate in this scenario stands for the number of packets of own data that a mote sends per hour. It is not permitted to set the rate of a sink node, due to it is suppose that a gateway will not send own data packets, but forward the sensor data to the global level. Similarly, **Set all rates** allows setting the sending rates of all the nodes at once.

Delete node removes a selected node from the WSN. In order to delete a node, the user has to select the node to delete first, and then click the button. Deleting the sink node is also permitted. When the sink is removed from the WSN, the links are no longer displayed. If one of the remaining nodes of the WSN is set as sink, then new links will be generated and displayed.

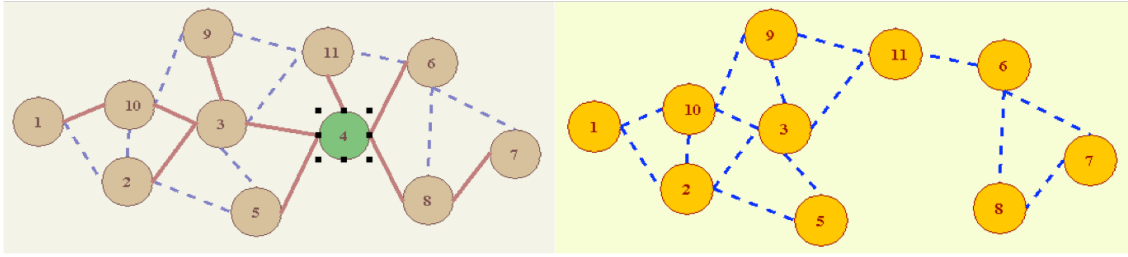


Figure 32: Topology after deleting the sink node

Adding a new sensor after deleting a node will create a node with an id not assigned previously. for instance, if a node is added to the topology on the right image above, its node id would be 12, not 4. **Clear map** is used for removing all the nodes of the WSN at once. It also resets the id assignment, starting from 1 on.

b) Open and save topologies

Besides enabling creating a topology from the scratch, GOAT allows the user to both save and open WSN topologies. To do so, two buttons are included in the GUI: “Open” and “Save”.



Figure 33: *Open* and *Save* buttons

When the user wants to save a topology already generated on the GUI, he has to click “Save”. Then, a window pops up asking for a name and the location of the file. This created file has to be txt format. The topology file format generated when the save process is finished consists of the following five columns.

- ID: Node identifier. It has no impact on the topology. The only thing to keep in mind is that two nodes cannot have the same identifier
- X: Abscissa of the node.
- Y: Ordinate of the node.
- S: Sending rate of the mote (own data packets sent per hour)
- K: Kind of the node. It determines whether a node is a mote or a sink. A maximum of one sink is permitted per WSN. There are, as mentioned, two possible values for K:
 - STA (or mote): 0
 - Sink (or gateway): 1

An example of a topology file is shown below.

ID	X	Y	S	K
1	50.0	50.0	10.0	0
2	125.0	100.0	10.0	0
3	200.0	125.0	10.0	0
4	275.0	100.0	10.0	0
5	350.0	130.0	10.0	0
6	425.0	80.0	0.0	1
7	500.0	100.0	10.0	0

Figure 34: GOAT topology file

Both opening and saving features display on the GUI console the opened and save topology file, respectively.

4.4 Graphical information displaying

GOAT implements several functionalities that allow the user to check information related with WSN being designed. These functionalities act on the console (matrices) and on the map (links and rates).



Figure 35: *Distance*, *Power*, and *Reachability* buttons

The **matrices** represent an amount regarding a pair of nodes. There are three type of matrices that can be displayed:

- **Distance:** The “Distance” button displays the nodes distance matrix. The user can check the distance between any pair of nodes in that matrix.
- **Power:** In order to check the power that a node receives from another one, the user can click the Power button. A matrix with all the power received by a node from the rest of them will be displayed.
- **Reachability:** The reachability button

All the mentioned matrices follow the same format. Let us suppose a power matrix P . Its component $p_{i,j}$ will represent the power received by the node i from the node j . In order to avoid misunderstandings, each matrix component where $i = j$ will be set to 0. An example of power matrix is shown below.

POWER MATRIX [dBm]						
1:	0	-78.3	-78.9	-84	-84.2	-87.4
2:	-78.3	0	-78.5	-81.3	-84.6	-86.8
3:	-78.9	-78.5	0	-77.5	-78.8	-83.4
4:	-84	-81.3	-77.5	0	-79.8	-80.6
5:	-84.2	-84.6	-78.8	-79.8	0	-78.5
6:	-87.4	-86.8	-83.4	-80.6	-78.5	0

Figure 36: Power matrix display at console

In the figure above we note that node 1 is receiving -78.3 dBm from node 2, -78.9 dBm from node 3, and so on.

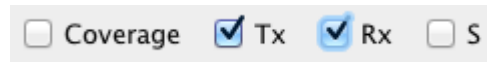


Figure 37: Graphical information displaying checkboxes

The **checkboxes** implemented on the GUI are listed above.

- **Coverage:** Displays the reachability between two nodes. That is, if a pair of nodes can hear each other, a blue dotted line will be painted between them.

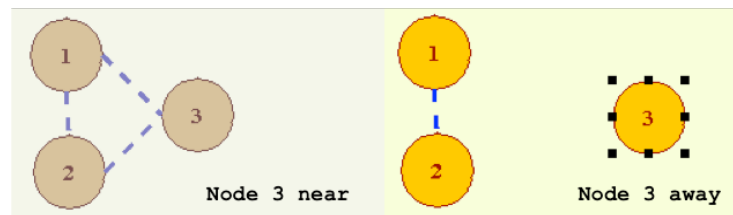


Figure 38: Coverage effect

- **Tx:** Aggregated sending rate. It displays the number of packets per hour that a node sends. These packets consist on both own information and forwarding packets.
- **Rx:** Listen rate. It displays the number of packets per hour that a node listens. The number inside the parenthesis indicates the number of packets per hour that are overhearing, that is, that are not going to be forwarded.
- **S:** Sending rate. It displays the number of own data packets per hour that a node transmits to the network.

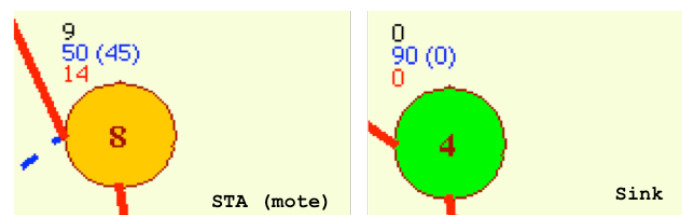


Figure 39: Sending, Reception, Overhearing and Transmission rates

In order to access the user manual, a button with a question image set as icon has been implemented. By clicking on the mentioned button, a web page will be loaded in the default browser displaying a document with all the functionalities and instructions to properly use GOAT.



Figure 40: Help button and user manual

4.5 Modules

Several physical, battery, MAC and routing modules have been considered in GOAT. In this subsection, the mentioned modules are depicted.

a) Physical models

The physical module implements different radio propagation models in order to know if a node can be reached from another one, that is, if a node is in the coverage area of another one. A radio propagation model is an empirical mathematical formulation for the characterization of radio wave propagation as a function of frequency, distance and other conditions [28]. We will use these models to estimate the path loss given in a link connecting two nodes, and determine whether they can be reached or not.

The physical models implemented in the first version of GOAT are the ones explained below:

- **Free space (Friis)**

The free space propagation model assumes the ideal propagation condition that there is only one clear line-of-sight (LOS) path between the transmitter and the receiver [29]. The power received at distance d can be calculated by the following equation:

$$P_R = P_T \frac{G_T G_R \lambda^2}{(4\pi d)^2} [W]$$

Where P_T is the transmitted signal power. G_T and G_R are the antenna gains of the transmitter and the receiver respectively. L is the system loss, and λ is the wavelength, which will vary depending on the used carrier frequency. The power received can be also given in dB:

$$P_R = P_T + G_T + G_R + 20 \log_{10} \left(\frac{\lambda}{4\pi d} \right) [dB]$$

The free space model basically represents the communication range as a circle around the transmitter. If a receiver is within the circle, it receives all packets. Otherwise, it loses all packets [29].

- **Log-normal fading**

Log-normal fading, also known as slow fading, arises when the coherence time of the channel is large relative to the delay constraint of the channel. In this regime, the amplitude and phase change imposed by the channel can be considered roughly constant over the period of use [30]. This phenomenon can be originated when a large obstruction such as a hill or building obscures the main signal path between the transmitter and the receiver.

In conclusion, log-normal fading is a well-established model for distance dependent average power attenuation. It can be modeled following the equations above.

$$P_r(d) = P_r(d_0) \left(\frac{d}{d_0} \right)^{-\gamma}, 2 \leq \gamma \leq 5$$

Where d_0 is the reference distance. Equivalently, the pass loss in dB can be calculated by the following equation.

$$L(d) = L(d_0) + 10\gamma \log \left(\frac{d}{d_0} \right)^{-\gamma}$$

A table of typical γ values in different scenarios is shown below.

Table 8: Typical log-normal values (based on [31])

Scenario	γ
Ground-wave reflection	4
Urban cellular radio	2.7 – 3.5
Shadowed cellular radio	3 – 5
In-building LOS	1.6 – 1.8
Obstructed in-building	4 - 6

The first version of GOAT implements the following two default log-normal fading models:

- $\gamma_1 = 2.2$
- $\gamma_2 = 2.5$

b) Battery models

Three well-known types of node batteries have been considered for the first version of GOAT. The most relevant parameters taken into account regarding batteries are the charge and the voltage feed. The three implemented battery models are considered for

nodes feeded at 1.5 V. The charges, however, will differ depending on the implemented mode as shown in the table below.

Table 9: Battery models implemented in GOAT

Battery model	Charge	Voltage	Stored energy
AA	1500 mAh	1.5 V	8100 J
AAA	7500 mAh	1.5 V	40,500 J
Lithium	2200 mAh	1.5 V	11,880 J



Figure 41: AA batteries¹⁷

The battery model will deeply affect the node and overall network lifetimes. The higher energy stored in the batteries, the higher the time the node could be operative.

c) MAC models

Multiple Access Control (MAC) provides addressing and channel access control mechanisms that make it possible for several terminals or network nodes to communicate within a multiple access network that incorporates a shared medium [40]. Bus, ring, and wireless networks are some examples of shared physical media. Packet mode contention based MAC methods detect or avoid data packet collisions. Instead, circuit-switched or channelization-based MAC methods reserve resources to establish a logical channel [40].

The MAC models implemented in the first version of GOAT are depicted in this subsection.

B-MAC

B-MAC is a carrier sense media access protocol for wireless sensor networks that provides a flexible interface to obtain ultra-low power operation, effective collision avoidance, and high channel utilization [41]. B-MAC employs an adaptive preamble-sampling scheme to reduce duty cycle and minimize idle listening in order to achieve low power operation. With B-MAC, each sensor node periodically wakes up, for few milliseconds, only to check if there is a transmission in the air, and remains awake if it finds activity, otherwise goes again to sleep [41]. A node willing to transmit sends a

¹⁷ Retrieved 1 June from <http://rightbattery.com/wp-content/uploads/2013/08/1-5v-aa-duracell-alkaline-battery.jpg>

long preamble before the packet transmission. The long preamble overlaps with the listening time of the receiver, thus it is assured that the receiver will listen to the packet.

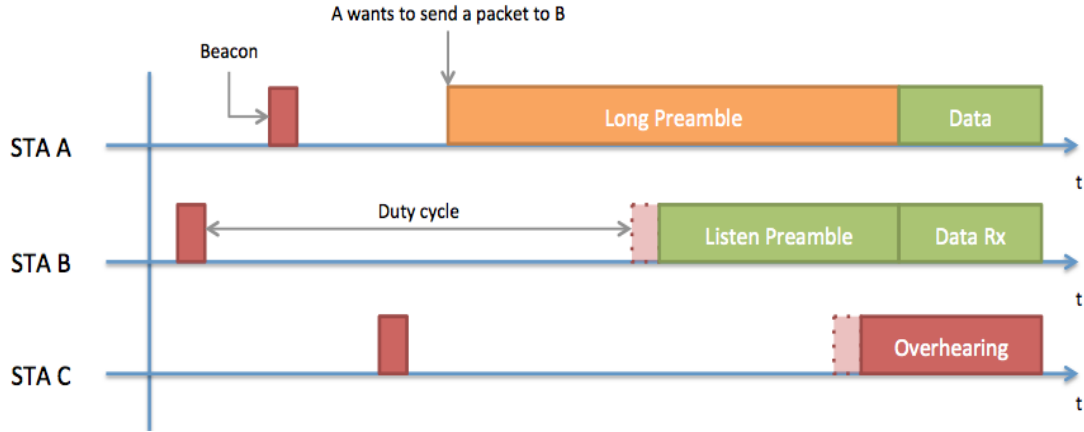


Figure 42: Example of B-MAC with three channels

The energy consumed by a node can be calculated adding the energy consumed in each of the possible node states. For wireless sensor network applications, the energy used by a node consists of the energy consumed by receiving, transmitting, listening for messages on the radio channel, sampling data, and sleeping.

$$e_{node} = e_{tx} + e_{rx} + e_{ov} + e_d + e_{sl}$$

Where the consumed energies are: e_{tx} when transmitting, e_{rx} when receiving, e_{ov} when listening packets that are not directed to the node (overhearing), e_d when the node samples the channel, and e_{sl} when sleeping.

In the equation above, e_{rx} and e_{ov} are equal because the energy consumed when listening a packet will be the same no matter whether the packet is directed to the node or not. Then,

$$e_{RX} = e_{rx} + e_{ov}$$

$$e_{node} = e_{tx} + e_{RX} + e_d + e_{sl}$$

Also, each of the state energies can be calculated as the state power multiplied by the time the node is in that state:

$$e_{state} = PT_{state}$$

Transmitting state

The time a node spends to transmit a packet is given by

$$T_{tx|pkt} = T_{preamble} + T_{pkt} = \frac{L_{preamble}}{2} + \frac{L_{pkt}}{R}$$

Then, the time a node will spend transmitting during an observation time T_{obs} is given by the number of packets it has transmitted, N_{tx} (which will depend on the node's sending rate, λ), multiplied by the time spent transmitting a packet.

$$N_{tx} = \lambda T_{obs}$$

$$T_{tx} = N_{tx} T_{tx|1pkt}$$

The energy consumed while transmitting during an observation time will be

$$e_{tx} = P_{tx} T_{tx}$$

Receiving state

The time a node spends listening for a packet is given by

$$T_{rx|1pkt} = T_{preamble}/2 + T_{pkt}$$

The equation above considers that probabilistically a node will listen half preamble as mean. Then, the time a node will spend receiving or listening a packet during an observation time, T_{obs} , is given by the number of packets it has listened, N_{rx} , multiplied by the time spent listening to a packet.

$$N_{RX} = N_{rx} + N_{ov} = (R_{rx} + R_{ov}) T_{obs}$$

$$T_{RX} = N_{RX} T_{rx|1pkt}$$

It is important to note that sending, receiving and overhearing rates will be given depending on the WSN topology and the sending rates of each node. That is one of the most relevant values of GOAT, allowing analyzing all kind of WSNs without depending on the topology throughout the simulation.

The energy consumed while receiving or listening to a packet will be

$$e_{RX} = P_{tx} T_{RX}$$

Sampling and sleeping state

The time a node is not listening neither transmitting is shared between the sampling and the sleeping time. The proportions are given by the time of a duty-cycle, T_{DC} , and the “awake” beacon time, T_{beacon} . Then, the time a node is sampling the channel during an observation time T_{obs} is given by

$$T_d = T_{beacon}/T_{DC} (T_{obs} - (T_{tx} + T_{rx}))$$

Respectively, the time a node is sleeping can be calculated as

$$T_{sl} = (T_{DC} - T_{beacon})/T_{DC} (T_{obs} - (T_{tx} + T_{rx}))$$

Or also,

$$T_{sl} = T_{obs} - (T_{tx} + T_{rx} + T_d)$$

Energy consumed

Finally, the energy consumed by a node during an observation time is given by

$$e = P_{tx}T_{tx} + P_{rx}T_{RX} + P_dT_d + P_{sl}T_{sl}$$

And the node's lifetime expressed in observation time units will be

$$l = e_{battery}/e [T_{obs}]$$

B-MAC + ACK

B-MAC with ACK is an expanded model of B-MAC. In this case, acknowledgment functionalities are considered. For that, any time a node receives a packet destined to it, it will generate an ACK and send it to the node that previously sent the data packet. Some considerations have to be taken into account:

The time taken to transmit a preamble should be greater than a duty cycle in order to ensure any preamble will be listened.

$$T_{Preamble} \geq T_{DC}$$

It is possible that a node would listen to data packets not directed to it, but also, ACK packets for other nodes could be listened by the mentioned node. Then, the transmission and reception times in the B-MAC + ACK model will be given by

$$N_{tx} = \lambda T_{obs}$$

$$N_{RX} = N_{rx} + N_{ov} = (R_{rx} + R_{ov})T_{obs}$$

$$T_{tx} = N_{tx}T_{tx|1pkt} + N_{rx}T_{ACK}$$

$$T_{rx} = N_{rx}T_{rx|1pkt} + N_{tx}T_{ACK}$$

The sampling and sleeping will remain the same as the B-MAC model ones.

$$T_d = T_{beacon}/T_{DC} (T_{obs} - (T_{tx} + T_{rx}))$$

$$T_{sl} = (T_{DC} - T_{beacon})/T_{DC} (T_{obs} - (T_{tx} + T_{rx}))$$

IEEE 802.11ah

The IEEE 802.11ah Task Group is currently working on the standardization of a new amendment with the focus placed on sensor and actuator networks. It will operate at sub-1GHz bands; ensure transmission ranges up to 1 Km, minimum network data rate of 100 kbps and very low power operation. This new amendment will extend IEEE 802.11 potential to applications such as smart metering; plan automation, eHealth or surveillance. The CAS-based channel access protocol for IEEE 802.11ah WLANs

presented in [42] optimizes the number of Channel Access Slots (CASs), their length and their allocation to the stations (STAs), while maximizing the time they remain in sleep mode in order to keep the energy consumption low.

GOAT implements a simplified version of the presented scheme. The following considerations have been taken:

- Low sending rate: It is considered that only one STA (or none) per group will transmit in a same TIM.
- Only uplink is considered: The sink node will receive data packets from the STAs but will not send packets to them.
- Single-hop routing model is considered: All the STAs will be directly linked to the sink (one-hop). That means that no packet forwarding will be given.

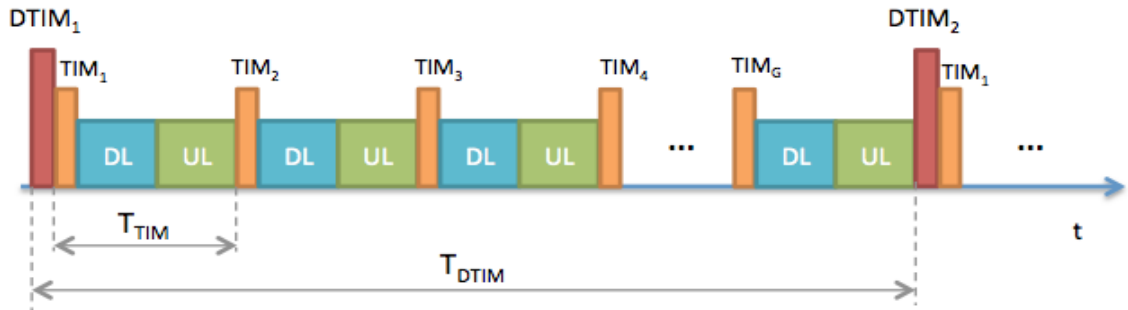


Figure 43: 802.11ah time slots distribution

The energy consumed by a node can be calculated adding the energies consumed in each of the possible node states. The possible node states in this scheme are transmitting (uploading), receiving, idle and sleeping.

$$e_{node} = e_{tx} + e_{rx} + e_{idle} + e_{sl}$$

During an upload, or transmission, time several packets and interframe spaces are given as shown in the figure below.

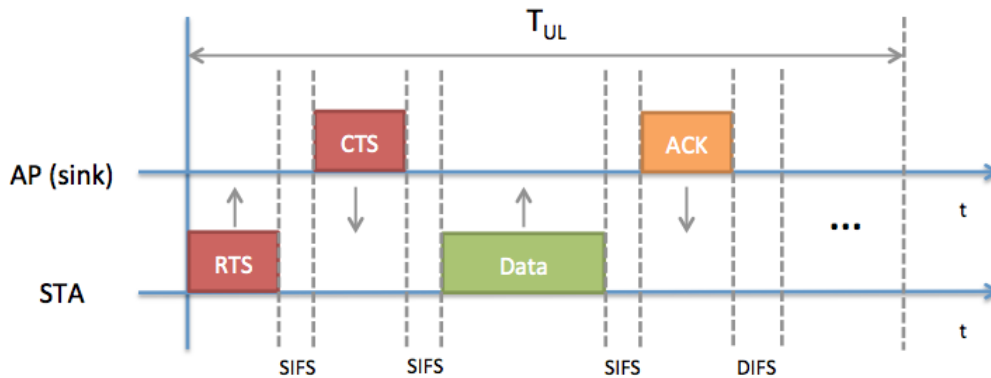


Figure 44: 802.11ah upload time slot distribution

Then, the times a node spends to transmit and receive a packet, respectively, are given by

$$T_{tx} = N_{tx}T_{tx|1pkt} = \lambda T_{obs}(T_{RTS} + T_{data})$$

$$T_{rx} = N_{DTIM}DTIM + N_{TIM}(TIM + T_{CTS} + T_{ACK})$$

And developing the reception time equation above,

$$T_{rx} = T_{obs}/T_{DTIM}DTIM + \lambda T_{obs}(TIM + T_{CTS} + T_{ACK})$$

A node is in idle state when, during an uplink time, it is not receiving or sending. That is, when the interframe spaces take place.

$$T_{idle} = N_{TIM}(3 SIFS + DIFS) = \lambda T_{obs}(3 SIFS + DIFS)$$

Finally, as for the previous cases, the sleep time can be calculated by subtracting the transmitting, receiving, and idle states times of the observation time.

$$T_{sl} = T_{obs} - (T_{tx} + T_{rx} + T_{idle})$$

d) Routing models

The routing module defines which wireless links will be built in order to allow any sensor node to reach the sink or gateway depending on several conditions: coverage, quality of service (QoS), energy saving, etc. Below, the first three implemented routing models are depicted.

Single-hop

The single-hop module just takes into account the reachability between the sink node and the rest of nodes. If an STA is able to reach the sink, a direct link will be built between them. Then, no forwarding is considered. All those nodes that could not reach the sink directly due to the lack of coverage will not be part of the network.

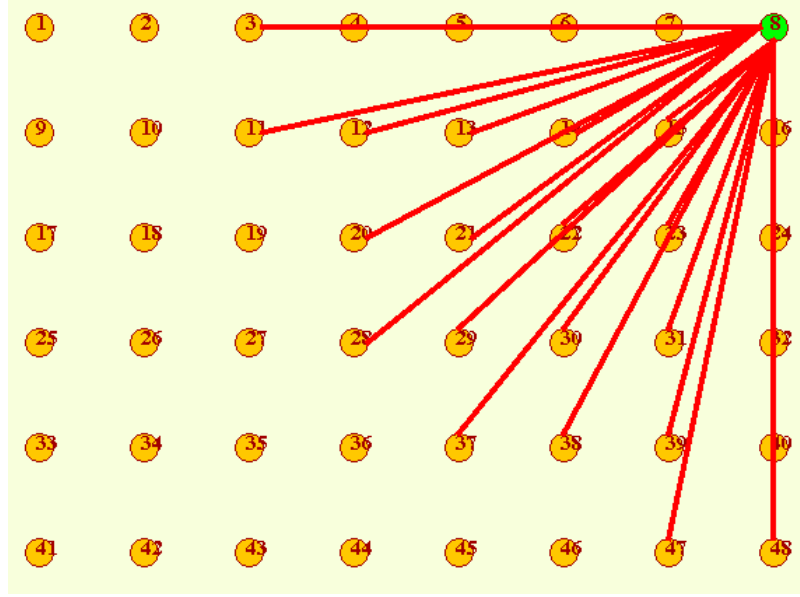


Figure 45: Single-hop routing

In the picture above a grid WSN topology is shown where the node 8 is set as the sink. We can note that those sensor nodes that are close enough to the sink are linked to it. On the other hand, the rest of nodes are out of coverage, then, they cannot reach the sink, which implies remaining out of the network. GOAT automatically changes the frequency value from 2.4 GHz to 868 MHz when the single-hop routing model is selected. That is done in order to allow greater distances between the sink and the rest of nodes as the 802.11ah standard demands.

Next-node (level based)

Next-node model determines the network links based on the levels composed by the sensor nodes. GOAT has defined 8 levels in its first version (from level A to level H). Nodes belonging to further levels will not be part of the network.

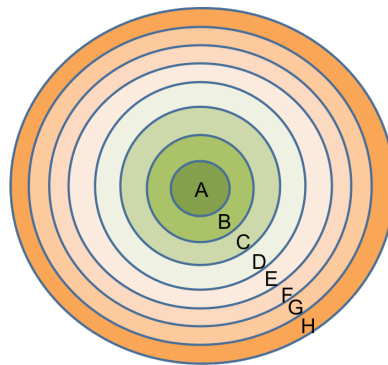


Figure 46: Routing levels (hops)

The procedure given in next-node module is the following:

- If a node N reaches the sink, that is, if the N belongs to level A , it gets linked to the sink directly.

- If a node N does not reach the sink:
 - It gets linked to the first node it senses belonging to the highest level. For instance, if N cannot reach the sink but reaches a node belonging to levels C and B, it will be linked to the node in level B.
 - If N reaches two or more nodes belonging to the same level, it will be linked to the first one it reaches without taking into account proximity or power levels.

Below it is shown how would the links be built on the grid topology presented in the previous case, but now implementing next-node routing model.

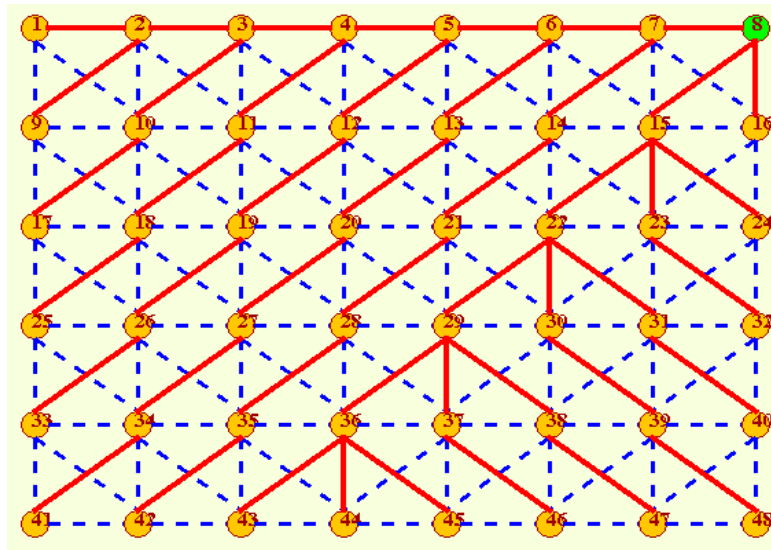


Figure 47: Next level routing

On the example above we clearly observe a routing design generated using next-node routing module. For instance, let us study the behavior of node 24. Instead of being linked to the node 16, which is reached from the node 24 and also belongs to level A, it gets linked to node 15. As explained before, this routing module does not take into account the proximity or power levels. That is why node 24 gets linked to node 15 at the moment it senses it, without finishing sensing the rest of nodes.

Closest-node (Power based)

Closest-node routing, as next-level routing, bases the links structure depending on the levels mentioned above. In this case, however, proximity and power levels are considered in order to optimize the WSN performance in real scenarios. The procedure used by this model is explained below:

- If a node N reaches the sink, that is, if the node belongs to level A, it gets directly linked to the sink.
- If a node N does not reach the sink:

- It sweeps all the nodes it can reach and determines the level and proximity to each of them. Then N will be linked to that node belonging to the closest level to the sink and also closer to the node N .
- If N reaches two or more nodes belonging to the higher level it can hop to, N will be linked to the closer node of both. That is, N will be linked to the node from which it senses a higher power.

In the figure above, the closet-node routing model has been used. As expected, now node 24 gets linked to node 16, which is the closest node belonging to level A. In terms of packets loss, this improvement will not affect the simulation results due to collisions nor errors are considered. However, for further GOAT versions it could be worth to have this model and analyze its better performance.

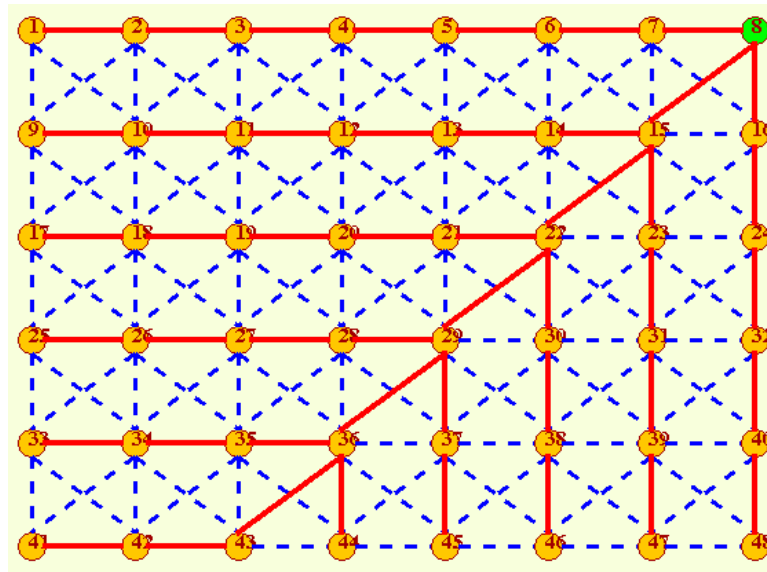


Figure 48: Power level routing

4.6 Configuration

Some of the variables and parameters defined in GOAT have been *hard-coded* in the java classes and are not accessible from the GUI. These parameters are not supposed to be usually modified by the user. Examples of these parameters are hardware configurations, battery standard charges or canvas colors. In case the user wants to change one of these values, he would have to modify the open source code (java classes) and compile it again. Then, running the simulator again will allow the user to design and simulate with the previously configured variables.

Table 14, which can be found at annex **A2. GOAT configuration variables**, lists all the mentioned parameters and their corresponding variable names, types and default values. The Java classes where the variables are defined are also included on the table in order to make it easier for the user to modify the code.

```
// Hardware
double pTx = 60;           // Transmit power [mW]
double pRx = 70;           // Receive power [mW]
double pIdle = 20;         // Idle power [mW]
double pSleep = 3;         // Sleeping power [mW]
double sensitivity = pow(10, (-81 / 10)); // Sensitivity [mW] (-81 dBm)
```

Figure 49: Hardware variables code

4.7 Simulation

After setting up the WSN topology and the scenario to analyze, the user can run a simulation by clicking on the button with a start icon. This subsection describes how the simulation's input and output are, and which associated processes take place.

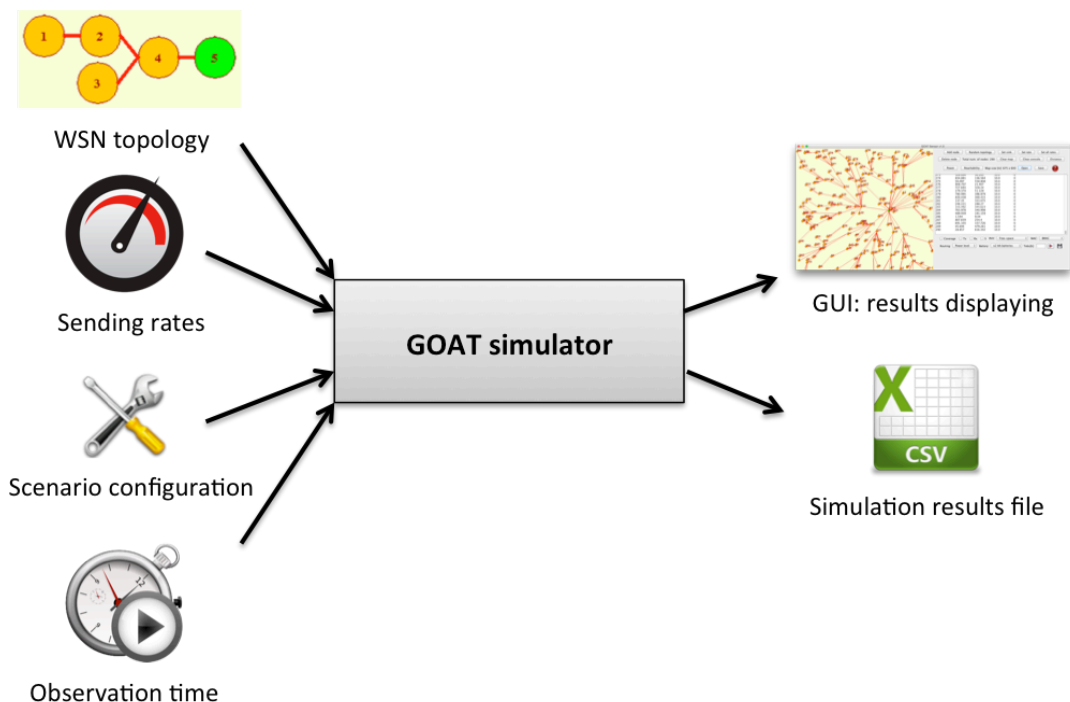


Figure 50: Simulation input and output

a) Input

The following elements are needed as input in order to run a simulation:

- **WSN topology:** Two or more nodes topology and a defined sink are required. Any kind of topology is allowed no matter the number of nodes or links. It is important to note, however, that a huge number of nodes may reduce the simulation processing speed. The WSN to be analyzed can both be designed from scratch or by opening a previously save topology file.
- **Sending rates:** The GOAT simulator requires the transmitting, receiving and overhearing rates for calculating the energy consumption independently from the selected models. These rates are calculated based on the sending rate of each

node (λ), the topology, and the physical model, which determines the reachability among nodes.

- **Scenario configuration:** In order to fully configure the scenario, the user has to select the physical, battery, MAC and routing models from the GUI, and set the hardware and fixed model parameters on the code.
 - Models selection: The scenario models have to be selected at the GUI. The implemented models in the first version of GOAT are the following.
 - Physical: Free-space, Log-normal (2.2), and Log-normal (2.5).
 - Battery: 2x AA, 2x AAA, and Lithium.
 - MAC: B-MAC, B-MAC/ACK, and 802.11ah.
 - Routing: Single-hop, Next-level, and Power-level.
 - Parameters configuration: There are two types of fixed scenario parameters.
 - Models fixed parameters: Those parameters related with the scenario models (e.g. 802.11ah TIM duration).
 - Hardware parameters: Those parameters related with the system's components hardware (e.g. node transmitting power).
- **Observation time:** Period of time during the WSN will be analyzed.

b) Output

When the user presses the simulation icon, GOAT will check if all the input elements are properly entered: two or more nodes topology, a valid observation time (i.e. numeric value), etc. If the input is properly entered, the simulation algorithm will generate a MAC model for each of the nodes. This model will determine the energy consumed by the node during the observation time, its lifetime, and its remaining battery charge. Then, by comparing the results of all nodes, the analyzer calculates the proportion of time the nodes spend in each of the possible states, and the network's overall lifetime, which is equal to the lifetime of the node that runs out of battery first.

The calculated variables are presented in two different ways: per console and in a comma-separated value (csv) file.

- **Console:** The GUI console displays the following information.
 - Table with node identifier, main rates (transmission, reception, and overhearing), total energy consumed during the entered observation time, the node's lifetime, and the remaining charge in percentage.
 - Network overall lifetime in days and years.
 - Critical node (first node to run out of battery) identifier.
 - Time proportion of the states (Transmitting, receiving, sampling, idle, and sleeping) of:
 - Network mean.

- Critical node specific values.
- Csv file: The user can save the simulation results by clicking the save button. The generated file will be structured as shown in the figure below. The first column represents the node identifiers; the first row indicates the time instant when a measurement is taken (in hours), and the rest of the cells represent the energy consumed by the node in a specific instant.

	A	B	C	D	E
1		0	1	2	3
2	Node 1	0	13,2	26,5	39,7
3	Node 2	0	11,7	23,3	35
4	Node 3	0	12,7	24,3	36

Figure 51: Simulation results csv file

4.8 Limitations

This first version of GOAT presents some limitations. The main ones are listed below.

- At the GUI, collisions among nodes cause uncontrolled nodes reallocation. It is highly recommendable to avoid dragging nodes over others.
- No packet collisions are considered.
- The number of nodes is limited to those that fit in the map. At the moment, no zoom feature is implemented.
- In order to modify some of the parameters the user has to modify the code and compile it again.
- Analyses have been done considering low dense WSN (up to 1,000 nodes). Proper estimations for WSN with a greater number of nodes are not assured.
- The number of levels, or hops from one node to the sink cannot be higher than 8. Nodes out of coverage from level 8 will not be linked to the network.

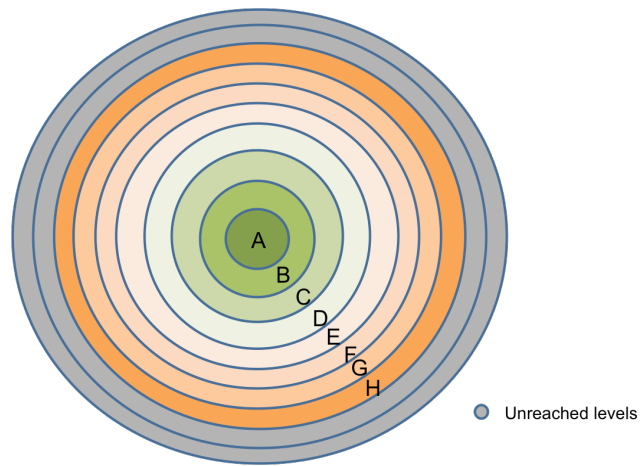


Figure 52: Unreached levels

5. EVALUATION

This section presents four different scenario evaluations made by GOAT as a proof of concept. As explained in sections before, GOAT aims to allow the WSN designer to estimate the network behavior in terms of energy consumption and lifetime before actually building it. Regarding that, the following results are an example of how the analyzer can help to that end.

5.1 Street parking

a) Scenario definition

This subsection analyzes different WSN designs for an outdoor parking area management in Juan de Garay Street, in Barcelona. The scenario is based on an abstraction of the real street parking deployed in the city of Santander throughout the SMART SANTANDER project, which has been developed by several companies and institutions including Telefonica I+D and University of Cantabria¹⁸.

The presented WSN will detect public parking sites availability at Juan de Garay Street, which is about 550 meters long. There are 90 parking sites located along this street, which will be detected by 90 sensors nodes, and 30 repeaters including the gateway. The repeaters, installed in the streetlights and trees, will be fed by their own battery and will listen to parking sensors based on ferromagnetic technology buried under the asphalt. As shown in the figure below, each of these repeaters will be wirelessly connected to 3 parking sites availability sensors, allowing full outdoor parking management along the street.

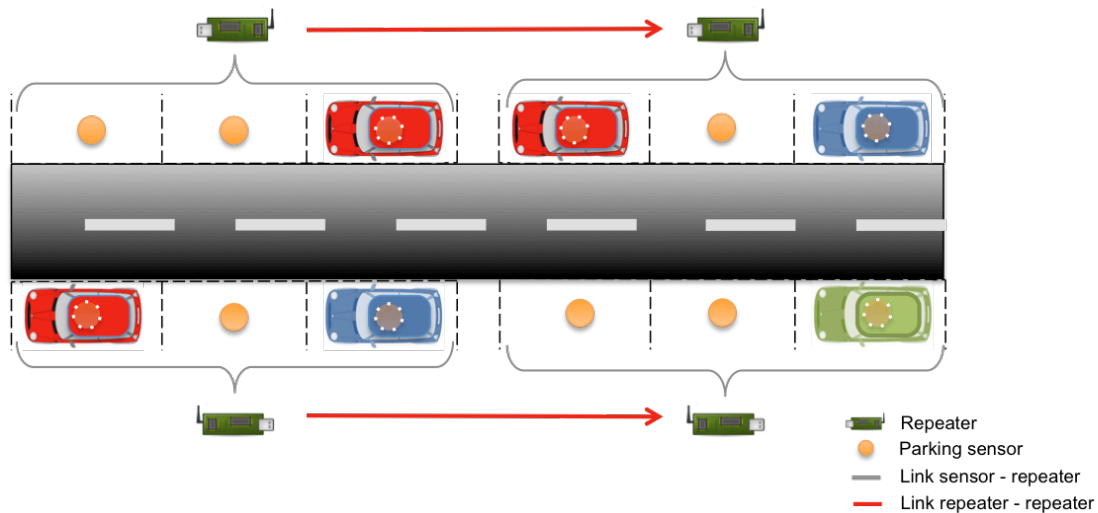


Figure 53: Smart parking scenario

These sensors will send the data to their respective repeater periodically by single-hop routing, and the repeaters will gather that data and forward it to the sink using next-level

¹⁸ SMART SANTANDER project - http://www.libelium.com/smart_santander_smart_parking/

or power-level routing models. The gateway will be also placed in a street light approximately in the middle of the street in order to reduce as much as possible the maximum number of hops among the nodes and the sink. This gateway will be constantly plugged to the steam and will store the WSN data in a server on the Internet through a 3G connection.

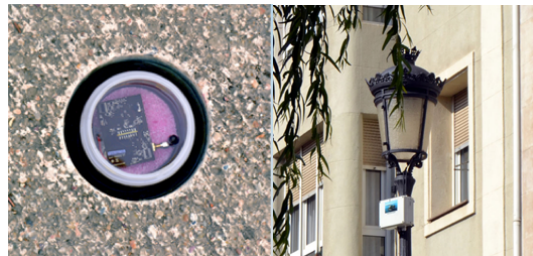


Figure 54: Parking sensor (left) and repeater (right)¹⁹

This study does not focus on the sensor - repeater local parts of the network, but on the 30 repeaters network which can be modeled as a WSN composed of 30 nodes with a sending rate of three times the packet rate a parking sensor generates.

b) Modeling and results

It is required to locate the nodes along the 530 meters of the street in a way that all the parking sites are covered. We are considering a log-normal physical model with γ factor of 2.2 corresponding to not crowd urban scenarios. The proposed solution is shown in the map below. The spotted blue lines indicate the reachability among nodes. Then, for instance, node 9 reaches (listens) nodes 10, 25 and 26.

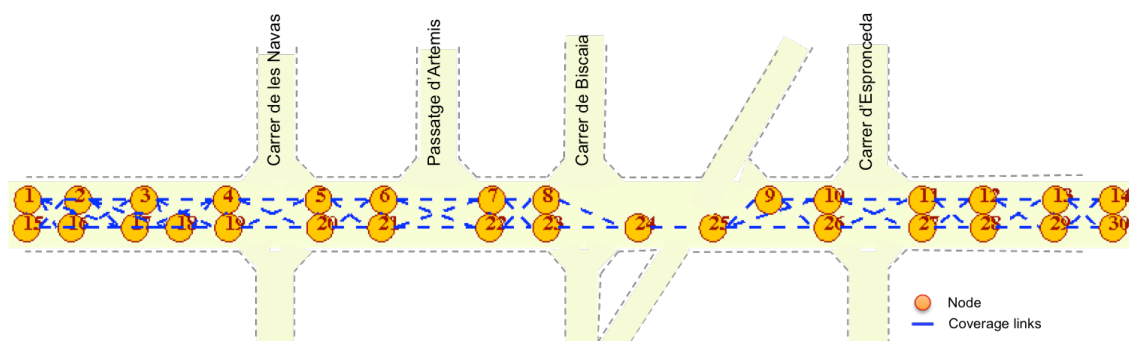


Figure 55: Smart parking - Repeaters reachability

As at SMART SANTANDER project, we consider that a parking sensor sends a packet each 5 minutes. Then, a node will forward 3 packets each 5 minutes, which implies a sending rate of 36 packets per hour. We will also consider that nodes have two AAA batteries installed (7500 mAh charge). For this scenario, we are going to study the B-

¹⁹ Retrieved 20 May 2015, from - http://www.libelium.com/smart_santander_smart_parking/

MAC and B-MAC/ACK MAC protocols. In order to start the first analysis, the node number 24 will be set as sink due to it seems to be located near to the middle of the street, minimizing the maximum number of hops from the farthest node to the sink. Lately, we will study which node is the better candidate to be set as sink in order to reduce the overall energy consumption.

Next-level vs. Power-level routing

If we implement **next-level** routing, the generated links will compose the following network topology.

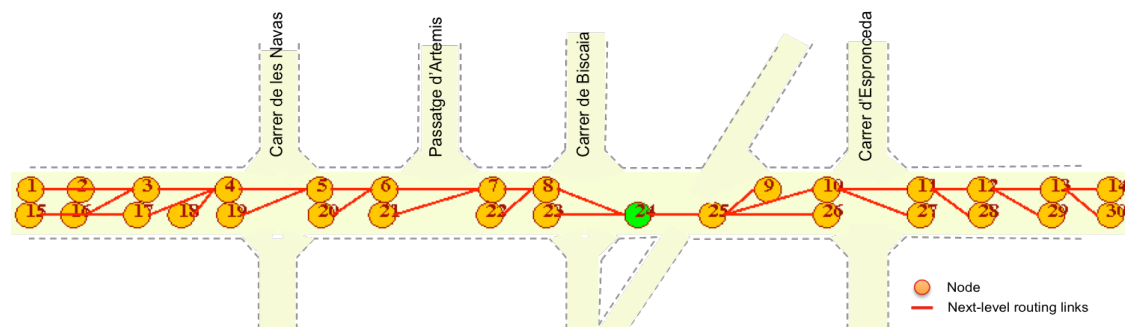


Figure 56: Smart parking - Next-level routing

As expected each node is linked to the first node it reaches belonging to a higher level. In principle, it is not an efficient way of routing, but is simple and could fit in an scenario like the presented. Running a simulation with GOAT and the mentioned parameters, these are the results generated.

- Network overall lifetime:
 - **119.51 days** (118.63 days with ACK)
 - Critical node (first to get out of charge): Node 22
- Time proportion in each state:
 - Overall (mean):
 - Transmitting: 0.07 %
 - Receiving: 0.44 %
 - Sampling: 2.00 %
 - Sleeping: 97.49 %
 - Critical node:
 - Transmitting: 0.02 %
 - Receiving: 1.16 %
 - Sampling: 1.97 %
 - Sleeping: 96.85 %

For the scenario above, we obtain that the network will be operative during 119.5 days (about 4 months). This value is determined by the critical node lifetime. The critical node, as shown in the results, is the node number 22, which is in the receiving state about 2.5 times the time in receiving state of the nodes mean. Also, the sleeping time is

reduced to 96.85 %. It is important to note that the maximum energy consumption at a node is given while receiving (70 mW considered in this scenario), and the minimum one is given while sleeping (3 mW considered in this scenario). That explains why the node 22, which listens to a lot of packets, is the first node to get out of charge.

Regarding the B-MAC/ACK MAC protocol, it can be observed that in an scenario where no packet collisions or loss are considered it makes no sense to implement acknowledgment features. The results given while implementing B-MAC/ACK show an increase on the energy consumption due to the extra ACK packets generated at the network, which will be transmitted, and also listened. In this case, the WSN overall lifetime can be elongated by approximately one day.

Now, let us consider the Power-level routing model. This model makes a node to get linked to the closest node belonging to a higher level. The topology obtained in this case is shown in the map below.

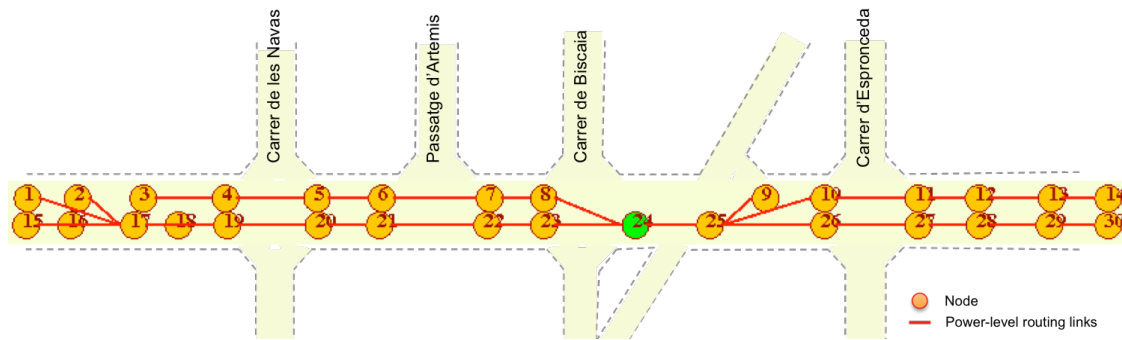


Figure 57: Smart parking - Power-level routing

The results for this scenario, as shown below, confirm the improvement in terms of energy consumption given when implementing Next-level routing. In this case, this improvement is not very great, however any increase in energy saving will allow the WSN to operate for a longer time (2 days more in this case). Now, the node 24 is no longer the critical node, but the node 7. This behavior could be explained by the increase on the number of packets that node 7 listens and the decrease given on node 24. Node 7 receives 2736 packets per hour (1296 overhearing packets), instead node 22 receives 2196 packets per hour (936 overhearing packets).

- Network overall lifetime:
 - **121.48 days** (120.57 days with ACK)
 - Critical node (first to get out of charge): Node 7
- Time proportion in each state:
 - Overall (mean):
 - Transmitting: 0.07 %
 - Receiving: 0.45 %
 - Sampling: 1.99 %
 - Sleeping: 97.49 %
 - Critical node:

- Transmitting: 0.09 %
- Receiving: 1.00 %
- Sampling: 1.98 %
- Sleeping: 96.93 %

Comparing the critical nodes performance in each of both cases, we note that the time spent sleeping by node 7 is greater than the time in node 22. Also, the receiving time is minor on node 7. Then, the critical node and the overall lifetime will be greater applying Power-level routing instead of Next-level routing.

Sink location

Having elected the routing model, let us now determine which the best location of the sink is in order to reduce the energy consumption to the minimum. Due to the peak of transmitted and listened packets is given in the center nodes of the network (nodes 7, 8, 22, 23 and 25), we will run a simulation for each of the mentioned nodes. The results are shown in the table below.

Table 10: Smart parking - Overall WSN lifetime depending on the sink location

Sink	Overall lifetime [days]
Node 25	123.82
Node 8	124.72
Node 23	124.72
Node 7	127.38
Node 22	126.64

The node 7, which was the critical node in the previous case, presents the higher overall lifetime (127.38 days). Then, we can conclude than the best WSN design in terms of energy saving is given when the following rules are applied:

- MAC model: B-MAC
- Routing model: Power-level
- Sink: node 7

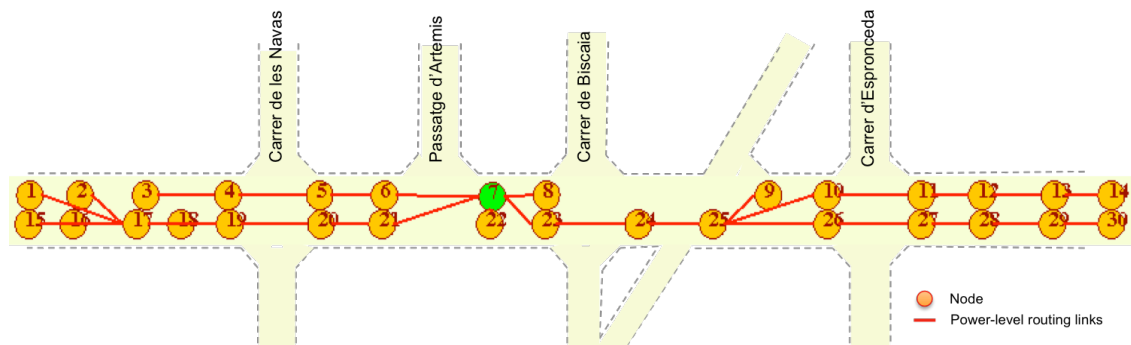


Figure 58: Smart parking - node 7 set as sink

The chart below displays the energy consumption of the critical node (node 5), the node consuming the minor energy (node 30) and the mean of all the nodes during 3,000 hours. This data has been gathered in a simulation file generated by GOAT indicating the node identifier and the energy consumed at each hour from 1 to 3200. The overall lifetime of the network is given when the node 5 energy consumption curve gets constant at the 3057th hour (corresponding to 40,500 J, the energy stored in 2 AAA batteries), when the node 5 battery gets out of charge and the WSN is considered to be inoperative.

Energy consumption

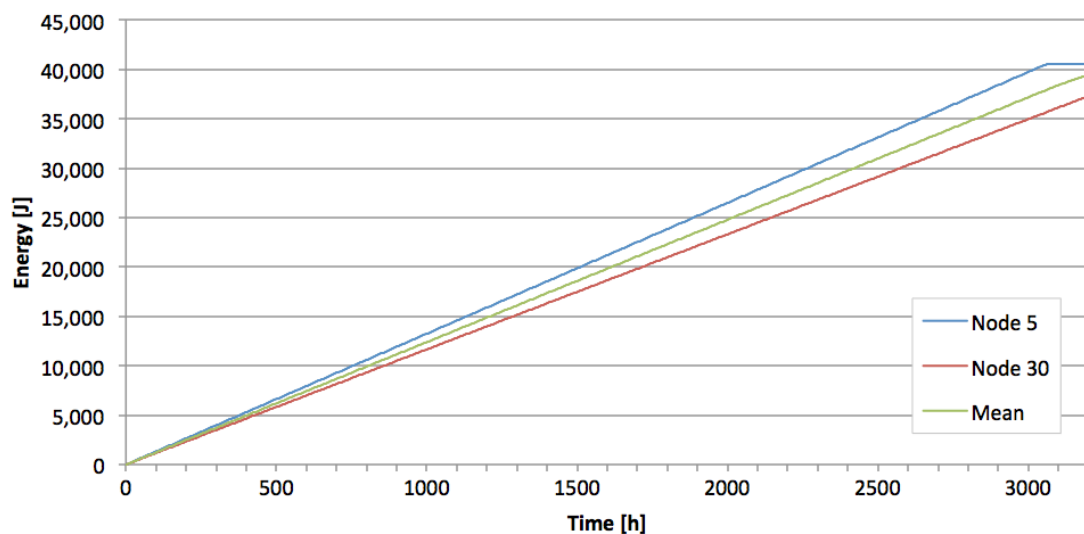


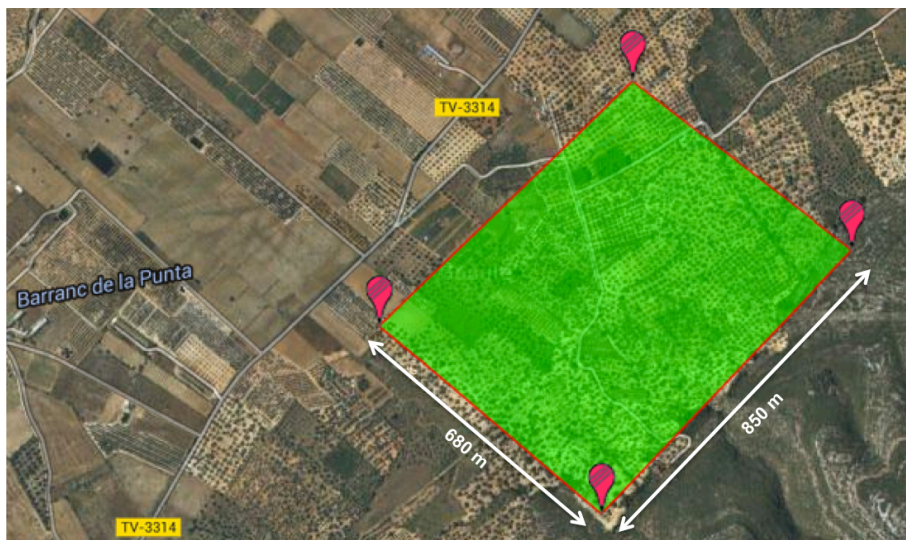
Figure 59: Smart parking - WSN energy consumption over time

5.2 Plague tracking

a) Scenario definition

This subsection presents an animal-mapping scenario based on the ENTOMATIC project²⁰. ENTOMATIC aims to solve a major problem faced by EU Associations of Olive growing small and medium-sized enterprises (SMEs): the Olive fruit fly (*Bactrocera oleae*) [43]. The estimated economic losses caused by the *Bactrocera oleae* are approximately 600 € per hectare. The solution proposed at ENTOMATIC is to develop a WSN composed by autonomous bioacoustic-recognition in order to detect and control the olive tree fly population at olive trees. This infrastructure will allow olive producers to track the fly populations and receive advice on precision pesticide application. When the project is working, it is expected a reduction of damage to olive fruit and oil production and to promote the sustainable use of pesticides [43].

The selected geographic zone for the analysis is an olive tree field placed near “El Barranc de la Punta” at Secans del Montsià, Tarragona. We will consider a rectangular area of 680 x 850 meters (57.8 ha). The scenario will implement a simplified version of the 802.11ah MAC model due to it has been designed for communicating WSN nodes being distanced by large distances (up to 1 kilometer) and its associated routing model, the single-hop. The implemented MAC model details can be consulted in **section 4.5.c MAC models**.



This study aims to analyze the impact of the propagation and battery model on the network overall lifetime. We will also study how to address the lack of coverage depending the physical model selected by dividing the WSN in two or more new smaller WSNs by setting new gateways. As in the case above, we have not considered packet loss or collisions and regarding the 802.11ah MAC model, we are considering the nodes to have a low sending rate (1 measurement sent every hour), which allows us

²⁰ Entomatic project - <http://entomatic.upf.edu>

to simplify the model and suppose that only one sensor node (or none) per group will transmit in a TIM.

b) Modeling and results

The topology for this scenario will be randomly generated on GOAT. The area is given by the part of the olive tree field we have selected. We will start considering that there will be 5 nodes installed per hectare. Hence, there will be 290 nodes (5 nodes * 58 nodes/ha), 289 sensor nodes randomly distributed over the olive tree field and one gateway centrally located. To do so we enter the following values on the “Random topology” pop-up window:

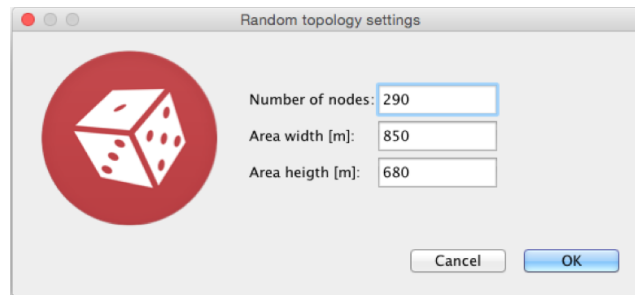


Figure 60: Random topology input values

Below it is shown the generated topology. This capture has been taken implementing free-space propagation model, hence each of the nodes of the WSN reach the sink and are able to transmit its measurements. The sink has been randomly located in a centered part of the area in order to maximize the coverage from the gateway to the rest of nodes.

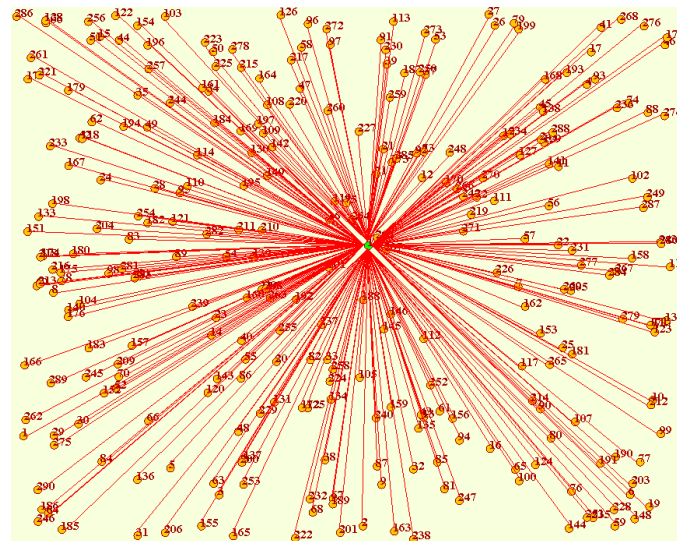


Figure 61: Plague tracking WSN topology (Free-space)

Battery model impact

For the case above, let us study the overall lifetime of the network depending on the battery model elected. Due to the abstraction done for modeling this case, all the nodes

of the network will consume exactly the same amount of energy. Thus, each of the nodes lifetime will be equal to the overall lifetime (all nodes will fall at the same time).

Table 11: Network lifetime depending on the battery model

Battery model	x2 AA (1500 mAh)	Lithium (2200 mAh)	x2 AAA (7500 mAh)
Network lifetime	30.47 days	44.69 days	152.38 days

The table below shows the network lifetime for different battery models. We note that installing two AAA batteries per node will allow the network to be operative for approximately 5 months. On the other hand, the couple of AA batteries present the minimum overall lifetime (about 1 month). In this scenario, and usually for any kind of WSN, it is interesting to maximize the network autonomy as much as possible in order to avoid having to replace manually the batteries of each of the 289 sensor nodes. However, it has to be taken into account the economic impact of the chosen battery model. Battery with higher charges will be usually more expensive and larger, so any scenario has to be studied accurately depending on all these variables.

It is remarkable the high amount of time a node spends in sleeping state on this scenario (99.89%). The minimum energy consumption is given while sleeping, so it is a clear objective to maximize the time a node spends in this state in order to increase the overall lifetime.

Physical model impact

Let us prioritize the overall lifetime and fix the battery model to a couple of AAA batteries and focus on the physical model impact. On the figure below it can be seen the coverage decrease when applying log-normal propagation models instead of free-space.

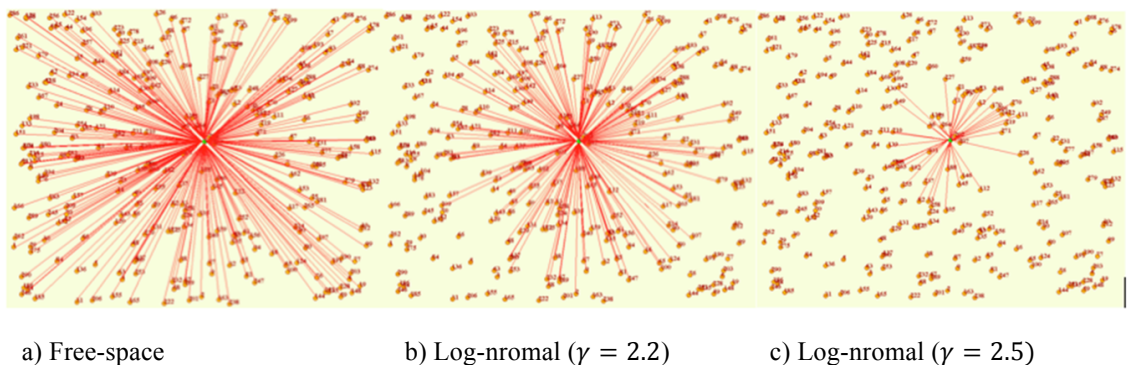


Figure 62: Physical model impact on coverage

If we want to keep the connectivity of each of the sensor nodes, it will be needed to install new sinks for reaching all the area. One possible solution would be to divide the area in smaller sectors with other sinks located in their centers. Then, new independent WSNs will be generated covering the entire olive tree field. The table below shows the

number of reached sensors nodes keeping just one sink, and the number of sinks needed to cover all the area.

Table 12: Physical model impact on topology

PHY model	Free-Space	Log-normal (2.2)	Log-normal (2-5)
Sensor nodes connected	289	187	45
Sinks needed	1	3	9

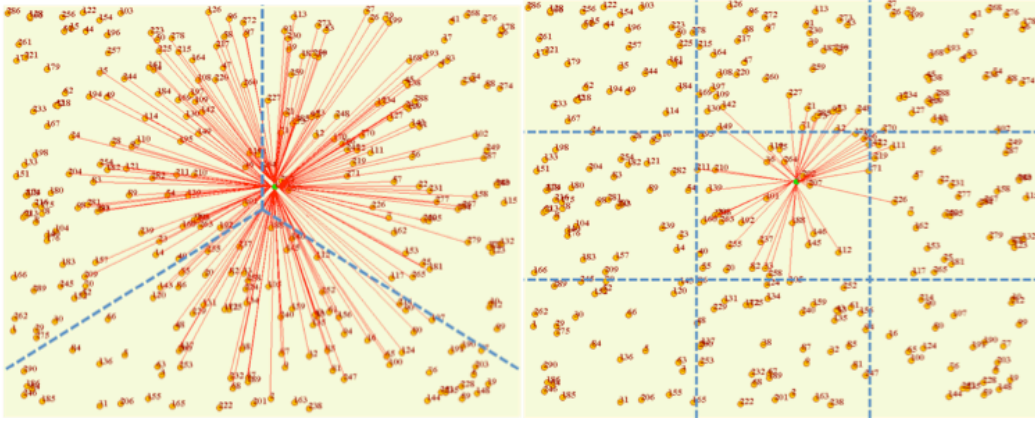


Figure 63: Area sectoring. log-normal 2.2 (left), and log-normal 2.5 (right)

B-MAC

Finally, we present what the simulation results are when implementing the B-MAC model. As mentioned before, the 802.11ah MAC model seems to fit best with this scenario due the high number of sensor nodes and the area dimensions. Nevertheless, let us check if the assumption was right by comparing the overall lifetimes got in both scenarios. The simulation was made for Free-space physical model and 2x AAA battery model. The new routing links are shown below.

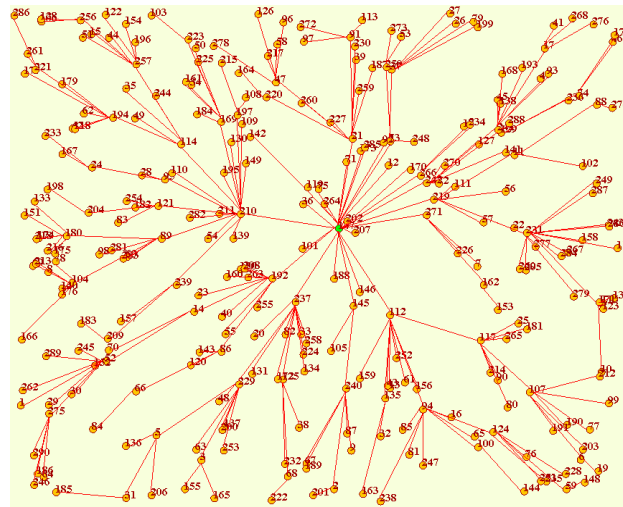


Figure 64: B-MAC topology

A comparison of the simulations results is shown in the table below.

Table 13: MAC model impact on lifetime

MAC model	B-MAC	B-MAC/ACK	802.11ah
Overall lifetime	110.96 days	109.22 days	152.38 days

As expected, 802.11ah offers longer lifetime. One of the key factors is the time the nodes are in sleeping mode. For the 802.11ah scenario, the nodes were sleeping a 99.89% of the time, instead, when B-MAC is applied; the critical node (node 23) is sleeping just the 96.04% of the observation time.

The overall lifetime is even smaller when the physical model level of restriction increases. If the coverage among the nodes gets smaller, the number of hops will grow, making the nodes to forward more packets and increasing its energy consumption. Below it is shown the WSN lifetime depending on the physical mode selected.

Table 14: Physical model impact on coverage

PHY model	Free-Space	Log-normal (2.2)	Log-normal (2-5)
Overall lifetime	110.96 days	103.34 days	No coverage

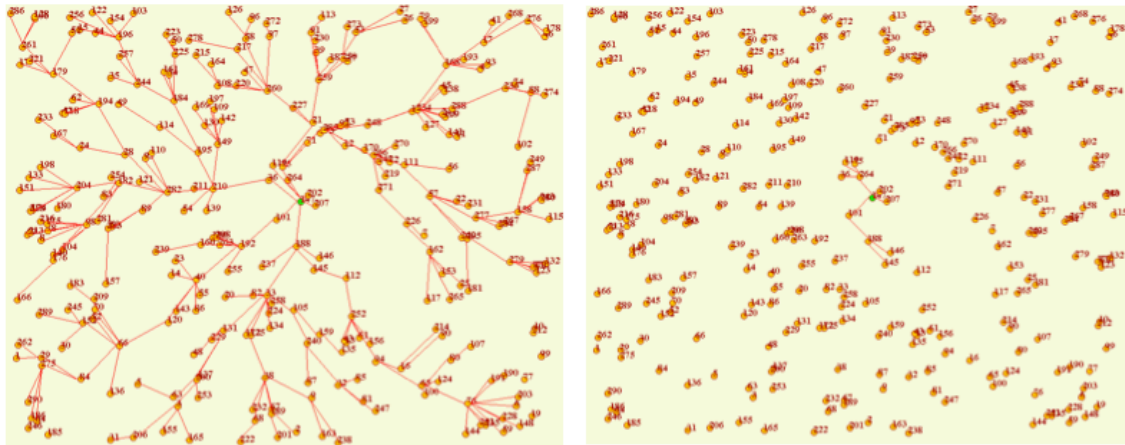


Figure 65: Log-normal 2.2 (left), and log-normal 2.5 (right) topologies

In both studied cases, B-MAC and 802.11ah, packet collisions were not considered to happen. This abstraction may be not valid on real scenarios due to the high number of sensor nodes that are able to reach each other even having a low sending rate. In the figure below the coverage links among the nodes of the WSN with log-normal and $\gamma = 2.5$ as physical model is shown. The high number of coverage links hinders avoiding packet collisions and properly designed MAC protocols should be implemented. However, these collisions may be negligible when implementing WSNs where all the contenders send packets with a very low rate.

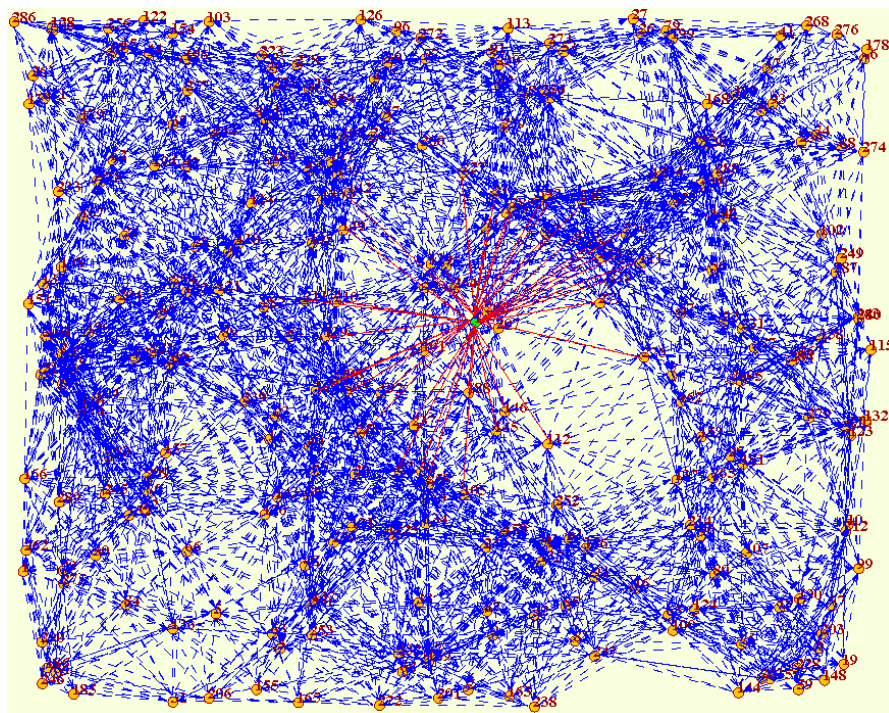


Figure 66: Coverage links

6. CONCLUSIONS AND FUTURE WORK

Conclusions

This project introduces wireless sensor networks and presents a software tool to analyze them. The field of WSN has been identified as one of the top emerging technologies that will change the way we understand communications, with an expected huge growth rate during the next decades. However, the limited energy resources of sensor nodes is a top constraint in this kind of networks due to the fact that, in most cases, nodes are battery-powered devices and, consequently, energy-constrained. Hence, the main concern is how to reduce the energy consumption in order to extend the overall network lifetime while providing a proper enough performance.

In order to face that issue, it is almost imperative to test WSN designs and try to optimize the energy saving mechanisms before actually building them. Nonetheless, running real experiments on WSN testbeds is costly and challenging. That is one of the main reasons why we have developed the GOAT tool.

GOAT is a graphical WSN analyzer that allows designing WSNs and estimating its energy consumption in configurable scenarios. The implemented models (physical, battery, MAC, and routing) can be thoroughly set, which offers a vast number of possible scenarios and allows designing and testing future real operating WSNs. As a first version, the tool can be enhanced in several aspects and in terms of performance; nevertheless, we have managed to present two feasible real WSN scenarios where GOAT has served to determine the optimal MAC protocol and network design. Also, GOAT is intended to be an open project, and due to its software modularity, it is a prototype where new models and protocols can be included and improved.

This project has been a major learning experience for me. I have had to face several obstacles during the end to end GOAT development, which have allowed me to delve deep into software design and Java programming. Moreover, I have honed my knowledge about WSN, a field in which I am deeply interested and in which I expect to work in the near future.

Future work

The next steps for continuing with this project can be divided in two main points: code improvement and project sharing. Below, a list of steps for each of these points is presented.

- **Improve the analyzer:**
 - Parameters configuration from the GUI (avoid modifying hard-coded parameters).
 - Improve the GUI design achieving professional looking.
 - Add zoom feature.

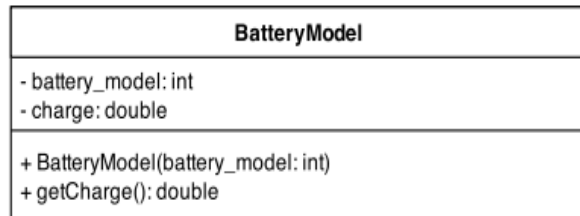
- Include packet collisions option for making the simulation results more accurate.
 - Implement a loader bar when a simulation carrying lots of data is being executed.
 - Allow up to 10,000 nodes simulations with proper performance.
 - Solve the issues related with the graphical collision among nodes.
 - Optimize window repainting and make it smoother.
 - Include more physical, battery, MAC and routing models.
 - Add charts and graphical results displaying at the GUI.
 - Display different types of node icons (instead of circles) with own hardware parameters.
 - Allow naming the nodes.
 - Implement pop-up windows with relevant data when the user hovers over a node.
- **Share the project:** Perform the needed steps for making GOAT open source in order to allow future collaborations.
 - Review the final code to ensure the code is idiomatic and follows best practices for variable names, whitespace, etc. for the Java programming language.
 - Determine software legal and property issues.
 - Upload the project's code to Github or similar platforms.
 - Identify the best way to promote collaboration (post at blogs, social networks, research conferences...).

ANNEX

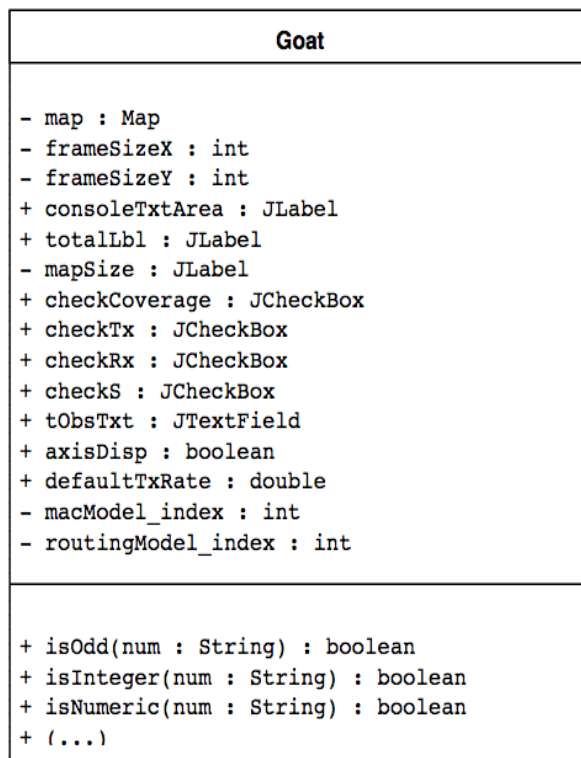
A1. UML Class diagrams

The classes' items and methods are shown in the UML class diagrams below.

a) BatteryModel.java



b) Goat.java



c) MacModel.java

MacModel
<pre> - mac_model : int - tObs : double - rTx : double - rRx : double - rOv : double - txTime : double - rxTime : double - samplingTime : double - idleTime : double - sleepTime : double - energy : double - lifetime : double - dataLength : int - ackLenght : int - (...) </pre>
<pre> - bmac() - bmacAck() - ah() + getEnergy() : double + getLifetime() : double </pre>

d) Map.java

Map
<pre> - canvas : Canvas - frameSizeX : int - frameSizeY : int + consoleTxtArea : JLabel + totalLbl : JLabel - mapSize : JLabel + checkCoverage : JCheckBox + checkTx : JCheckBox + checkRx : JCheckBox + checksS : JCheckBox + tObsTxt : JTextField + axisDisp : boolean + defaultTxRate : double - phyModel_index : int - batteryModel_index : int - macModel_index : int - routingModel_index : int </pre>
<pre> + addNode() + setSink() + setRate() + deleteNode() + simulate() + clearCanvas() + appendDisMat() + appendPowerMat() + appendReachMat() + saveFile() + openFile() + getPositionPerId(id: int): int </pre>

e) Node.java

Node
<pre> - id: int - kind: int - shape: Ellipse2D - color: Color - rate: double </pre>
<pre> + Node(id: int, kind: int, shape: Ellipse2D) + getId(): int + setId() + getKind(): int + setKind() + getRate(): double + setRate() + getColor(): Color + setColor() </pre>

f) PhyModel.java

PhyModel
<pre> - phy_model: int - nodeList: List<Node> - distance: List<List<Double>> - power: List<List<Double>> - reachability: List<List<Integer>> - powTx: double - powSens: double - freq: double - txG: double - rxG: double - loss: double - gamma: double </pre>
<pre> + PhyModel(phy_model: int, frequency: double, nodeList List<Node>, gamma:double) - freeSpace(powTx: double, dis: double): double - logNormal(powTx: double, dis: double): double + getDistance(): List<List<Double>> + printDistance(): String + getPower(): List<List<Double>> + printPower(): String + getReach(): List<List<Integer>> + printReach(): String + getTxPower(): double + getSensitivity(): double </pre>

g) Point.java

Point
<pre> - x: int - y: int </pre>
<pre> + Point(x: int, y: int) + getX(): int + getY(): int + toString(): String + distance(point: Point): double </pre>

f) RoutingModel.java

RoutingModel
<ul style="list-style-type: none"> - routing_model : int - topologyMat : List<List<Integer>> - nodeList : List<Node> - phyModel : PhyModel
<ul style="list-style-type: none"> + RoutingModel(int routing_model, PhyModel phyModel, List<List<Integer>> reachMat, List<Node> nodeList) - singleHop() - nextLevel() - powerLevel() + getTopologyMat() : List<List<Integer>> + printTopology : String

A2. Configuration variables

The table below lists the hard-coded variables values defined in the first version of GOAT.

Table 15: GOAT configuration variables

Java class	Parameter	Variable	Type	Default value
Goat	Frame size X	frameSizeX	int	1350
	Frame size Y	frameSizeY	int	600
	Sending rate (λ)	defSendingRate	double	10 pkts/h
	Canvas font size	canvasFontSize	int	12
Node	STA color	staColor	Color	(252, 201, 139)
	Sink color	sinkColor	Color	(100, 100, 100)
	Node width	w	double	50
	Node height	h	double	50
Battery	AA battery charge	charge2A	double	1500 mAh
	AAA battery charge	charge3A	double	7500 mAh

	Lithium battery		chargeLi	double	2200 mAh
PhyModel	Hardware	Transmission power	powTx	double	1 mW
		Sensitivity power	sensitivity	double	-81 dBm
		Transmission rate	rate	double	100 kbps
		Transmitting power consumption	pTx	double	60 mW
		Receiving power consumption	pRx	double	70 mW
		Sampling power consumption	pSampling	double	10 mW
		Idle power consumption	pIdle	double	8 mW
		Sleeping power consumption	pSleep	double	3 mW
	Friis	Transmission gain	Gtx	double	0 dBi
		Reception gain	Grx	double	3 dBi
		Systeml Loss	L	double	0 dBi
	Log - normal	γ	gamma	double	2.2 and 2.5
MacModel	Generic	Data length	dataLength	int	100 bytes
		ACK length	ackLength	int	14 bytes
	BMAC	Preamble length (BMAC)	lPreamble	int	128 bytes
		DC time awake (BMAC)	tDCAwake	int	2 ms
		DC time sleeping (BMAC)	tDCSleep	int	98 ms

	802.11 ah	RTS length	rtsLength	int	20 bytes
		CTS length	ctsLength	int	14 bytes
		Duration of a TIM group	tTim	int	200 ms
		Number of TIM groups	nTim	int	8
		Number of DTIM groups	nDtim	int	1000
		Duration of a DTIM group	tDtim	int	$tTim * nTim$
		Length of a TIM beacon	lTim	int	62 bytes
		Length of a DTIM beacon	lDtim	int	102 bytes
		Duration of a SIFS interval	tSifs	int	160 ms
		Duration of a DIFS interval	tDifs	int	264 ms
RoutingModel	-	-	-	-	-

References

- [1] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver. (October 2014). *The IEEE 802.11ah Wi-Fi Approach to M2M Communications*. University of Pompeu Fabra. Retrieved March 21 2015, from <http://arxiv.org/pdf/1402.4675.pdf>
- [2] Cisco. (February 2015). *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper*. Retrieved March 23 2015, from http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf
- [3] R. Cheng. (October 2011). *Wireless execs see connected devices as 'next big thing'*. CNET. Retrieved March 23 2015, from <http://www.cnet.com/news/wireless-execs-see-connected-devices-as-next-big-thing/>
- [4] S. R. Thampuran. (June 2005). *Routing Protocols for Ad Hoc Networks of Mobile Nodes*. University of Massachusetts. Retrieved April 7 2015, from <https://cs.wmich.edu/wsn/doc/adhocrouting/AdhocRouting.pdf>
- [5] A. Sahu, E. B. Fernande, M. Cardei, and M. VanHilst. (2011). *A Pattern for a Sensor Node*. Florida Atlantic University. Retrieved April 7 2015, from <http://www.hillside.net/plop/2010/papers/sahu.pdf>
- [6] E. E. Flores. (October 2012). *Wireless Sensor Networks applied to the Medicine*, master's thesis (pp. 5). University of Cantabria. Retrieved April 5, 2015, from <http://repositorio.unican.es/xmlui/bitstream/handle/10902/1288/349251.pdf?sequence=1>
- [7] S. Gajjar, N. Choksi, M. Sarkar, and K. Dasgupta. (2014). *Comparative analysis of Wireless Sensor Network Motes*. 2014 International Conference on Signal Processing and Integrated Networks (SPIN). Retrieved April 8 2015, from http://www.researchgate.net/profile/Mohanchur_Sarkar/publication/267393943_Comparative_analysis_of_WSN_motes/links/545254a70cf2bccc49089c73.pdf
- [8] ARM. (n.d). *ARM7 Processor Family*. Classic Processors. Retrieved April 26, 2015, from <http://www.arm.com/products/processors/classic/arm7/index.php>
- [9] A. Elsts, G. Strazdins, A. Vihrov, and L. Selavo. (2012). *Design and Implementation of MansOS: a Wireless Sensor Network Operating System*. University of Latvia. Retrieved April 26 2015, from <http://mansos.edi.lv/wp-content/uploads/2012/11/mansos-lu-2012.pdf>
- [10] R. Singh, and A. K. Virk. (July 2013). *Review of Key Management Schemes in WSNs*. International Journal of Science and Research (IJSR). Retrieved April 26, 2015, from <http://www.ijsr.net/archive/v2i7/MTIwMTM0NA==.pdf>

- [11] D. K. Gupta. (2013). *A Review on Wireless Sensor Networks*. International Conference on Recent Trends in Applied Sciences with Engineering Applications. Karnataka State Open University. Retrieved April 26, 2015, from <http://iiste.org/Journals/index.php/NCS/article/download/6062/6018>
- [12] F. Xia. (2008). *QoS Challenges and Opportunities in Wireless Sensor/Actuator*. Sensors. Queensland University of Technology. Retrieved April 26, 2015, from <http://arxiv.org/pdf/0806.0128.pdf>
- [13] M. Rouse. (n.d.). *Transceiver definition*. TechTarget. Retrieved April 26, 2015, from <http://searchnetworking.techtarget.com/definition/transceiver>
- [14] M. Zennaro, H. Ntareme, and A. Bagula. (2008). *On the design of a flexible gateway for Wireless Sensor Networks*. First International Workshop on Wireless Broadband Access for Communities and Rural Developing Regions. Retrieved April 27, 2015, from <http://arxiv.org/pdf/0806.0128.pdf>
- [15] M. Raluca, M. Razvan, and A. Terzi. (June 2008). *Gateway Design for Data Gathering Sensor Networks*. Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks. Retrieved April 27, 2015, from <https://www.cs.jhu.edu/~ralucam/papers/secon08stargate.pdf>
- [16] Cunningham & Cunningham, Inc. (November 2014). *Event Driven Programming*. Retrieved April 27, 2015, from <http://c2.com/cgi/wiki?EventDrivenProgramming>
- [17] P. Gupta, and R. Sangwan. (2014). *Wireless Sensor Network*. International Journal of Innovative Research in Technology (IJRT) 324. Retrieved April 27, 2015, from http://ijirt.org/paperpublished/IJIRT100225_PAPER.pdf
- [18] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver. (October 2014). *IEEE 802.11ah: The Wi-Fi Approach for M2M Communications*. University of Pompeu Fabra. Retrieved April 7, 2015, from <http://arxiv.org/pdf/1402.4675.pdf>
- [19] N. Srivastava. (February 2010). *Challenges of Next-Generation Wireless Sensor Networks and its impact on Society*. Journal of telecommunications. Retrieved April 7, 2015, from <http://arxiv.org/pdf/1002.4680.pdf>
- [20] M. Potnuru, and P. Ganti. (2003). *Wireless Sensor Networks: Issues, Challenges and Survey of Solutions*. Proceedings of the IEEE, University of Illinois Urbana. Retrieved April 8, 2015, from http://www.academia.edu/890321/Wireless_Sensor_Networks_Issues_Challenges_and_Survey_of_Solutions

- [21] S. Mitra. (n.d.). *Wireless Sensor Network*. Calcutta Institute of Engineering and Management. Retrieved 25 March 2015, from <http://ciemcal.org/wireless-sensor-network/>
- [22] B. Musznicki, and P. Zwierzykowski. (September 2012). *Survey of Simulators for Wireless Sensor Networks*. International Journal of Grid and High Performance Computing (IJGHPC). Retrieved 3 April 2015, from http://www.sersc.org/journals/IJGDC/vol5_no3/3.pdf
- [23] C. Cano. (2011). *Medium Access Control in WSNs*. Design of Telecommunications Infrastructures (DIT) subject material, University of Pompeu Fabra.
- [24] T. Adame, and E. Ducheyne. *DI.2 Update of system specifications*. ENTOMATIC project. Retrieved 21 April 2015, from <http://entomatic.upf.edu/>
- [25] G. Bauerfeind. (July 2012). *Which radio data transmission protocol matches to my application?*. Dresden elektronik ingenieurtechnik gmbh. Retrieved 6 May 2015, from http://www.sersc.org/journals/IJGDC/vol5_no3/3.pdf
- [26] Q. Wang, and I. Balasingham. (December 2010). *Wireless Sensor Networks*. Norwegian University of Science and Technology. Retrieved 2 May 2015, from <http://www.intechopen.com/books/wireless-sensor-networks-application-centric-design/wireless-sensor-networks-an-introduction>
- [27] J. Lee, and Y. Su. (November 2007). *A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi*. The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON). Retrieved 2 May 2015, from <http://www.cs.odu.edu/~nadeem/classes/cs795-WNS-S13/papers/advance-005.pdf>
- [28] M. A. Alim, M. M. Rahman, M. M. Hossain, and A. Al-Nahid. (2010). *Analysis of Large-Scale Propagation Models for Mobile Communications in Urban Area*. International Journal of Computer Science and Information Security (IJCSIS). Retrieved 25 April 2015, from <http://arxiv.org/pdf/1002.2187.pdf>
- [29] K. Lakhtaria. (March 2012). *Connectivity as a Fundamental Characteristic of Mobile Ad Hoc Networks*. Technological Advancements and Applications in Mobile Ad-Hoc Networks: Research Trends (pp. 82-84).
- [30] Essays, UK. (November 2013). *Analysis Of K Connectivity Computer Science Essay*. UKEssays.com. Retrieved 15 April, 2015, from <http://www.ukessays.com/essays/computer-science/analysis-of-k-connectivity-computer-science-essay.php?cref=1>
- [31] R. Mathar. (November 2009). *Wireless Channel Modeling and Propagation Effects*. RWTH Aachen University. Retrieved 15 April 2015, from https://www.ti.rwth-aachen.de/teaching/ti/data/channel_modeling.pdf

- [40] R. Manikandan, and K. Selvakumar. (May 2013). *Power Optimistic with Throughput Improved Adaptive CSMA MAC Protocol Design for Wireless Ad Hoc Network*. International Journal of Computer Applications. Retrieved 14 April 2015, from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.404.1033&rep=rep1&type=pdf>
- [41] J. Polastre, J. Hill, and D. Culler. (2004). *Versatile low power media access for wireless sensor networks*. In: Proceedings of the Sensys'04, San Diego, CA, 2004. Retrieved 14 April 2015, from <http://www.cs.berkeley.edu/~culler/papers/sensys04-bmac.pdf>
- [42] T. Adame, A. Bel, B. Bellalta, J. Barcelo, J. Gonzalez and M. Oliver. (October 2014). *CAS-based channel access protocol for IEEE 802.11ah WLANs*. European Wireless (EW) 2014. Retrieved 8 April 2015, from <http://arxiv.org/pdf/1402.4675.pdf>
- [43] University of Pompeu Fabra. *ENTOMATIC: bioacoustic identification of the olive fruit fly*. Project. Retrieved 23 February 2015, from <http://entomatic.upf.edu/project>

