Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

Energy-efficient wireless sensor networks via scheduling algorithm and radio Wake-up technology

Relatore:
Chiar.mo Prof.
Marco Di Felice

Correlatore:
Chiar.mo Prof.
Tullio Salmon Cinotti

Presentata da:
Luca Sciullo

Sessione II
Anno Accademico 2016-2017

Ne pudeat quae nescieris, te velle doceri.
Scire aliquid laus est, culpa est nihil discere velle.

Cato Censor

Abstract

One of the most important requirements for wireless sensor networks (WSNs) is the energy efficiency, since sensors are usually fed by a battery that cannot be replaced or recharged. Radio wake-up - the technology that lets a sensor completely turn off and be reactivated by converting the electromagnetic field of radio waves into energy - is now one of the most emergent strategies in the design of wireless sensor networks. This work presents Scheduled on Demand Radio Wake-Up (SORW), a flexible scheduler designed for a wireless sensor network where duty cycling strategy and radio wake-up technology are combined in order to optimize the network lifetime. In particular, it tries to keep sensors sleeping as much as possible, still guaranteeing a minimum number of detections per unit of time. Performances of SORW are provided through the use of OMNet++ simulator and compared to results obtained by other basic approaches. Results show that with SORW it is possible to reach a theoretical lifetime of several years, compared to simpler schedulers that only reach days of activity of the network.

Index Terms – Wireless Communication, Wireless Sensor Network, Energy Efficiency, Scheduler, Radio Wake-Up, Duty Cycling.

Contents

Chapter 1

Introduction

Today, Internet of Things (IoT) is used as a catchphrase by many sources. This expression encompasses a galaxy of solutions and technologies somehow related to the world of intercommunicating smart objects, i.e., devices with a unique identity capable of being addressed through the network, with the ability to sense and/or perform some tasks, perhaps remotely controlled.

The technological improvements of the last years made possible to virtually turn everything into a smart object, leading to the definition of smart environments, i.e., settings where IoT is particularly suitable, such as cities, houses, factories, fields, and so on, and where a large number of smart things collaborate together and interact to provide a particular service. This wide heterogeneity unravels new challenges for researchers and developers on a daily basis and despite a constant growth in its utilization, IoT still faces very important issues that sometimes, together with strict requirements, need a different solution for each kind of application.

Wireless Sensor Networks (WSNs) can be considered as a particular instance of the IoT world, but because of their nature, they often address even more difficulties and stricter requirements.

One of the fundamental requirements of sensors, for instance, is the need of very efficient strategies for optimizing the energy consumption. In fact, they often use only a small battery as power supply and in environments like forests or underground it is impossible to recharge or replace. One of the most effective ways for reaching this goal is to optimize and reduce as much as possible the radio transmissions, since they require a bigger amount of energy compared to all the other operations. The radio wake-up is a set of technologies that let sensors completely turn off and be awaken when needed. In particular, they are equipped with a radio interface that is able to convert the electromagnetic fields of radio waves into energy required for the boot phase. Of course, this implies a significant amount of energy saved, especially for those applications that require long-term detection and hence sensors are used less frequently.

This work is based on a wireless sensor network with the following characteristics: for each unit of time, a number λ of different detections are required from λ different sensors. Furthermore, sensors are equipped with a radio wake-up system that lets them completely turn off or be awaken, depending if it is their turn to send data or not. For this reason, they can assume the *Switch (SW)* or the *Low Power (LP)* mode. The first means the sensor is turned off and requires a defined amount of irradiated energy for booting. The second means the sensor is in sleep mode, consuming a constant amount of energy also in case it is not used. In particular, the objective of this work is to find a good algorithm to maximize the network lifetime, i.e., its period of activity, by scheduling the utilization of sensors and keeping those not involved in sleeping mode. Then, through the radio wake up system, sensors can be awaken on demand.

For this purpose, first I will model the generic problem for the WSN with such characteristics, then I will introduce the *SORW* (scheduled on demand radio wake-up) scheduler, my proposal as solution for this problem.

In particular, I will describe the algorithm it is based on and I will implement it in the OMNet++ simulation environment, in order to understand how it would actually work.

In the end, I will compare the results obtained by simulations both with the analytic results of the SORW algorithm and some other algorithms based on comparable approaches.

The rest of this document is organized as follows: chapter 2 introduces the IoT world with its application domains, its challenges, and an overview of the wireless technologies involved, classifying them based on different criteria. Chapter 3 introduces the wireless sensor networks, specifically treating which solutions exist to maximize the lifetime of the network. Chapter 4 is about the definition of the generic problem this work is based on and which particular instance of it. The SORW scheduler is presented as a solution to the problem, together with its analytic point of view. More in detail, this scheduler is based on an algorithm that takes the best from both the on demand and the scheduled schemes for the WSN schedulers and merges them with the radio wake up technology. In order to understand how SORW performs over realistic network scenarios, it is implemented in the OMNet++ simulation environment and compared with other kinds of algorithms. Chapters 5 and 6 respectively explain how simulations are built and what results are obtained. In particular, the first briefly describes how OMNet++ works and how SORW is implemented by using this framework. In the second, SORW is compared with other algorithms, while varying some factors like the number of different detections requested per unit of time, the network size or the duration of the interval between two different requests. Chapter 7 is about the conclusions and the future developments of this project.

Chapter 2

Internet of Things

2.1 Definition

Internet of Things (IoT) represents an innovative paradigm in the Information and Communication Technology (ICT) world and a rising field with a strong impact on global economy (about 11 trillions of dollars before 2025 [3]).

IoT is based on the idea that every thing or every physical object can be interconnected at the same time, regardless of its nature and its geographical position, by unique addressing schemes. Most of the times, devices are connected to the Internet network, becoming part of a wide digital ecosystem.

According to the Cluster of European research projects on the Internet of Things [32] - 'Things' become active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information sensed about the environment, while reacting autonomously to the real/physical world events and influencing it by running processes that trigger actions and create services with or without direct human intervention -.

This perspective turns common devices into smart devices, where smartness is the ability of a device to operate interactively and autonomously. In particular, devices can sense (through sensors), elaborate, and communicate some kind of information, or, given a precise input, perform some specific actions (through actuators).

The idea of smart devices leads to a natural definition of smart environment. By [6], "A smart environment is that making its "employment" easy and comfortable thanks to the intelligence of contained objects, be it an office, a home, an industrial plant, or a leisure environment."

In order to make a thing smart, there are some steps to follow:

- give the thing a unique identity
- give the thing the ability to communicate
- give the thing senses, putting sensors on/in it
- give the thing the chance to be remotely controlled, by using very small embedded electronics

2.2 Technologies

IoT has become so important with the technological improvements of the last 15 years, that it has made possible almost for everyone to follow the process for making things smart.
Day after day embedded devices become smaller and cheaper and a huge number of new technologies are unveiled.

Some macro-areas of technologies used in IoT can be identified, especially technologies for identification and sensing, and for communication.

Identification and sensing One of the first technologies that has been used, even before the concept of IoT, was RFID (Radio-frequency Identification). RFID is a technology useful for identifying and tracking entities using electromagnetic fields. In particular, it counts two different elements: tags and readers. Tags are labels physically attached to what has to be identified and they contain an electronic circuit that is activated by the induction of an electromagnetic field. Once a tag has been activated, it transmits unique information stored on board. There are two main types of tags: passive or active. Passive tags only use radio energy transmitted by readers, while the active ones have a battery and periodically transmit on their own. Tags can also be read-only or read/write. In the first case, a database is required to keep track of the association ID-tag, while in the second case it is highly customizable. Readers are responsible for tags activation by irradiating them with some radio waves and/or for reading data transmitted by them.

Another common way to identify objects is by using IPv6. In fact, while IPv4 was able to identify a group of cohabiting devices and was strongly depending on a gateway for distinguishing a single device, IPv6 with its 128 bits can define 10^{38} different addresses and so it should be enough to identify each single object.

Thousands of different sensors exist (see chapter 3 for more details), for several different applications, but the crucial innovation of IoT is the ability to connect

them to the network for sharing and analyzing all the collected data. For this reason, a lot of new platforms have been released, with the chance to connect sensors and to be connected to the Internet by using network interfaces already on board. These platforms are small and cheap (with the size of a credit card and a price below 10 dollars), as for instance the recent Raspberry Pi Zero or the new C.H.I.P. one.

Communication The most common way for making devices communicate is by using wireless technologies. For this reason, several protocols exist with different characteristics, depending on multiple factors. A first discriminant for differentiating the technologies is the Spatial Range, i.e., the distance at which they work. Based on their communication range, technologies can be used for different kinds of networks, like Wireless Body Area Network (WBAN), or Wireless Personal Area Network (WPAN), or Wireless Local Area Network (WLAN), or Wireless Wide Area Network (WWAN). Depending on the network the technologies belong to, they can be categorized as Proximity technologies, or Short Range ones, or Long Range ones. The first, like RFID or NFC (Near Field Communication), typically have a range of a few meters and they are usually used for small data transfers or identification. The second, with a range going from a few meters to around a hundred, are typically suitable for areas like rooms, small buildings or a house. For instance, the Bluetooth (BL) standard and all the IEEE 802.15.4-based technologies belong to this family. The third, finally, are used for covering areas of kilometers. They are suitable for applications that serve factories, rural fields, big buildings and so on. These technologies count all the cellular ones (2G, 3G, 4G) and all those used for LPWAN (Low Power Wide Area Network) networks, as for instance LoRa and SigFox.

Another way to differentiate the technologies involved is by clustering them depending on their data rate. The data rate determines what a technology is developed for. It ranges from a hundred bps to more than 10 Mbps.

Technologies can require different network topologies. There are three main types: the star topology, the hierarchical tree topology, and the mesh topology, as shown in fig 2.1. In the star topology, a central node acts as a sink for the network, while the other nodes are directly connected to the central one, without being connected to each other. Most of the times, the sink node is also the gateway to external networks. A sub-case of the star topology is the star-of-stars topology, where more different star networks are related in some way. The mesh topology consists of nodes directly connected to each other, even if only few of them are connected to the one that acts as sink. All the others can reach the sink node through a multihop path starting from their neighbors. In the hierarchical tree topology, nodes are directly connected following a tree structure, consisting of

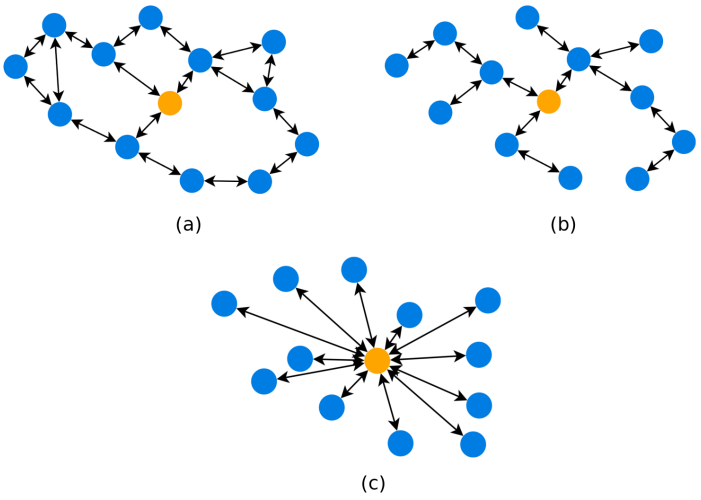

Figure 2.1: Image showing a view of different topologies. (a) Mesh topology, (b) Hierarchical tree topology, (c) Star topology.

layers and a main node (the root). The root acts as sink and gateway for the entire network.

Wire technologies are very important in the networking world as well as in the IoT one. Nevertheless, I will not discuss them, given their already consolidated use.

2.3 Applications

The potential of IoT allows the development of a huge number of applications, even if only few of them are currently available. The domains and the usage environments of IoT systems are really diverse, but all of them share the ability to improve the quality of human lives.

A European Consortium called IoT-A, made by both companies and universities, worked from 2010 to 2013 on a reference Model and Architecture for IoT. They focused on a reference Architecture for IoT in order to describe essential building blocks as well as design choices to deal with conflicting requirements regarding functionality, performance, deployment and security. For this reason, firstly they identified the main applications domains as well as the main environments for the IoT world, as shown in Figure 2.2. Most of them are also described in [6], [10].

Figure 2.2: Image from [13] showing a view of the main IoT environments/applications domains and some of the most used technologies.

Healthcare In the healthcare environment, there are many benefits that can be provided by IoT and, according to [6], they can be grouped mostly into:

- *Tracking*: like in all the other domains, the most useful benefit of tracking is related to the possibility to monitor and improve the workflow. In such sense, it means being able to follow patients in hospitals, or checking the access to designated areas, as for instance operating theaters.

- *Identification and authentication*: by the patients' point of view, the identification is useful for reducing incidents that can be very harmful for them, like wrong drug prescription, or dosage, or procedure, etc.. By the staff's point of view, identification and authentication grant access to specific areas in a more comfortable and efficient way.

- *Data collection*: the automatic data collection reduces processing time and

helps improve the goods supply, by making it faster and more efficient.

- *Sensing*: one of the most interesting fields where IoT is improving healthcare is sensing. Since sensors have become very small and less invasive, it is now possible to place them inside a body in order to provide real-time information on patient health indicators. These values can be monitored remotely and/or by the patient using a smartphone.

Sell Retail Sell Retail is a new frontier for smart stores. The idea is to put a tag on every good in order to be able to track it when people go shopping. In this way, it is possible to keep track of which products are taken, when they are bought and where. Check-out assistants can be removed, since there are some sensors before the exit that scan the tags inside the shopping bag and ask for an automatic payment directly on the client's smartphone. People not only save time, but can continuously access product information by using a smartphone that is able to read the tag. Recently, at the end of 2016, Amazon opened the first Amazon Go store[1] in Seattle (USA), a prototype grocery store where different technologies, like computer vision, deep learning algorithms and sensor fusion, made possible to test this kind of market for the first time in history.

Smart Transports Smart Transports are radically changing the way of thinking about transports. The chance to spread sensors and actuators over roads and rails means that people can control transportation vehicles to better route the traffic, provide the tourist with real-time transportation information, help in the management of the depots, and monitor the status of transported goods, especially if they are equipped with some tracking tags. But Smart Transports also include the assisted-driving world and the automatic one. Research in these fields is extremely important nowadays, with companies like Google and Tesla, for instance, investing billions of dollars in self-driving cars.

Logistics The idea of logistics is to improve the workflow. This can be realized by keeping track of every step of a process in order to analyze it and understand where it can be enhanced. In industrial scenarios, tags are associated with what needs to be monitored. After each step of the production, the tag is read by a sensor that can collect several information about that specific object. Another interesting scenario of logistics is the monitoring of goods transportation. Goods are associated with tags that can be read by sensors put on the vehicles used for transportation. Through the vehicle's GPS, products can be monitored in real-time during their travel, as well as in the moment they reach the destination.

[1] https://www.theverge.com/2016/12/5/13842592/amazon-go-new-cashier-less-convenience-store

Smart Cities The IoT paradigm can also be applied for cities, giving them several benefits [39]:

- *Structural Health of Buildings*: different kinds of sensors can be used in order to monitor the health of buildings, especially the historical ones. Vibration and deformation sensors are suitable for monitoring a building's stress level, atmospheric agent sensors in the surrounding areas for pollution levels, and temperature and humidity sensors for the environmental conditions.

- *Waste Management*: in order to improve the quality of recycling and to reduce the cost of waste, cities can use smart waste containers, which detect the level of load and allow for an optimization of the collector trucks route. Sensors can detect how much a container is full and can communicate it to people in charge of the waste service.

- *Monitoring*: some important factors can be monitored in cities, like the pollution level (or the quality of air), and the noise rate. Also in this case, some sensors are used in specific positions of the city in order to collect reasonable data about what is monitored. Data collected together can give an overview of such levels, letting the city government know right away about a problem and decide about possible solutions.

- *Smart Lighting*: since the optimization of costs is fundamental for public administration, the smart management of resources like electricity can save a lot of funds. Smart lighting reduces the electricity consumption by switching lights on and off or optimizing their intensity depending on some precise factors and smart policies. For instance, lights can be immediately turned off if there is no presence of people nearby, or in the morning after a specific value of natural light has been reached.

- *City Energy Consumption*: related to the energy optimization, cities can monitor the instant energy required by different services (public lighting, transportation, traffic lights, and so on) in order to set priorities in the resources management.

Smart Houses/Domotics Sensors and actuators spread over the house can help people make their lives more comfortable, most of the times by automating several processes that are usually done by hand. The room lighting can be automatically adjusted depending on the time of day, the heating system can behave according to the residents' preferences, and possibly everything can be managed remotely by a smartphone. By using this kind of systems, people are also able to save money and energy since they can manage their resources according

to some optimized policies. In the last few years, some of the most important companies in the ICT world, like Alphabet, Apple, Amazon, etc.., released several products for making the house smart[2,3,4]. Most of them are plug-and-play but require important investments at the beginning, while homemade solutions are quite cheap but require more effort.

2.4 Challenges

Since IoT is a recent field of research and because it involves different areas of ICT, like networking, data science, cloud computing, artificial intelligence, and so on, there are several challenging issues which need to be addressed, both from a technological point of view and concerning its social impact on humanity. Actually, technological and social challenges can be considered as requirements needed for a particular IoT environment, even if social challenges also represent a way of measuring how much a system weighs on people.

As technological requirements:

- *Low Power Consumption*: this is an important feature, especially for sensors and/or actuators that are powered only by batteries, maybe because of the physical distribution of the device. In this case, as explained in 3.3, various solutions have been deployed. One of the most common is trying to reduce the duty cycle of a device as much as possible. This can be obtained by switching it on or off depending on whether it has to be used.

- *low cost*: in order to obtain useful data, in most cases IoT networks involve a high number of nodes, i.e., a high number of devices. For this reason, necessarily, the cost per unit has to be as affordable as possible.

- *scalability*: an IoT application can require a scalable network, where the number of nodes can be increased over time. For this reason, since increasing nodes means also increasing possible collisions, load balancing issues, and bigger deployment costs, it is fundamental to keep the reconfiguration efficient.

- *Reliability*: it is an important feature for almost every kind of application. In particular, in some IoT environments, like in monitoring contexts, reliability is considered as a fundamental constraint.

- *Low latency*: latency can adversely influence the behavior of some applications and for this reason low latency is considered a very important requirement.

[2]https://madeby.google.com/home/
[3]https://www.apple.com/ios/home/
[4]https://www.amazon.com/smart-home/b?node=6563140011

It depends on several aspects, like the link strength between endpoints, or the number of interferences, or the choice of the MAC layer access method, etc..

- *Compatibility*: many companies are releasing new hardware and software for the IoT world. Compatibility becomes very important especially for the integration of these new solutions in the already existing ones.

- *Security*: it has to be considered a very strict challenging issue, because of the nature of IoT networks. In fact, these are exposed to several kinds of attacks that can compromise the CIA triad (confidentiality, integrity, and availability). In particular, some of them, like DoS (Denial of Service), can be made more easily by using new weak points, e.g., the packet fragmentation (which may involve long cryptoblocks)

As social requirements that can impact the humanity perspective:

- *Security*: it is also a social requirement, since it determines the level of trust of a service from the users' point of view. Some kinds of applications, like home automation for instance, require a very strong security level. But security for people also implies a guarantee that their data can be considered protected and that it cannot be stolen by someone else.

- *Privacy*: it is a concept deeply rooted in our society and recognized by all the legislations of civilized countries. Every application should explain what kind of private data it will be using, what kind of usage it will make of it, and if it will persistently be storing it. In the IoT context this requirement is amplified by the fact that a lot of different data is collected everywhere, sometimes even if people are not directly using any IoT service. For instance, people could be recognized and tracked by some private security systems without knowing.

Requirements have a different importance depending on the domain they belong to. Table 2.1 shows how the importance of requirements changes for some domains, previously discussed in 2.3. Security and reliability can be considered fundamental requirements for each domain, since both security and reliability faults can compromise the entire system, and in some cases it could mean danger for people. Another generic consideration is that costs should always be as low as possible, but sometimes, especially in very important situations like health, this can be considered a minor constraint.
In healthcare IoT systems all the qualitative requirements are fundamental, since errors can mean health problems for patients. Privacy is very important too.
Retail is quite more flexible, because only Low latency and Privacy can be considered

really important requirements. The first is due to the fact that several operations in this domain require an instant feeling for clients, while the second one to the fact that people's shopping is considered sensitive information that has to be protected. Logistics require that new solutions, both by the hardware point of view and the software one, can be applied in the future without the need to reconfigure all the entire system. For this reason, Compatibility and Scalability can be considered important requirements for this domain.
Privacy and low costs are very important requirements for smart cities and home automation, since most of the times these domains treat very sensitive information and have to deal with limited budgets, especially when Public Administration is involved.

Domain / Requirement	Healthcare	Sell Retail	Logistics	Smart Cities	Home automation
Low Power Consumption	Required for personal/wearable devices, since battery replacement could require some invasive surgery	Not necessarily required	Not necessarily required	Not necessarily required	Required, since low power consumption also means reduced costs for clients
Low Cost	Not necessarily required	Not necessarily required	Not necessarily required	Required, since Public Administration always tries to save funds	Required in order to make a home automation system affordable for everyone
Scalability	Not necessarily required	Not necessarily required	Required, especially when the same solution has to be replicated	Required, especially when the same solution has to be replicated	Not necessarily required
Low Latency	Strongly required, since timeliness represents a fundamental requirement in every healthcare system	Required, especially for some operations that have to give an immediate feeling to clients	Required, especially when collected data has to be analyzed in real-time	Not necessarily required	Required, especially when the home automation system includes some Security feature for the house
Compatibility	Required when devices replacement could require some invasive surgery and some new features have to be integrated in the already existing ones	Not necessarily required	Required if new features have to be integrated in the already existing ones	Required if new features have to be integrated in the already existing ones in order to save funds	Required if new features have to be integrated in the already existing ones
Privacy	Strongly required, since privacy is a fundamental requirement in every healthcare system	Required, since people's shopping is a sensitive information	Not necessarily required	Strongly required especially for services that involve citizens	Strongly required, since a house is the most intimate place for people

Table 2.1: View of how requirements have different importance depending on the application domain they belong to. Security and reliability are considered strongly required for each domain, so they are not included in this table.

Chapter 3

Wireless Sensor Networks

Even if wireless sensor networks can be considered as part of the IoT ecosystem, they have gained so much importance that they need to be treated as a separate world.

The concept of wireless sensor networks has been made possible by the micro-electro-mechanical systems (MEMS) advances of recent years and the improvements in the wireless communications, that enabled the development of cheap, multifunctional and low-power devices.

A sensor network consists of a large number of sensor nodes, which are usually arranged close to a phenomenon or inside it. These nodes can sense, make some data processing, communicate and potentially perform some operations through specific actuators. Battery is the main power source and so power efficiency and low energy consumption are considered strong requirements for WSN, although there are other kinds of sources that can harvest power from the environment, as for instance solar panels.

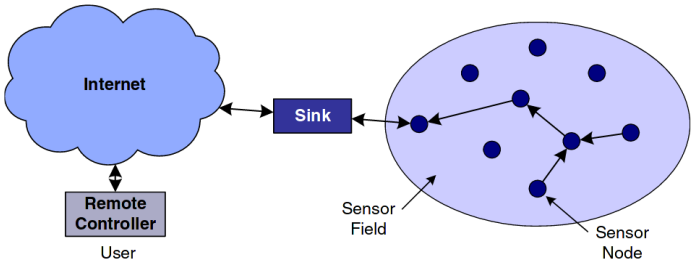

Figure 3.1: Image from [5] showing a WSN generic architecture. In a sensor field there are some nodes that (usually) communicate through multi-hop, until they reach the sink node. The sink node acts as collector of information coming from sensors and as gateway, i.e., it is in charge of forwarding network packets to external networks, as for instance the Internet one.

Most of the times, sensors send their data to a sink node, that is in charge of collecting data and possibly spreading it in the sensor network or in an external

one, acting as gateway for the WSN. In figure 3.1, a generic architecture for WSNs is shown. In a sensor field some sensor nodes are deployed according to different policies or application requirements.

Many protocols and algorithms from traditional wireless ad hoc networks have been proposed, but they do not seem to be suitable for the WSN applications, since sensors networks and ad hoc ones have different characteristics. According to [4], [38], the differences can be summarized in this way:

1. the number of nodes in a sensor network can be several orders of magnitude bigger than the one of ad hoc networks
2. sensor nodes usually have a reduced power, memory and computational capacity than nodes in an ad hoc network
3. topologies of sensors networks are frequently inclined to change
4. sensors networks have to deal with a high number of failures
5. sensor nodes are densely deployed

3.1 Sensor

Platforms available for WSN support a huge number of different sensors. It is a big challenge to directly integrate them on-board, since platforms and sensors have different radio components, processors, and storage. For this reason, the embedded software, like the OS, must be designed to deal with several different hardware, trying to optimize the computational costs and to maximize the power life of the device.

There are several kinds of sensors, in order to be able to monitor for instance:

- lighting conditions
- pressure
- humidity
- temperature
- noise level
- pollution level
- the presence/absence of particular objects
- vibrations

- soil conditions
- speed, directions, etc..

As shown in figure 3.2, a sensor is composed of four main components: a *sensing unit*, a *process unit*, a *transceiver unit*, and a *power unit*, but other optional components can be added, like a *location finding system*, a *power generator*, and a *mobilizer*.

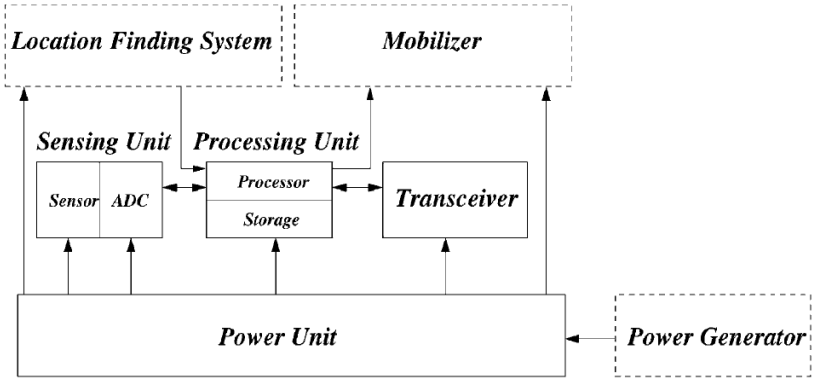

Figure 3.2: Image from [4] showing the components of a sensor. There are four main components: a *sensing unit*, a *process unit*, a *transceiver unit*, and a *power unit*. There can also be additional components, like a *location finding system*, a *power generator*, and a *mobilizer*.

The sensing unit is made up of sensors and analog to digital converters (ADCs). Sensors produce an analog signal of the observed phenomenon that is converted into a digital one by the ADCs, before being processed by the processing unit. This one makes some computations with digital signals, in order to carry out the assigned task, and potentially stores the collected information on a little memory. The transceiver unit connects the node to the network, enabling the communication with other sensors. The power unit is usually connected to a battery, but it can also be supported by a harvesting unit, such as a solar panel. If the assigned task depends on the knowledge of the location, a location finding system can be included in the sensor unit. Finally, a mobilizer may be needed if sensor requires movements to achieve its goal. All the subunits should fit into a small module, sometimes even smaller than a cubic centimeter.

3.2 Topology and environments

WSNs are usually composed by few tens to thousands of sensor nodes working together, most of the times with little or no infrastructure at all. Furthermore, they can also be structured or unstructured: the first are the ones that contain a

sparse collection of nodes, while the second contain a dense collection. The way in which nodes are deployed influences the network topology, that is a very important aspect of WSNs. Since nodes can be inaccessible and unattended, with or without an infrastructure, the topology design and maintenance have to be studied a priori. There are two ways for spreading nodes: in an ad-hoc manner or one by one, also called pre-planned manner. The first way means dropping nodes from a plane, or throwing them by catapult or missile, without having the absolute certainty of the positions the nodes will assume. The second one means placing sensors one by one either by a human or a robot, according to a well designed initial plan. After the deployment of the network, its topology can be modified by some changes in nodes, like their position, their reachability, their movements, their residual energy, and so on. The placement and the number of sensors in a network determines the *degree of network coverage*. Basing on the kind of applications, a higher degree of coverage should be required, in order to achieve more accurate values of data.

As described by [38], WSNs can be deployed on land, underground and underwater. Basing on the environment, a WSN faces different issues and constraints. There exist five types of WSNs: terrestrial, underground, underwater, multi-media and mobile ones.

Terrestrial WSNs In a terrestrial WSN, there are hundreds to thousands of wireless sensor nodes deployed in a specific area. Reliable communication, especially in a dense environment, becomes very important. Since battery power is limited and may not be recharged due the position of the sensor, it can have a secondary power source in order to harvest energy from the environment, like solar panels. In any case, sensor nodes should conserve as much energy as possible. For this reason, several strategies have been studied and are used, like multi-hop optimal routing, short-transmission range, low duty-cycles, etc..

Underground WSNs In an underground WSN, several sensor nodes are buried underground, like under a field, or a cave, or a mine. There can be some sink nodes above the ground for collecting data from sensor nodes. Of course, the underground WSN is more expensive than a terrestrial one, because its costs for the maintenance and deployment, but also because the equipment has to be very specialized in order to ensure communication through soil, rocks, other minerals, and so on. For the underground WSNs the biggest challenge is reliable wireless communication, since there are several chances of signal losses and high levels of attenuation.

The deployment of sensors requires a precise planning and an important energy consideration. Like in the terrestrial WSNs, nodes are equipped with a limited power source, but there are no possibilities for replacing or recharging it and

almost none for harvesting energy from the environment.

Underwater WSNs In an underwater WSN, several nodes and some vehicles are deployed underwater. Unlike terrestrial WSNs, sensor nodes are very expensive and only few of them are used, arranged in a sparse deployment. Vehicles are autonomous and useful for explorations or collecting data from sensors. The biggest challenge for this kind of network is the wireless transmission, since the communication is established by acoustic waves. The acoustic communication faces several issues, like limited bandwidth, long propagation delay and signal fading, and the entire network encounters many failures due to environmental conditions. Also in this kind of network sensors are connected to a battery power source, and there is no chance at all to replace it or recharge it.

Multi-media WSNs In a multi-media WSN there are several low cost sensors equipped with cameras and microphones, whose goal is to monitor and track events in the form of multimedia, like video, imaging and audio. Sensors are interconnected to each other through a wireless communication, in order to retrieve, process and store collected data.

One of the most important requirements for this kind of network is the perfect coverage of the interested area and for this reason nodes are deployed into the environment in a pre-planned manner.

The biggest challenges for multi-media WSNs are related to the quality of service (QoS), the high demand of bandwidth and the high consumption of energy.

Mobile WSNs In a mobile WSN nodes can move by themselves and can interact with the physical environment. Furthermore, they act as static nodes, i.e., they can sense, compute, and communicate.

Through autonomous movements nodes have the ability to reposition and organize themselves, possibly changing the initial network topology. Communication is made possible when mobile nodes are within range of each other.

The most important challenges are the nodes' localization, the self-organization, the autonomous navigation and control, the coverage of the area, and the energy management and optimization.

3.3 Network Lifetime

As previously explained, one of the most important requirements of WSNs is to maximize the network lifetime, i.e., the maximum period of time the network is able to perform its task. This is a fundamental requirement especially in those

environments in which it is impossible to recharge or replace the battery of sensor nodes, like underwater, underground, etc..

According to [5], the following remarks about energy consumption of a sensor need to be taken into account:

- the communication components of a sensor consume much more than the computation ones. Transmitting only one bit has the same order of consumption as executing a few thousands of instructions. For this reason, it is better to trade communications for computations.
- reception, transmission, and idle states of radio components is of the same order of magnitude. Instead, the sleep state requires at least one order of magnitude less. This means that radio should be turned off whenever possible.
- sensing components can require a high consumption of energy, so they should be used only when required.

Basing on [27], another key point for saving energy is to reduce the extra load on sensor nodes in a bottleneck zone, by using some efficient bandwidth utilization schemes. A bottleneck zone is defined as the area close to a sink node, where network traffic is significantly increased, since it follows a many-to-one pattern. Nodes in the bottleneck zone consume all their energy very quickly, a problem known as the *energy hole problem*. A failure of such nodes can lead to a wastage of network energy and network reliability.

Several solutions have been studied and proposed in order to increase the network lifetime. In particular, I report the low duty cycle strategy and the network coding technique: the first is thought to decrease the energy consumption by turning off the radio components of a sensor according to some precise policies, while the goal of the second is to decrease the amount of data flow in bottleneck zones, in order to avoid the energy hole problem.

3.3.1 Low Duty Cycle

The duty cycle of a sensor is the ratio between the time of activity and the period of time, and it is typically referred to the radio subsystem of a sensor. For example, with a 1% of duty cycle, a sensor node has the radio subsystem active for just 1% of the time. As previously explained, this solution leads to a very effective power save, since it reduces the idle listening, i.e., when the radio transceiver is waiting in vain for a frame, or the overhearing time, i.e., when a node wastes energy listening to an uninteresting frame.

Even if duty cycling seems to be a very intuitive solution, as stated in [8], it has some downsides here briefly discussed.

End-to-end message delay In a duty cycling multi-hop network, it can happen that a packet has to wait for the next hop to wake up. This effect is called *sleep waiting*, and since it considerably increases the latency, it is not acceptable for some kinds of application (see table 2.1 for instance).

Collision Rates Duty cycling implies shorter transmission and reception time windows, and so if a contention-based medium access control (MAC) is used, smaller windows mean an increment in the probability of collisions.

Control packet overhead Duty cycling can require a synchronization among nodes, increasing the extra control traffic, especially in fine-grained synchronization schemes, where frequent resynchronizations are done to deal with clock skews.

The concept of duty cycle is strictly connected to the MAC protocol a sensor uses, since it determines how to use the radio subsystem, and for this reason since the first outset of sensors networks several different MAC protocols have been studied and improved. These MAC protocols are based on different strategies and schemes. In particular, they can be mainly grouped into synchronous schemes and asynchronous ones. In the first approach nodes are time-synchronized, so they keep a global time, while in the second one there is no common clock. Actually, since pair-wise synchronization is easier to obtain than a global one, an hybrid approach, also called semi-synchronous scheme, groups neighbors into synchronized clusters that communicate asynchronously with each other. Figure 3.3 shows how different duty cycling schemes can be grouped.

3.3.1.1 Synchronous Schemes

In synchronous schemes nodes keep a common time reference, that is not necessarily a global time, but it implies an extra exchange of information. These schemes can be sub-grouped into the rendezvous (or strictly synchronous) and the skewed/staggered ones.

Rendezvous or Strict Synchronous Schemes The rendezvous scheme is the most intuitive one, since all nodes turn the radio on/off at the same time. This approach is very hard to achieve in a multihop environment, where pair-wise synchronization errors tend to be bigger. Most of these schemes are based on the TDMA (Time Division Medium Access), i.e., a contention free MAC protocol, like TRAMA (traffic-Adaptive MAC protocol) [26]. TRAMA tries to save energy by removing possible collisions and by putting nodes in sleep mode if they do not participate in a communication. It assumes that the global synchronization is

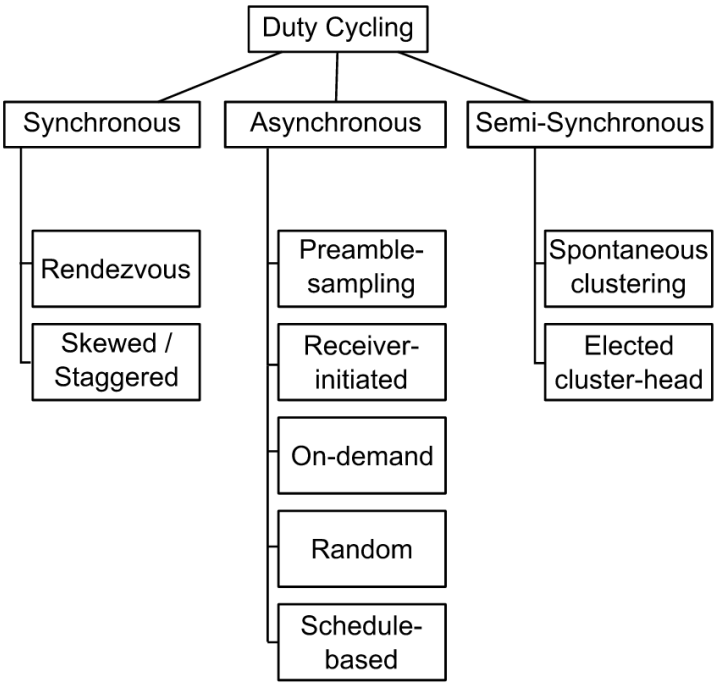

Figure 3.3: Image from [8] showing how different duty cycling schemes can be grouped. The main distinction is on the fact if they are synchronized or not.

provided by other mechanisms, and despite this, it reaches only a 12,5% of duty cycling.

With a given global time synchronization, rendezvous schemes would be advantageous, since they precisely coordinate transmissions and reduce idle listenings and collisions at the same time. However, global synchronization is almost never provided and depends on extra hardware and/or extra control messages.

Skewed/Staggered Schemes Rendezvous schemes can suffer from a problem that previously was defined as *sleep waiting*, i.e., when a frame gets stuck during a multihop path because the next node goes to sleep according to its duty cycle policies. Skewed/staggered schemes were designed to solve this problem by forming a network topology tree, with the sink as root, and scheduling nodes' wake up in a ladder-like way, following their depth in the tree.

A first solution was DMAC [19]. DMAC sets the nodes' awakenings with an offset based on the number of hops between them and the sink but suffers from intense interference, since nodes at the same hop distance contend the medium simultaneously. Other solutions, like PELLMAC [23], improved DMAC by suggesting to use different schedules for different branches of the tree. However, this approach increases complexity and adds extra control traffic.

Staggered schemes use other kinds of solutions, as for instance graph coloring: nodes wake up according to their colors and guarantee that a node has always at least one neighbor of the same color active. Since traffic in a WSN is not only upstream but also downstream (configuration traffic), authors of [15] proposed a solution that takes into account a bidirectional ladder scheme.

Staggered or skewed schemes solve some problems of rendezvous schemes, but are strongly topology-dependent and require the topology discovery and maintenance.

3.3.1.2 Semi-Synchronous Schemes

In semi-synchronous schemes nodes are grouped into synchronized clusters that asynchronously interact with each other, thus they can be considered as hybrid schemes. The main advantage of clusters is that synchronizing neighbors is easier than achieving a global synchronization. However, clusters can require election mechanisms for their maintenance, therefore adding extra control traffic. Basing on this requirement, semi-synchronous schemes can be grouped into Spontaneous Clustering or Elected Cluster-head ones.

Spontaneous Clustering Spontaneous clustering schemes are those in which nodes coordinate themselves with no cluster head. A first proposal was S-MAC (Sensor-MAC) [37], in which nodes spontaneously organize themselves in virtual clusters by just exchanging timestamps between neighbors. Clusters are formed in this way: node A broadcasts its schedule and if a node B listens to this message before its own decision, it follows A adopting its schedule, i.e., a cluster is formed. Instead, if B receives the message after it has decided by itself, it follows both schedules, meaning that it belongs to both clusters.

S-MAC is considered outdated because it reaches only a 20% duty cycle and it suffers from frequent sleep waiting.

T-MAC (Timeout-MAC) [33] has been proposed as an improvement of S-MAC, adding an adaptive strategy to it. Nodes dynamically switch off whenever the traffic activity of its neighborhood stops and so it is more energy-efficient, even if it sacrifices listen-synchronization among nodes of the same cluster. This can cause the so-called early sleep problem, i.e., when a node goes to sleep even if there is still traffic for it from a neighbor.

Elected Cluster-head In the elected cluster-head schemes, one node of the cluster is in charge (temporarily in most cases) of coordinating the cluster activity.

LEACH (Low-Energy Adaptive Clustering Hierarchy) [12], represents one of the first schemes of this type. Cluster-heads have to coordinate the activity of their cluster and perform some operations that can reduce the power consumption, as

for instance the network coding 3.3.2, and randomly rotate to guarantee fairness in energy consumption.

Cluster-heads are one hop far from the sink, but some other improvements of LEACH, like multihop-LEACH [36], set them some hops further.

Concerning the duty cycling, these approaches usually use TDMA for the communication inside the cluster and rendezvous mechanisms for the synchronization of the duty cycles schedule with the cluster-head.

Since election-based mechanisms require significant and complex control traffic, the most important challenges in these schemes are about finding efficient elections for cluster-heads and efficient ways for the inter-cluster traffic forwarding.

3.3.1.3 Asynchronous Schemes

In a multi-hop wireless network, nodes' synchronization is hard and costly to reach and maintain. For this reason asynchronous schemes represent a valid alternative: in these schemes, in fact, there is no synchronization among nodes, so no global time reference is kept. There are several strategies that can be grouped into *Preamble Sampling*, *Receiver-Initiated*, *Ondemand wakeup*, *Random duty cycling*, and *Schedule-based*.

Preamble Sampling B-MAC [25] was one of the first solutions for WSNs in which the LPL (low power listening) technique, also called preamble sampling, was used. The goal is to reduce idle listenings by charging the sender (only one), instead of receivers (possibly many), with the energy required for listenings. Every node, after going to sleep, asynchronously wakes up and checks the channel. Before a frame, the sender relays a long preamble (longer than the duration of active and sleep phases together) so that every node has the chance to wake up, listen to the frame and potentially stay awake to receive it.

There are some immediate drawbacks of this approach: firstly, the long preamble keeps the channel busy for a while, preventing other nodes to transmit. Secondly, there can be a significant end-to-end latency and an excessive overhearing, since also uninterested nodes have to listen to the preamble.

X-MAC [7] improves B-MAC by replacing long preambles with short frames (strobes) and with short intervals between them. This gives the receiver the chance to acknowledge the sender, hence interrupting the frames train before its end. Short frames can also contain the address of the receiver, so all the uninterested nodes can switch off, reducing their overhearing time.

Receiver-Initiated In Receiver-Initiated schemes, the sender, instead of using preamble frames, waits for a periodic beacon from a node that is ready to receive and then transmits the frame. RI-MAC [31] adopts this approach for a

wide range of traffic rates and patterns. The idea is that every node periodically wakes up, sends a beacon and stays awake if it receives a *stay awake* signal from the sender, otherwise it goes back to sleep.

As in the preamble sampling case, instead of receivers, the sender is charged with the extra energy needed to remain active and receive the beacon.

Ondemand wakeup In on-demand wakeup schemes the main idea is that a node can be awaken if necessary. In particular, another hardware interface, that usually is a low power radio and is called *wakeup radio*, continuously listens to a wakeup signal and then sends an interrupt to the CPU that reactivates the main radio interface.

Although the advantage of keeping the primary radio interface turned off is clear, the question is if keeping the secondary one turned on is more or less expensive, in terms of power, than the energy saved from the first one. Some studies ([17], [20]) pointed out how the extra device should not consume more than tens of microwatts for the on-demand scheme and that for reaching better results the range of operation should be very short, not always possible for WSNs.

Other variants have been proposed [9], where the secondary interface is radio-triggered by the induction of the electromagnetic field of a radio signal. In this case the question is how strong the signal has to be in order to reactivate the secondary interface, a value that strongly depends on the distance.

Although several attempts have been done, this approach still remains an interesting and rising field for research, and together with the schedule-based scheme, is what this work is based on.

Random duty cycling If a network is sufficiently dense, the random duty cycling strategy can result very effective. It is based on the idea that nodes go to sleep and randomly wake up, guaranteeing through randomness an adequate number of active nodes anytime.

RAW (Random Asynchronous Wakeup Protocol) [24] implements this approach: a random wakeup scheme is proposed in which node activity is inversely proportional to the number of its neighbors.

Results based on this kind of schema illustrate how dense the scenarios have to be in order to work properly. Furthermore, to avoid low delivery rates, the duty cycle and the randomness criterion must have been carefully decided. If well designed, random duty cycling leads to a fair distribution of the traffic load and a low end-to-end delay.

Schedule-based In schedule-based schemes the idea is just to design wakeup/sleep schedules a priori. Time is divided into slots and nodes assume an awake/asleep

status for each slot. In order to properly work, this strategy requires that two nodes overlap part of their active time, property defined [14] as *rotation closure*.

Clearly, the main advantage of this kind of approach is that there is no need at all of extra control traffic and it is very simple to implement, since it also does not depend on the topology.

3.3.2 Network Coding

Several solutions exist in literature for avoiding energy holes. Most strategies are about which distribution should be used to spread nodes in the network, like for instance the uniform [21] or the nonuniform [35] ones. Instead, the network coding approach [18], sometimes combined with duty cycle [27], appears as a more generic solution, since it is based on the idea that traffic inside bottleneck zones can be significantly improved by using algorithmic techniques, i.e., obtaining a better utilization of bandwidth and more reliability.

In particular, network coding makes intermediate nodes encode data packets received from neighbors into one and then sends it. The receiver nodes decode the message and gets original data packets again. In this way, instead of n different messages, only a single message is sent to the receiver.

Encoding operation The node designated to encode n packets chooses a sequence of n different coefficients a = (a_1, a_2, a_3,..., a_n), that is called *encoding vector*, from GF(2^t). The n packets P = (p_1, p_2, p_3,..., p_n) are linearly encoded into only one in this way:

$$X = \sum_{i=1}^{n} a_i P_i, \qquad a_i \in GF(2^t) \tag{3.1}$$

All the coefficients used for encoding the packet are transmitted together with it, in order to use them for decoding the message.

Decoding operation Decoding a message means solving a set of linear equations. Together with the encoded data, the encoding vector is received too and is used for the solution. Let a set (a^1,X^1),..., (a^mm,X^m) be received by a node, it solves the following set of of m linear equations with n unknowns.

$$Y^j = \sum_{i=1}^{n} a_i^j P_i, \qquad j = 1, \ldots, m \tag{3.2}$$

In order to be solvable, the set has to be composed by at least n different coded packets. The unknown P_i contain the original packets that were previously encoded.

Chapter 4

Problem

In this chapter I will introduce the problem this project is based on, together with the technologies involved. In particular, I will describe which kind of sensor nodes have been considered, the system model and the instance problem with its assumptions and the goal description. Finally, the SORW (Scheduled Ondemand Radio Wakeup) scheduler is presented as proposal for the objective of this work.

4.1 Scenario

Nodes considered in this WSN have a battery power source, so they need to respect strict energy constraints in order to extend the network/application lifetime as much as possible. For this reason, together with the use of duty cycling (3.3.1) schemes, nodes have some hardware features that let them save a significant amount of energy. In particular, the wake-up technology further optimizes the duty cycling approach, since nodes can be awaken ondemand by using an ultra-low power radio receiver. This additional hardware, called Wake-Up receiver (WuRx), is intended to listen to specific wake messages, sent by some other Wake-Up transmitter (WuTx). This approach lets the node completely disconnect from the power supply, waiting for a wake signal before being reconnected. In this way, sensor does not consume energy while sleeping, while WuRx is designed to consume a very small amount of energy compared to the main radio interface [22].

A node that uses this kind of technology can be defined as a *Wake-up Node* and can be classified depending on how the wake-up system and the node's circuits are fed.

The first distinction is therefore:

- Node in *Active* status (figure 4.1 (a)):
  the wake-up node is in an active status if the WuRx uses an internal battery.

The consume is about a few μA, but the range of communication is larger than the passive case.

- Node in *Passive* status (figure 4.1 (b)):
  the wake-up node is in a passive status if the WuRx harvests energy from a RF irradiator to supply itself. There is no consume of energy at all, but the range of communication in this modality is shorter than the previous one.

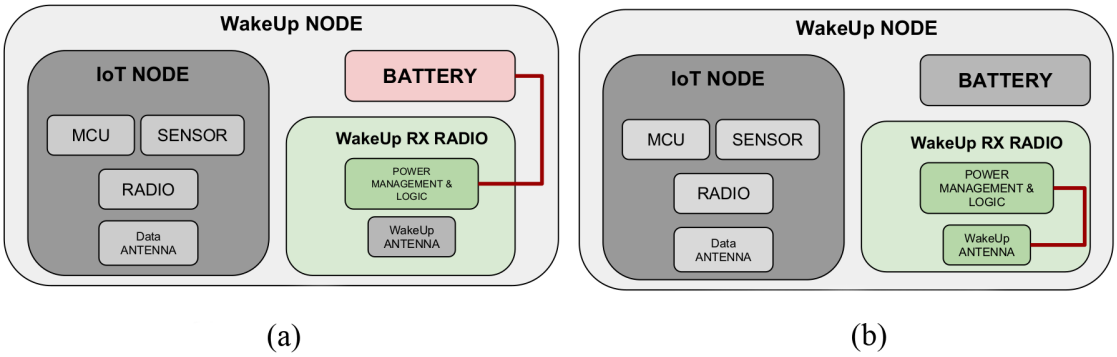

Figure 4.1: Representation of a node in active status (a) and in passive status (b). In the first case the WuRx uses an internal battery, in the second one instead it has to harvest energy from a RF irradiator.

The second distinction is:

- Node in *Switch* mode (figure 4.2 (a)):
  a node is in switch mode if it is connected to a hardware switch that connects/disconnects the power supply to it. This switch is controlled by the Wake-Up radio, that can then enable/disable the node power supply.

- Node in *Low Power* mode (figure 4.2 (b)):
  a node in low power mode has an output trigger connected to an interrupt MCU input that is used to wake up the node from its sleep state.

For this project, sensor nodes (figure 4.3) are composed of a subGHz radio device for data communication (SPIRIT1 [28]), an ultra-low power microcontroller (STM32L1 [29]), a battery, and a temperature sensor (STTS75 [30]). The microcontroller acquires the temperature and sends its value by using DASH7 [34] in request/response mode. The node is also equipped with a radio wake-up ([16], [11]) developed by *STMicroelectronics* that has a high-sensitivity RF harvester (-18 dBm @ 868MHz), an adjustable LDO, and an ultra-low power management unit for RF to DC power conversion and control. In active mode the device consumes about 1 μA with a response time lower than 0.5 sec and a radio sensitivity of -38dBm, which in free-space allows to turn a WuRx node on with power transmitted by the WuTx

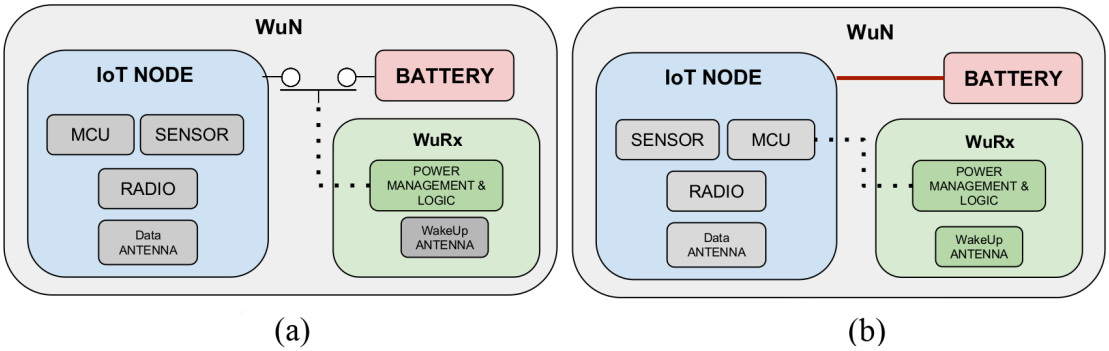

Figure 4.2: Representation of a node in switch mode (a) and in a low power mode (b). In the first case the node is connected to a hardware switch that can enable/disable its power supply. In the second case, the node is connected to an interrupted MCU used to wake up the node.

at 27 dBm, from a distance ranging between 30 and 50 meters. In passive mode, it harvests energy from an RF irradiator with a sensitivity of -18 dBm (5m range with 27 dBm RF transmitted power).

For a single node, E_{ON} is defined as the energy needed for the wake-up/sensing/transmission sequence, E_{BOOT} as the energy required for the microcontroller booting phase and E_{STB} as the energy used when both MCU and radio are in sleep mode or not powered. The latter depends on the duration of the standby phase t_{STB}. In table 4.1, the energy measurements of the wake-up node. In this work, results are provided by the *Advanced Research Center on Electronic Systems "Ercole De Castro" (ARCES)*[1].

4.2 System model

The system model considers n wake up sensor nodes and 1 *irradiator*, in the following denoted as *coordinator*. Let N be the sensor set, with $N = s_0, s_1, \ldots, s_{n-1}$. The coordinator is in charge of transmitting the wake up signal and is placed at the center of the scenario, while sensor nodes are randomly located within the RF range of the coordinator. Let also *time* be divided into slots, i.e., $T = t_0, t_1, t_2, \ldots$ of the same T_{slot} duration (seconds).

The application model works as follows: at each time slot, the coordinator must read λ measurements from λ different sensors. For this purpose, let $\phi : NxT \rightarrow [0:1]$ be the selector function, indicating which sensors are used at each slot: if $\phi(i, t_j) = 1$, then the s_i sensor is selected at slot t_j. Let also $d\ (i)$ be the distance between the s_i sensor and the coordinator node, and with $D = d(0), d(1), \ldots, d(n-1)$ the corresponding distance array. As explained in 4.1,

[1] http://www.arces.unibo.it/en/advanced-research-center-on-electonic-systems

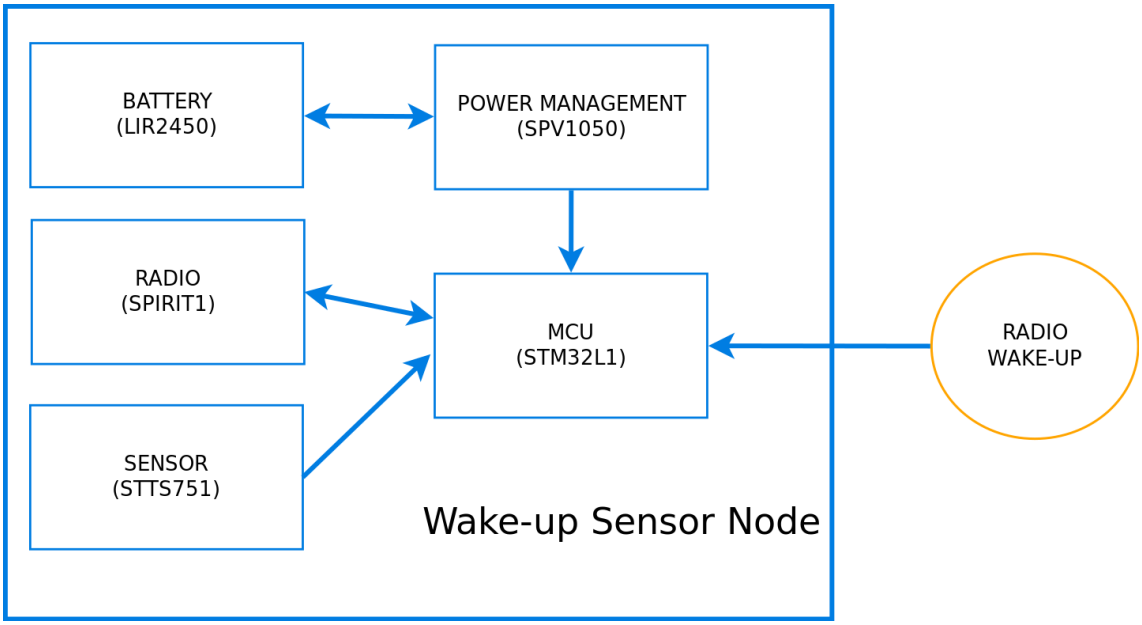

Figure 4.3: Sensor node used in this project are composed by a subGHz radio device, an ultra-low power microcontroller, a battery and a temperature sensor

Mode	***Active***	
	Low Power	***Switch***
E_{ON}	494 μJ	
E_{BOOT}	0	7.5 mJ
E_{STB}	10 μW * t_{STB}	0

Table 4.1: Energy measurements for the wake-up node. E_{on} is the energy needed for the wake-up/sensing/transmission sequence, E_{boot} the energy required for the microcontroller booting phase and E_{stb} the energy when both MCU and radio are in sleep mode or not powered. It depends on the duration of the standby phase t_{stb}.

a sensor node can be in one of the following statuses, depending on its distance from the coordinator:

- **Active status**: the wake-up radio is fed by a battery, consuming a E_{WKP} constant amount of energy per slot t.
- **Passive status**: the wake-up radio harvests energy from the coordinator, so $E_{WKP} = 0$

Let $M : N \rightarrow [0 : 1]$ be the function that maps a sensor node to its status: M (i) = 1 if sensor s_i is in *active* status, otherwise M (i) = 0 if it is in a *passive* status. The sensor's status is an input of the problem, since it depends on the scenario topology. In fact, there exists a threshold distance χ such that: if d (i) $>$ χ then M (i) = 1, otherwise M (i) = 0. This means that a sensor is in a passive status only when it is sufficiently close to the coordinator in order to use energy harvesting.

When a sensor node is awaken by the coordinator, it sends its data to it, consuming E_{ON} amount of energy. This value takes into account the full sequence of operations performed by a sensor node, i.e., wake-up→sensing→transmission.

As explained in 4.1, a sensor node can also assume one of the following modes:

- **Switch mode (SW)**: the node circuits are connected to a hardware switch that can enable/disable the node's power supply of the battery. When disconnected, i.e., in standby phase, there is no consumption of energy ($E_{STB} = 0$), while the boot phase (from standby to active one) requires a significant amount of energy ($E_{BOOT} >> 0$).
- **Low Power mode (LP)**: the node circuits are fed by a battery, implying a constant per-slot energy consumption even if in standby mode, while there is no boot phase. This means that $E_{STB} > 0$ and $E_{BOOT} = 0$.

While the status of a node is pre-decided because of the distance from the coordinator, the node's mode can be dynamically changed, both by the coordinator and the sensor itself. In order to do so, let $S : NxT \rightarrow [0 : 1]$ be a function such that: S (i, t_j) = 1 if node s_i is in LP mode at time slot t_j, and S (i, t_j) = 0 if node s_i is in SW mode at time slot t_j. Furthermore, let's define E (i, t_j) as the residual energy of sensor s_i at slot t_j. For every slot t_j, E (i, t_j) $\forall s_i \in N$ is updated as follows:

$$
\begin{aligned}
E(i, t_j) = E(i, t_{j-1}) & \quad (1)\\
- M(i) \cdot E_{WKP} & \quad (2)\\
- \phi(i, t_j) \cdot [E_{ON} + E_{BOOT} \times (1 - S(i, t_{j-1}))] & \quad (3)\\
- S(i, t_j) \cdot E_{STB} & \quad (4)
\end{aligned}
$$

The alive function is defined as $A : NxT \rightarrow [0 : 1]$: if sensor s_i still has some energy left at slot t_j (E (i, t_j) > 0), then A (i, t_j) $= 1$, otherwise A (i, t_j) $= 0$.

4.2.1 Problem formulation

Given N, D, λ, the goal is to determine the state function S (i, t_j) and the selection function ϕ (i, t_j) so that the system lifetime L is maximized. In addition, the following requirements must be guaranteed:

1. $\forall t_j \in T, \sum_{0 \leq i < n} \phi(i, t_j) = \lambda$, i.e., the coordinator has to collect exactly λ measurements per slot.
2. $\forall t_j \in T, \sum_{0 \leq i < n} T_{BOOT} \cdot \phi(i, t_j) \cdot (1 - S(i, t_{j-1})) \leq T_{slot}$, i.e., the coordinator is able to wake up all the selected sensor nodes in SW mode within the duration of the time slot (T_{slot}). The T_{BOOT} factor is the time required by the node to perform the boot when in SW mode, and it is assumed to be constant.

The system lifetime is actually defined in two different ways:

- **Network Lifetime**: it is the slot (t_{final}) after the first sensor runs out of battery.

$$L = \max_j \mid A(i, t_j) = 1, \forall s_i \in N \tag{5}$$

- **Application Lifetime**: it is the slot (t_{final}) after it is not possible to take λ different measurements anymore, i.e., there are less than λ devices with enough energy.

$$L = \max_j \mid \sum_{0 \leq i < n} A(i, t_j) \geq \lambda \tag{6}$$

4.2.1.1 Assumptions

In order to be able to treat the analytic results, for this first part of the project, I made some assumptions:

1. Distance is not taken into account since all nodes are equally distant from the coordinator.
2. All the nodes are in a *passive* status
3. The first λ used nodes start with a LP mode, while all the others with a SW mode.
4. Only the application lifetime is considered

4.3 SORW scheduler

The goal of this work is to find a scheduler that tries to maximize the lifetime of the problem previously defined in 4.2.1.

The Scheduled Ondemand Radio Wake-Up (SORW) scheduler tries to reach exactly this result.

As the name suggests, the approach is based on both a scheduled and ondemand scheme 3.3.1.3, with a wake-up radio used by the sensor to be awaken.

In fact, the idea is to keep the best of the scheduled schema and the ondemand one: on one hand there is the possibility to completely turn off sensors and then wake them up, on the other hand there is no need of extra control traffic to make sensors synchronized.

First, I introduce the Switch Benefit (SW_B) index, defined in this way:

$$SW_B = \left\lfloor \frac{E_{BOOT}}{E_{STB}} \right\rfloor \tag{7}$$

Essentially, it represents the minimum number of slots in which a sensor node should keep the SW mode in order to save energy compared to the LP mode. Clearly, this value strongly depends on the slot duration, since it influences the E_{STB} index.

I assume that $SW_B > 1$ in the following, otherwise the computation of S (i, t_j) becomes trivial, i.e., $S(i, t_j) = 0 \ \forall t_j < t_{final}$ and $\forall s_i \in N$. The cardinality of N, that is the total number of sensor nodes, is represented by n.

The rationale of the SORW algorithm is to make each sensor node save energy by entering the SW mode only once and keeping it for the biggest number of consecutive time slots, at least equal to the SW_B index defined above. In particular, the algorithm is split into two stages, respectively called `ROUND_ROBIN` and `GREEDY`. At the beginning, if $(n\%\lambda) \neq 0$, the index m is computed as

$$m = \max_i \mid n - (i \cdot \lambda) > \lambda \tag{8}$$

i.e., the maximum number of groups of λ nodes that can be taken from n (total number of sensors), but still leaving out at least $\lambda + 1$ nodes. Otherwise, m assumes the $\frac{n}{\lambda}$ value. Sensors are split into two groups: the first is made up of $k = n - (m \cdot \lambda)$ nodes ($G_0 = \{s_{j=0}, s_{j+1}, \ldots, s_{(k-1)}\}$), while the second is composed of all the last $m \cdot \lambda$ sensors, ($G_1 = \{s_{(n-k)}, s_{(n-k+1)}, \ldots, s_{n-1}\}$). At this point the `ROUND_ROBIN` stage can begin: it starts taking measurements from the subgroup $G'_0 = \{s_j, s_{j+1}, \ldots, s_{(j+\lambda-1)\%k}\}$ in G_0, keeping its nodes in LP mode and repeats the same operation for ST consecutive slots, where:

$$ST = \left\lfloor \frac{E_{INIT} - E_{BOOT}}{\lambda \cdot E_{STD}} \right\rfloor \tag{9}$$

and that maximizes the duration of the SW mode. If $ST < SW_B$ the problem is trivial, i.e., $S(i, t_j) = 1\ \forall t_j < t_{final}$ and $\forall s_i \in N$. After that, the subgroup G'_0 is updated on a rolling base way, considering the next λ sensors in range $[j+1 : (j+\lambda)\%k]$, and setting sensor s_j in SW mode, while those used in LP.

The `ROUND_ROBIN` stage ends when all the nodes in G_0 have been used exactly $\lambda \cdot ST$ times, meaning that they have approximately the same energy left. Furthermore, there are exactly λ sensors of G_0 in LP mode, while all the rest is in SW mode. At this point the `GREEDY` stage can begin: let t_{greedy} be the slot in which it starts, the algorithm proceeds by selecting λ sensors in turn from G_1 and completely discharging them, before switching to the next λ sensors.

Again, the selected sensors are moved from the SW to the LP mode (consuming E_{BOOT}), and used until completely discharged. The algorithm ends when there are no λ nodes left in G_1 with enough energy for another slot.

The figure 4.4 shows how the SORW scheduler works for $N = \{s_0, s_1, \ldots, s_9\}$ and $\lambda = 3$, assuming $ST = 2$, without considering a specific battery capacity. First, groups of nodes for the `ROUND_ROBIN` and the `GREEDY` stages are composed (line 9, 10 of the SORW algorithm). Then, the `ROUND_ROBIN` stage starts: it computes the `ST` value, selects nodes from G_0, and uses them for exactly `ST` times, keeping them in LP mode (from 16 line to 20), before changing the selected nodes and setting the first of the set in SW mode. After this stage, there can be a micro-optimisation phase, in which nodes that are already awake could be used for a last round. This is due to possible approximation issues while computing the `ST` value. After that, the `GREEDY` stage starts: it selects the first λ nodes from the G_1 set and uses them until they have enough energy, before switching nodes with the next λ ones in the set.

4.3.1 Analytic Results

According to the previous definition of the problem and the SORW scheduler algorithm, the application lifetime can be computed as the sum of lifetimes obtained by the `ROUND_ROBIN` (included the `micro-optimisation` phase) and the `GREEDY` stages.

Algorithm 1 SORW scheduler algorithm

1: $time = 0$
2: $r = n\%\lambda$
3: **if** $r = 0$ **then**
4: $\quad m = \frac{n}{\lambda}$
5: **else**
6: $\quad m = \max_i \mid n - (i \cdot \lambda) > \lambda$
7: **end if**
8: $k = n - (m \cdot \lambda)$
9: $G_0 \leftarrow \{s_{j=0}, s_{j+1}, \ldots, s_{(k-1)}\}$
10: $G_1 \leftarrow \{s_{(n-k)}, s_{(n-k+1)}, \ldots, s_{n-1}\}$
11: ▷ `ROUND_ROBIN`
12: $ST = \frac{E_{INIT} - E_{BOOT}}{\lambda \cdot E_{STD}}$
13: **for** $j = 0$ to $j < |G_0| - 1$ **do**
14: $\quad G'_0 \leftarrow \{s_j, s_{j+1}, \ldots, s_{(j+\lambda-1)\%k}\}$
15: ▷ use all nodes and set them in LP mode
16: $\quad$ **for** $i = 0$ to $i < ST - 1$ **do**
17: $\quad\quad \phi(s_j, t_{time}) = 1, \forall s_j \in G'_0$
18: $\quad\quad S(s_j, t_{time}) = 1, \forall s_j \in G'_0$
19: $\quad\quad time = time + 1$
20: $\quad$ **end for**
21: **end for**
22: ▷ micro-optimisation phase
23: **while** $A(s_j, t_{time}) = 1, \forall s_j \in G'_0$ **do**
24: $\quad \phi(s_j, t_{time}) = 1, \forall s_j \in G'_0$
25: $\quad time = time + 1$
26: **end while**
27: ▷ `GREEDY`
28: $j = n - k$
29: **while** $\sum_{s_i \in G_1} A(s_i, t_{time}) \geq \lambda$ **do**
30: $\quad G'_1 \leftarrow \{s_j, s_{j+1}, \ldots, s_{(j+\lambda-1)}\}$
31: $\quad$ **while** $A(s_j, t_{time}) = 1, \forall s_j \in G'_1$ **do**
32: ▷ iterate while nodes are still alive
33: $\quad\quad \phi(s_j, t_{time}) = 1, \forall s_j \in G'_1$
34: $\quad\quad S(s_j, t_{time}) = 1, \forall s_j \in G'_1$
35: $\quad\quad time = time + 1$
36: $\quad$ **end while**
37: $\quad j = j + \lambda$
38: **end while**

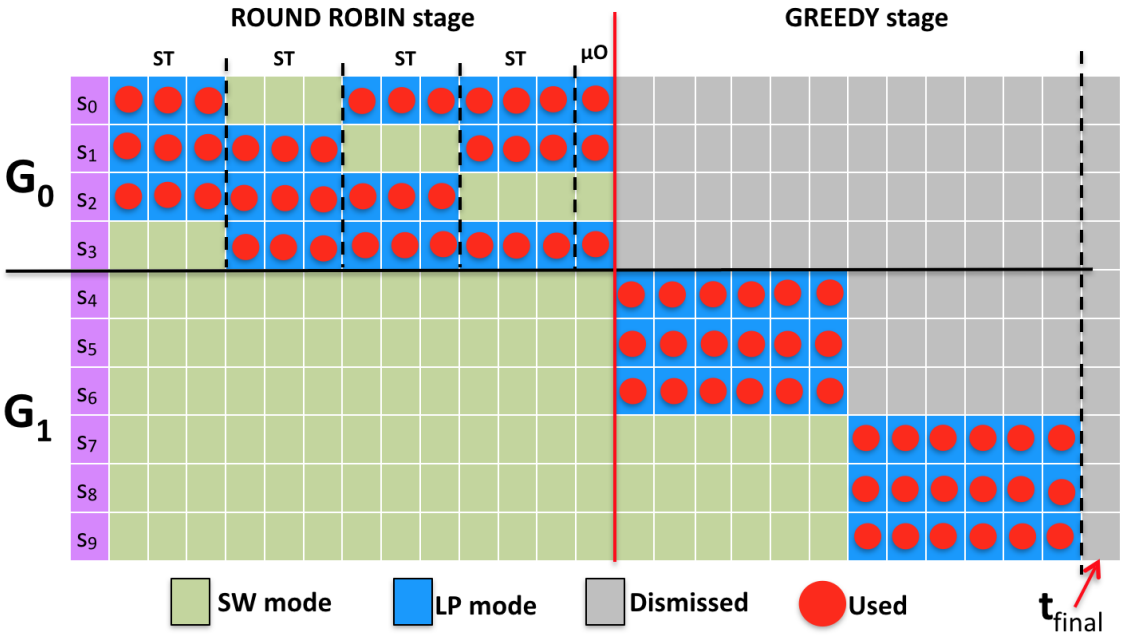

Figure 4.4: Image showing how the SORW scheduler works for $N = \{s_0, s_1, \ldots, s_9\}$ and $\lambda = 3$, assuming $ST = 2$ and no particular battery capacity for nodes.

By an analytic point of view, the first can be defined as:

$$LT < ST \cdot |G_0| + \lambda - 1 \tag{10}$$

In fact, the `ROUND_ROBIN` stage ends after each node in the first set (G_0) has been used at least $ST \cdot \lambda$ times, and more precisely ST times multiplied by `r`, where $\lambda < r < |G_0|$. For this reason, the previous formula represents an upper bound (figure 4.5 and table 4.2) for the lifetime. Furthermore, because of approximation issues resulting from the `ST` estimation, it could happen that nodes have still enough energy for one final round. For this reason, the `micro-optimisation` step is also taken into account, meaning that at the end of the `ROUND_ROBIN` stage, the λ nodes that are already awake could be used for one more detection.

The second can be defined as:

$$LT = \left\lfloor \frac{|N| - |G_0|}{\lambda} \right\rfloor \cdot \left\lfloor \frac{E_{INIT} - E_{BOOT}}{E_{STD}} \right\rfloor \tag{11}$$

In fact, the `GREEDY` stage takes groups of λ sensors from the $G_1 = N \setminus G_0$ set, and uses them until they have enough energy before switching them with the next λ sensors. In particular, the number of times that they can provide a detection

is their total amount of initial energy (after paying E_{BOOT} to wake up) divided by E_{STB}, i.e., the energy required for the sequence wake-up/sensing/transmission and that for taking sensors in LP mode.

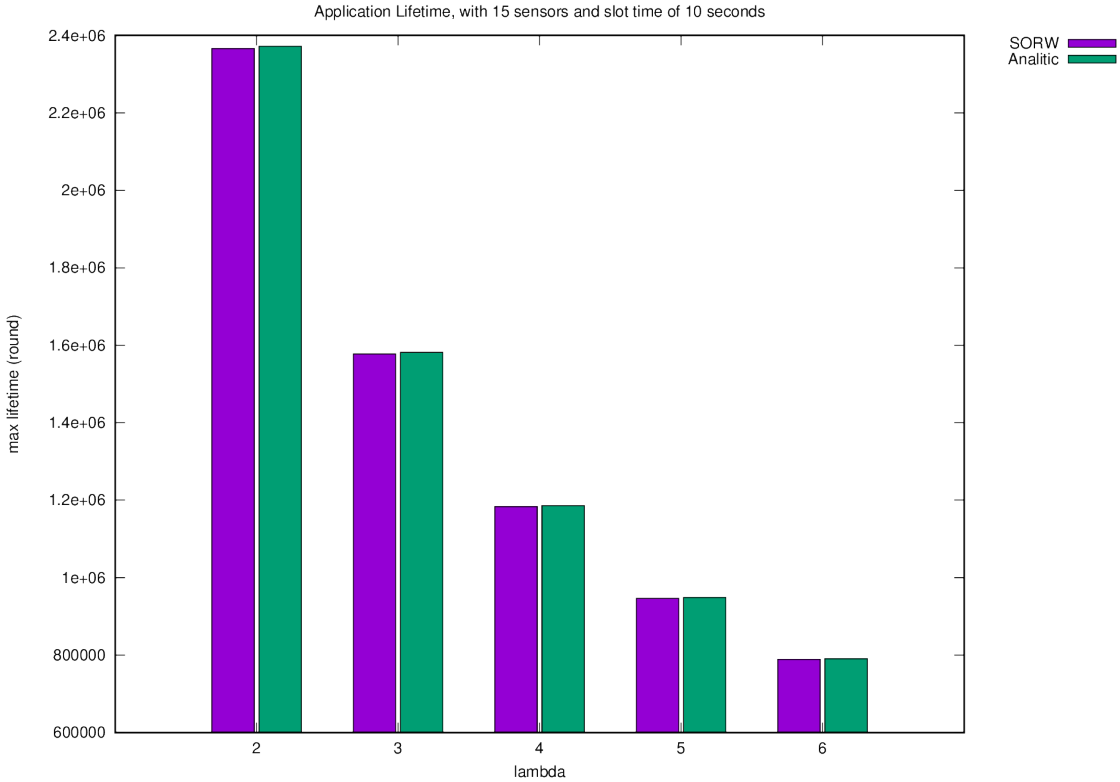

Figure 4.5: Comparison of results obtained by the SORW scheduler and the analytic approach for a network composed of 15 sensors, with $\lambda \in [2, 6]$, a time slot of 10 seconds, and a battery capacity of $20mAh$.

λ	**SORW**	**Analytic**	**difference**	**difference (%)**
2	2366658	2372482	5824	0.25
3	1577508	1581655	4147	0.26
4	1183742	1186239	2497	0.21
5	946510	948993	2483	0.26
6	789160	790825	1665	0.21

Table 4.2: Results obtained by the SORW scheduler and the analytic approach for a network composed of 15 sensors, with $\lambda \in [2, 6]$, a time slot of 10 seconds, and a battery capacity of $20mAh$. Analytic values are an upper bound for the SORW's ones, with a difference of less than 0.30%.

Chapter 5

Implementation

In order to test and obtain some results related to the problem discussed in chapter 4, I used OMNeT++ [1], one of the most popular discrete-event simulators. This chapter is organized as follows: I will firstly provide a brief discussion about how OMNET++ works and which kinds of tools it offers, and then I will introduce the implementation choices for this work.

5.1 OMNeT++

OMNeT++ is an open-source discrete-event simulator, extensible and modular, and its components are based on a C++ simulation library and frameworks. Even if it is particularly useful for networks, it can be used for almost every kind of simulation. In particular, some important applications are the validation of an hardware architecture, the evaluation of software-system performances, the modeling of generic protocols or distributed system, and the modeling of systems that require mobility, frequently in collaboration with SUMO [2].

In general, it is suitable for all those systems in which a discrete-event approach can be used and entities can communicate by exchanging messages.

OMNeT++ is not only a simulator, but a platform of simulation that offers:

- a simulation kernel library
- NED (Network Description) topology description language
- an OMNeT++ IDE based on the Eclipse platform
- a GUI for simulation execution, links into simulation executable (Tkenv)
- a command-line user interface for simulation execution (Cmdenv)

- a C++ class library, that usually developers use and customize according to their needs
- some utilities (makefile creation tool, etc.)
- documentation, sample simulations, etc.

OMNeT++ provides a component architecture for models that are programmed in C++ and then assembled into larger components and models using a high-level language (NED).

The flow of operations is the following: as first step the developer defines modules and so the model structure using the NED language, either through a graphical interface or a text editor. In the second step, he implements the components of each module by using the C++ class library and the kernel provided. In the last step, the developer makes a custom configuration of some parameters of the simulation, builds the simulation program and runs it.

5.1.1 Module and Model

Modules are the fundamental components of frameworks and they can be structured in a hierarchic way, making a tree: leafs are called *simple modules* and represent entities or their behaviors, while the root is considered as the whole *system*.

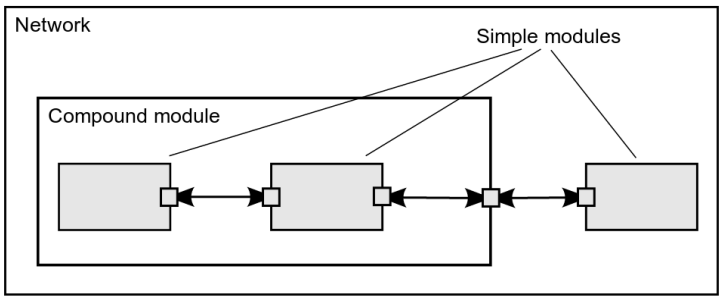

Figure 5.1: Image from [1] showing how single modules can be grouped together.

As shown in figure 5.1, simple modules can be grouped together to make *compound modules* or *models* by using the NED language. Parameters in simple modules are used as configuration data, while parameters in compound modules are used to define the number of submodules, the number of gates or connections. In both cases, they can be assigned either in a *.ini* configuration file, or interactively by the user.

5.1.2 Gate

Messages are sent and received by simple modules through *gates*, i.e., some particular input/output software interfaces. A gate can be linked to a specific *connection* of the same hierarchical layer or of a compound module, while there is no chance for interconnecting modules at different layers.

As in the reality, connections can be characterized by three parameters:

- Bit error rate: probability that a bit is transmitted in an erroneous way
- Propagation delay: amount of time required for the head of the signal to reach the destination from the sender
- Data rate: amount of data that can simultaneously travel in the connection

5.1.3 Message

Since in OMNeT++ the communication among entities is based on message exchange, messages can represent frames, packets, tasks, events, and all the other user-customized structures and can be sent/received by different modules or by themselves (*self-messages*) through the use of a timer.

Message headers are defined in separate files (*message definition files*) in a C language similar syntax that OMNeT++ automatically converts into a class with its header file. All the generated classes are subclasses of the OMNeT++ message class, i.e., the *cMessage* class.

5.2 Implementation

According to how OMNeT++ works and to the problem previously defined, there are few design choices that need to be discussed and that concern which modules have been defined, which classes have been implemented and how the SORW scheduler algorithm works.

5.2.1 Modules

Two single modules are defined in this project, as well as a compound module: a module for the irradiator, a module for the sensor (called device) and a module for the network, composed by both.

Irradiator The module *irradiator.ned* represents the single Irradiator/coordinator of the network, i.e., that special device that is in charge to wake up selected nodes that were sleeping and to collect all λ detections from them.

Since the NED module needs for the network definition, the position and the gate definition are the only pieces of information required in this step. In particular, the first is randomly assigned when launching the simulation, while the second is a *radioIn* gate, i.e., a radio interface that is able to receive all radio waves surrounding it. Actually, instead of listening to all radio waves, and so avoiding problems like collisions, the irradiator only listens to messages explicitly addressed to it.

Unlike sensors, an irradiator is supposed to have an infinite power source, since, for the goal of this first part of the project, it can be represented by a fixed device directly connected to electricity.

Device The module *device.ned* represents a generic *sensor* abstracting from three criteria: which kind of data it is able to sense, how it is physically done, and which wireless technology/protocol is used for the transmission.

Since a sensor is only in charge of replying to the irradiator when required and potentially waking up, also in this case its position and its *gate* definition are the only pieces of information required for the module. The first is randomly assigned when launching the simulation and following a uniform distribution, while the second , like in the irradiators case, is a *radioIn* gate, i.e., an abstraction of a radio interface capable of sensing radio waves around it.

All the other information about the physical device, as for instance the power source capacity, are stored in the class file.

Network The module *network.ned* represents the network abstraction of the project, involving a single irradiator and a variable number of devices (sensors) as shown in figure 5.2. For this reason, it is defined as a composition of both the irradiator sub-module and the device's.

This module is also used to record some input parameters useful for the simulation, like the value of λ, or the number of sensors, or the kind of scheduler that will be utilized.

5.2.2 Classes

Classes defined in this project reflect the single modules defined and contain all the routines for the message exchange, for the scheduling, and for the energy management. There was no need to declare also a message class, since in this first

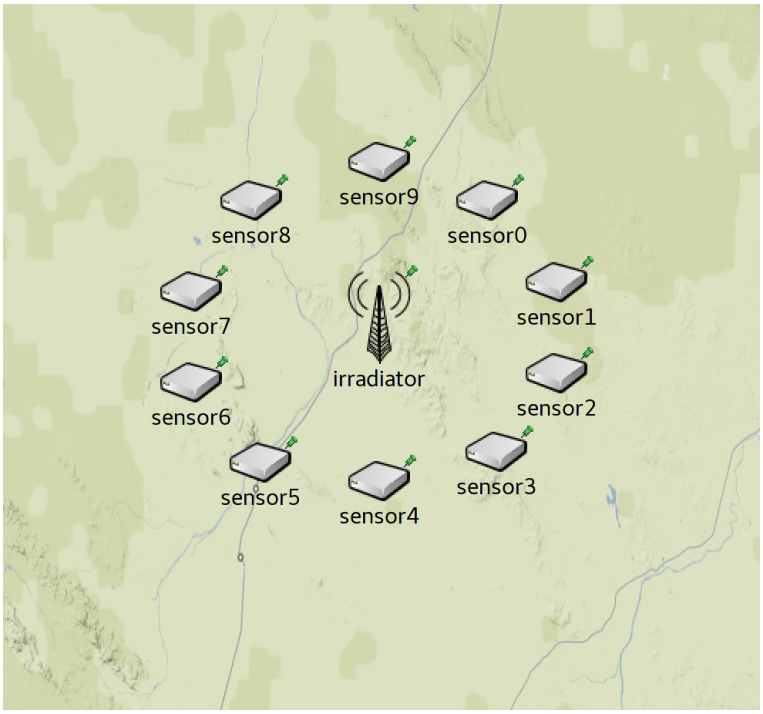

Figure 5.2: The network module representation. In particular, this instance shows the irradiator surrounded by 10 sensors equally distant from it.

part of the project the only information required is an ack from sensors instead of the message content.

Irradiator and device classes contain some variables about the wireless technology too, like the propagation delay, the packets size, etc. . . , but for now they are not so useful, since messages always arrive and their size is not taken into account.

5.2.2.1 Irradiator

The irradiator class contains all the routines needed for the message exchange between irradiator and nodes, the routines for the scheduling and the routines for the energy management of the entire network. In fact, in this first part of the project, even if it is nodes that store their energy information, the system is considered centralized, since the irradiator can just invoke a method of them in order to retrieve how much energy they have left.

At the same time, the irradiator class contains several member variables used for network topology and result collection. Here a brief discussion about the most important member variables and methods of an irradiator.

- **simsignal_t networkLifetimeSignal**
  simsignal_t applicationLifetimeSignal

These two variables are used respectively to keep track of the network lifetime and the application lifetime. In particular, each time a new round of scheduling starts, a signal is emitted and caught by the default routines of OMNeT++, in order to make some statistics. These signal are usually used for recording events, and together with the values, they also save the simulated time in which they occurred. In this way, through the OMNeT++ GUI it is possible to monitor these data in real-time during the simulation, and to use them offline for statistics.

- **int lambda**

  It represents exactly the λ variable of the defined problem, i.e., the number of detections that an irradiator has to perform each round from λ different sensors. It is taken by the network module, since it is a parameter passed before the simulation.

- **int new_mode_if_used** and **int new_mode_if_not_used**

  These two variables are used to indicate the mode in which a sensor should be set respectively if it has sent its value or not during the current round. A sensor's mode can be, according to the definition of the problem, Low Power or Switch, meaning how the wake-up radio is connected to the sensor.

- **int n_sensors**

  It represents the cardinality of the set of sensors `N`, as defined in problem (chapter 4), i.e., the number of nodes currently participating in the network. It is taken by the network module, since it is a parameter passed before the simulation.

- **int detections_per_slot** and **int switch_benefit**

  The first variable represents exactly the n variable of the defined problem (chapter 4) in the SORW algorithm case, i.e., the number of consecutive rounds that a node has to stay awake for and send its data before its sleeping turn. The second one represents the Switch Benefit (SW_B) index, used for understanding if a simple round robin schema has to be used instead of the SORW algorithm.

  Both of them are compute if and only if the SORW algorithm is required.

- **virtual void initialize(int stage) override**

  The initializing method overrides the default method of OMNeT, where all the initializations should be done. In particular, in case of the SORW algorithm, the two sets of sensors are composed assuming that they are all equal at the beginning.

- **virtual void handleMessage(cMessage *msg) override**
  virtual void send_msg(cModule *sensor, cPacket *pk)

  These methods are in charge of performing the message exchange with sensors. The first is called when a message from a sensor arrives or to handle a scheduled event (that for OMNeT++ is considered a message too), while the second is used for sending a data request to sensors.

  Since a new event implies the start of a new round, handleMessage is also in charge of emitting the signals previously described, in order to keep track of the network and application lifetime. Events are scheduled by the irradiator in this function as soon as it has received exactly the λ different messages from sensors.

Here the code of the routine:

```
1   void
2   Irradiator::handleMessage(cMessage *msg) {
3
4     int index = -1;
5     //check if the input message is a scheduled event
6     if (msg == event) {
7       //increase the application lifetime
8       emit(applicationLifetimeSignal, application_lifetime);
9       application_lifetime++;
10      if(check_network_lifetime(NULL)) { //if all sensors are alive
11        //increase the network lifetime
12        emit(networkLifetimeSignal, network_lifetime);
13        network_lifetime++;
14      }
15      which_scheduler(); //select the right scheduler
16
17      for (int i = 0; i < lambda; i++) {
18        //take sensors previously selected by the scheduler
19        index = sensors_index[i];
20        cPacket *pk = new cPacket("msg");
21        pk->setBitLength(pkLenBits);
22        cModule *sensor =
23          getParentModule()->getSubmodule("sensor", index);
24        if (sensor == NULL) {
25          throw cRuntimeError("sensor not found");
26        }
27        send_msg(sensor, pk);
28      }
29    //the input message is a message received from a sensor
30    } else {
31      cPacket *recv_pk = (cPacket*) msg;
32      tmp_received_msgs++;
33      delete recv_pk;
34
35      if (tmp_received_msgs == lambda) { //check if λ messages arrived
36        tmp_received_msgs = 0;
37        //schedule immediately a new event, i.e., a new round
38        scheduleAt(simTime(), event);
39      }
40
41    }
42  }
```

- **virtual int check_networkLifetime(std::vector<int> *enabled_sensors)**
  virtual int check_applicationLifetime1(std::vector<int>*enabled_sensors)

  These methods are in charge of checking respectively the network and the application lifetime. The first just checks that each sensor is still alive and has enough energy to make a new round, i.e., to send a new message. The second one checks that at least λ different sensors can send their value.

  While the second operation is performed after the end of a round, in order to stop the computation in case there are not λ sensors left, the first is executed before the beginning of a new round when a new scheduled event starts, as shown at line 6 of the previous routine.

- **virtual void deduct_energy()**
  virtual float compute_sensor_energy(cModule *sensor, int to_be_used, int new_mode)

  The first method is in charge of deducting energy from sensors, according to what the second estimates. This one, in fact, computes how much energy is used by a sensor according to its previous status, the new one and if it has sent a message during the last round or not. See the formula in section 4.2 for more details.

- **virtual int still_energy_left()**

  This method simulates a new round in order to understand how many sensors can effectively do it. If at least λ different sensors can send their data for another round, the scheduling can go on, otherwise the simulation ends.

- **virtual void SORW_scheduler()**

  For the sake of simplicity, the implementation of the SORW algorithm (section 1) is included in the irradiator class. Nevertheless, in future developments it will be implemented as a distinct class. The pre-allocation of groups, one for the ROUND_ROBIN and one for the GREEDY stage, is done in the *initialization* routine. Here is the code of the routine:

```
44  void
45  Irradiator::SORW_scheduler() {
46    int last_one = sensors_index[lambda-1];
47    int first_one = sensors_index[0];
48    int i = 0;
49    int vector_index = -1;
50    int real_index = -1;
51    int change_sensors = 0;
52     //n_sensors is a multiple of lambda
53    if(n_sensors%lambda == 0) {
54      //first round it's round robin
55      if(last_one == -1) {
56        round_robin_scheduler();
57        return;
58      //it is not the first round, let's check if still enough energy
59      } else {
60        real_index = enabled_sensors[vector_index];
61        for(i = 0; i < lambda; i++) {
62          //take the vector index
63          vector_index = return_index_from_value(enabled_sensors,
64              sensors_index[i]);
65
66          /*if sensor has not enough energy, just remove it
67          from the available ones*/
68          if(!can_i_take(sensors_index[i])) {
69            enabled_sensors.erase(enabled_sensors.begin() + vector_index);
70            change_sensors = 1;
71          }
72        }
73        if(change_sensors) {
74          /*if here, it means that the system
75          has still enough energy but previous sensors have
76          to be swapped with bigger indexes*/
77          for(i = 1; i < lambda+1; i++) {
78            sensors_index[i-1] = last_one + i;
79          }
80        }
81      }
82    } else {
83      //first round it's round robin
84      if(last_one == -1 && current_detection == detections_per_slot) {
85        round_robin_scheduler();
86        current_detection--;
87        return;
88      } else {
89        /*not the first step, but it could be that
90        current_detection is still not 0*/
91
92         //use the same indexes again
```

```
93          if(current_detection) {
94            //do nothing, sensors_index have to be the same
95            current_detection--;
96            if(current_detection == 0 &&
97                strcmp(&which_s[0], "SORW") == 0) {
98              //set the sensor offset
99              sensor_off = last_one-lambda+1;
100           }
101         } else {
102           current_detection = detections_per_slot;
103           sensor_off = -1;
104            //first round, since first_one is still not the last sensor
105           if(first_one != enabled_sensors[n_sensors-1]) {
106             for(i = 1; i < lambda+1; i++) {
107               sensors_index[i-1] = (first_one+i)%n_sensors;
108             }
109            //micro-optimization phase
110           } else {
111             //change the scheduler for making the optimization phase
112             memset(&which_s[0], 0, 50);
113             strncpy(&which_s[0], "uoptimization",
114                 strlen("uoptimization"));
115             uoptimization_scheduler();
116             return;
117           }
118           current_detection--;
119         }
120       }
121     }
122     //deduct energy from sensors
123     deduct_energy();
124   }
```

5.2.2.2 Device

For this first part of the project, it is not useful to focus on how sensors collect data and which kind. For this reason, the only important member variables and methods defined in this class are about the information of the Device, i.e., its energy and its mode.

Here is a brief discussion about the most important member variables and methods of device.

- **int status**, **int mode**, **int energy**

  The status variable reflects the status of the sensor, i.e., *passive* or *active*, the mode variable is the mode of the sensor, i.e., *SW* or *LP*, and the energy variable is the energy left for the device.

- **Getter/Setter methods**

  For all the previous variables, device has a getter and setter, in order to make the irradiator able to read these values. In a future development of the project, these values will be sent through the network instead of just reading them by using the module reference.

- **virtual void handleMessage(cMessage *msg) override**
  virtual void send_msg(cModule *irradiator, cPacket *pk)

  These methods are in charge of performing the message exchange with the irradiator. The first is called when a message from the irradiator arrives, while the second is used for sending data to the irradiator.

Chapter 6

Results

In this chapter, I will firstly introduce some algorithms that could be used for a WSN like the one used in this work, and for every one of them I will select its best policy. Then I will compare SORW with the algorithms in order to show its flexibility, especially when there is a change in the parameters of the application, as for instance the duration of the time slot or the number of sensors deployed.

6.1 Algorithms

According to the definition of the problem (chapter 4), a node can dynamically assume a *Switch* or a *Low Power* mode, meaning how the wake-up radio is connected to it. Since this mechanism is guaranteed by a hardware switch controlled by a software, the proposed algorithms should implement a similar feature as well, in order to be able to behave accordingly. For this reason, depending on the fact if a sensor has been used or not, I defined 4 different policies:

- **LPSW**: nodes that have been used assume a *LP* mode, while those not used assume the *SW* one. Even if this solution seems the most interesting one, especially for the SORW algorithm, as shown in this section it is not the best one for all algorithms, since there are situations in which it is more efficient to put sensors to sleep after they have been used.

- **LPLP**: both nodes that have been used and those that have been not used assume a *LP* mode. This policy is very useful for those cases in which the scheduler alternates nodes very quickly, the time slot is short, and there are not so many available nodes.

  For instance, this is the policy used by SORW if *SWITCH_BENEFIT* is greater than the *ST* value together with a simple round robin scheme, since

it becomes less convenient to pay energy for the boot phase than keeping sensors always awake.

- **SWLP**: nodes that have been used assume a *SW* mode, while those not used assume the *LP* one. This solution is particularly suitable for situations in which it is better to make sensors sleep after they have been used, perhaps because they will be used again after a long time.

- **SWSW**: both nodes that have been used and those that have been not used assume a *LP* mode. This strategy is particularly suitable for those cases in which the slot duration is very high, as well as the number of available nodes. In fact, it is very convenient to turn off all sensors, since a long time passes before a new detection, both between two consecutive nodes and for the same node. For instance, this is the strategy adopted by SORW if $SWITCH\_BENEFIT < 1$ (equation 7 in chapter 4), meaning that it is more convenient to always put nodes in SW mode, since keeping them in LP would consume a lot more.

The following scheduling algorithms have been deployed in order to make a comparison with the SORW one. For this reason, together with the nodes selection, they also have to implement the detection phase and the consequent update of the sensors information, like their energy and their mode. The algorithm in 2 shows exactly which kind of operations are performed according to the policy required. This step is identical for these algorithms, and for this reason, each algorithm calls the following routine before each new step.

Algorithm 2 Update sensors function

function UPDATE_SENSORS(`G`, `mode`)
  $\phi(s_j, t_{time}) = 1, \forall s_j \in G$
  if mode = LPSW **then**
    $S(s_j, t_{time}) = 0, \forall s_j \in N \setminus G$
    $S(s_j, t_{time}) = 1, \forall s_j \in G$
  else if mode = LPLP **then**
    $S(s_j, t_{time}) = 1, \forall s_j \in N$
  else if mode = SWLP **then**
    $S(s_j, t_{time}) = 0, \forall s_j \in G$
    $S(s_j, t_{time}) = 1, \forall s_j \in N \setminus G$
  else if mode = SWSW **then**
    $S(s_j, t_{time}) = 0, \forall s_j \in N$
  end if
  $S(s_j, t_{time}) = 1, \forall s_j \in G$
end function

6.1.1 Simple Round Robin Scheduler

Each turn, the simple round robin scheduler selects λ nodes from N, according to a classic round robin policy. In particular, the $G_0 = \{s_j, s_{j+1}, \ldots, s_{(j+\lambda-1)\%n}\}$ set of nodes is used, with n that indicates the total number of nodes and j that is incremented by one each turn. Then, depending on the mode used for sensors, they are updated accordingly.

Algorithm 3 Simple Round Robin algorithm

1: $time = 0$
2: $j = 0$
3: **while** $\sum_{s_i \in N} A(s_i, t_{time}) \geq \lambda$ **do**
4: $\quad G_0 \leftarrow \{s_j, s_{j+1}, \ldots, s_{(j+\lambda-1)\%n}\}$
5: $\quad update\_sensors(G_0,\ mode)$
6: $\quad j = j + 1$
7: $\quad time = time + 1$
8: **end while**

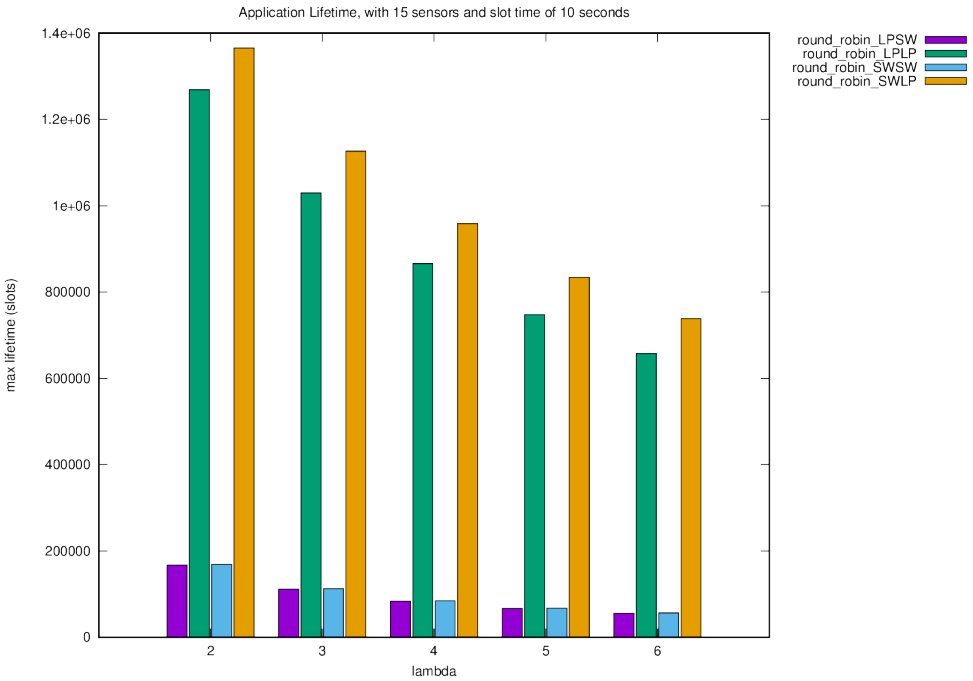

Figure 6.1: Comparison of lifetimes obtained by using all the different policies defined for the round robin algorithm, with $\lambda \in [2, 6]$, 15 sensors and slot time of 10 seconds.

As shown in figure 6.1, for this kind of algorithm, the *SWLP* policy seems to work better than the others. This is due to the fact that sensors nodes are quickly

alternated, and they have to wait for a complete round robin lap before being used again. For this reason, the best solution is to set them to sleep after the detection and to wake-up the next ones, that will hence be ready for their turn. Basing on this assumption, the *SWLP* policy is used for the rest of the comparisons in the simple round robin case.

6.1.2 Greedy Scheduler

The greedy scheduler selects λ nodes from N, each turn choosing the nodes that are most charged and have the lowest indexes. For this reason, every turn the list of nodes G is sorted in order to keep the nodes with more energy and the lowest indexes in the head. In this way, it is just a matter of selecting the first λ nodes starting from the head of the list.

Algorithm 4 Greedy algorithm

1: $time = 0$
2: $List\ L = newList()$
3: **for all** $l \in N$ **do**
4: $\quad L.push(l)$
5: **end for**
6: **while** $\sum_{s_i \in N} A(s_i, t_{time}) \geq \lambda$ **do**
7: $\quad \triangleright$ sort L to have sensors with more energy and with lowest indexes in the head
8: $\quad L \leftarrow L.sort()$
9: $\quad G_0 \leftarrow \{L[0], L[1], \ldots, L[\lambda - 1]\}$
10: $\quad update\_sensors(G_0,\ mode)$
11: $\quad time = time + 1$
12: **end while**

Figure 6.2 shows the behaviour of policies for the greedy algorithm. Round robin and greedy algorithms obtain exactly the same values, since sensors start with the same energy and because round robin can be considered as a greedy algorithm, where the strategy is to firstly consider nodes with lowest indexes and lowest times of usage. Of course this would not be true if sensors had different energy at the beginning, or if the distance influenced the amount of energy used in each slot, but for this particular instance of the problem, the two algorithms can be considered as the same one and for this reason only round robin will be considered in the next analysis.

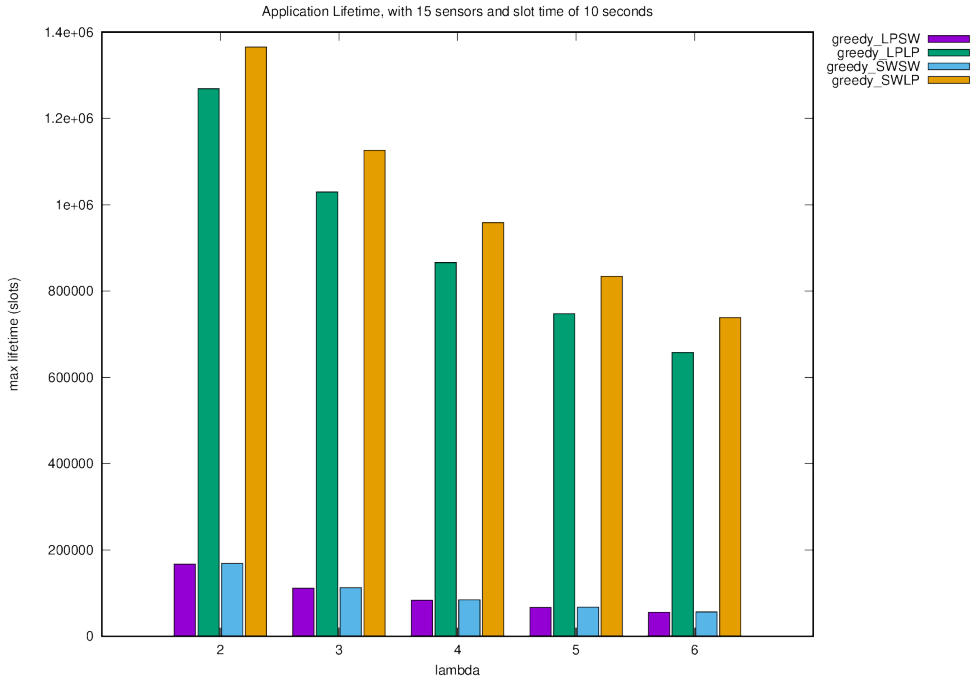

Figure 6.2: Comparison of lifetimes obtained by using all the different policies defined for the greedy algorithm, with $\lambda \in [2, 6]$, 15 sensors and slot time of 10 seconds.

6.1.3 Probabilistic

The probabilistic scheduler randomly selects λ nodes from N each turn and repeats the same operation until there are at least λ sensors with enough energy for a new round. Each turn a sensor can be selected with a probability equal to its residual energy and normalized with respect to all the others. In this way, at the end of the simulation, sensors should have more or less the same energy left.

As shown in figure 6.3, the best policy for the probabilistic algorithm is the *LPLP*. This is due to the fact that nodes are randomly selected and so their previous mode is not taken into account while deciding which mode will actually be adopted. This means that, since nodes can be used/awaken very frequently, it is more convenient to keep them always ready for a new detection instead of paying for several boot phases.

Results obtained with different simulations in OMNet++ are always identical, since the same parameters are used even for each instance of the simulation. While this is true for the *round_robin*, the *greedy*, and the *SORW* case, where initial configurations are identical to each repetition, this is no longer true for the probabilistic case. In fact, randomness introduces uncertainty about which nodes will be selected each turn, and so about how their energy will be used. Since this affects the lifetime, more repetitions of the probabilistic simulation are required

Algorithm 5 Probabilistic algorithm

```
1: time = 0
2: while ∑_{s_i∈N} A(s_i, t_time) ≥ λ do
3:     i = 0
4:     G_0 ← ∅
5:     while i < λ do
6:         l ← RANDOMLY_SELECT_NODE(N)    ▷ Randomly select a node with enough energy
7:         if l ∉ G_0 and A(l, t_time) = 1 then
8:             G_0 ← G_0 ∪ {l}
9:             i = i + 1
10:        end if
11:    end while
12:    update_sensors(G_0)
13:    time = time + 1
14: end while
```

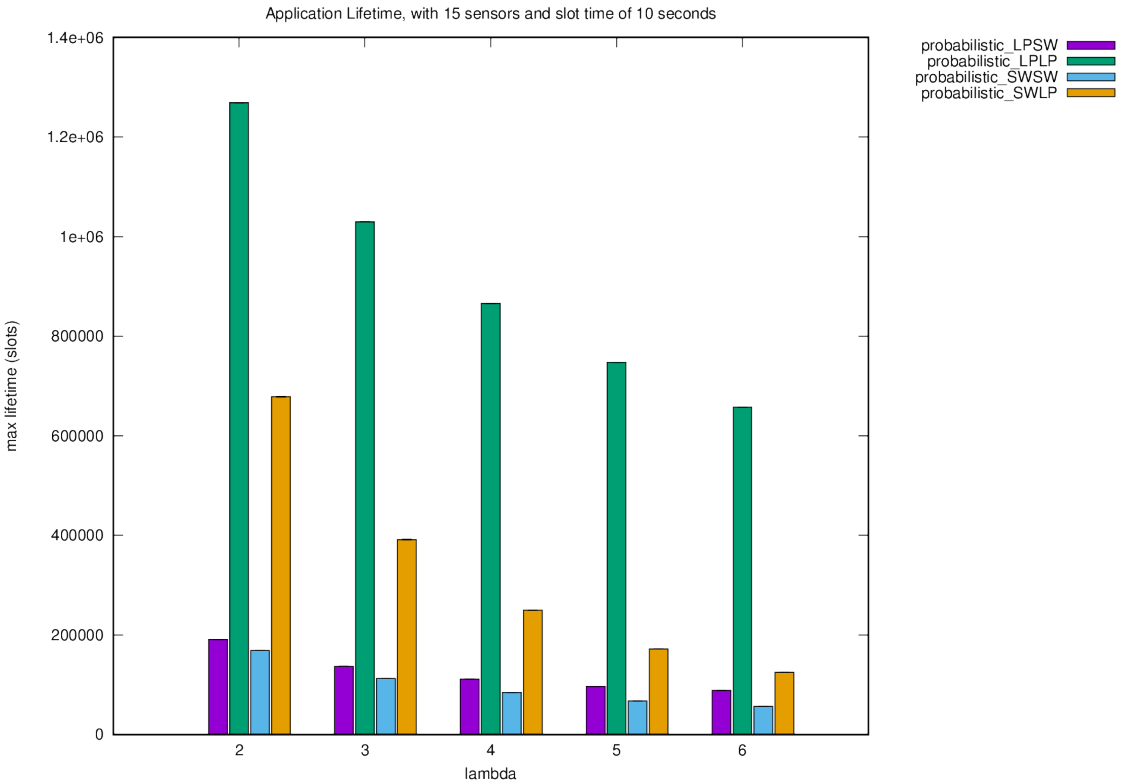

Figure 6.3: Comparison of lifetimes obtained by using all the different policies defined for the probabilistic algorithm, with $\lambda \in [2, 6]$, 15 sensors and slot time of 10 seconds.

for obtaining an interesting value. Nevertheless, figure 6.3 includes the confidence interval for the lifetime that actually is not visible because of its shortness, as confirmed by results in table 6.1, also including their confidence interval.

λ	**mean**	**confidence interval**
2	1269048	[1269042.01937, 1269053.98063]
3	1029661	[1029659.41032, 1029664.25635]
4	866260	[866256.639517, 866264.360483]
5	747616	[747614.417645, 747618.915689]
6	657557	[657552.487301, 657562.179366]

Table 6.1: Results and their confidence interval obtained by making 6 different simulations of the probabilistic scheduler using the *LPLP* policy, for a network composed of 15 sensors, $\lambda \in [2, 6]$, a time slot of 10 seconds, and $\alpha = 0.95$.

6.2 Analysis

In this section I will make some comparisons of lifetimes obtained by varying the parameters which may affect the system performance. In particular, it is interesting to see how SORW reacts to a variation of the λ value, since this parameter strongly affects the way groups are formed in the two stages of the algorithm. At the same time, another important element that can produce different results is a change in the number of sensors involved. In fact, again, this is another way to build the sets of nodes required by SORW. Finally, modifying the slot duration can be important to understand if SORW is suitable for applications that require a frequent number of updates and those that instead require only long-term updates, perhaps for monitoring environments that rarely alter their state.

6.2.1 Load Analysis

A first evaluation of SORW can be made by looking at its behaviour while varying the λ value, since this parameter affects the way sets of nodes are made before the beginning of stages. In fact, a good composition for sets means a small waste of resources: the bigger the set of nodes for the `ROUND_ROBIN` stage is, the more residual energy will exceed, since nodes are not discharged uniformly. So, for these cases it is important to see if other algorithms obtain better results. As results in figure 6.4 confirmed, SORW keeps obtaining the best results, even if reducing its performance, i.e., the number of detections from sensors. This is due to the fact that a bigger number of requests for the same set of sensors implies that they are used more intensively and so they are discharged faster.

Another important aspect is that, if increasing λ while keeping fixed the number of sensors and the slot duration, differences with other algorithms are less important. This is because the G_0 set of nodes for the `ROUND_ROBIN` stage in SORW becomes larger, and since the approach is very close to the round robin algorithm, results are still different but with a smaller gap.

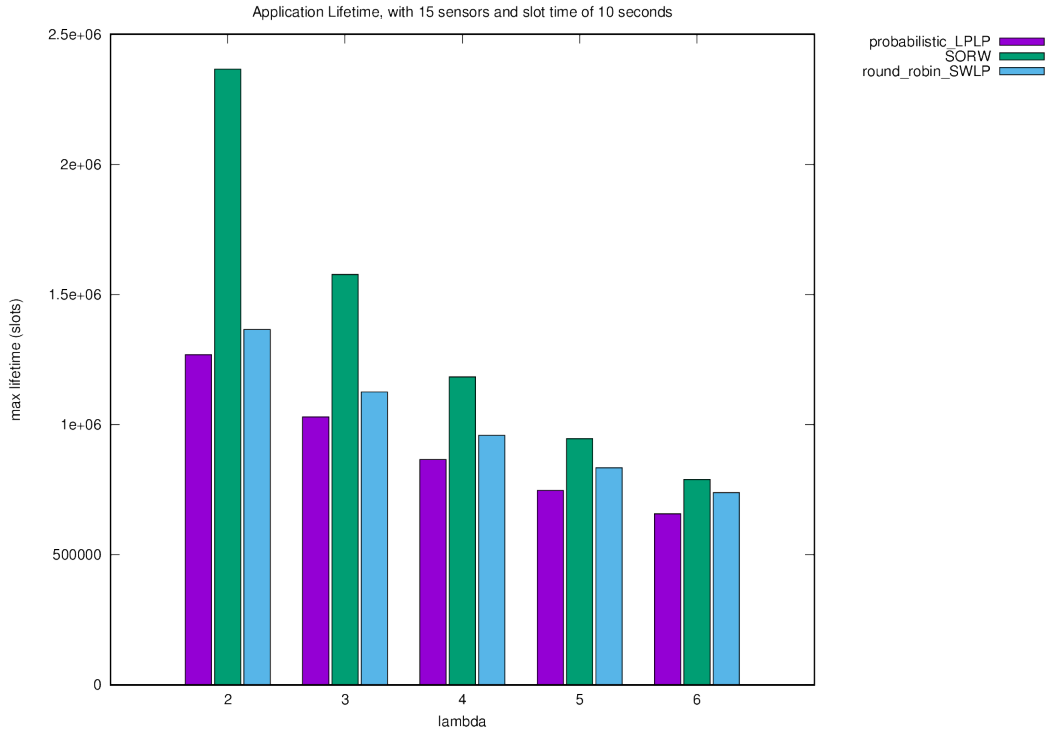

Figure 6.4: Comparison of lifetimes obtained by using SORW, *probabilistic_ LPLP*, and *round_ robin_ SWLP*, with $\lambda \in [2, 6]$, 15 sensors and slot time of 10 seconds.

6.2.2 Nodes Density Analysis

Increasing the number of nodes while keeping fixed the λ value is another key point to understand if SORW obtains better results than other algorithms, since, again, the composition of sets for the algorithm is strongly affected by the number of nodes involved and the number of detections required. Nevertheless, in this situation results are quite predictable: as represented in figure 6.5, the same number of data requests for bigger numbers of nodes leads to a constant increment of the number of rounds the network supports, and so a bigger lifetime.

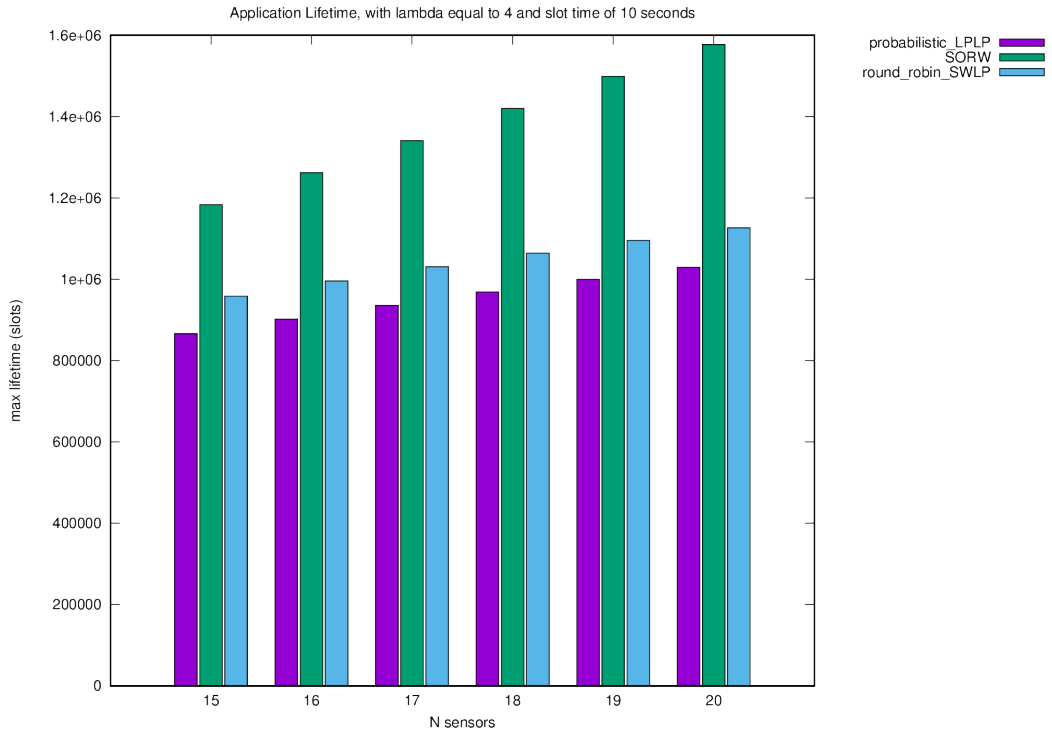

Figure 6.5: Comparison of lifetimes obtained by using SORW, *probabilistic_LPLP*, and *round_robin_SWLP*, with $\lambda = 4$, number of sensors $\in [15, 20]$ and slot time of 10 seconds.

6.2.3 Duty Cycle Analysis

The most important variation of parameters is the slot duration one. In fact, this affects the amount of energy E_{STB} required by the low power mode and consequently the total amount of energy $E_{STD} = E_{ON} + E_{STB}$ required for a data transmission. As underlined in chapter 4, SORW reacts differently depending on the variation of these values: in particular, if the slot time is quite big, SORW could disable the SW mode, since for two different detections for the same sensor it would be better to pay just a E_{BOOT} amount of energy instead of E_{STD} multiplied by the number of slots that sensor has to wait between the two detections. On the contrary, with a small value of the slot duration, possibly with also a high number of detections and a small number of sensors still available, SORW could use just the LP mode since nodes are frequently alternated and it is more convenient to always stay awake than to pay several E_{BOOT} quantities of energy.

Slot duration is also a typical parameter that characterizes the QoS of an application, especially for monitoring ones. Hence, it is very important to understand if SORW is suitable for specific kinds of applications or can be used for different ones.

Figure 6.6 shows a comparison of lifetimes obtained by using *SORW*, *probabilistic_LPLP*,

probabilistic_SWSW, *probabilistic_SWSW*, and *round_robin_SWSW* algorithms, while using 15 sensors and asking for 4 detections per slot. Two fundamental aspects can be underlined:

- For small time slots SORW works better than all the others. This can be explained by the fact that it smartly manages the LP and the SW modes, while the others alternate them in a static manner. Differences are amplified if considering only the algorithms used for all the previous analysis and bigger time slots, since it becomes too expensive to use the LP mode. In fact, it is always more convenient to put a sensor to sleep than to keep it awake and ready for a new request, since it means a significant amount of energy saved. For this specific scenario, there is a value of time slot between 60 seconds and 3600 seconds that makes SORW use only the SW mode. From that value on, the lifetime obtained is exactly the same, since LP mode is not used and then E_{STB} value will not influence the energy consumption of a sensor.

- For longer time slots, differences disappear if comparing SORW with the *probabilistic* and the *round_robin* algorithms both in SW mode. This is due to the fact that in these cases they use the same exact strategy: they select λ nodes, wake them up and wait for the data. For this reason, even if the selection operation can be different, sensors are always discharged uniformly and so in the end the lifetime is the maximum number of times that algorithms can repeat the previous sequence of operations.

Considering only SORW, despite lifetimes obtained are numerically identical starting from a precise value of slot time, results should not be interpreted in the same way. In fact, according to 4.2.1, the application lifetime is defined as the maximum number of time slots in which the WSN is able to provide λ detections, but it could also be read as the maximum period of time, expressed for instance in days, of activity of the WSN. In this way, clearly, the longer is the slot time, the higher number of days sensors will conserve their energy and be able to provide the service requested. Figure 6.7 shows the same scenario for the previous plot, but considering lifetime expressed in days instead of the number of slots. As shown, differences in terms of days among high-intensive applications, i.e., with a small slot time, and not-intensive, i.e., with a high slot time, are strongly amplified. In particular, it is interesting to see that for a slot time of 10 seconds the lifetime is about 137 days, while for a slot time of 1 day, the maximum duration is more than 231 years. Clearly, this is a theoretical value that takes into account only a single transmission for each data, i.e., without considering all the possible problems related to the network. Furthermore, lithium batteries suffer from self-discharge, typically losing a constant amount of charge each month. Despite this, the value obtained for lifetimes can be interesting for monitoring applications, where the

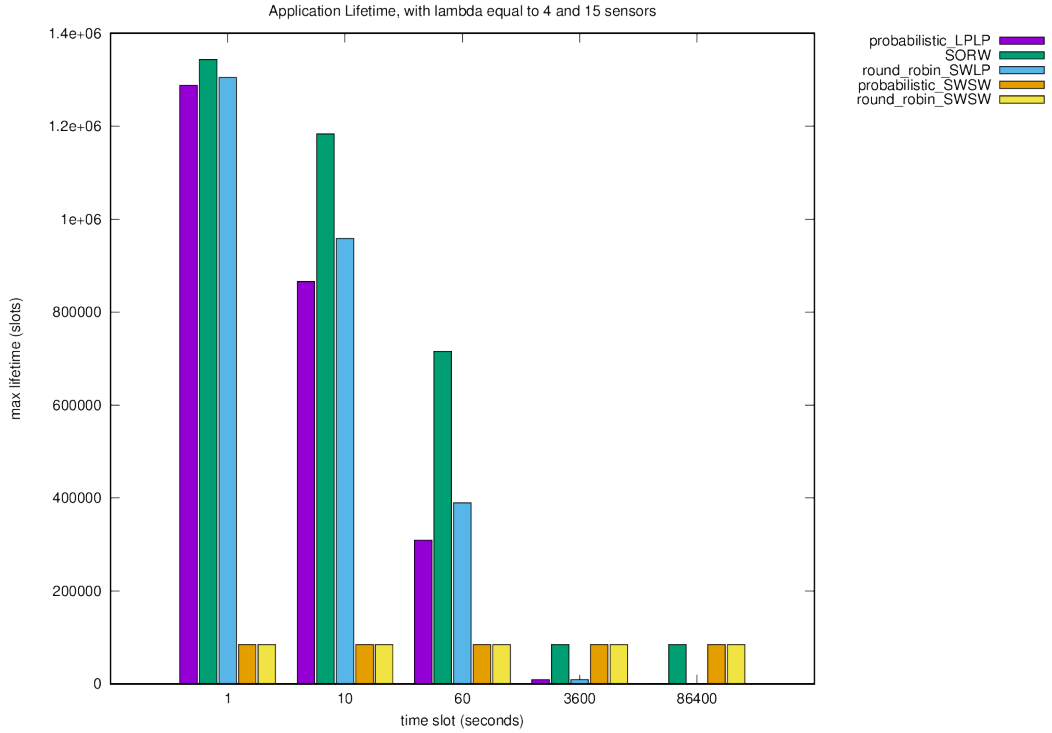

Figure 6.6: Comparison of lifetimes obtained by using SORW, *probabilistic_LPLP*, *probabilistic_SWSW*, *round_robin_SWSW*, and *round_robin_SWLP*, with $\lambda = 4$, 15 sensors and slot time $\in \{1, 10, 60, 3600, 86400\}$.

replacement or recharge of batteries is often impossible. In conclusion, the most important result obtained is that SORW is suitable for applications with different requirements in terms of time slots and it seems very flexible, since it can adapt its behaviour if parameters change, without decreasing its performance.

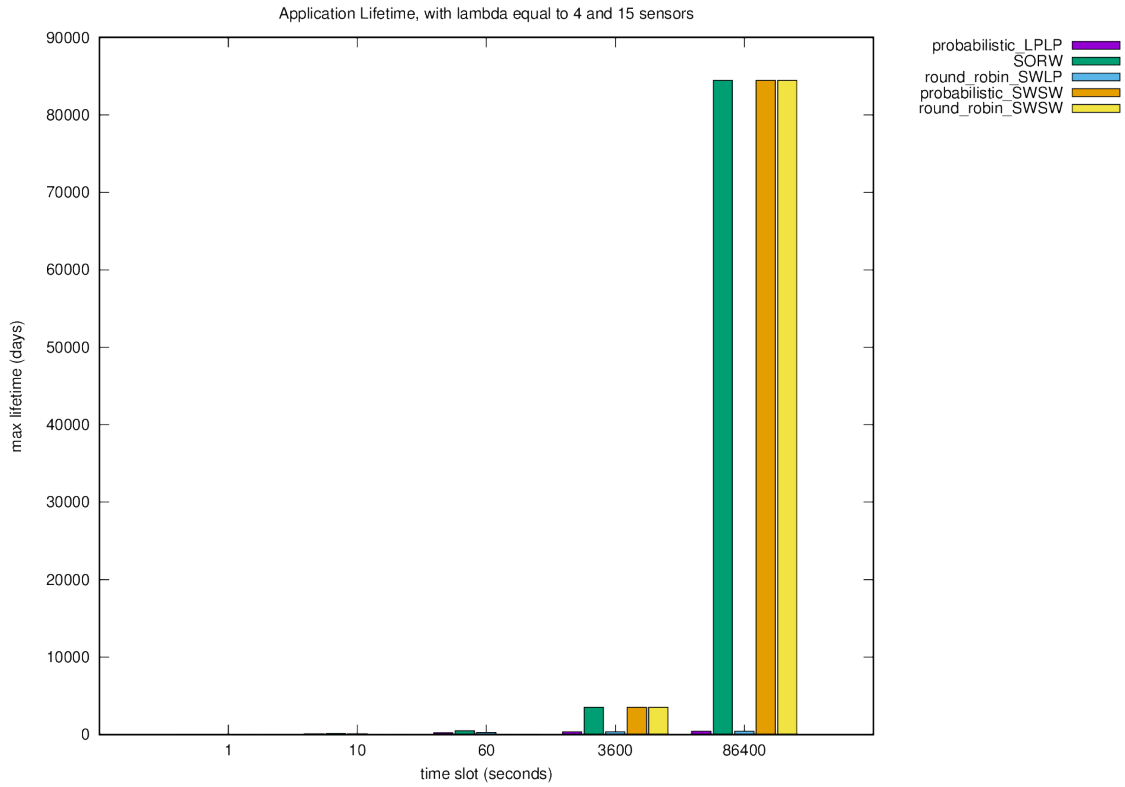

Figure 6.7: Comparison of lifetimes (in days) obtained by using SORW, *probabilistic_LPLP*, *probabilistic_SWSW*, *round_robin_SWSW*, and *round_robin_SWLP*, with $\lambda = 4$, 15 sensors and slot time $\in \{1, 10, 60, 3600, 86400\}$.

Chapter 7

Conclusions

The objective of this work was to prove the feasibility of the deployment of a wireless sensor network (WSN) with the possibility to wake-up and put in sleep mode each sensor involved on demand, through the use of a radio wake-up energy harvesting system. In particular, the goal was to find a good scheduler to maximize the period of activity of such network.

For this purpose, I analyzed the solutions currently available for the energy optimization in WSNs, most of the times based on an attempt to improve the approach of the MAC protocol. Then, I described the characteristics and the requirements of a WSN that could be used in this project. In particular, sensors are equipped with a radio wake-up system that lets them completely turn off when they do not have to send their data, or otherwise be awaken on demand. Hence, they can assume two different modes: the SW mode indicates the sensor is sleeping and it is not consuming energy at all, while the LP mode means the sensor is awake and ready for sending its data if needed. The first requires a well defined amount of energy irradiated by an external device in order to reactivate the sensor's circuit, i.e., waking up the sensor, the second instead requires a constant amount of energy per unit of time, regardless the fact the sensor will send its data or not. In this way, the system reduces the energy consumption by optimizing the sensors' intervals of activity, and consequently maximizing the application lifetime of the network, namely the period in which the network is able to provide the service it is designed for.

Then I defined the system model and I formulated the generic problem for a WSN with such sensors and the following requirements: a number λ detections should be acquired in each unit of time from λ different sensors, and for each unit of time, sensors can assume the SW mode or the LP one.

After that, I presented the *Scheduled OnDemand Radio Wake-Up (SORW)* scheduler, i.e., a scheduler that keeps the best both from the schedule based 3.3.1.3 and the on demand schemes 3.3.1.3. SORW is designed to integrate a *Radio*

Wake-Up System, capable to awake sensors when required and to let them go into sleeping mode otherwise.

Through the OMNet++ simulation environment, I implemented SORW and I made some tests in order to understand its behaviour if really deployed. Results obtained from simulations were first compared to those obtained by the analytic model and then to those of other kinds of algorithms that could be suitable for a WSN with such characteristics. In particular, I studied the performance of different schedulers while varying some parameters that strongly affect the lifetime of the network, such as the number of detections requested per unit of time, the duration of the interval time between two different requests, and the number of sensors involved in the WSN.

By using SORW, results are definitely improved with respect to another simple scheduler that could be used for this kind of WSN and the lifetime obtained is about years, an important value that makes the radio wake-up technology an interesting field of IoT where research and investments should be focused on.

7.1 Future Developments

This work can be considered as a first step of a more complex and complete project, in order to understand if this approach is suitable for a WSN with sensors that can be energy harvested by using the radio wake-up technology. In particular, some important requirements will be taken into account in future developments, like:

- considering also a variable distance between the radio wake-up irradiator and the sensor to be awaken. Nevertheless, another way could be considering the possibility of using mobile irradiators, that hence will keep a fixed and constant distance by physically reaching the sensor.

- considering also the application this WSN is designed for. In fact, different kinds of applications can require different policies of QoS (quality of service), that can affect the packet size and the protocol for the data communication. Different approaches in this sense can affect the number of transmissions required as well as the energy consumption.

- considering also the *active* status of a sensor.

- using clusters of sensors as abstraction of a node. In large-scale environments, where a huge number of sensors are deployed, it is more useful to consider groups of them as a node instead of single ones. Of course, they need both an inter-scheduler and an intra-scheduler solution.

- making simulations also for larger environments, taking into account also all the other requirements for a network, such as the throughput, the error data rate, the interferences, the delivery ratio, etc..
- validation in a real test-bed by using sensors described in 4.1

In conclusion, this work proved the feasibility of a more complex project that involves innovative technologies, and showed how a well defined scheduler for the radio wake-up technology can definitely improve the efficiency of a WSN in terms of energy saving and lifetime.

Bibliography

[1] OMNeT++ Discrete Event Simulator. `https://omnetpp.org/`. Accessed in 2017.

[2] Simulation of Urban MObility. `http://sumo.dlr.de/index.html`. Accessed in 2017.

[3] Techtimes - news about silicon valley companies, technology innovations, and other cool stuff.

[4] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[5] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad hoc networks*, 7(3):537–568, 2009.

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[7] Michael Buettner, Gary V Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320. ACM, 2006.

[8] Ricardo C Carrano, Diego Passos, Luiz CS Magalhaes, and Celio VN Albuquerque. Survey and taxonomy of duty cycling mechanisms in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 16(1):181–194, 2014.

[9] Lin Gu and John A Stankovic. Radio-triggered wake-up for wireless sensor networks. *Real-Time Systems*, 29(2):157–182, 2005.

[10] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[11] Ranieri Guerra, Alessandro Finocchiaro, Giuseppe Papotto, Benedetta Messina, Leandro Grasso, Roberto La Rosa, Giulio Zoppi, Giuseppe Notarangelo, and Giuseppe Palmisano. An rf-powered fsk/ask receiver for remotely controlled systems. In *Radio Frequency Integrated Circuits Symposium (RFIC), 2016 IEEE*, pages 226–229. IEEE, 2016.

[12] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System sciences, 2000. Proceedings of the 33rd annual Hawaii international conference on*, pages 10–pp. IEEE, 2000.

[13] IoT-A Consortium. Internet of things – architecture, iot-a, 2013.

[14] Jehn-Ruey Jiang, Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Hwang Lai. Quorum-based asynchronous power-saving protocols for ieee 802.11 ad hoc networks. *Mobile Networks and Applications*, 10(1-2):169–181, 2005.

[15] Daniela Kruger, Dennis Pfisterer, and Stefan Fischer. Cupid-communication pattern informed duty cycling in sensor networks. In *Systems and Networks Communications (ICSNC), 2010 Fifth International Conference on*, pages 70–75. IEEE, 2010.

[16] Roberto La Rosa, Natale Aiello, and Giulio Zoppi. Rf remotely-powered integrated system to nullify standby power consumption in electrical appliances. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 1162–1164. IEEE, 2016.

[17] Philippe Le-Huy and Sébastien Roy. Low-power wake-up radio for wireless sensor networks. *Mobile Networks and Applications*, 15(2):226–236, 2010.

[18] S-YR Li, Raymond W Yeung, and Ning Cai. Linear network coding. *IEEE transactions on information theory*, 49(2):371–381, 2003.

[19] Gang Lu, Bhaskar Krishnamachari, and Cauligi S Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 224. IEEE, 2004.

[20] Stevan Marinkovic and Emanuel Popovici. Nano-power wake-up radio circuit for wireless body area networks. In *Radio and Wireless Symposium (RWS), 2011 IEEE*, pages 398–401. IEEE, 2011.

[21] Stephan Olariu and Ivan Stojmenovic. Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform

distribution and uniform reporting. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12. IEEE, 2006.

[22] Joaquim Oller, Ilker Demirkol, Jordi Casademont, Josep Paradells, Gerd Ulrich Gamm, and Leonhard Reindl. Has time come to switch from duty-cycled mac protocols to wake-up radio for wireless sensor networks? *IEEE/ACM Transactions on Networking*, 24(2):674–687, 2016.

[23] Sumeet N Parmar, Sukumar Nandi, and Atanu Roy Chowdhury. Power efficient and low latency mac for wireless sensor networks. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, volume 3, pages 940–944. IEEE, 2006.

[24] Vamsi Paruchuri, Shivakumar Basavaraju, Arjan Durresi, Rajgopal Kannan, and S Sitharama Iyengar. Random asynchronous wakeup protocol for sensor networks. In *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pages 710–717. IEEE, 2004.

[25] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107. ACM, 2004.

[26] Venkatesh Rajendran, Katia Obraczka, and Jose Joaquin Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless networks*, 12(1):63–78, 2006.

[27] Rashmi Ranjan Rout and Soumya K Ghosh. Enhancement of lifetime using duty cycle and network coding in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 12(2):656–667, 2013.

[28] STMicroelectronics NV. SPIRIT1 radio. `http://www.st.com/resource/en/datasheet/spirit1.pdf`, 2015.

[29] STMicroelectronics NV. STM32L1 MCU. `http://www.st.com/resource/en/datasheet/stm32l151c6.pdf`, 2016.

[30] STMicroelectronics NV. STTS751 digital temperature sensor. `www.st.com/resource/en/datasheet/stts751.pdf`, 2016.

[31] Yanjun Sun, Omer Gurewitz, and David B Johnson. Ri-mac: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 1–14. ACM, 2008.

[32] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36, 2010.

[33] Tijs Van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180. ACM, 2003.

[34] Maarten Weyn, Glenn Ergeerts, Rafael Berkvens, Bartosz Wojciechowski, and Yordan Tabakov. Dash7 alliance protocol 1.0: Low-power, mid-range sensor and actuator communication. In *Standards for Communications and Networking (CSCN), 2015 IEEE Conference on*, pages 54–59. IEEE, 2015.

[35] Xiaobing Wu, Guihai Chen, and Sajal K Das. Avoiding energy holes in wireless sensor networks with nonuniform node distribution. *IEEE Transactions on parallel and distributed systems*, 19(5):710–720, 2008.

[36] Fan Xiangning and Song Yulin. Improvement on leach protocol of wireless sensor network. In *Sensor Technologies and Applications, 2007. SensorComm 2007. International Conference on*, pages 260–264. IEEE, 2007.

[37] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576. IEEE, 2002.

[38] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer networks*, 52(12):2292–2330, 2008.

[39] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.