Energy Efficient Programming for SONAR Networks

Vladir Vicente
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2016

Orientador
Luís Lopes, Professor Associado, Faculdade de Ciências da Universidade do Porto

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /

Pa nhâ mai ku nhâs irmás!

Acknowledgments

I would like to say thank you, to my adviser and Professor, Dr. Luís Lopes, for the guidance, patience, corrections, and the opportunity to work in this project, which I am very grateful for.

I would also like to thank my friend and colleague, Gil Ferro, for the help and the guidance he provided along the way.

A special thanks to my Aunt Lucília for all the support and help provided in the pursuit of a college education.

Thank you, to all my family, whom i miss dearly, and never stopped supporting me, through thick and thin.

Thank you, to my all my friends, who have been my family in all this years I have been away from home.

A very special thanks to my little sister Laussa, and my big sister Vânea, who have always been there for me, no matter what.

Finally, I would like to thank the best person in the world, whom I have no words to describe, my mother. Thank you for my being my mother, my father, and my friend. Thank you for everything. May I never let you down.

Resumo

A disponibilidade de energia é uma das maiores limitações das redes de sensores sem fios. Normalmente alimentadas por baterias, estas redes são frequentemente implantadas em áreas inacessíveis onde a substituição das baterias pode ser difícil ou até mesmo impossível. Por este motivo as estratégias de conservação de energia são cruciais para garantir um tempo de vida útil adequado das redes depois de colocadas no terreno. Em virtude do seu desenho, o rádio é a componente que mais consome energia e por isso os protocolos de comunicação por elas usados são desenhados para minimizarem a actividade do rádio.

Neste trabalho partimos de uma plataforma de software que pretende agilizar a recolha de séries temporais de dados a partir de redes de sensores, designada por Sensor Observation aNd Actuation aRchitecture (SONAR), que não utiliza nenhuma técnica de conservação aplicada a priori. O trabalho desenvolve-se no sentido de testar e integrar na plataforma esquemas de conservação de energia, focando-se principalmente na utilização do rádio, especificamente na percentagem de tempo em que o rádio está activo, um mecanismo conhecido como "duty-cycling". Os testes foram realizados com duas topologias diferentes de rede (estrela e mesh) e duas tecnologias diferentes (ZigBee e DigiMesh).

Os esquemas de "duty-cycling" testados permitiram dimunuir significativamente o consumo energético dos dispositivos. A sua implementação teve impactos no desenho e na implementação da plataforma SONAR original que também foram avaliados.

Abstract

The energy availability is one of the biggest limitations in Wireless Sensor Networks. Typically powered by batteries, these networks are frequently deployed in inaccessible environments where battery replacement is a difficult or even impossible task. Because of this reason, energy conservation strategies are crucial to guarantee an adequate life spawn in the networks after their deployment in the field. By the virtue of its design, the radio is the main power sink component, thus the communication protocols used by it, are designed to minimize the radio activity.

In this work we started from a software platform that aims to optimize the collection of time series of data from a sensor network designated by SONAR, which does not use any energy conservation technique beforehand. The work develops in the sense of testing and integrating energy conservation schemes in the platform, mainly focusing on the radio utilization, specifically in the percentage of time the radio is active, a mechanism known as "duty-cycling". The tests were performed with two different topologies (star and mesh) and two different technologies (ZigBee and DigiMesh).

The tested "duty-cycling" schemes allowed a significant energy conservation in the devices. Their implementation also had impacts in the design and implementation of the original SONAR platform, which was also evaluated.

Acronyms

6LoWPan	IPv6 over Low power Wireless Personal Area Networks. 11
AODV	Ad-Hoc On-demand Distance Vector. 10
API	Application Programming Interface. 6
APS	Application Support Sub-layer. 10
ASCENT	Adaptive Self-Configuring sEnsor Networks Topologies. 14
ATT	ATTribute Protocol. 12
BLE	Bluetooth Low Energy. 11, 12
CCA	Clear Channel Assessment. 10
CPU	Central Processing Unit. 2, 15
CSMA	Carrier Sense Multiple Access. 10
CSMA-CA	Carrier Sense Multiple Access-Collision Avoidance. 9, 10
DPM	Dynamic Power Management. 7
DSSS	Direct-Sequence Spread Spectrum. 10
EDF	Earliest Deadline First. 7, 33

FHSS Frequency-Hopping Spread Spectrum. 10

GAF Geographical Adaptive Fidelity. 14, 15

GATT Generic Attribute Profile. 12

HCI Host Controller Interface. 11

HMTL5 Hypertext Markup Language, version 5. 34

HTTP Hypertext Transfer Protocol. 33

IEEE Institute of Electrical and Electronics Engineers. 9–11

IP Internet Protocol. 11

IPv6 Internet Protocol version 6. 11

ISA Instruction Set Architecture. 9

JVM Java Virtual Machine. 8

L2CAP Logical Link Control and Application Control. 12

MAC Medium Access Control. 9, 10, 17, 18

MTU Maximum Transmission Unit. 11

OS Operating System. 2, 5–9, 24, 31

P/S Publish/Subscribe. 21, 23, 25, 34

PAN Personal Area Network. 9

PHY PHYsical layer. 9, 10

RF Radio Frequency. 10, 12, 23, 24, 27

RTC real time clock. 7, 24, 31, 33

SIG Special Interest Group. 11

SMP Security Manager Protocol. 12

SONAR Sensor Observation aNd Actuation aRchitecture. 2, 4, 7, 9, 21–24, 30, 31, 33, 34

SSE Server Sent Events. 33, 34

SSI Single System Image. 8

STEM Sparse Topology and Energy Management. 15

STL Sonar Task Language. 9

SVM Sonar Virtual Machine. 9, 31

TDMA Time Division Multiple Access. 17, 18

VM Virtual Machine. 8, 24

WPAN Wireless Personal Area Network. 10, 11

WSN Wireless Sensor Network. 1, 2, 4, 5, 10, 12, 18, 24, 25, 27, 30, 31

ZDO ZigBee Device Objects. 10

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless Sensor Networks (WSN) can be defined as a set of tiny, programmable, low-cost embedded devices, which have the functionality of sensing the physical world and disseminate the sensed information via radio links [22].

Their manifold applications in a variety of contexts (e.g., scientific, industrial, social) combined with the need to monitor and measure the physical world contributed for their evolution and their increase of use.

Figure 1.1 is an example of a WSN with a mesh topology. The figure presents a set of nodes responsible for acquiring physical data and route it to the gateway (described as “Collecting Node”) via radio links. The data can be processed, stored, and accessed by the authorized personal locally or via a web client.

1.1 Motivation and Goals

WSN present several limitations, such as computational power, memory and power consumption, the latter being crucial for lifespan [18, 21]. Indeed, WSN are often deployed in hostile or extreme environments, e.g., when used for industrial or environmental monitor-

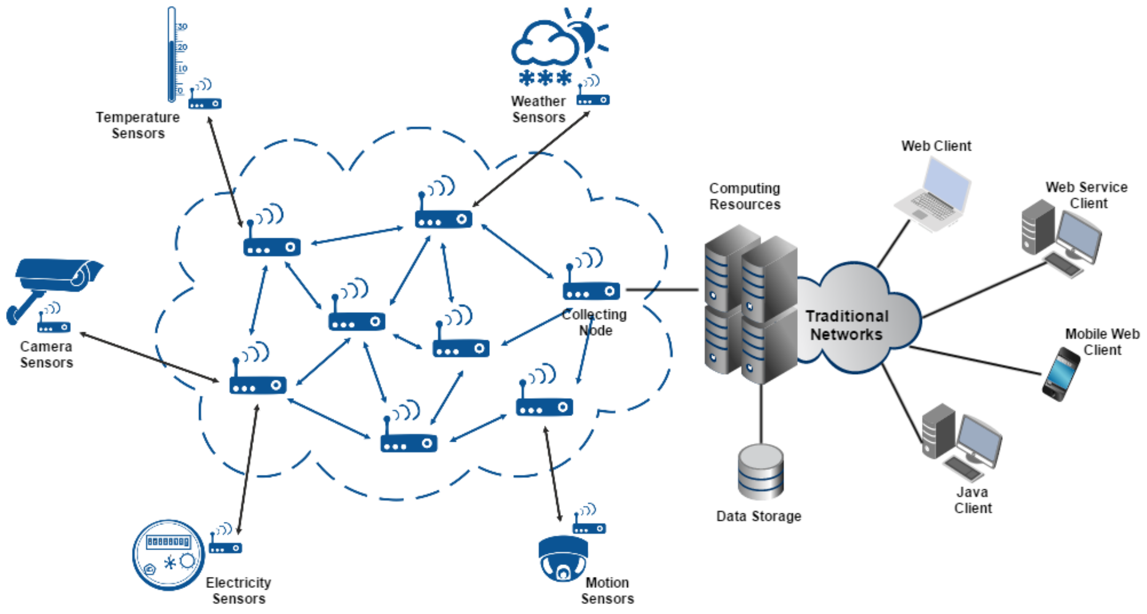


Figure 1.1: A wireless sensor network.

ing [27]. In these conditions battery replacement is often quite dangerous, difficult or even impossible.

The use of mesh topologies is of great value since they provide the redundancy required to make data from all nodes available at the gateway. Mesh networks, however, can be challenging since nodes in such a topology usually behave in more complex ways and require more energy when compared to nodes in other topologies, e.g., star.

We use a 3-layer – client, broker and data – middleware called SONAR [11,12]. The client layer provides graphical and command line interfaces to the end users for accessing data streams from WSN deployments. The broker layer implements a publish/subscribe bus that allows data from deployments to flow to interested clients and a WSN management service that can be used to setup and administer SONAR deployments. The data layer abstracts the low level complexity of the hardware of the WSN deployments by using two software components – the SONAR virtual machine and operating system.

The work presented in this thesis focuses on the SONAR data layer and aims to:

- implement energy conservation schemes for deployments using the SONAR middleware;
- evaluate the extent of the changes required to the middleware, especially in the data layer, to implement such schemes;
- evaluate the gains in the lifespan of the deployments when using the implemented energy conservation schemes.

1.2 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents a survey about relevant work developed in the WSN area. We describe the state of the art for operating systems and virtual machines for WSN. We also present the most used wireless communication technologies for WSN which are, by definition, based on low-power technology and protocols. We overview current energy conservation schemes for wireless networks, focusing in duty-cycling mechanisms. Chapter 3 presents an overview of the SONAR middleware. Chapter 4 presents possible energy conservation schemes for SONAR and describes their implementations. Chapter 5 presents an evaluation of the implemented energy conservation schemes and discusses the results. Chapter 6 presents the conclusions and discusses possible directions in future work.

Chapter 2

Related Work

In this chapter we will present a brief overview of the work already developed related with Wireless Sensor Networks. We will cover some of the most notable Operating Systems and Virtual Machines designed specifically for WSN. We will also present a brief overview of the wireless communication technologies used in WSN's, and at last the existing energy conservation schemes.

2.1 Operating Systems

Operating Systems for WSN's follow a different design approach than traditional operating systems. This is due to WSN's specific characteristics, it's numerous constraints, and inaccessible deployment [21].

TinyOS

TinyOS is an open-source, application-specific OS for WSN's. It was designed to tackle the main constraints of WSN's, such as the limited computational resources and the need for low-power operations [18]. It has become the main development platform for WSN's. In TinyOS code executes either asynchronously in response to a hardware event, which are interrupts fired by a timer, sensor, or communication device, or in synchronously scheduled tasks which are stored in a queue. When the queue is empty, the system enters a sleep state until the next interrupt is fired, allowing some energy saving [7, 18].

TinyOS provides an Application Programming Interface (API) in order to conserve and manage power properly with a high level of efficiency. It manages the radio, which is the most power hungry hardware we normally find in a sensor node, as well as the processor [21]. Power management uses, among others the following elements: any service running on TinyOS can be stopped by a call to its `StdControl.stop()` command; `HPLPowerManagement.Enable()` can be called by the application, this allows the processor to go into the lowest power mode under certain conditions (radio is off, clock interrupts are disabled, task queue is empty, and serial peripheral interface interrupt is enabled) [18]. TinyOS's power management API implements cross-layer control at a low level, the application can call the TinyOS component `HPLPowermanagement` which goes directly to the hardware and decide when and if the processor can be switched into it's various low power modes [18].

MantisOS

MantisOS is a multi-threaded OS [6,29]. Its thread-driven model gives flexibility in writing applications, since the developer doesn't have to worry about the task size, which is a must in event-driven operating systems [21]. Scheduling among the threads is achieved by a priority scheduling algorithm with round-robin semantics. Its main disadvantages is the fact that it suffers from context switching overhead [21].

MantisOS achieves energy-efficiency by implementing a power-efficient scheduler. This scheduler makes the processor enter the sleep state after all the application threads have called the MantisOS `sleep()` function [21]. To do so, the application thread must enable power-save mode. This can be done by calling it's `mos_enable_power_mgt()`. All thread must call the latter function as power-save mode is turned off by default. Next, the `mos_thread_sleep(p)` is called, with `p` being the parameter representing the sleep duration [6]. If no threads are ready for execution, the system enters the sleep state automatically. The sleep state can vary depends on the system state, if the system is suspended on I/O (it enters the ATMEL's moderate idle sleep mode), or if all application threads have called the `sleep()`, in this case the system enters a deep power-save sleep mode [6].

SenOS

SenOS is a finite state machine based OS for network sensor nodes [16,29]. It's structured in three main components, A Kernel that constrains a state sequencer and an event queue, a callback library which provides a set of libraries to the developer for hardware access, and a state transition table which defines an application. Using multiple state transitions tables, we can switch among them, supporting multiple applications in a concurrent manner [16, 21,29].

For effiency purposes, SenOS provides the network management protocol as an application layer protocol. Its callback library contains node hardware access function such as `turnOn()`, `turnOff()`, `receive()`, `tellNeighbor()` and location aware functions such as `isNeighbor()` [16]. Combining these functions it can shutdown unnecessary nodes in a cluster.
This approach is not considered a very efficient one, because of the sleep-state transition overhead involved. With this in mind Dynamic Power Management (DPM) algorithm is used. DPM algorithm can generate a sleep/wake state transition policy based on the events that occur in the network [16,21].

Sonar Operating System

In SONAR the nodes have some pre-installed components, one of these components is the operating system. The OS is responsible for processing incoming control messages, managing memory resources for the non-preemptive tasks and for scheduling these tasks. The scheduling is done in an Earliest Deadline First (EDF) fashion.

The OS initiates the node's components, such as, the radio and the micro-controller and it attaches interrupts in order to detect any radio activity or tasks deadline signaled by the real time clock (RTC). Its four main functions are the pre-defined functions `run()`, `schedule()`, `sleep()`, `listen()`. Combining these functions the OS runs the task (in case there is a task to run), schedules the next task, and puts the micro-controller in a sleep state until an interrupt occurs (task activation or incoming message). This process is a loop for as long as the node is active.

2.2 Virtual Machines

MagnetOS

MagnetOS [5] is a distributed Virtual Machine (VM) based OS designed specifically to enable power-aware adaptive, and efficient ad-hoc networking applications. It is a Single System Image (SSI) , the whole system can be viewed as a single unified Java Virtual Machine (JVM) [29]. The system adapts to resource constraints and changes the underlying network in a rapid, yet temporally stable manner. It adapts depending on the changes in the network topology, and application behavior [29]. MagnetOS works by automatically partitioning applications into components and it migrates these components to the best-suited node. To achieve this it takes advantage of some existing object placement and power aware algorithms such as NetPull and NetCenter. MagnetOS policies and adaption mechanisms yield good power utilization, thus increasing the network longevity [19].

Mate

Mate is a tiny communication-centric VM designed to work on top of TinyOS [17, 29]. It is a byte-code interpreter that enables rapid and efficient programming of sensor networks. A Mate program code is composed of capsules. A capsule contains up to 24 instructions and the size of each instruction is 1 byte. Mate programs can deploy themselves into the network. It can program the entire network through hop-by-hop code injection [17].

Mate has a concise representation of programs, which is responsible for energy savings, when compared to TinyOS binary upload. Programs can fit in a few capsules, and it is preferable for a small number of executions, since the CPU overhead increases rapidly with the number of executions. Mate's energy efficiency comes from the fact that it uses little bandwidth for its transmissions [17].

Sonar Virtual Machine

Another pre-installed component in SONAR nodes is the Sonar Virtual Machine (SVM), and it complements with the SONAR OS. The SVM executes the tasks scheduled by the OS. These tasks are written in Sonar Task Language (STL), then translated into byte-code by a compiler. SVM is stack based, with a simple Instruction Set Architecture (ISA). The byte-code is composed by a header, a data, and a text segment. The header contains the size of the data and the text segments, and the maximum run-time stack size. The stack segment exists only at run-time and it is allocated between the data and the text segment. The text segment is composed of instructions, which are interpreted and executed by the virtual machine. A more complete description of the virtual machine (byte-code, translation function, semantics) will be presented in the next chapter.

2.3 Wireless Communication Technologies for WSN

802.15.4

The Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 is a standard developed for low cost, low complexity, low power consumption, and low data rate wireless communication. It provides a low data rate (250Kb/s), but high enough to support a set of application, and it can be scaled down as needed for energy efficiency purposes [3]. The standard compromises speed for low power consumption.

The standard specifies the PHYsical layer (PHY) layer, and the Medium Access Control (MAC) sub-layer specifications for low data rate wireless communication with limited battery consumption requirements [3]. The MAC sub-layer presents an interface between the upper layer and the PHY. It is responsible for all the access to the physical radio channel. It is also responsible for a set of tasks such as generating network beacons, synchronizing to network beacons, supporting Personal Area Network (PAN) association and disassociation, managing the Carrier Sense Multiple Access-Collision Avoidance (CSMA-CA) mechanism, among others [3].
PHY provides an interface between the MAC and the physical radio channel, through Radio Frequency (RF) firmware and hardware. It is responsible for data reception and transmission, toggle the radio transceiver state, energy detection within the current channel, and Clear Channel Assessment (CCA) for CSMA-CA [3]. Thus the standard allows the easy implementation of duty-cycling mechanisms in order to save energy.

ZigBee

ZigBee is a specification for wireless communication in a Wireless Personal Area Network (WPAN). It was developed on top of IEEE 802.15.4 MAC/PHY layers as an open global standard to address the necessity of low-cost, low-power wireless networks, with the goal

of providing mesh capabilities to the 802.15.4 standard. Like IEEE 802.15.4 it operates in unlicensed band such as 2.4GHz, 900MHz, and 868MHz. ZigBee provides multi-hop and advanced mesh routing capabilities to the IEEE 802.15.4 standard [2].
This standard also targets the application domain of low-power, low-duty-cycle, and low-data-rate devices, such as XBee [2]. Beside the IEEE 802.15.4 MAC/PHY, it implements three more layers: (1) network layer, which adds routing capabilities and multi-hop to the RF packets; (2) Application Support Sub-layer (APS), which is the application layer that defines the addressing objects such as clusters, endpoints, and profiles; (3) ZigBee Device Objects (ZDO), the application layer which presents service discovery features and advanced network management capabilities [2].

DigiMesh

DigiMesh is a proprietary protocol developed by Digi International. It was designed to tackle the need for low power consumption in WSN's, in which battery power routers are required. Just like ZigBee, DigiMesh was developed on top of 802.15.4, and operates under the same unlicensed bands, as of those two. It operates under 2.4GHz using Direct-Sequence Spread Spectrum (DSSS) modulation, and under 900MHz using Frequency-Hopping Spread Spectrum (FHSS). It uses a enhanced Ad-Hoc On-demand Distance Vector (AODV) for message routing, and route discovery, which means routing tables are built only when necessary for needed destinations. Channel Access is accomplished by using a time-synchronized Carrier Sense Multiple Access (CSMA) technique.

DigiMesh contains some interesting features, such as, self healing (nodes may enter and/or leave the network without causing its failure), peer-to-peer architecture (no parent-child relationship is established, instead, routes are discovered and created when needed), selective acknowledgement (only the destination node replies to route requests), and sleep mode for all nodes in the network due to the accurate time synchronization in DigiMesh.

6LoWPAN

IPv6 over Low power Wireless Personal Area Networks (6LoWPan) is a networking technology that allows Internet Protocol version 6 (IPv6) packets to be carried efficiently within small link layer frames. It is an adaption layer that stands between IEEE 802.15.4 and the IPv6 stack. Its main functionality is to allow communication between low power WPAN and Internet Protocol (IP) based networks in a simple manner. 6LoWPan provides a adaption layer where the transition from IPv6 to 802.15.4 is performed. This adaption layer performs three main functions: (1)the header compression which compresses the IPv6 payload (which size is greater than that of 802.15.4) by assuming some fields are the same in every communication and by eliminating the fields that can be inferred in the 802.15.4 layer, *e.g.,* payload length, (2) IPv6 has a Maximum Transmission Unit (MTU) of 1280 bytes and 802.15.4 maximum payload length is 127 bytes, so the IPv6 packets are fragmented into several smaller segments in order to pass through 802.15.4 radio links [30]. Depending on the routing protocol being used, the reassemble of these segments can be performed either in the destination node or in each node they hop through, and (3) stateless auto configuration, which allows nodes to choose their own IPv6 address. 6LoWPan provides mechanisms to prevent several nodes to choose the same address [23].

Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless technology developed by Bluetooth Special Interest Group (SIG) for short range communication. It was developed as a low power, single-hop solution for control and monitoring applications [14].

The BLE stack protocol has three main components, the Controller, the Host, and the Host Controller Interface (HCI) which stands between the first two. The controller is composed by the physical layer and the link layer.

The physical layer defines the mode of operation of BLE in 2.4GHz ISM band and defines

40 RF channels. It divides the channels into two types, the advertisement channels and the data channels. The advertisement channels are used for device discovery, connection establishment, and broadcast transmissions. The data channels are used for bidirectional communication between connected devices [14].

The link layer defines two device types for a created connection, the master and the slave, which act as initiator and advertiser during the connection creation, respectively. The advertiser announces that it is a connectable device, while the initiator listen for said advertisements, and chooses to either connect or not to the initiator. A master can manage several simultaneous connections with different slaves, whereas each slave can only connect to one master, thus forming a star topology network. For energy saving purposes, slaves spend most of their time in sleep mode, waking up periodically to listen for incoming packages [14].

The host is composed by the Logical Link Control and Application Control (L2CAP), ATTribute Protocol (ATT), Generic Attribute Profile (GATT), and the Security Manager Protocol (SMP). L2CAP is the lowest layer among them and its main goal is to multiplex the data provided by three upper layers. In BLE the data of these services are handled by L2CAP in a best-effort approach, without the of retransmissions or flow control, which are available in other Bluetooth versions. GATT defines the server and client role in the connections, which are independent of the slave and master roles, and it manages the exchange of information from one device to another, *e.g.,* sensed data. ATT defines the communication between the server and the client. The client can access the server information by sending requests, which triggers responses from the server [14].

2.4 Energy Conservation Schemes - Duty-Cycling

Designing a sustainable WSN is a challenging task, and in some cases WSN's are expected to run autonomously for long periods of time. This is because in some situations it is quite difficult or even impossible to replace exhausted batteries in hostile environments [28].

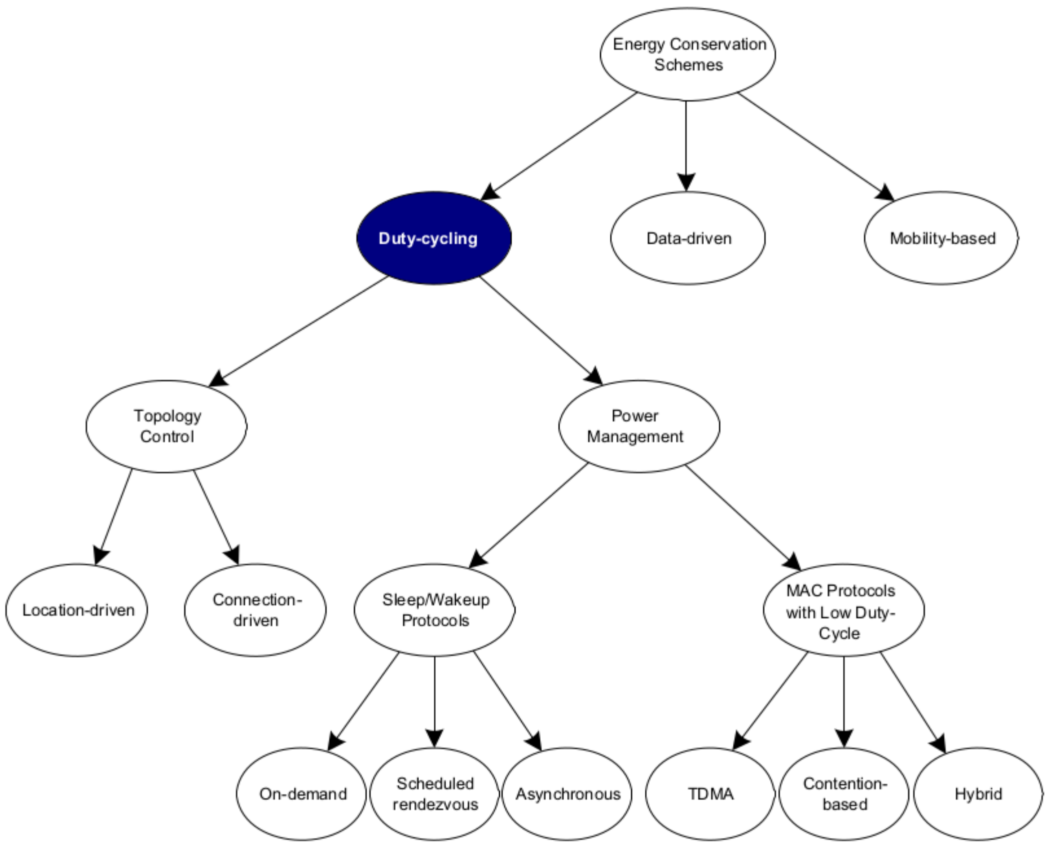


Figure 2.1: Taxonomy of duty cycling schemes (adapted from [4])

Putting the radio module in low power consumption mode (sleep mode) whenever its services are not required is the most effective operation regarding energy conservation, as the radio transceiver is normally the most power hungry hardware in a sensor node. In this approach nodes switch between a sleep/wake state [8]. This is the basis of duty cycling, it can be defined as the amount of time the node is in the awake state.
As presented in Figure 2.1 [4] duty cycling can be accomplished by two methods: (1) topology control, and (2) power management. Those two mechanism are complementary as will be explained in the following subsections.

Topology Control

In some cases, sensor network deployment is done at random, e.g., dropping a certain amount of nodes from an airplane [31]. In these cases it may be practical to deploy a number greater than necessary, in case of possible node failures or other undesired events. But this leads to some degree of network redundancy [31].
The idea behind Topology Control is to exploit network redundancy in dense networks to increase their lifetime. Nodes that are not necessary to ensure connectivity or coverage can be turned down in order to save power. Discovering these optimal subset of nodes that ensure connectivity and coverage is referred to as Topology Control. Thus, Topology Control aims to dynamically adapt the network topology, prolonging its longevity [8]. The taxonomy of Topology Control is quite simple, as it divides in two methods, *Location Driven* and *Connectivity Driven* [4].

Location Driven approaches define which node should be turned on based on the node location, assuming the location is previously known. Geographical Adaptive Fidelity (GAF) is a location driven protocol. GAF [32] reduces the network energy consumption while keeping a constant level of routing accuracy. GAF divides the sensing area into small virtual grids, where nodes from a grid have one-hop connection with all nodes from any adjacent grid. Nodes within the same grid are equivalent for routing, therefore only one needs to be up and running. In this way, nodes must coordinate with each other to decide which should go to sleep and for how long.

Connectivity Driven approaches dynamically activate or deactivate nodes in order to ensure network connectivity or sensing area coverage. Span [10] is a connectivity driven protocol that dynamically chooses ”coordinators” from the nodes in the network, once again we are assuming the network is dense and presents a certain degree of redundancy. These coordinators are basically routing nodes. Coordinators stay wake continuously in order to perform multi-hop packet routing. Span is implemented in a way that it ensures that enough coordinators are elected so every node can be in the radio range, it rotates the coordinator to guarantee that all the nodes participate in the task of providing global connectivity, and it attempts to minimize the set of coordinators, since they are the nodes with higher energy cost. Other protocols such as Adaptive Self-Configuring sEnsor Networks Topologies (ASCENT) [9] and Naps [13] also implement the connectivity driven approach in a similar manner.

Power Management

Duty-cycling mechanisms implemented in active nodes are refereed to as power management, therefore, topology control and power management complement each other in the sense that both are duty cycling techniques with different granularity [8]. Power Management has a more complex taxonomy than Topology Control, as presented in Figure 2.1.

Sleep/Wake Schemes

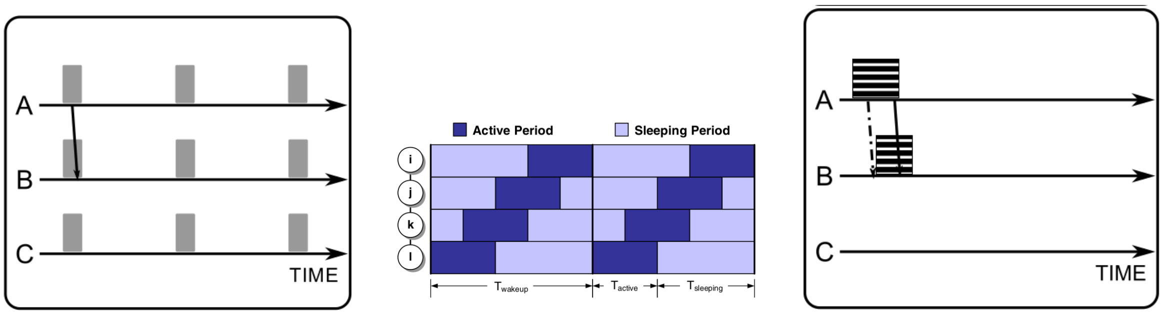
On-Demand (Figure 2.2c) schemes are quite straightforward. The approach is based on the concept that a node can be switched from the sleep to wake state whenever necessary, normally when it has to receive data from a neighboring node [4]. Although it is possible to use two channels in a single radio module, proposals for on-demand schemes rely on two radio modules, the *wakeup radio* and the *data radio*. Ideally the wakeup radio is a lower power module that would listen to a wakeup signal and send an interruption signal to the Central Processing Unit (CPU) which would activate the primary radio (data radio). It is trivial to conclude that an extra piece of hardware will add an extra energy cost, but most on-demand proposals claim that the advantage of having a second radio is the fact that ongoing transmissions will not affect the wake up signaling, thus justifying that the extra energy cost (which is not as high as the data radio energy cost) is tolerable. Sparse Topology and Energy Management (STEM) is one of the most influential on-demand mechanism, and it actually uses two similar radios under the pretext that the use of an ultra-low radio interface would result in different transmissions range and a mismatch between the wakeup signal and transmitted data signal. Duty-cycling mechanisms are applied in both radios. Some good results were obtained by combining STEM with GAF, significantly reducing the network energy consumption thus prolonging the network lifetime [8].

Scheduled rendezvous (Figure 2.2a) presents itself with two approaches, the *fully synchronized pattern* and the *staggered rendezvous*. Those methods have the similarity that each node has a pre-defined wake up time, thus the scheduled term in the name of the mechanism. Typically, nodes wake up periodically, listen for potential communication and transmit right away, depending on the application, go back to sleep and wake up in the next rendezvous point. They also require the node's clock to be synchronized, in order for them to stick to the schedule, so clock synchronizing algorithms might be necessary. The difference between the scheduled rendezvous protocols is the timing when nodes sleep and wake up.

The fully synchronized pattern is considered the simpler one, since nodes wake up all at the same time, and there is no worrying about the neighbors. TinyDB implements the fully synchronized scheme due to its simplicity and its effects on the power consumption. Since nodes wake up at the same time, there is a high probability of collisions to occur, which is the main draw back of this protocol [8].

The Staggered Wakeup pattern 2.2b, is a little bit more interesting, and complex, yet it presents a great level of simplicity and energy conservation. In this approach the nodes have different wake up time, but this time must overlap. The children nodes wake up, and it's parent wakes up a bit later, just in time to receive the children's packet. Now it is time to the current parent to become the children and send the packet to its parent, which follows the same process as the node before it. This method minimizes energy dissipation from sleep to active mode. As we can predict this method also requires the clock synchronization.
If global time synchronization is provided, *e.g.,* an external hardware, rendezvous schemes

become more advantageous, because they can coordinate transmissions, reduce idle listening and collisions [4].



(a) Scheduled Rendezvous (b) Staggered Wake up (c) On Demand

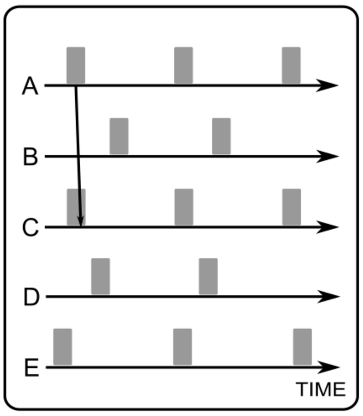
Figure 2.2: Sleep/Wake up Schemes

Asynchronous Schemes

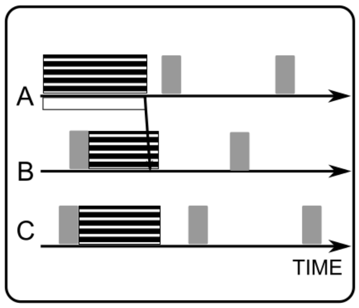
Synchronizing nodes in a multi-hop wireless network can be a costly task, involving extra hardware to achieve the desired result [8]. Asynchronous schemes allow nodes to have independent wake up schedules. The only concern is to guarantee that neighbors nodes up time always overlap in a specific number of cycles. Asynchronous methods are usually easier to implement, and can guarantee connectivity in scenarios where synchronous schemes can not. This flexibility comes with a drawback, lower power efficiency, since in asynchronous schemes nodes have to wake up more often than in most synchronous schemes [20].

Preamble Sampling, Figure 2.3b, is an asynchronous technique that reduces idle listening by switching the energy waste to the sender which is always one, instead of the receiver which can be many. Nodes go to sleep asynchronously and then wake up periodically to listen for channel activity. The preamble is the time the sender is waiting for the receiver to wake up, this time is longer than the sleep and active time put together, so other nodes have enough time to wake up, detect the preamble and receive the intended data. Drawbacks of this method are large end-to-end latency and the appropriation of the channel by the preamble, which can be wasteful and also prevent other nodes to transmit since the channel is already in use [8].

Random duty-cycling, Figure 2.3a is another asynchronous technique, maybe the most simplistic of them all. The basis is that in sufficiently dense networks, nodes randomly go to sleep and wake up, since the probability that there will be enough active nodes any time is very high [8].



(a) Ramdom



(b) Preamble

Figure 2.3: Asynchronous wake up Schemes

MAC protocols with low duty-cycle

Several MAC protocols have been proposed for wireless networks. These protocols are divided essentially in three categories:(1) Time Division Multiple Access (TDMA)-based,(2) contention-based, and(3) hybrid protocols [20].

TDMA-based schemes enable duty-cycling since the channel access is done via a slot-by-slot mechanism. In these schemes time is divided by frames, which contain time slots and each node is assigned one or more slot per frame. Since nodes must turn on their radio only on the allocated slot, the energy consumption is reduced to the minimum required to the transmission and reception of data, ideally [4]. TRAMA [25], FLAMA [26], and LEACH [15] are example of protocols that implement this mechanism, the later also takes advantage of clustering mechanisms.

Contention-based are the most popular MAC protocols proposed for WSN's. These protocols achieve duty-cycling by integrating channel access functions with sleep/wake schemes, such as those described earlier in this thesis, the difference being that these schemes are now coupled with MAC protocol and not independent of it. B-MAC [24] is one of the most famous contention-based protocols, and it is used in TinyOS operating system.

Hybrid protocols try to take advantage of both schemes described above, they adapt the contention-based scheme when the contention level is low, and TDMA-based, when the contention level is high.

2.5 Summary

In this chapter we presented a brief overview of the state of the art in the WSN area concerning this thesis, specifically the Operating Systems and Virtual Machines, specifying their power saving protocols. We also presented an overview of the communication technologies currently currently in use on WSN. Later we analyzed and presented the energy saving mechanisms. These study of these subjects were carried in order to find out each one of the mechanism could be applicable in the SONAR network.
In the next chapter we will present an overview of SONAR network, its components, implementation, and the data-flow in the network.

Chapter 3

SONAR

In this chapter, we will focus on the SONAR platform. We will present a high level overview of SONAR, and then specify the architecture of each one of its layers. We will also present and describe the data flow between layers and internally in the Data layer. We will finish this chapter by describing the implementation of SONAR layers and their components.

3.1 Architecture

SONAR follows a 3-layer architecture. Figure 3.1 presents a high level view of system's architecture. It is based on a Publish/Subscribe (P/S) architecture where a set of clients connected to the internet, access the data generated by the Data Layer at each of SONAR's deployment through the Broker.

A client must connect to SONAR Broker in order to have access to the previously registered deployments. Once the client has access to the deployments, it can consult their context information and the data streams they produce. The client can then subscribe to the desired data stream and receive the respective data.

A simple example application for the SONAR system is the monitoring of a greenhouse.

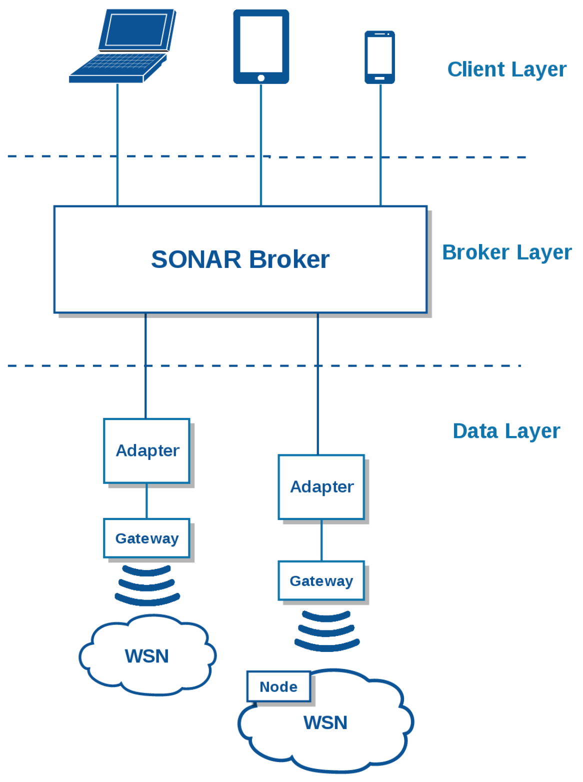


Figure 3.1: The SONAR architecture.

A set of nodes are placed in strategic positions, they measure the desired environmental variables, such as, humidity, temperature, luminosity, and forward these sensed data to the client. The nodes can also actuate in the environment in case some conditions are met, e.g., irrigate a plant in case the soil is too dry, or open ventilation window.

The Data-layer abstracts all the sensors and actuator in each WSN deployment, and it is composed by three parts:

- adapter;
- gateway;
- nodes,

each one with specific functions. The data-layer is responsible for producing and transmitting the requested data stream to the upper layer.

The Broker-layer is the middle layer. It maintains the connection between the the Data-layer and the Client-layer in order for these two to communicate. It implements a P/S service for this purpose.

The Client-layer is composed by a P/S client used to connect to a SONAR Broker, and an administration client used to manage the deployments and allows users to access a deployment.

3.1.1 Data Layer

The Data layer is the lower layer of the SONAR architecture. It abstracts all the sensors and actuators present in each deployment, facilitating the use of SONAR. Its components are the SONAR adapter, a gateway, and a set of nodes that maintain connection with the gateway via RF links.

Just like SONAR itself, the Data layer can be seen as 3-layer structure. The adapter being the upper layer, the gateway being the middle layer, and the set of SONAR nodes can be viewed as the lower layer.

Data Layer Components

The three components of SONAR Data layer have distinct functionalities.

1. The adapter functions are to establish the connection between the Administration client from the Client Layer, and the deployment gateway, and establish the connection between the gateway and the Broker. These connections are both unidirectional. It allows the administration user to manage its respective deployment, working as a bridge between the administrator and gateway. It does so by forwarding the control messages received from the administrator to the gateway, and by forwarding the data

received from the gateway to the Broker.

2. The gateway is responsible for forwarding commands to the nodes, and to gather the data produced by the nodes and pass it to the adapter. It is itself a typical node, excluding the sensors and the actuators hardware. Just like the nodes it is equipped with a radio interface. The communication between the gateway and the adapter is done via serial communication ports, and the communication between the gateway and the nodes is done via RF links.

3. The nodes differentiate themselves from the gateway because they are equipped with extra hardware: sensors, actuators, and a ChronodotRTC. They also have two pre-installed modules, the SONAR OS, and the SONAR VM.

3.1.2 Broker Layer

The Broker layer was implemented using a simple Java Web Service with two server endpoints, one reserved for the client connections and the other for the connections with SONAR adapters. It has three main functionalities:

1. registering WSN deployments;
2. publishing data provided by the deployments;
3. handling subscriptions requests from the Client layer.

In order for Broker layer to carry out these functionalities, three main structures are used, each one with specific purpose:

- a task structure, which stores the parameters of the data streams registered in the Broker. Each task representation is composed by five parameters:(1) an ID which identifies the task,(2) a period which represents the periodicity of the task execution,(3) a value parameter that presents the data produced by the task;(4) the unity used to represent each value in the values field, and (5) the task info, which is an optional field used to store any extra and relevant information regarding the task.

- a deployment table, which keeps information about all the registered deployments. Each deployment is accompanied by a pair of parameters: (1) a general information field, which contains general information about said deployment, and (2) a list containing the ID of each task running in the deployment.

- a subscribers table, which maps each client with the data streams they subscribed to.

3.1.3 Client Layer

As stated above, the Client layer has two main components, the P/S Client and the Administration Client.

The P/S Client allows the user to communicate with the Broker layer.The user can list and/or subscribe/unsubscribe a given data stream. It also allows the user to query the Broker for the specific type of data being produced in the WSN Deployment. The data available to the stdout of the client can be piped to processing scripts or stored in local databases. The communication between the P/S Client and the Broker is described in the section above, in the Figures 3.4, and 3.5 respectively. Table 3.1 describes the commands available for the P/S client.

The Administration Client allows the user to manage its deployments, specifically to register or unregister deployments in the Broker, as well as managing the deployment's tasks. Table 3.2 presents the commands available for the Administration Client along with their respective descriptions.

Command	**Description**
`sonar -l`	List the available Deployments and Tasks
`sonar -s -t ID_TASK -d DEP`	Subscribe Task ID_TASK running in Deployment DEP
`sonar -u -t ID_TASK -d DEP`	Unsubscribe Task DEP running in Deployment ID_TASK
`sonar -f -t DATATYPE`	Query the Broker for tasks producing data with type DATATYPE
`sonar help`	Show a list containing the available commands

Table 3.1: Commands available in the P/S Client interface.

Command	**Description**
`reg -g MAC`	Register the Deployment in the Broker.
`unreg -g MAC`	Unregister the Deployment in the Broker.
`list`	List the running Tasks
`task -p PER -b BC_PATH -d DESC_PATH`	Add a new task with period PER, byte-code file BC_PATH, and description file DESC_PATH;
`period -t ID_TASK -p PER`	Change the period of task ID_TASK to PER.
`kill -t ID_TASK`	Remove task ID_TASK from the runing tasks pool.
`reset MAC`	Remove the tasks from the running tasks pool.

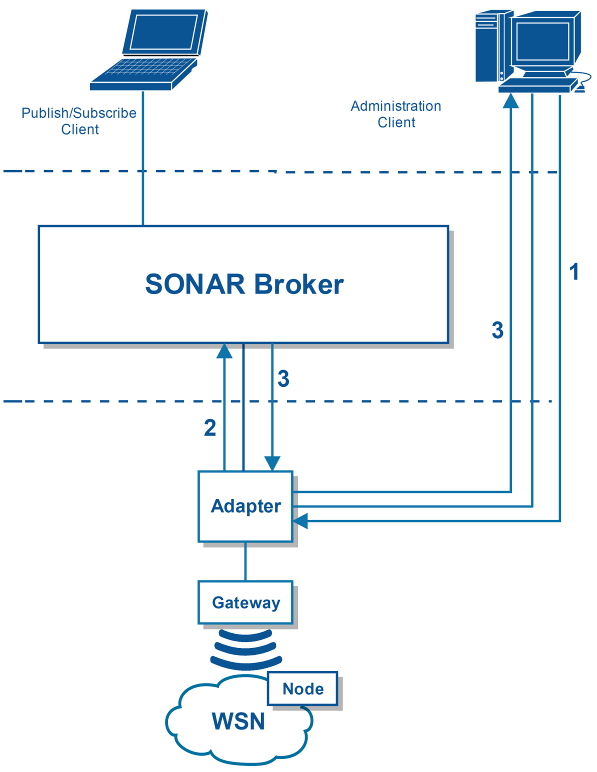
Table 3.2: Commands available in the Administration Client interface.

3.1.4 Data Flow

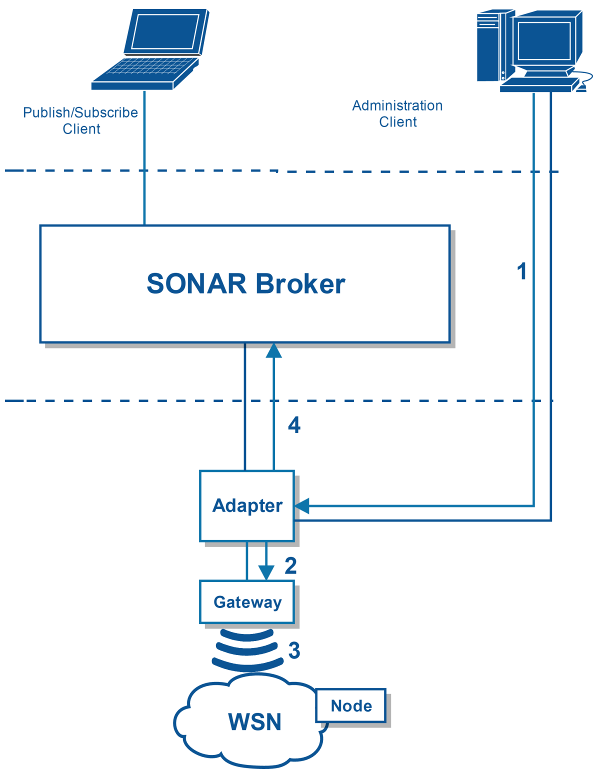
To register a deployment, the Administration Client sends a command directly to the adapter, which sends a register request to the Broker containing the gateway's MAC address. After processing the request, the Broker responds with a confirmation that the deployment has been registered. The Administration Client can manage its deployment by issuing the commands in table 3.2. These commands are sent to the adapter, and then forwarded to the gateway, which sends it to all the nodes in the deployment. Figure 3.2a and 3.2b illustrate the register command data flow, and the management commands data flow, respectively. When issuing the `register` command, the Administration Client sends control messages to the adapter, which issues a register request to the Broker. This request contains the MAC address of the gateway. After receiving the request, the Broker processes it and responds with a confirmation that the register process was successful. To manage the tasks running in the network, the Administration Client sends the desired command to the adapter, which submits the command to the deployment and informs the Broker about the new command that was issued.

Figures 3.3a and 3.3b present the data flow inside the data-layer, task submission and data collection, respectively. After the task submission by the Administration Client, the information about the task is passed to the adapter, the adapter then sends this information to the gateway via serial communication, the adapter radios it to the WSN deployment. The generated data is sent from the WSN deployment to the gateway via RF links, the gateway passes the data streams to the adapter, which sends the data streams to the broker.

Data streams produced in the Data layer are made available to the client through the Broker layer, which is responsible for receiving the data from the Data-layer and forward it to the Client layer, thus the Broker exchange data both with the client and the data layer. Regarding the client, it has to receive a command to list, subscribe, or unsubscribe a data stream and manage it appropriately. Regarding the Data layer, it has to receive



(a) Register command data flow

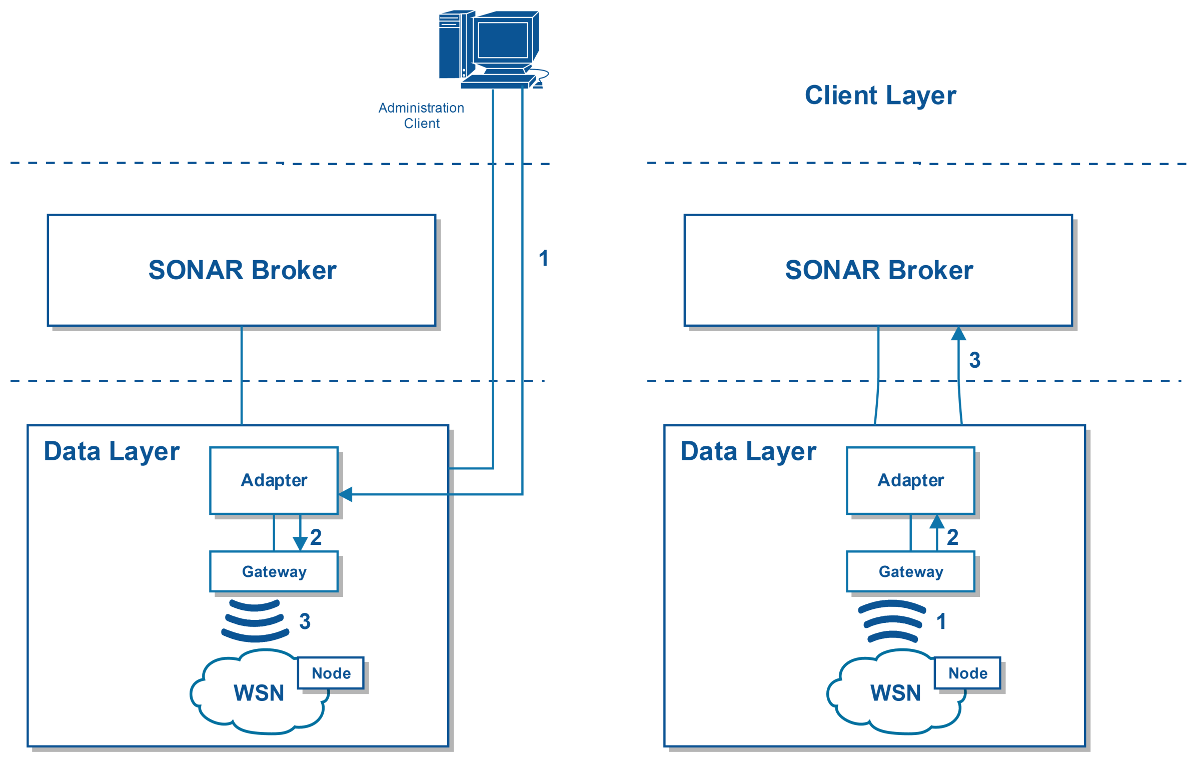


(b) Tasks management data flow

Figure 3.2: Communication between Adapter and administration client

the data streams and forward it to the respective subscribers.

Figure 3.4 describes the data flow between the client and the broker. The client asks the Broker to list all available data streams, upon receiving the request, the broker queries its deployments table and creates a message with the available deployment and their respective data streams, the Broker then responds with the constructed message. After receiving the list, the client creates a message containing the IDs of the desired data streams and sends it to the Broker, the Broker then receives the message and adds the client ID in the respective entry to the subscribers table and responds with a confirmation of the subscription. The client sends another message to unsubscribe one of its subscription and the Broker deletes the said client from the respective entry. Finnaly, the Broker responds confirming that the subscription is canceled.



(a) Task submission data flow

(b) Data collection data flow

Figure 3.3: Data Flow in Data Layer

After subscribing a task, the client will periodically receive the data streams that have been subscribed, in a format that is specified by the Broker at the moment of subscription.

The data exchange between the Data layer and the Broker is quite simple, as we can observe in Figure 3.5. When the data stream becomes available, the Adapter constructs a message containing the deployment identifier, the data stream identifier, the respective data stream, and sends the respective message to the Broker, which receives the message, retrieves the the deployment identifier, and the task identifier. Using the retrieved information, the Broker queries its deployments table to discover its subscribers and forward the received data stream to its respective subscribers.

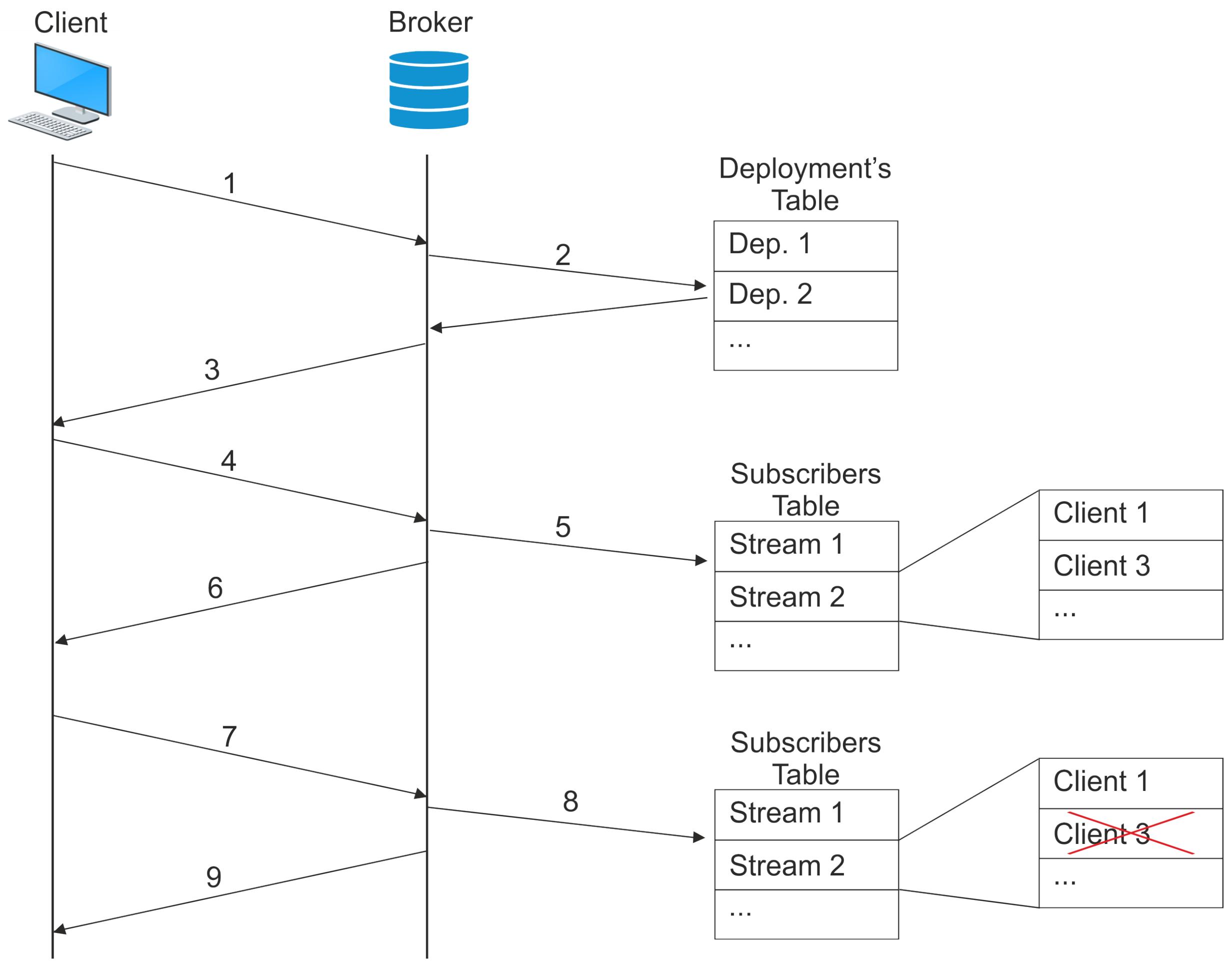


Figure 3.4: Message exchange between client and Broker

3.2 Implementation

The different layers of SONAR required distinct technologies and methods for their implementation.

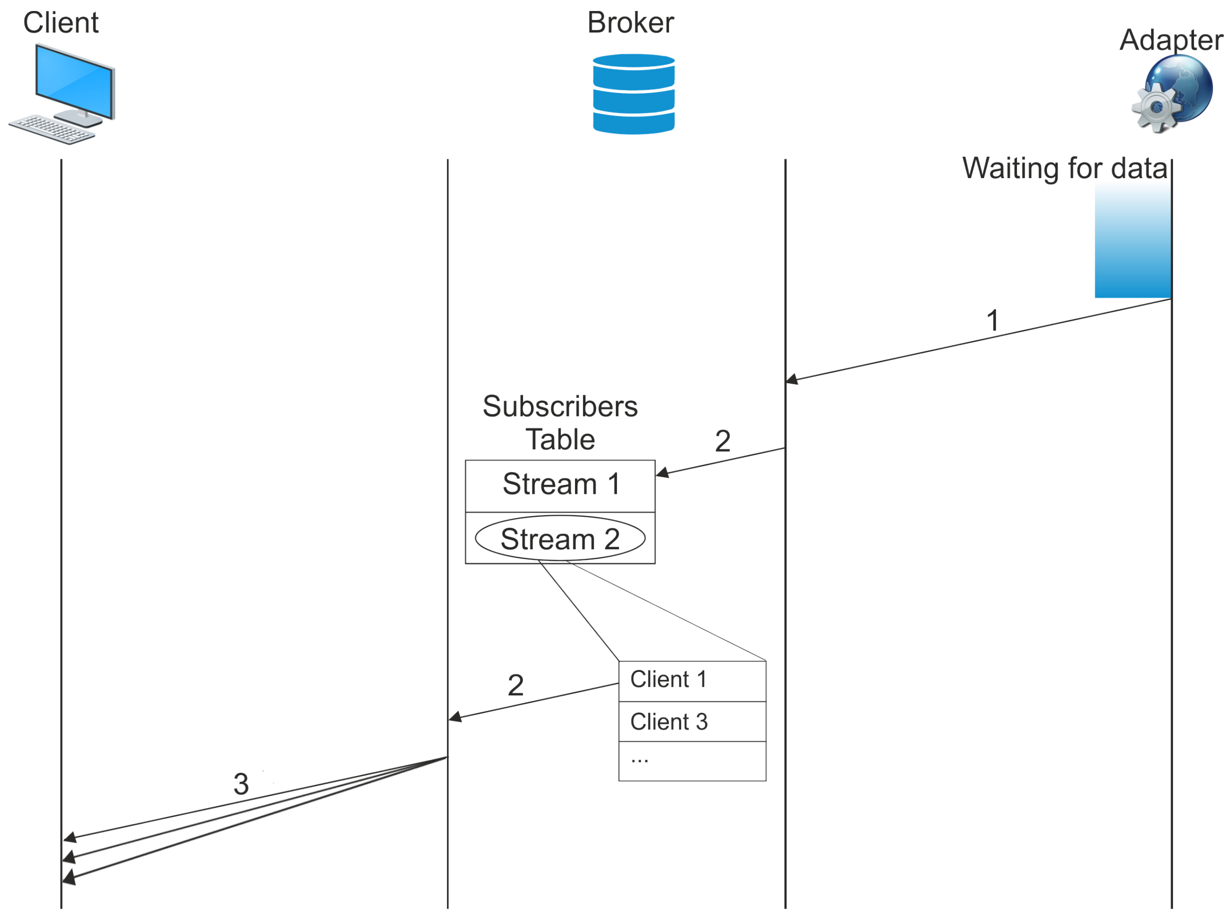


Figure 3.5: Message exchange among the Adapter, the Broker, and the client

3.2.1 Data Layer

Adapter

The adapter is the component responsible for establishing the connection between the gateway and the Administration Client, and it perform two roles in the system. It allows users to administer the SONAR WSN deployment, and it forwards messages from the Administration Client to the gateway, and from the gateway to the SONAR Broker.

The adapter and all of its components were implemented in JAVA. The connections to the gateway is done via serial communication, and the communication to the Broker is done using Web Sockets.

Gateway

Algorithm 1 depicts the gateway functionalities in a simple manner. The interrupts are attached, in order to detect the source of the incoming message. The source can be either the radio for data streams produced in the WSN deployment, or the serial port for commands sent the Administration Client. Detecting this message and its source, it is captured and transmited to the other end. Methods were implemented to process some of these messages in the gateway, e.g., the fragmentation of large packets in order to transmit them via radio, but were omitted for simplicity purposes, since they are not relevant for this topic. The algorithm was implemented C/C++/Wiring programming language which is the native language for Arduino. The communication with the WSN deployment is done through radio links, and the communication with the adapter is done through serial communication.

The gateway is equipped with a Arduino Mega2560, which is connected to an Arduino Wireless Protoshield, a Adafruit RTC Chronodot, and a XBee radio module.

Nodes

Each node in SONAR is composed by a receiving and scheduling loop, responsible for receiving control messages sent by the adapter, and the scheduling and execution of the tasks, the SVM, responsible to run by received bytecode, and a set of hardware libraries used to control and access the sensor and actuators available on the system. Both the SONAR OS and the SVM, which are the main components of the nodes, were implemented in C/C++/Wiring.

Algorithm 2 depicts the node's main loop. A complete description of the SVM can be consulted in [11]. As we can observe in the figure, two interrupts are attach in order to detect the relevant events, such as incoming packet arrival or RTC alarm for task execution, then the node enters a loop where it executes a given task, in case there is one to execute.

Algorithm 1 The gateway program

```
function MAIN()
    ATTACH(RADIO_RCV, HANDLERADIOMSG)
    ATTACH(SERIAL_RCV, HANDLESERIALMSG)
    loop
        MICROSLEEP()
        switch ( src )
            case RADIO:
                $msg \leftarrow$ READRADIORCVBUFFER()
                FORWARDTOADAPTER($msg$)
            case SERIAL:
                $msg \leftarrow$ READSERIALRCVBUFFER()
                FORWARDTONODES($msg$)
        end switch
    end loop
end function

function HANDLERADIOMSG()
    src ← RADIO
end function

function HANDLESERIALMSG()
    src ← SERIAL
end function
```

This happens by calling the function `RUN()`, which calls the SVM with the task identifier as parameter. After the `RUN()` method, a schedule is made for the next task to be executed. The scheduling process is done using a EDF-method method. After this the node attempts to enter a sleep mode in case there is nothing pending, otherwise it wakes up on a RTC generated interrupt to execute the next task. When detecting a radio interrupt it calls the `LISTEN()` method, which is the method that handles incoming radio messages.

SONAR are equipped with a Arduino Mega2560, which is connected to an Arduino Wireless Protoshield, a Adafruit RTC Chronodot, and a XBee radio module, so for those are the same components as the gateway. The nodes also possess other components, a SHT-15 temperature and humidity sensor, a Light-Dependent Resistor, an a red LED, which can serve as an actuator, or for feedback and debugging purposes.

Algorithm 2 The node main loop

```
function MAIN()
    ATTACH(RADIO_RCV, HANDLERADIOMSG)
    ATTACH(RTC_ALARM, HANDLERTCALARM)
    loop
        RUN()
        SCHEDULE()
        SLEEP()
        LISTEN()
    end loop
end function
```

3.2.2 Broker Layer

It was already stated that the Broker layer was implemented using a simple Java Web Service with two server endpoints, one reserved for the client connections and the other for the connections with SONAR adapters.

In an early stage the communication model between the client and the server was implemented using the Server Sent Events (SSE) paradigm. SSE are mechanism that allows the server to push notifications through Hypertext Transfer Protocol (HTTP) connections to a set of clients. The basic idea is that a client subscribes to a set o data streams, which would be replied to, when the data becomes available. This idea was later abandoned because using a SSE P/S scheme, each subscription would be a pending request from the client to the server, and therefore increasing the number of subscription would result in performance decrease. With this in mind, another mechanism was considered: Web Sockets.

Web Sockets are part of Hypertext Markup Language, version 5 (HMTL5) specification. It is a protocol for two way communication between a server and a remote host client through a full-duplex channel. In order for the two parts to establish the connection and communicate with each each other, a handshake must be performed in advance. The server maintains the connection open so it can transmit the data to the client and vice versa until the client requests to close the connection. Web Sockets were the chosen mechanism to perform the communication between a client and the SONAR Data layer, given the fact that it allows the implementation of a two-way communication channel, and this single channel can be used to send data from different subscriptions made a client.

3.2.3 Client Layer

The two components of the Client layer P/S Client and the Administration Client were both implemented as Java Web Services, using Web Sockets Clients to communicate with the Broker and the Adapter, respectively. The usage of a Web Sockets allows the access to the Broker and to the Adapter anywhere using a common terminal with Internet access. Both clients are implemented as a shell interface. The available commands can be consulted in Tables 3.2 and 3.1, for the Administration Client and the P/S Client respectively.

3.3 Summary

In this section we presented an overview about SONAR's architecture, explaining its three layer architecture, and taking a little detail into each layer. We described the functionalities of each layer,their respective components, and their implementation. We described the communication process among the layers, and internally within the Data Layer. We also explained the flow of a STL program, beginning with its writing until its execution by the SVM. To do so, we explained the basics of the STL, the bytecode, and the implementation of the algorithms running in the gateway, and in the nodes.

Next chapter will focus on the implementation of energy conservation techniques in SONAR.

Chapter 4

Energy Conservation in SONAR

In this chapter we describe the implementation of energy conservation schemes in SONAR. We start by analyzing the energy consumption of each SONAR's hardware component, next we present the small but necessary changes made in the SONAR OS and in the SONAR hardware in order to be able to apply some of the energy conservation schemes presented in Chapter 2. We also present the functionalities that we introduced or modified in order to make the implementation of these schemes possible. Furthermore we explore two different approaches, using the ZigBee communication protocol, and the DigiMesh communication protocol.

Our main goal is to reduce the network energy consumption while keeping the mesh topology functional. In order to achieve this goal, some mechanisms presented in Chapter 2 were tested, but not all of the presented mechanisms are applicable to SONAR. Regarding operating systems, SONAR presents its own, in which we will be working to achieve our goal. The same applies to virtual machines. Regarding wireless communication technologies, we are limited to work with either ZigBee or DigiMesh. Both implement mesh functionalities on top of 802.15.4. BLE does not provide mesh support, and 6LoWPan would bring excessive overhead and energy consumption because of its adaption layer. On the tested energy conservation schemes we discarded topology control, as SONAR does not possess any mobile node and the network is not dense. We use the same argument (SONAR networks are not dense) to explain why we also discarded some power management schemes such as random duty-cycling. Our tests will focus mainly in scheduled rendezvous and a variation of preamble sampling where we synchronize the clock beforehand and use an asynchronous method to overlap the preamble time.

4.1 Energy Consumption in SONAR

Table 4.1 presents the current consumption for each component in a SONAR node. This will help understand why we choose to focus on the radio, to try and lower the energy consumption.

Hardware	Current Supply		
	OnSleep	Base	Max
XBee	3.5 µA	15 mA	45 mA
RTC Chronodot		110 µA	300 µA
Arduino Board		50 mA	
SHT15		28 µA	550 µA

Table 4.1: Current Consumption for each Component

The table shows that the radio module needs a current supply almost as much as the Arduino board itself. We need the board active the whole time, but the same does not apply to the radio module. We only need the radio module active when there is available data to either receive or transmit. The sections to come, focus on optimizing the radio utilization, specifically to try and activate the radio only when data is available to receive or transmit. From the table we can observe that while on sleep the radio module has a current supply which is almost negligible (3.5 µA). We can also observe that, excluding the Arduino board, the other hardware components have a negligible current supply, thus we did not try to apply any mechanism to lower their current supply. Note that the presented value for the Arduino board is for its integrated circuit, and each I/O pin can draw up to 20 mA.

4.2 SONAR Commands

In order to achieve our goal some new commands had to be implemented and introduced into the SONAR platform and made available to the Administration Client:

- `status`: the `status` command can be used by the administrator to retrieve information about the tasks running in each node. This command will return a list with the ID of each task running in the node, the period and the remaining time until the execution of each of these tasks. The difference between this `status` command and the `list` command (presented in Table 3.2), is that the first one, queries the nodes directly, instead of retrieving the information stored in the Broker. Thus, this command allows us to verify if a task submitted to the deployment is in fact running in the nodes.

- `rtc`: This command was implemented in order to verify the RTC Chronodot drift between the nodes. It is not relevant that the clocks have the exactly same time, but it is important that once we start our network, the clocks advance all at the same rate. The first implementation of the `rtc` command did not have any arguments, and by sending this command all the nodes would periodically send their exact time, the period was statically defined in the code, but it was improved so the period and the time limit it would be running could be passed dynamically as arguments. The final version of the command looks like this:

```
rtc -p RTC_PER -l LIMIT
```

Where RTC_PER is the period in which the adapter will query the nodes for their time and LIMIT the amount of time before the adapter stops querying the nodes.

This command can be used with the command line argument `-s` to set the time in the RTC. In this form, it is not available after the WSN deployment, rather it is used beforehand. Its use is different from the other commands. The node must be connected to the adapter via a serial port and upon issuing the command the node's time is set to the same as the adapter. The reason why it is implemented this way is to eliminate the problems associated with WSN clock synchronization, such as different latency for different nodes.

- `task`: the task command already existed in the previous version of SONAR (presented in Table 3.2), so we did not implemented it from scratch, but a little change was made. The command is now followed by the period of the task and a communication period, the later being a new parameter. So the command that used to be:

  ```
  task -p PER -b BC_PATH -d DESC_PATH
  ```

  was changed to:

  ```
  task -p PER -c COM_PER -b BC_PATH -d DESC_PATH
  ```

  where `COM_PER` is the new communication period parameter. Communication period is the time interval between two communication windows.

Table 4.2 summarizes the new commands available to the Administration Client.

Command	**Description**
`status`	Queries the nodes about their running tasks.
`rtc -p RTC_PER -l LIMIT`	Periodically retrieves the rtc Chronodot current time.
`rtc -s`	sets the nodes time to the current gateway time
`task -p PER -c COM_PER -b BC_PATH -d DESC_PATH`	Add a new task with period PER, communication period COM_PER, byte-code file BC_PATH, and description file DESC_PATH;

Table 4.2: New Commands available in the Administration Client interface.

4.3 Duty-Cycling with SONAR

The network structure is not limited to one design. While designing the network, the developer has few options for topologies, based on the application requirements and the available resources. On a first approach, we attempted to maintain the communication model where the task is executed, and the generated data is instantly transmitted to the gateway, thus preserving the real time model already implemented, and trying to reduce the energy consumption in the process. To do so, we took advantage of the asynchronous sleep provided by the ZigBee protocol.

ZigBee allows users to control the state of the radio by setting its SM (Sleep Mode) parameter value to 0 (router), 1 (pin hibernate), 4 (cyclic sleep), or 5 (cyclic sleep with pin wake). Configuring the parameter to 1, 4, or 5 will force the node to act as an end device. The values 2 and 3 are reserved by the radio manufacturers and the parameter can not be configured to these values. The cyclic sleep values 4 and 5, cause the node to go to sleep in a predefined time, and once its configured it will remain the same through the whole time the network is active. The difference is that with the value 5 the node can be awoken up by sending a signal to XBee. The value 1 allows us to dynamically send the XBee to sleep and wake it up in the same way, so we chose to work with this sleep mode. Setting the parameter value to 1, allows us to control the state of the radio by asserting or de-asserting its SLEEP_RQ pin (XBee pin 9). We connected the SLEEP_RQ pin to the analog pin 4 (A4) on the Arduino Mega board (the A4 pin was configured to act as a digital pin), using a 10k Ω (Ohm) resistor. To put the XBee to sleep, A4 is configured as an output pin and set high (5 V). To wake the XBee, Arduino's pin A4 is also configured as output and set low (0V). Figure 4.1 depicts the connection we made in order to send the desired signal to the XBee sleep pin.

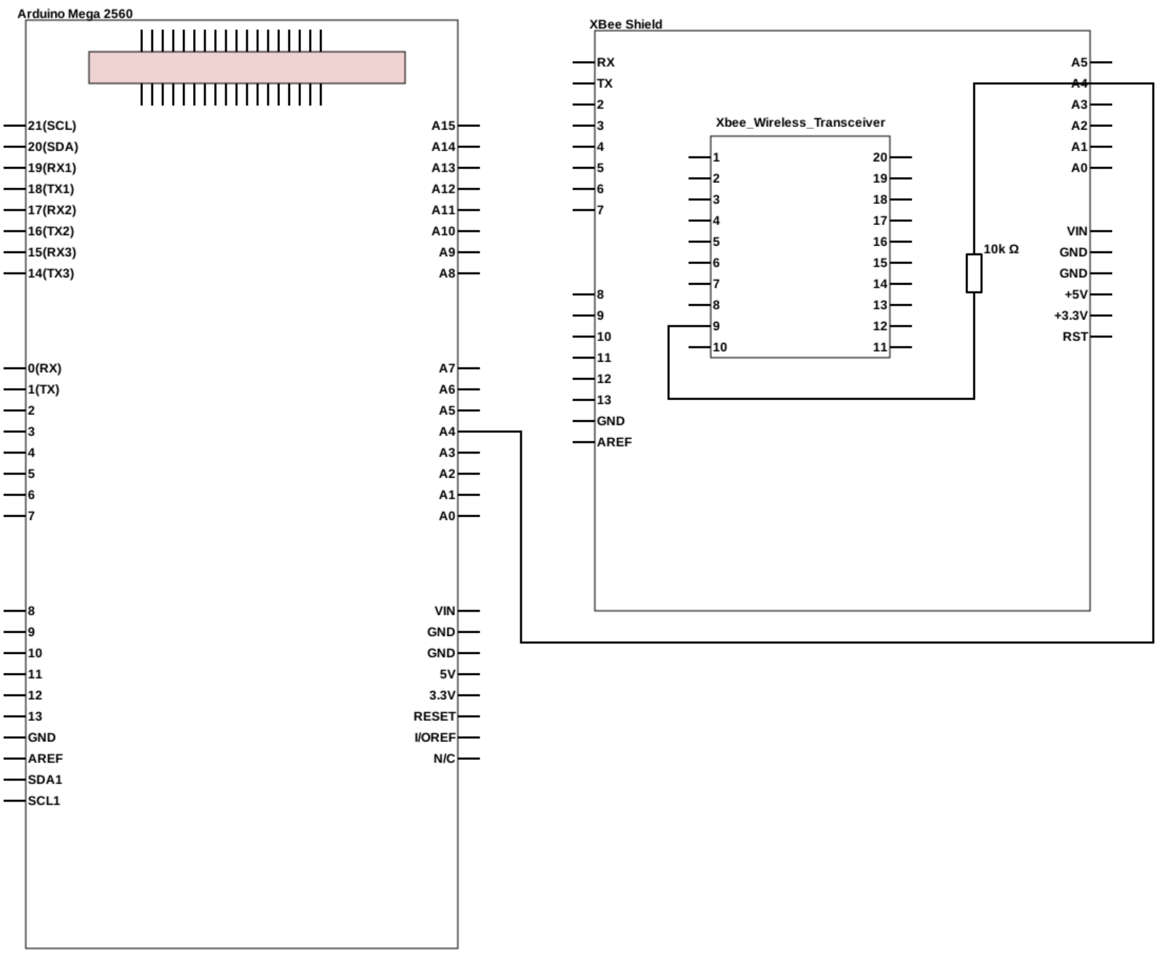


Figure 4.1: XBee Pin9 Connection to Arduino Mega

The asynchronous sleep methodology provided by ZigBee does not allow a node to sleep and be able to perform router functions. In consequence of this limitation, we first considered a network which uses a gateway and a set of nodes directly connected to it.

4.3.1 Star Topology (Case Study I)

The XBee radio modules with support for ZigBee do not support sleeping routers and our goal being to have the whole network in a low power state in order to conserve the most amount of energy possible, we started by testing a star topology network. Thus, all routers where eliminated from the network, ending up with the gateway and a set of nodes directly connected to it via radio links as depicted in Figure 4.2. These nodes, acting as end devices could now be put to sleep, since their absence do not affect the functioning of the network in any negative way, in particular in packet routing, since there is none.

By configuring the XBee SM parameter value to 1, suppressing the routing capabilities, and connecting the XBee SLEEP_RQ pin to the Arduino Mega Board A4 pin, we were in conditions to write the code necessary to send the signals and control the radio state.

In [11, 12] the authors take advantage of the RTC Chronodot to program an alarm and attach an interrupt in order to trigger the micro-controller to execute a scheduled task. Taking advantage of the same piece of hardware, we programmed another alarm with the goal to control the radio module sleep/wake state. This alarm is triggered whenever a certain established deadline is met. At first, since a star topology wasn't our final goal, this alarm was programmed statically within the code, instead of passing this deadline as a parameter. Whenever the alarm occurs, the radio module awakes, and prepares to receive any incoming message and respond to it if necessary. The previously implemented alarm to execute tasks was also convenient, since we took advantage of it to wake the radio for transmission purposes only whenever a task is executed. Summarizing, we ended up with two alarms, one for task execution and the instantaneous transmission of the produced data, and another for a two way communication between the gateway and the

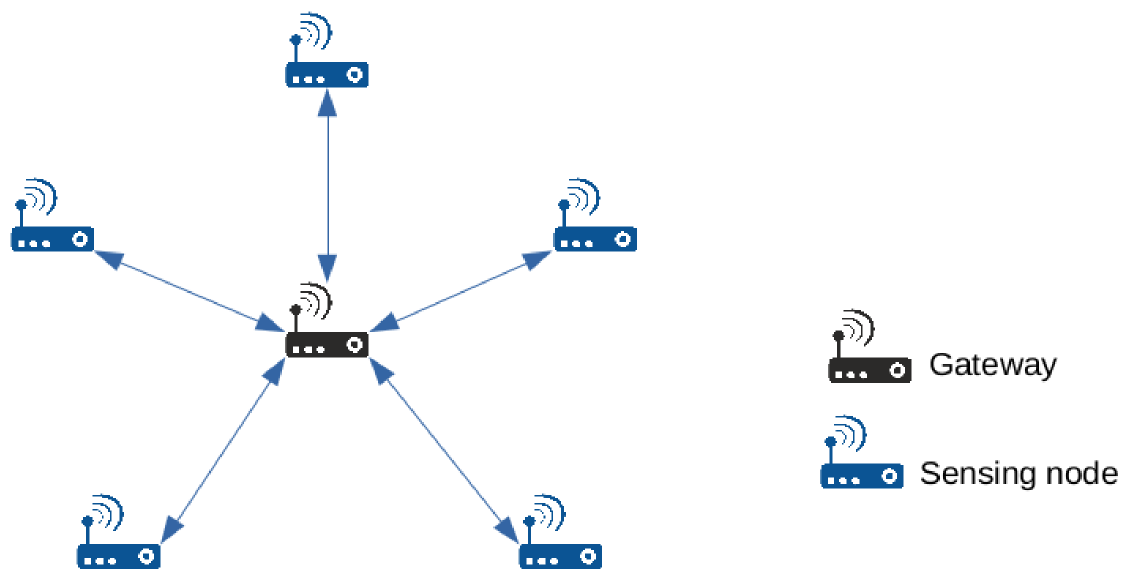


Figure 4.2: WSN Star Topology

node. Note that in this scenario the clocks do not have to be synchronized, since each node acts independently and do not intervene with the correct functioning of one another. The final result is the one presented in Figure 4.3. From the time t_0 to t_1 the network is initialized and the commands can be sent, including task submissions. In instant t_1 the radio module is put to sleep. From t_1 to t_2, the radio module is down most of the time, toggling its state to active only upon the execution of a task in order to send the generated data to the gateway. In instant t_2 the radio is activated to communicate with the gateway and between t_2 and t_3 available commands can be sent to the nodes. Data is not sent to the gateway in this communication window. In t_3 the radio is once again deactivated. This process is repeated while the network is active. The dotted lines represented as **A**, **B**, and **C** serve as guidelines to observe the energy consumption.

Table 4.3 shows the main advantages and disadvantages of using a star topology WSN. A star topology is recommended for small networks, since the number of nodes depend on

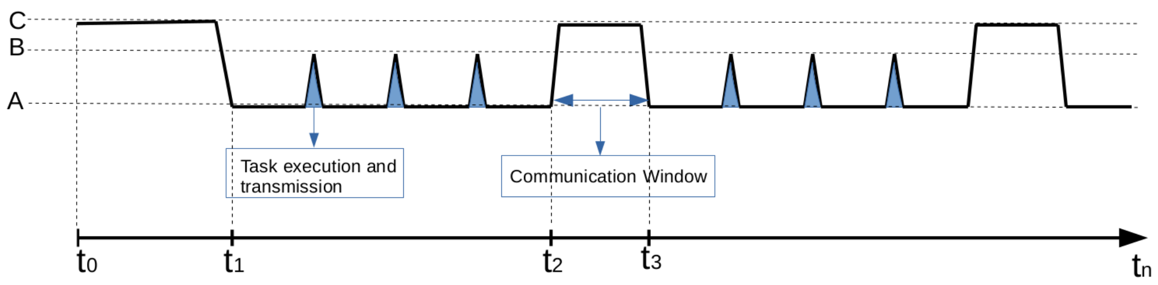


Figure 4.3: Case Study I

Advantages	**Disadvantages**
Easy to add extras nodes	Number of nodes depend on coordinator capacity
Node failure does not affect network performance	Single point of failure
Simplicity of implementation	Coordinator must be in radio range of every node

Table 4.3: Advantages and Disadvantages of Star Topology

how many the gateway/coordinator can manage. This parameter is normally predefined by radio module. Since star networks are also 1-hop network (the nodes do not route data), each node can only send data to the gateway. The range is another parameter that is defined by the radio module. In SONAR we worked with XBee ZigBee S2, and XBee DigiMesh S1, both with a indoor range of 30 meters and outdoor range of 100 meters [1,2]. Through several experiences we observed that these were merely suggestive, since in radio transmissions, factors such as interference, refraction, reflection, and diffraction must be considered, and these factors can alter the radio signal range significantly.

4.3.2 Mesh (Case Study II & III)

The star topology allows us to reduce energy consumption, but to do so we must compromise our mesh topology. As stated in Chapter 1, our main goal is to reduce the energy consumption while preserving the mesh topology, so our next step was to try to implement this low power mesh topology. A first attempt consisted in the implementation of a standard mesh network, composed by sleeping end devices and ZigBee routers, as presented in Figure 4.4.

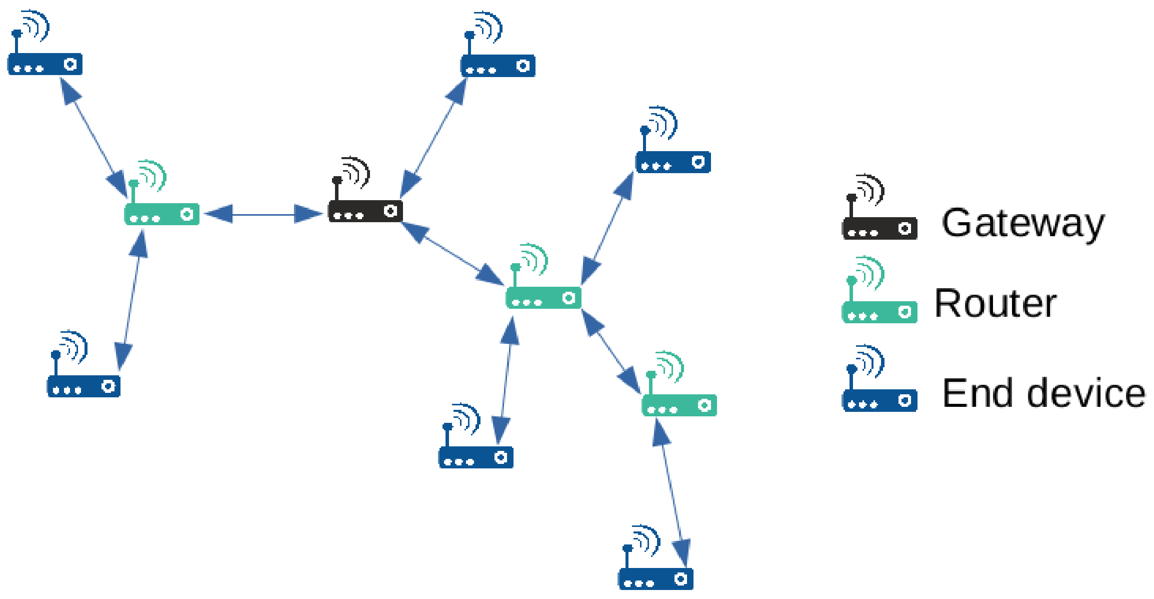


Figure 4.4: WSN Mesh Topology

For the sleeping end devices, we took advantage of the hardware and software setup already developed for the star topology, and consequently, the end devices have the same behavior as the nodes from the star topology, as represented in Figure 4.3. The routers have a slightly different behavior. Because of their need for routing purposes, combined with the inability for having ZigBee sleeping routers, they never enter a low power state. Instead, they are always listening for possible messages requiring routing to its final destination. This happens because ZigBee recognizes it lacks a reliable clock for time synchronization among its nodes, nor any mechanisms to compensate possible clocks drift which is crucial for sleeping routers. If one router fails to wake up when it is supposed to, it can compromise the entire network [2].

The communication between the gateway and the nodes, follows the same patter as that of the star network. If we compare Figure 4.5, corresponding the ZigBee router behavior, with Figure 4.3, corresponding to the end device we can observe the differences between them. The first one is the lack of the radio sleep event at moment t_1. Since the router does not deactivate the radio, we represented the events involving the radio with a smaller line, this way showing that the difference of energy consumption in these events is greater in the end devices. We can also observe the occurrence of another event immediately after the task execution and transmission, the router proceeds to transmit packets originated in other nodes to their final destination. The reason this happens immediately after the execution and transmission is that the nodes are programmed to execute the tasks in the same moment, this way after the router sends its own data it is ready to receive and transmit data from other nodes. This communication from t_1 to t_2 continues to be a one way communication (from the nodes to the gateway), and from t_2 to t_3 we have the two way communication, where tasks can be submitted and other commands can be sent as well.

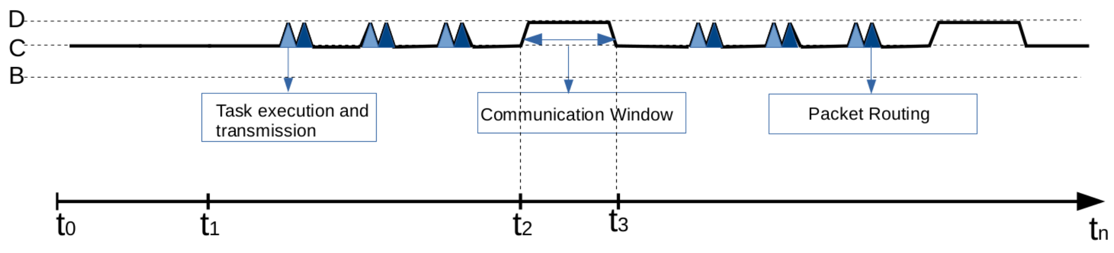


Figure 4.5: Case Study II

The model described above does not implement any mechanism to reduce energy consumption in the router nodes, therefore, we proceeded by trying to implement some energy conservation mechanism in them.

Sometimes the produced data is not necessary in real time, in other occasions, it is not even necessary in their raw format, so data aggregation was considered. Data aggregation allows us to conserve energy by holding some transmissions, and only transmit a set of processed readings, instead of each reading by itself. Duty-cycling being our main focus, we did not make a exhaustive study of the data aggregation area, considering it is a extensive area of study. By working with STL we observed we could apply some simple data aggregation techniques with few changes in the STL program and in the SONAR OS. For this we also took advantage of the communication period parameter already implemented in the task command.

STL program 4.3.1 represents a task that is executed, and in each execution the produced value is transmitted to the gateway. STL program 4.3.2, represents another task, this one with a simple data aggregation technique applied to it. Every time the task is executed, it tries to radio the average value of the produced values until the last execution. The fact that the loop block contains the radio instruction might make us think that the value is transmitted in each execution, which is a valid thought. To avoid this to happen, the node calculates how many times it must execute a task before transmitting it, upon receiving such task. We know we receive the task period p, the communication period c, and we also know that this information can only be received on the communication window. With this in mind we can calculate $N = c/p$, with p being divisible by c, whre N defines the number of executions before the aggregated value is transmitted to the gateway. After each execution the number of executions n is incremented by one, and if the condition $n = N$ is met, the data is transmitted. With the limitation that p is divisible by c, we can guarantee the transmission attempt is always performed in the communication window ($c = N * p$).

Figure 4.6 describes the router behavior in this model. From t_0 to t_1 it has the standard behavior described in the previous model. From t_1 to t_2 it performs the executions but do not transmit any data, nor it routes any packet, instead the data is aggregated and stored.

STL Code 4.3.1 Standard STL code

```
sensors {
  temperature: void -> float ,
  humidity: void -> float
}

init {
  float x = 5.0;
  float y = 0.0;
}

[ float @ "temperature:Celsius",
  float @ "humidity:%" ]
loop {
  x = temperature();
  y = humidity();
  radio [x,y];
}
```

From t_2 to t_3 it perform its Nth execution and transmits the aggregated data. Note that t_2 to t_3 is the communication window and the nodes continue to perform the two-way communication in this interval, as described in the previous model. The average value is just a example and other local data-aggregation techniques can be applied, minimum value, maximum value, median, sliding average, etc.

After we succeeded to have sleeping end devices and a communication interval where we perform all the necessary communication, our next goal was to have the routers sleeping when outside of the communication interval (t_2 to t_3). The documentation was not quite clear whether this was possible to accomplish using ZigBee. We tried to implement this kind of routers anyway. The premise was that if we could switch the XBee mode of operation from router to end device locally and instantly, we could then put the radio module to sleep, and changing it back from end device to router upon waking the radio

STL Code 4.3.2 STL example code with data aggregation

```
sensors {
  temperature: void -> float ,
  humidity: void -> float
}

init {
  float x = 0.0;
  float y = 0.0;
  float n = 0.0;
}


[ float @ "temperature:Celsius",
  float @ "humidity:%" ]
loop {
     x = x + temperature();
     y = y + humidity();
     n = n + 1.0;
     radio[x/n, y/n];
}
```

module. The problem with this approach is that upon issuing the command to change the XBee mode of operation all routing information and parent-child relationship would be lost, and unfortunately ZigBee does not grant access to this information locally. Taking into consideration that when the mode of operation is changed the node looses thisinformation and leaves the network, we introduced a delay in order to have enough time for the nodes to rejoin the network and rebuild the routing paths. When running experiences with this model implemented we observed that it came with a new set of problems, for example, the end device would not immediately associate itself with the router. For this to happen, the end devices would also have to be restarted to force it to issue a request to join the network. But this wasn't all, we also observed that when altering the XBee mode of operation, the

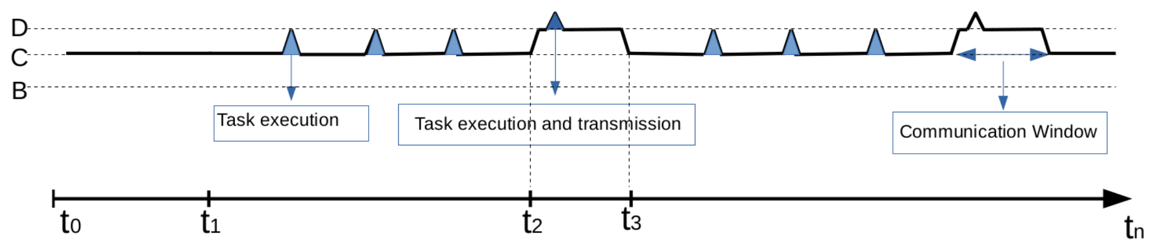


Figure 4.6: Case Study III

Arduino Board itself suffers a reset, forcing it to start all the process from the beginning, which means that all the information already there would be lost. We later abandoned this idea, but we kept researching and pursuing other ways to have sleeping router, it was then that we came across DigiMesh.

4.4 DigiMesh (Case Study IV)

At this point we have implemented some models, moving toward the goals set in the beginning of this thesis, implementing the star topology, a standard mesh topology, and a mesh topology with a simple local data aggregation technique applied to it. This work was accomplished using the ZigBee protocol. Considering ZigBee's limitations, namely the impossibility of having sleeping routers, we acquired new radio modules, the XBee S1 XB24-DM, and consequently adopted the protocol associated to it, DigiMesh.

Unlike ZigBee, which defines three types of node (coordinator, router, and end device), DigiMesh defines only one node type, and all nodes can participate in packet routing, and are interchangeable. DigiMesh nodes does not define any parent-child relationship and do not keep information about paths which are not currently being used. Figure 4.7 illustrates a mesh topology network using DigiMesh nodes, one of these node behaves as the gateway of the network. Although the figure shows only one path to each node, DigiMesh networks can be represented with connections from each node to every node in its direct range. The purpose behind this is to demonstrate its self healing capabilities.

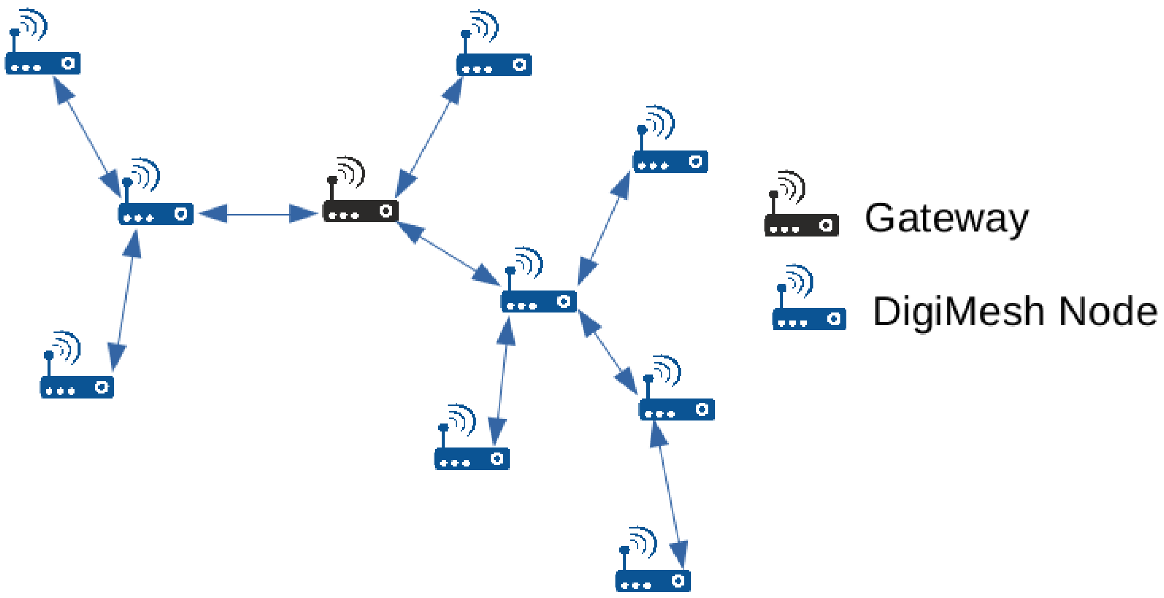


Figure 4.7: WSN Digimesh Topology

Even though we changed our radio module, we could still take advantage of all the functions already implemented. The XBee S2 with support for ZigBee, and the XBee S1 with support for DigiMesh were both conceived by Digi International, and from one to another they preserved the hardware configuration. They also incorporated the data frame structure used in ZigBee for DigiMesh in order for the developers to reuse any function already implemented for ZigBee.

DigiMesh provides a set of low power modes to extend its time of operation, and like ZigBee these modes can be used by configuring the SM parameter. The available sleep modes are categorized as either synchronous or asynchronous. For asynchronous modes the SM parameter can be configured as follows:

- Asynchronous pin sleep: the SLEEP_RQ pin controls the state of the module, the same as in ZigBee (SM=1);

- Asynchronous cyclic sleep: the module periodically sleeps and awakesbased on the values of SP (Sleep Time), and ST (Wake Time) (SM=4);
- Asynchronous cyclic sleep/Pin wake: The module sleeps periodically and can be awoken by asserting the SLEEP_RQ pin (SM=5).

For synchronous mode the available configurations are:

- Synchronous sleep support mode: The module with this configuration coordinates the sleeping status of the network, but does sleep itself. It is responsible for synchronizing the network sleep time, and respond to new nodes attempting to join the network (SM=7);
- Synchronous cyclic sleep mode: The module sleeps for a specified time and awakes in unison with the network to exchange data and sync messages (SM=8).

The values 2, 3, and 6 are reserved by the manufacturers.

A DigiMesh network cannot have both synchronous and asynchronous nodes belonging to it. Using asynchronous mode would eliminate the possibility of having sleeping routers, which was the main reason we adopted DigiMesh. This is because using asynchronous mode the nodes would control their sleeping time locally with no coordination among them, and a small clock drift in a single node can compromise the entire network. So, obviously, we opted to use synchronous sleep mode. We elected the gateway as the sleep coordinator, setting its SM parameter to 7 (synchronous sleep support mode), because this mode does not allow the node to sleep, and the gateway is supposed to be a mains powered node. The remaining nodes were configured to operate in synchronous cyclic sleep mode. Upon initializing the network, we provide the desired sleep and wake time to the gateway. The gateway broadcasts these values to every node configured to participate in the network. Each node keeps an internal sleep timer to know when the sleep time expires. The nodes then wake up to transmit and receive sync messages.

Using synchronous cyclic sleep we only need one alarm from the RTC Chronodot, the one used to trigger task execution. With ZigBee we needed to keep the the sleep timer ourselves, so we programmed the alarm for this purpose. DigiMesh exchange time synchronization messages to keep the clocks synchronized. The synchronization messages are exchanged periodically between the gateway and the nodes, if clock drift is detected the gateway commands the drifted node to adjust its clock. In consequence of this, DigiMesh nodes can sleep and route packets. They can be trusted to not fail the wake cycles, and if it happens, there are mechanisms to put the node back in sync with the rest of the network. DigiMesh is more limited than ZigBee regarding packet size and bandwidth, so a few changes had to be made in the code we use to divide tasks into blocks in order to fit DigiMesh message buffers. The communication period parameter that had two functions, to determine the sleeping time and calculate the maximum task execution number, would now serve only for the later.

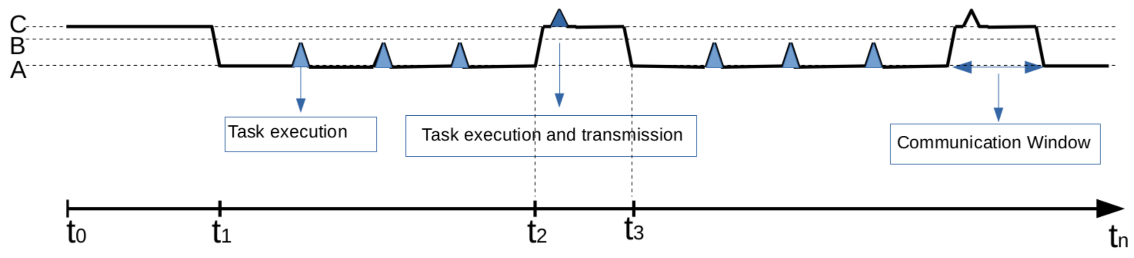


Figure 4.8: Case Study IV

Figure 4.8 shows the node behavior of this last model, which applies to every node in the network. From t_0 to t_1 we have the network initialization. In t_1 the nodes enter a sleep state for the amount of time specified by the gateway. During their sleep time (t_1 to t_2), they periodically execute the tasks and aggregate the generated data. From t_2 to t_3 we have the communication window where, once again, commands can be sent, and sync messages are exchanged. The data transmission from the nodes to the gateway also occurs in this interval. The DigiMesh node's behavior is similar to the end device's behavior in the previous model, the difference being that in this model this behavior is global to all nodes instead of subset of nodes.

4.5 Summary

In this chapter we identified the possible solutions for energy conservation in SONAR, presented the different current supply for each of SONAR hardware component and made a comparison among them in order to highlight the one worth focusing on. Then we presented the newly introduced commands in SONAR, focusing on the changes we made in some of them and in the newly implemented commands. Later on we described the different case studies we implemented and tested, alongside with the node's behavior in each of these models, describing what changes from one to another.

Chapter 5

Evaluation

In this chapter we present the collected results obtained from the different case studies we tested. We describe and analyze this results independently, then compare them to each other. We evaluate the trade offs associated with each case study and provide a general model for calculating the average energy consumption per cycle in the applied energy conservation schemes.

5.1 Setup

The results presented on this thesis were collected under the following configurations:

- Network Configuration: our experiments were conducted with small networks. In ZigBee star topology we worked with six SONAR nodes acting as end devices, and one gateway. In ZigBee mesh topologies we also worked with six nodes, two of them acting as routers and four acting as end devices, and a gateway. In DigiMesh we worked with one gateway and three DigiMesh nodes.
- Measurement Setup: to measure the energy consumption we connected a multimeter in series with a node, as presented in Figure 5.1 and measured the electric current

variation in the node. The multimeter used was a *TENMA 72-7732A*

Several experiments were conducted with each case study. The presented results represent one of these experiments.

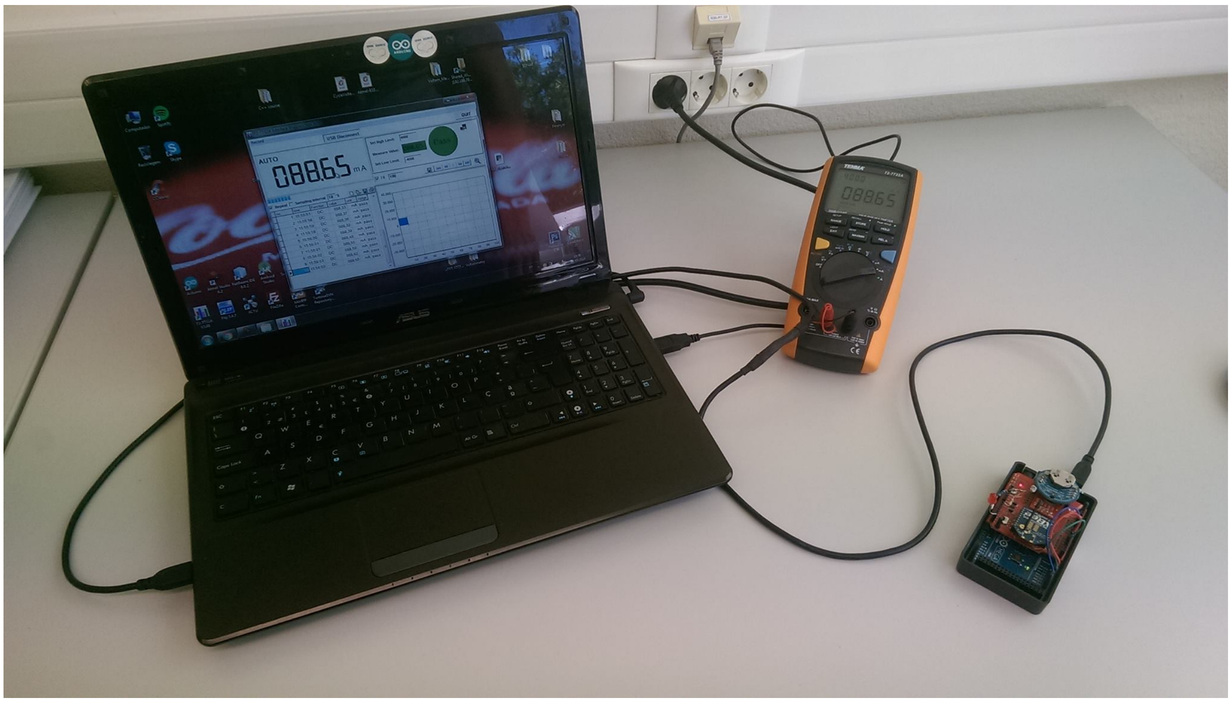


Figure 5.1: Energy Measurement Setup

5.2 Results Report

Figure 5.2 illustrates the current drawn from a node in the star topology. The result is as expected and described in Section 4.3. In the beginning the radio module is active, and the node is in its initialization process (I), thus we observe a relatively high current peak. Then, we have the stage where the tasks are executed and transmitted (T). A single task was submitted in each test for visualization purposes. Note that, even though the radio module becomes active after the task execution, the current draw at that moment is not as high as in the communication window (W). This is because we do not allow the radio to listen to any incoming messages at that moment. In the same figure we can also observe when the task is discarded from the network, resulting on the deactivation of the radio module until the next communication window. In this model the base current drawn is approximately 51 mA and a peak current drawn of 90 mA. The percentage of duty-cycling applied is this test case is around 20%, which means that the radio module is deactivated for 80 % of the time, and the 51 mA current draw occurs is 80% of the network lifetime.

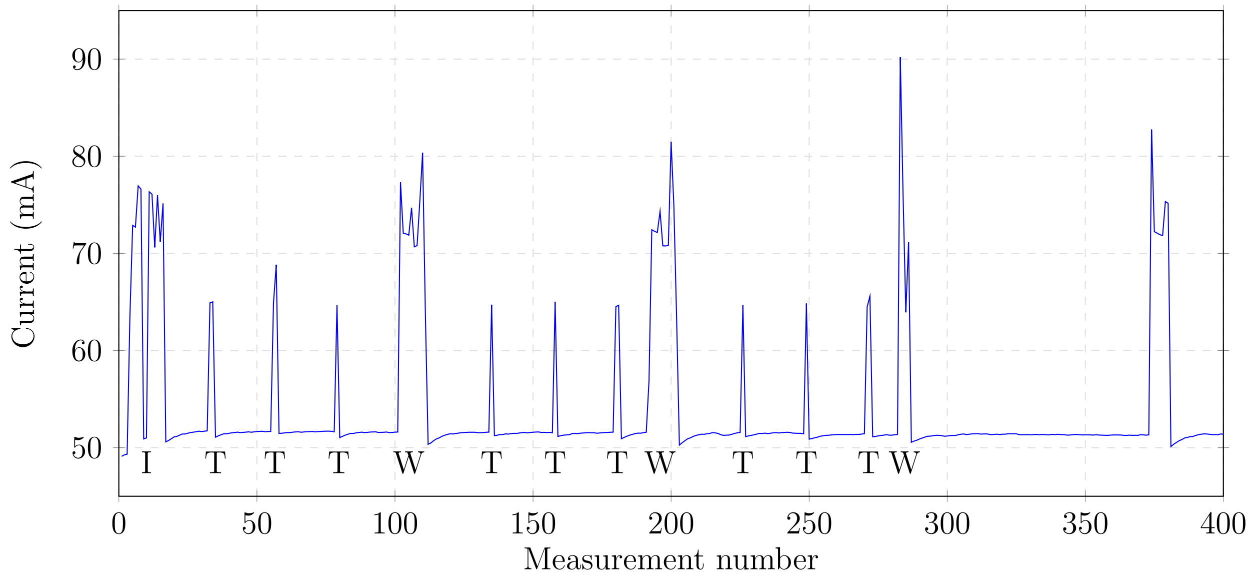


Figure 5.2: End Node Current Draw (Case Study I)

Figure 5.4 illustrates the current drawn from a router node in the ZigBee mesh network without the local data aggregation. In this model the data collected is transmitted right after the task execution (T). The router node has 100% duty-cycling, but the highest values of current draw are achieved in the communication window (W), where the module is also listening for incoming messages. In the communication window we can observe the routing activities (R). Most of the energy conservation is achieved by the sleeping end devices in the mesh network, which did not exist in the original SONAR networks, which we can observe in Figure 5.3. The figure depicts a simple radio transmission of 64bytes, without any task execution, and yet it presents a higher current consumption than any of our case study.

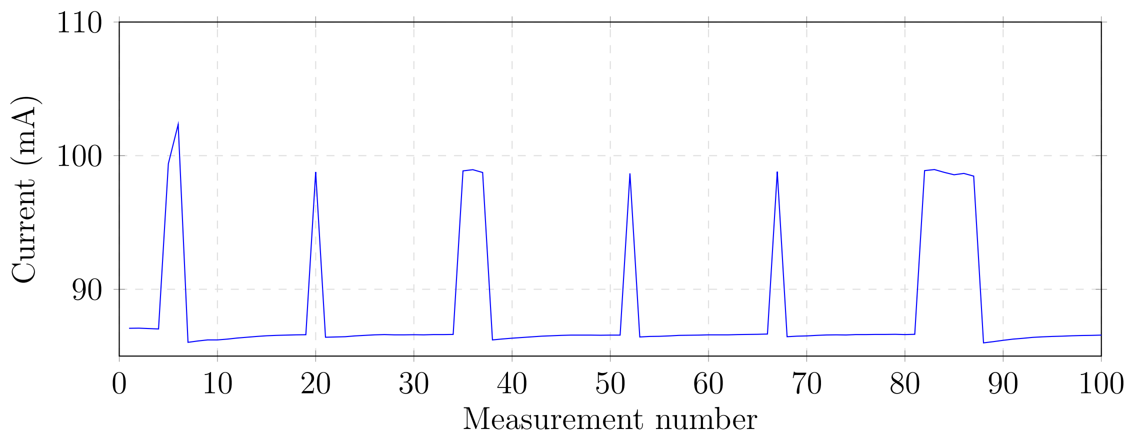


Figure 5.3: SONAR 64bytes Radio Message, from [11]

Figure 5.5 presents the result by applying the already mentioned local aggregation technique in the ZigBee mesh network. By taking a closer look we can verify that the Figures 5.4 and 5.5 present a subtle difference in the task execution peak (T): it lasts longer in Figure 5.4. This is due the absence of the transmission following the task execution in Figure 5.5, rather it occurs in the communication window. This allows us to save a small amount of energy in the router nodes. Furthermore, the sleeping end devices implement the same aggregation technique and can also save this small amount of energy, and only transmit in the communication window (W). In both Figures 5.4 and 5.5 the base current draw is approximately 81mA and the peak is around 97mA, but the average values are 84.6mA and 83.8mA, respectively. This difference is due the small amount of energy saved by aggregating the data in Case Study III. The presented average values presented were calculated by adding all the measurements and dividing it by the total number of measurements. The end nodes in these case studies follow the same pattern as the star end end node, thus the absence of their current consumption illustration.

Figure 5.6 illustrates the results obtained from a DigiMesh node. In this model all the nodes have the same configuration. When active, the DigiMesh radio module has a higher current draw than the ZigBee module, but by applying a low percentage of duty cycling this difference becomes irrelevant. The result is from an experience with 20% duty cycle

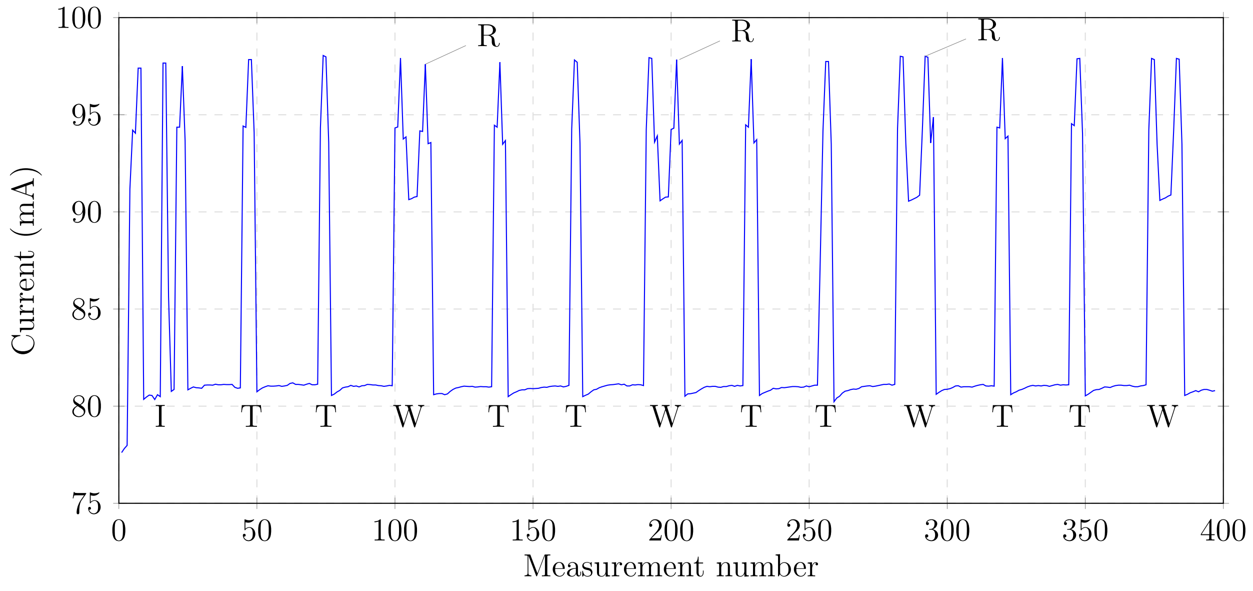


Figure 5.4: ZigBee Router Current Draw (Case Study II)

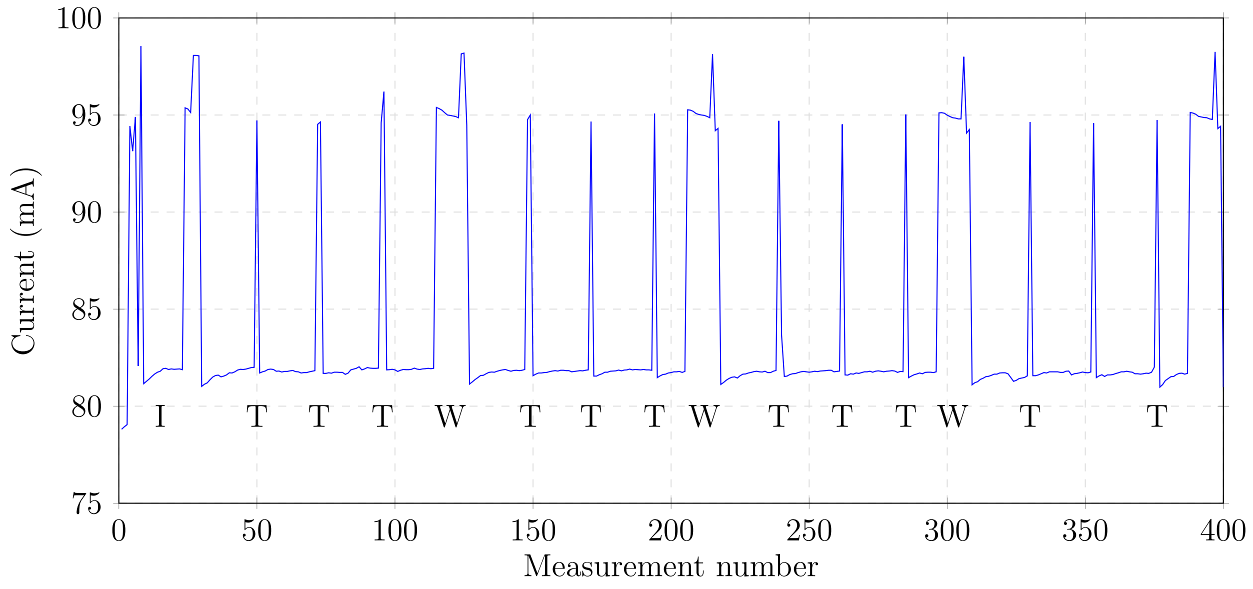


Figure 5.5: ZigBee Router with Aggregation Current Draw (Case Study III)

and the model also performs the local data aggregation technique. The radio module is only active for a short period of time in the communication window (W). In this case we can observe a base value of current draw of 51mA, and a peak value of 110mA, which occurs when the radio is either transmitting or receiving messages. We can also observe the task executions (T) inside the communication window, as we stated in Chapter 4.

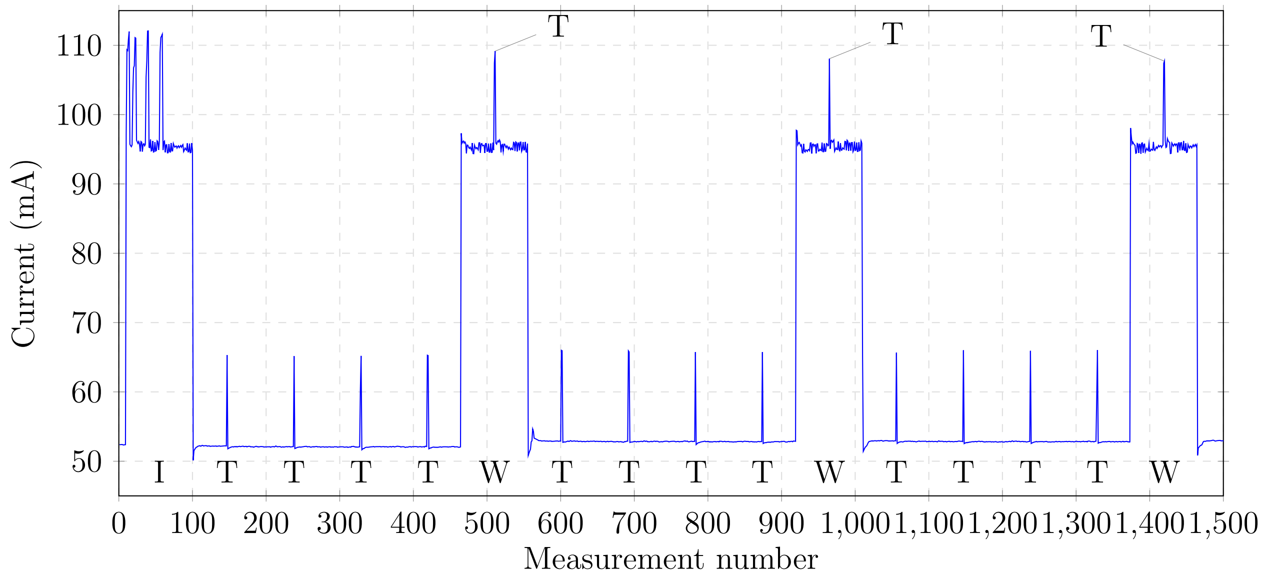


Figure 5.6: DigiMesh Node Current Draw (Case Study IV)

5.3 Comparison and Discussion

Table 5.1 summarizes the consumption values for each model. The unit of every value presented in the Table is milliampere. The table contains information about each case study: base consumption, average consumption per cycle, minimum, and maximum value achieved in each model. What we consider as the base value is the current that is draw outside the communication window, when the node is in its lowest power consumption mode. The average value presented is relative to one cycle in each case study, which we define as the time between the start of a communication window to the start of the next one. The min an max, are the lowest and the highest peak values, respectively.

Case Study	Base (mA)	Average (mA)	Min (mA)	Max (mA)
I	51	54.1	49.1	90.2
II	81	84.5	77.6	98.6
III	81	83.8	77.4	98.2
IV	52	66.8	50.1	112.7

Table 5.1: Case Studies Values Comparison

Analyzing the base and average values, we can observe that Case Study I is the one with the best behavior, but we must take into consideration that those values are from a star network, which means that in order achieve them we had to renounce the mesh network, ending up with a network with a limited range and scalability.

Case Study II and III have the highest base and average values, but we must not forget that these values are from the routers in a mesh network. The end devices in these networks follow the same energy consumption pattern as those in Case Study I. In this case we end up with networks that possess nodes that behave differently among themselves regarding power consumption. If we want to have the same life spawn in every node, the routers must have a stronger power source or the network must be periodically reorganized if possible. The values from case studies II and III have little difference between than, what leads us to conclude that the mechanism applied in Case Study III is only worth it if the network is intended to run for a extremely long period of time, otherwise we would sacrifice the real time communication of data for a insignificant amount of energy.

Although Case Study IV presents a low base value, its average value is still higher than Case Study I. This is because DigiMesh radio modules have a higher power consumption than ZigBee's when active. The advantage of using DigiMesh is that all nodes follow the same energy consumption model. We also observed that with each cycle the average value gets lower, as presented in Figure 5.7. This is due the smoothing of the initialization peaks.

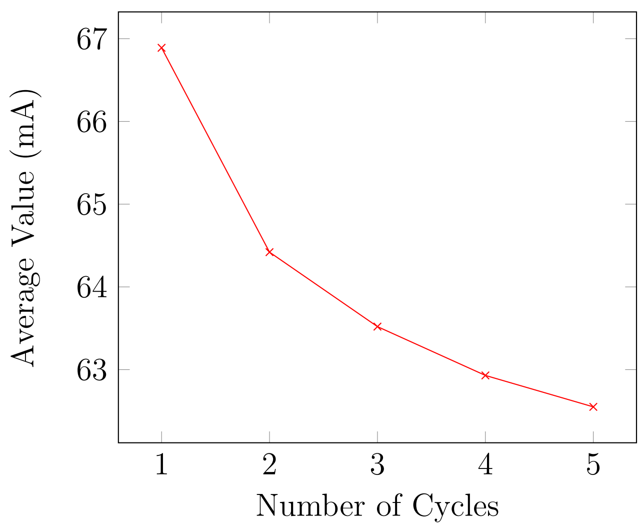


Figure 5.7: DigiMesh Average Values

The trade-off associated with this case study is real time communication of data and its raw format, but both of them can be recovered with little cost, by implementing the model where the task is executed, the radio is turned on to send the data and turned of again. This way we would end up with a model similar to that of the end nodes in Case Study III (execute task, turn on radio, send data, turn off radio), but in this case it would be applied to all nodes in the network, since DigiMesh guarantees the correct functioning of routing in sleeping networks. Setting the communication window low smaller intervals would cause this case study to have a lower average value. The smaller the communication window, the closer the average value gets to the base value.

If we observe the result yield by each case study we can see that all of them follow the same pattern: at first the current drawn is high, then it goes down until it executes a task, or in absence of a task, until it enters the new communication window, and this behavior is cyclic. If we ignore the tasks and, in consequence, the data transmission to the gateway we end up with a model for the average current draw per cycle as presented in Figure 5.8.

$$avg = \frac{i_0 * \Delta t_0 + i_1 * \Delta t_1}{\Delta t_0 + \Delta t_1} \tag{5.1}$$

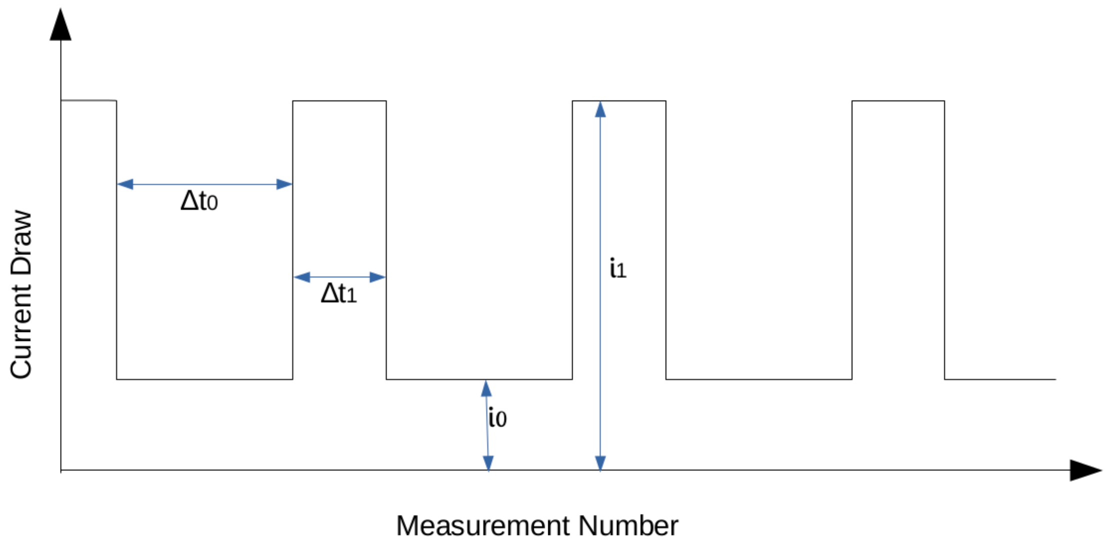


Figure 5.8: General Consumption Model

The values i_0 and i_1 are assumed to be constants in each model, although this does not happens in reality, the variation in quite small. The values Δt_0 and Δt_1 can be dynamically set by the user, and these values determine the percentage of duty-cycling. Equation 5.1 determines the average current draw per cycle based on these parameter. The formula can also determine the average power consumption since current consumption are positively correlated: $P = I * V$, where P is the power consumption, I is the current draw, and V is the voltage, which is constant.

5.4 Summary

In this chapter we presented the results obtained from each case study that we tested. We analyzed and commented these results, presenting the trade-offs of each one of them, and the situations in which each one of them is recommended. We summarized the obtained results in order to make a comparison amonf them. We also presented general formulas to calculate the power consumption in the implemented and test case studies.

Chapter 6

Conclusion and Future Work

In this thesis we researched and tested mechanisms to conserve energy in SONAR networks. From the studied mechanism some were applicable to SONAR, and some were not. This resulted in the testing of four case studies, each one with specific aspects. We tested a star topology, and three different implementation of mesh networks: a simple ZigBee-based mesh network, a ZigBee-based mesh network with a simple data aggregation scheme, and a DigiMesh-based network, also with the data aggregation scheme. The results obtained from these tests were compared in order to evaluate the different degrees of energy conservation.

All the case studies presented some kind of energy conservation when compared to the previous version of SONAR, which did not have any conservation scheme applied to it and all the nodes acted as routers with the radio module always on a listening state.

In order to conserve energy in WSN's compromises must be made, and trade-offs are required. In the case of the star topology we had to give up the the range of the network and the amount of nodes that can belong to it. In order to conserve more energy in a mesh network with sleeping end nodes we applied a data aggregation technique and had to compromise the real time communication, and the raw format of the generated data. This being done we tested another hardware, DigiMesh, where we could finally have sleeping routers, but we also compromised the real time communication and the raw format of the data. We also limited the bidirectional communication to a small communication window, thus compromising the real time management of the network.

Winding up, energy conservation in WSN's can be achieved through compromises and trade-offs. The amount of energy we can save depends on the application requirements and the compromises we are available to make. For our purposes and type of applications we use with SONAR, Case Study IV presented the best results, since it significantly decreased the energy consumption preserving the mesh topology.

The communication window we applied resulted in a 20% duty-cycling, this window can and should be optimized to the minimum interval possible that guarantees the correct functioning of the network. In this thesis of focused on duty-cycling schemes to achieve energy conservation, and not all of them were implemented. In order to increase the amount of energy conservation other techniques can be considered, *e.g.,* staggered wake up. Data aggregation along the network, which can decrease the amount of data flow in the network can be applied. Data compression algorithms should also be considered in future work since it can significantly decrease the amount of data flow in the network.

Bibliography

[1] *Xbee/Xbee-PRO DigiMesh 2.4 RF Modules User Guide, RevT.* `http://www.digi.com/resources/documentation/digidocs/pdfs/90000991.pdf`, Last accessed: 30-09-2016.

[2] *Xbee/Xbee-PRO ZigBee RF Modules User Guide.* `http://www.digi.com/resources/documentation/digidocs/PDFs/90000976.pdf`, Last accessed: 30-09-2016.

[3] *IEEE Standard for Local and metropolitan area networks-Part 15.4: Low Rate Wireless Personal Networks, IEEE Standard 802.15.4*, 2011.

[4] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537 – 568, 2009.

[5] R. Barr, J. Bicket, D. Dantas, B. Du, T. Kim, B. Zhou, and E. Sirer. On the need for system-level support for ad hoc and sensor networks. *ACM SIGOPS Operating Systems Review*, 36(2):1–5, 2002.

[6] S. Bhati, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantisos: An embedded multithreaded operating system for wireless micro sensor plataform. *Mobile Networks and Applications*, 10(4):563–579, 2005.

[7] P. Budhwar. Tinyos: An operating system for wireless sensor networks. *International Journal of Conputer Science and Technology*, 6, 2015.

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

[9] A. Cerpa and D. Estrin. Ascent: adaptive self-configuring sensor networks topologies. *Mobile Computing, IEEE Transactions on*, 3(3):272–285, July 2004.

[10] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494.

[11] G. Ferro. Towards out-of-the-box wireless sensor networks. Master's thesis, Departamento de Ciências de Computadores, Faculdade de Ciências da Universidade do Porto, 2015.

[12] G. Ferro, R. Silva, and L. Lopes. Towards Out-of-the-Box Programming of Wireless Sensor-Actuator Networks. In *Computational Science and Engineering (CSE), 2015 IEEE 18th International Conference on*, pages 110–119. IEEE, 2015.

[13] P. Godfrey and D. Ratajczak. Naps: scalable, robust topology management in wireless ad hoc networks. In *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, pages 443–451. IEEE, 2004.

[14] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734, 2012.

[15] A. Kaur and A. Grover. Leach and extended leach protocols in wireless sensor network-a survey. *International Journal of Computer Applications*, 116(10), 2015.

[16] T. Kim and S. Hong. Sensor network management protocol for state-driven execution environment. In *International Conference on Ubiquitous Computing*, pages 197–199, 2003.

[17] P. Levis and D. Culler. Mate: A Tiny Virtual Machine for Sensor Networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, October 2002.

[18] P. Levis, S. Maddem, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The emergence of networking abstractions and techniques in tinyos. Technical report, University of California, Intel Research Berkeley, CSAIL MIT, 2004.

[19] X. Li and S. Moh. Midleware systems for wireless sensor networks: A comparative survey. *Contemporary Engineering Sciences Vol 7*, 2014.

[20] Maheswar. A survey on duty cycling schemes for wireless sensor networks. *International Journal of Computer Networks and Wireless Communications (IJCNWC)*, 3(1):37–40, 2013.

[21] A. Mallikarjuna, P. Kumar, D Janakiran, and G Kumar. Operating systems for wirelles sensor networks: A survey. Technical report, Indian Institute of Technology Madras, 2007.

[22] L. Mottola and G. Picco. Programming wireless sensor networks: Fundamentals concepts and state of the art. *ACM Comput. Surv.*, 43(3), April 2011.

[23] J. Olsson. 6lowpan demystified. Technical report, Texas Instruments, 2014. `www.academia.edu/download/37904447/swry013.pdf`, Last accessed: 30-09-2016.

[24] J. Polastre, J. Hill, and D. 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.

[25] K. Rajendran, V.and Obraczka and J. J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78, 2006.

[26] V. Rajendran, J. J. Garcia-Luna-Aveces, and K. Obraczka. Energy-efficient, application-aware medium access for sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, pages 8–pp. IEEE, 2005.

[27] H. Rashvand, A. Abedi, J. Alcaraz-Calero, P. Mitchell, and S. Mukhopadhyay. Wireless sensor systems for space and extreme environments: A review. *IEEE Sensors Journal*, 14(11):3955–3970, Nov 2014.

[28] T. Rault, A. Bouabdallah, and Y Challal. Energy Efficiency in Wireless Sensor Networks: a top-down survey. *Computer Networks*, 67(4):104–122, July 2014.

[29] K. Sohraby, D. Minoli, and T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*, chapter Operating Systems for Wireless Sensor Networks. Jonh Wiley & Sons, Inc., 2007.

[30] P. Thubert and J. Hui. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, September 2016.

[31] D. Vujić. Wireless sensor networks applications in aircraft structural health monitoring. *Journal of Applied Engineering Science*, 13(2):79–86, 2015.

[32] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, pages 70–84, New York, NY, USA, 2001. ACM.