UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

Progettazione e sviluppo di un sistema cromoterapico mediante una rete di sensori wireless

RELATORE: Ch.mo Prof. Schenato Luca

LAUREANDO: Massimo Marra

Padova, 7 Dicembre 2010

UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

Master's Degree in Computer Engineering

Design and implementation of a chromotherapy system using a wireless sensor network

Supervisor: Prof. Schenato Luca

Author: Massimo Marra

ACADEMIC YEAR 2010-2011

I dedicate this thesis to my parents Danilo and Renata,
my brother Marco, my sister Silvia,
and to my love Marta

Abstract

The work of this thesis consists in the development and implementation of a chromotherapy system based on a WSN. The system is independent from the environment in which is installed and is very flexible. The nodes of the system interact with each other to synchronize themselves and to disseminate the color sequence to display.
Synchronization can be managed and controlled through a Java interface that allows the parametrization of many aspects of the algorithm.
The system is able to recognize if topology changes occur and is also able to reconfigure itself accordingly without affecting the nodes synchronization. This important characteristic is guaranteed by the algorithms proposed in this work. The network synchronization is based on the offset compensation of the local clocks of the nodes and is achieved through the local information exchange between neighboring nodes. A fast convergence to a common value of the virtual global clock, and a high accuracy is obtained thanks to a dynamic hierarchical overlay structure.
The color therapy sequence is generated in real-time from a Java application. This software divides the sequence and sends the portions to a reference node whose task is to communicate them to the rest of the network. The dissemination takes place with a multi-hop flooding process of all the sequence portions. The system must introduces a delay between the generation instant and the displaying instant of the sequence. This time interval is necessary for the multi-hop communication to take place. Also the color therapy functionality of the system is independent from the network

topology. Therefore the system can be implemented even in networks that change over time.

The synchronization algorithm and the chromotherapy system have been implemented on a Tmote sky/TinyOS v.2 architecture. The testing process of all the functionalities was performed on a real WSN. The excellent behavior of the system and the good performances obtained show the effectiveness of the proposed design methodologies.

Sommario

Questo lavoro di tesi è consistito nello sviluppo e nella relativa implementazione di un sistema cromoterapico basato su di una rete di sensori wireless. Il sistema è indipendente dall'ambiente nel quale viene installato risultando perciò molto flessibile nell'utilizzo. Ogni nodo della WSN interagisce con gli altri cercando di creare una rete sincronizzata e permettendo la diffusione e la visualizzazione di una sequenza di colori atraverso un device RGB esterno.

Il sistema può inoltre riconoscere se un cambiamento topologico sta avvenendo nella rete ed è in grado di riconfigurarsi di conseguenza senza influire sulla sincronizzazione dei nodi. Questa importante funzionalità è garantita dall'algoritmo di sincronizzazione proposto in questa tesi. Esso si basa sulla compensazione dell'offset dei clock locali dei singoli nodi e sullo scambio locale di informazioni temporali tra nodi vicini. L'ottima precisione dell'algoritmo ed una veloce convergenza dei nodi ad un unico clock globale di riferimento sono ottenute attraverso una struttura di overlay gerarchica. Anche questa struttura assicura dinamicità al sistema essendo robusta a variazioni topologiche. Il protocollo di sincronizzazione può essere gestito e controllato attraverso un'applicazione Java che permette la parametrizzazione di molti aspetti dell'algoritmo.

Anche la sequenza cromoterapica utilizzata dai nodi viene creata in realtime da un software Java. Questa applicazione, non appena ha generato una porzione della sequenza composta da un certo numero di colori, la inoltra ad un determinato nodo di riferimento il qui scopo è quello di comunicarla

ai restanti nodi della rete che dovranno emetterla attraverso una periferica RGB. La diffusione delle parti della sequenza è effettuata attraverso una comunicazione di tipo flooding multi-hop. Il sistema ha la necessità di inserire un piccolo ritardo tra l'istante della generazione di una porzione di sequenza e l'istante corrispondente alla sua effettiva visualizzazione da parte dei nodi. Questo lasso di tempo è necessario per permettere che avvenga la comunicazione multihop. Anche la funzionalità cromoterapica è indipendente dalla topologia della rete ed è robusta agli spostamenti spaziali dei nodi. Risulta quindi possibile implementare questo sistema cromoterapico anche in reti che possono cambiare la loro configurazione nel tempo.
L'algoritmo di sincronizzazione ed il sistema cromoterapico sono stati infine implementati su di una architettura composta da mote Tmote sky e dal sistema operativo TinyOS ver.2. L'intera realizzatione ottenuta è stata testata su di una WSN reale. L'ottimo comportamento del sistema e le performance ottenute dimostrano l'efficacia delle scelte progettuali adottate.

Contents

List of Acronyms

ADC	Analog-to-digital converter
AmI	Ambient Intelligence
API	Application Programming Interface
ATS	Average TimeSync
CBSE	Component-Based Software Engineering
CSV	Comma-Separated Values
DAC	Digital-to-analog converter
DS	Distributed Systems
EEPROM	Electrically Erasable Programmable Read-Only Memory
EDS	Electrostatic Discharge
FIFO	First-In First-Out
GUI	Graphical User Interface
HAA	Hardware Abstraction Architecture
HAL	Hardware Abstraction Layer
HIL	Hardware Independent Layer

HPL	Hardware Presentation Layer
IFA	Inverted F Antenna
ISM	Industrial Scientific Medical
LED	Light Emitting Diode
lsb	least significant bit
MAC	Media Access Control
MCU	Micro Controller Unit
MDB	Memory Data Bus
MIG	Message Interface Generator
ms	milliseconds
NTP	Network Time Protocol
O-b	Overlay-based
OC	Offset Compensation
OLS	Ordinary Least Squares
OS	Operating System
p2p	Peer-to-peer
PTP	Precision Time Protocol
RF	Radio frequency
RGB	Red Green Blue
RSSI	Received Signal Strength Indicator
SFD	Start Frame Delimiter
SPI	Serial Peripheral Interface

UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
ubicomp	Ubiquitous computing
WSN	Wireless Sensor Network
WSAN	Wireless Sensor and Actuator Networks

List of Figures

List of Tables

Chapter 1

Introduction

The recent technological improvement in the low cost miniaturization of electronic devices and in the wireless communication, has made possible the opportunity to create low-power consumption sensors with a good efficiency. The integration of computation, sensing, communication and storing activities on a single small device has opened new horizons for the Distributed Systems (DS)[1]. These kind of appliances are the fundamental elements of a Wireless Sensor Network (WSN).

A WSN consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions. Every single unity of a WSN can communicate with each other. Before the advent of this technology, the capability to cooperate among sensors was constrained by the use of cables for the information transmission. The nodes of a WSN have instead introduces many new fundamental characteristics, the mainly are:

- they are connected through their radio chips using free radio frequencies;
- they are miniaturized;
- they are less expensive;
- they can be deployed in wide areas, and must be easy to install;
- they need less maintenance;
- the network that they form must be scalable;
- they can be easily attached even to moving parts.

So it is easy to understand why WSN are widely studied, and the reason of their diffusion not only in R&D activities. WSN have some strengths, but have also weaknesses. In fact they are generally powered with batteries, and it is well known that with a limited power source, the energy consumption becomes a great problem. Another complication is the node short radio communication range necessary to limit power consumption. These aspects may lead to unreliable communication network.

WSN are a limited part of a greater field: the Pervasive Computing [2] also called Ubiquitous computing (ubicomp). This is a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities. In the course of ordinary activities, someone utilizing ubiquitous computing engages many computational devices and systems simultaneously, and may not necessarily even be aware that they are doing so. This model is usually considered a revolutionary advancement from the desktop paradigm. In fact pervasive computing devices are not personal computers as we tend to think of them, but very tiny devices[1] all communicating through increasingly interconnected networks. So networks give rise to an intelligent environment, able to interact with the man and the objects, trying to allow a perfect fulfillment of human needs. This is also known as Ambient Intelligence (AmI) which is a human-centric computer interaction design characterized by systems and technologies that must be integrated into the environment to recognize the human actions and the situational context in order to change in response of them. This model must also be personalized and finally in some cases should anticipate the humans desires. Even for example the concept of smart city, like CitySense[3], belong to AmI.

A WSN is a ductile instrument that could be exploited in many different application. It is sufficient a simple Internet search to find out that these networks are used in a lot of industrial sectors such as the domotics[2], agri-

[1]They can be mobile or embedded devices, even invisible, present in almost any type of imaginable object, for example cars, tools, appliances, clothing and various consumer goods.

[2]Also called home automation or home systems.

culture, livestock, logistics, environmental monitoring, construction, public works and infrastructure management and monitoring. Finally are also used in medicine and military applications.
A domotics use for example, that is increasingly became popular, consist in the integration into a single system of one or more personal computers, and in particular of typical consumer electronics such as TVs, audio and video equipment, gaming devices, smartphones and PDAs. In addition, we can expect that all kinds of devices such as kitchen appliances, surveillance cameras, clocks, light controllers, and so on, will all be hooked up into a single DS. Others examples are projects as SIMEA[35] or OPTICONTROL[36], that have the aim to study, design and realize novel sensor network systems and innovative data analysis algorithms, that allow precise profiling and evaluation of the main environment and energetic parameters in buildings. The goal is improving the indoor climate control and reduce energy consumption while maintaining high user comfort and work productivity at modest basic investment and operating costs.

The work presented in this thesis try to implement a wireless network in which every node has the control of a small RGB Light Emitting Diode (LED) device that is used to show a unique color sequence through the whole extension of the network. So two aspects become fundamental for us:

1. The coordination among nodes
2. The real-time nature of the system

We use WSNs as infrastructure for our project. This choice permits to exploit their advantages as for instance the reliability, the multi-hop communication and the adaptability.
The work is made in cooperation with an Engineering office that develop, among other things, chromotherapy devices. The aim of the project is to create a system that is something different from the commercial products that are available in the market today. In fact generally the devices for the color therapy are wired and centralized. Some other systems already uses wireless light devices but are remote controlled, and for this reason the extension of these systems is constrained by the radio range of the controller.

The possibility that a chromotherapy system can inherits all the capabilities of a network infrastructure is the innovative aspect that has driven our work.

The major contribution of this work is the development of a real-time chromotherapy system that lets to choose among some color effects, and to set up parameters as for instance the rate of color changes.
Chromotherapy[3] is based on the fact that certain colors could trigger moods or alter metabolism of the human body. In this method seven fundamental colors of the spectrum is associated with specific healing properties:

1. **Violet** promotes enlightenment, revelation, and spiritual awakening. The Holistic healthcare, for instance, use violet to soothe organs, relax muscles, and calm the nervous system.
2. **Indigo** is also sedative and calming. It is said to promote intuition.
3. **Blue** promotes communication and knowledge.
4. **Green** because it is located in the middle of the color spectrum, green is associated with balance and calm.
5. **Yellow** is a sensory stimulant associated with wisdom and clarity.
6. **Orange** promotes pleasure, enthusiasm, and sexual stimulation.
7. **Red** promotes energy, empowerment, and stimulation.

Is possible to observe that what we implement is a very original application from the others presented until now. A Chromotherapy system, is something radically dissimilar from an application that for instance sense and collect data.
The developed system is therefore able to generate a sequence of different colors in real-time with the possibility to accept instructions from a user through a software interface. So it is possible that the user fixes the color of the network according to his/her wants. A further development of the system could also create real-time colored sequences in relation to external events as for instance sounds or music.
In order to show a unique color sequence across all the network a master node sends broadcast messages containing portions of that sequence. The

[3]Sometimes called color therapy, light therapy or colorology.

wireless sensors that receive these messages repeat them with the purpose to forward the sequence to other nodes. It is a simple mechanism used to flood information in a multi-hop manner. We have also made a study of the timing of the master node messages. It becomes crucial to disseminate correctly the sequence across all the network without loose some packets because for instance are sent too often. So we must introduce an *initial delay* between the generation process and the displaying of each portion of the sequence. And this interval depends from the topology and from the extension of the WSN.
All the sequence parts received by a nodes are replicated trough the external LEDs sending tern of Bytes via the Universal Asynchronous Receiver Transmitter (UART) interface.
Every master message contains in addition to the sequence portion, a reference global time. It has the task to inform the "slaves" nodes when they must start to show the colors contained in the packets. So the synchronization of the network assumes a topic role for this project: RGB devices must be controlled by the sensors with the constraint that the global shade of the color showed in the entire network must change without differences visible by the human eyes. So it is crucial that all the motes act together, scanning the sequence with the maximum precision. Every color of the sequence must be showed by every node always in the same instant equal for all the sensors.
In the literature regarding synchronization algorithms for WSN there are many possible choices that we could implement. For the chromotherapy system was chosen to simplify the Average TimeSync (ATS)[4] algorithm. As first step, ATS was modified removing the skew compensation and so working only with offset compensation. This alternative has a low computational complexity and at the same time grant a sufficient precision for our purpose. In the second step, after a poor initial convergence capacity to a common global clock was verified, was implemented an overlay logical network that creates a hierarchical structure over the WSN. A predefined root node became the reference node in the synchronization process. The other nodes consume received information about the neighborhood time-

stamp according to a hierarchical model. If for instance a node **A** is closer to the root than node **B**, for another neighborhood node **C** that is able to listen messages from **A** and **B** (but not from the root), the informations received from **A** are more trustworthy than the informations get from **B**. This approach ensure a fast convergence of the network to a common virtual reference clock.
The entire system was implemented and tested on a Tmote/TinyOS-2.x architecture in order to verify if it works and what performance we are able to reach.

Contents of the chapters

The structure of the thesis is organized in seven chapter:

- Chapter 2: presents a brief introduction to the WSN. We familiarize with the application fields and the challenge that this technology introduce.
- Chapter 3: describes the Tmote Sky platform, the Tiny Operating System (OS) and finally the NesC program language.
- Chapter 4: presents the most used synchronization algorithms for WSN included the ATS one. Is also described the algorithm implemented in our work, the convergence problem and the approach to fix it.
- Chapter 5: explains the performance of the implemented synchronization algorithm.
- Chapter 6: describes the generation of the color sequence, the diffusion of it across the network and the way of how the colors are displayed.
- Chapter 7: explains briefly the implemented NesC code and the Java interfaces created to manage and set up the synchronization of the network and the creation of the color sequence.
- Chapter 8: shows the tests results of our work running on a real WSN and the limits of this architecture.
- Chapter 9: presents the conclusion of this work of thesis and the possible further developments.

Chapter 2

Wireless Sensor Network

2.1 Definition and characteristics of WSN

A WSN is a network of small nodes (or motes) with wireless communication capabilities and equipped with sensors. They can pick up data from the environment and process them through an on-board processor. These small devices are widely produced and distributed, and have a negligible cost of production. Each sensor has a limited and not-renewable energy reserve and after it is placed, it must work in autonomy. To obtain as much data as possible even thousands or tens of thousands of sensors are deployed. This type of networks are rapidly spreading because they offer a series of undeniable advantages: mobility, which allows the terminal to move, flexibility and low implementation costs.
However, wireless networks also face some problems. One of these is undoubtedly the characteristics of the transmission medium, which is unique and shared by all connected nodes. The existence of a single channel necessarily limits the maximum number of user that can utilize the service simultaneously. Similarly, the presence of more users leads to a reduction in transmission speed. In fact the capacity of the transmission channel must be shared between everyone who are using it.

There is also to consider the problem of security in case of absence of specific controls, it is easy for an attacker to intercept information transmitted in the ether or to access services without authorization. We should also

consider that the communication quality can also be influenced by external factors, such as electromagnetic interference and mobile obstacles. Finally, the energy consumption of radio transmission equipment is typically higher than wired one.
Each device has a control module, a communication module and one or more sensors that allow to create large networks that are able to communicate with each other through communication protocols developed for this purpose. The sensor networks can significantly improve the quality and the fidelity of information: for example providing real time data from hostile environments and reducing the cost to collect them. A WSN is only a part of a more complex system, called WSN System. It is composed by the WSN, the channel of the communication between the WSN and a database of collected data (that can be even an Internet server), and the interface between the database and the user. A CaRiPaRo project called WISE-WAI [5] is a clear example of what we have just presented.
Schematically a WSN system can be represented as in Figure 2.1.

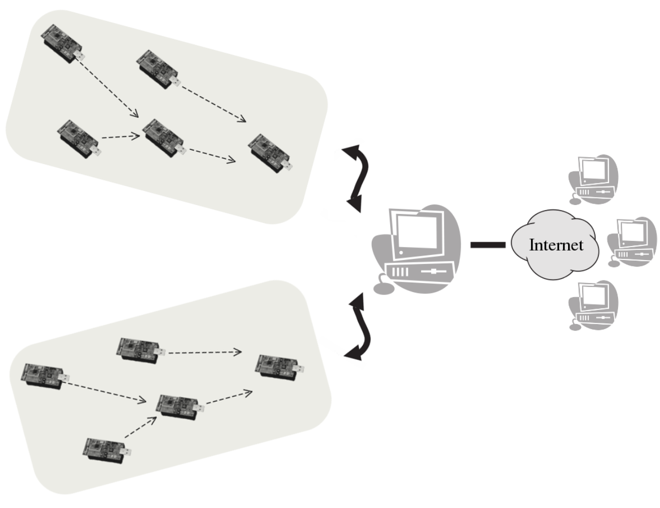


Figure 2.1: *Example of a WSN system.*

It is important to underline that a WSN is also able, through appropriate interfaces, to interact with the user: and we can assume that it is the only way to consider useful the sensing of the environment. By analyzing in detail the components of a WSN, it becomes clear the differences between network nodes responsible to manage the sensors and maintain the network infrastructure, from those who have the task to collect and transmit to the central server the data received from other nodes. Each of these can interact, according to the communication protocol adopted, with other nodes configured in a flat, hierarchical or mesh topology. The primary objective of each node is still to send their data to a collection point within the WSN called Gateway. It has the task to send all the data collected through a wired[1] or a wireless connection[2] to a central system, usually a server, which acts as a database. In the most advanced WSN the data flow and commands may also be transmitted from node to node, or from central server (and so the user) to nodes.

2.2 Architecture of a node

A node consists of four main modules:

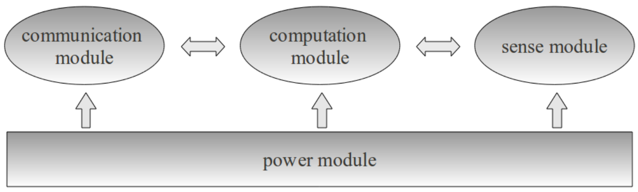


Figure 2.2: *Architecture of a mote.*

- **Sense module**: this is usually composed of two subunits: sensors-actuators and Analog-to-digital converter (ADC). A sensor is capable to detect and measure environment variables, and then transforms them into an electrical signal. Instead actuators are devices capable to act on the environment in different way, for instance actuators can be valves, speakers, as well as mechanical arms.
  The number of sensors and actuators of a node determines its capabilities, but also the cost, the size and the power consumption. The

[1]Ethernet, USB, LAN and firewire are some examples.
[2]For example GPRS, UMTS and HSDPA connection.

ADC is used to translate in digital form the electrical signals generated by sense device. Similarly, this unit is often connected to a DAC, which converts digital signals generated by the microprocessor into an electrical one in order to control actuators.

- **Computation module**: it is an Micro Controller Unit (MCU) that executes procedures and tasks. Microprocessors are often excluded from WSN due to their cost, furthermore microcontrollers consume less power than CPU and motes usually need to execute simple processes. In addition, microcontrollers are suited for WSN, because some parts of them may be turned off when not needed, reducing energy consumption and preserving battery life.
  The MCU is associated with a storage unit, generally integration of RAM and ROM, used to hold data, applications and the operating system. The memory usage involves high energy consumption, thus embedded memory blocks have limited capacity.
- **Communication module**: it connects the node to the network and can be an optical or a Radio frequency (RF) device. Among all node components, the radio chip is the device with the highest energy consumption. To reduce the cost and power utilization well-established and low complexity modulations are used and no high speed transmission are implemented. This module generally works on three different frequency ranges: 400 MHz, 800-900 MHz, 2.4 GHz or Industrial Scientific Medical (ISM) bands[3].
- **Power module**: it is a very important component for a sensor node, commonly made up by commercial batteries such a AA potentially supported by a photo-voltaic module. This last one perform the batteries recharge with the purpose to extend the mote power life.

The particular characteristics of the nodes require the development of platform specific applications with the aims to use less storage space and energy as possible. This implies the need to limit the usage of various interfaces (for example radio, sensors and actuators) and the processor. Even the operating system must have a very small storage image and must

[3]For these bands no government licenses are required

grant low power consumption during the execution of processes.

2.3 Challenges for the WSN

Most of the challenges are consequence of the WSN limited resource availability while others are constantly faced by the majority of the network technologies. The following list outlines the most important challenges that are presented today in the design and implementation of WSNs.

Battery Life

Nodes in the WSN are powered by batteries, and the lifetime of the network depends on the usage of the available energy. In wide wireless sensor networks, it is important to minimize the number of batteries replacement. In order to reach an energy autonomy one or more years long, we must ensure a low duty-cycle operating mode for the sensors. The use of sleep mode for the MCU and the radio becomes crucial.

Scalability

Some applications require thousands or more of wireless sensors. These large scale WSN present challenges not seen in WSN with a few sensors. Algorithms and protocols that work fine on small networks do not necessarily work well in large ones. A typical example is the Dijkstra's shortest path routing algorithm that works well in small networks while is not efficient in large network because of its energy consumption. For wide WSN for instance is preferable to implement location-based routing algorithms, in which the position of each node is known and is used to found paths to transmit information. Similar scalability problems occur for other features of the networks.

Connectivity among networks

WSNs need to be interconnected so that the data reaches the destination to be stored, analyzed, and to take appropriate action. We can imagine that the WSNs can be interconnected with many different network technologies such as phone, Internet, ad hoc wireless networks. This network interfacing is not trivial: new protocols and mechanisms must be designed to connect and transfer data among the WSNs. Normally these connections are realized through gateways, which require new capacity for understand and translate different communication protocols.

Reliability

The wireless sensors are inexpensive devices with a fairly high failure rate. Moreover, in many applications, these devices are launched on an area from a plane, or similar vehicles. As a result, different nodes fail, or alter their normal capability. The reliability of the nodes also depends on the amount of energy available on the node.

Variety

The new WSN are composed of wireless sensors with different capabilities and features. This differentiation requires new algorithms and protocols as for example cluster-based architectures that use devices with more power to aggregate and transmit data on behalf of nodes with limited resources. This strategy, however, include the need of clustering and data aggregation algorithms that are not trivial to design.

Privacy and Security

The privacy and security concerns are topic aspects in the network research field. However, the security mechanisms typically require a lot of resources, which are instead limited in wireless sensors. So there is the need for new security algorithms that require little computational power and energy.

2.4 Network topologies

As explained previously a network of sensors can be reflected in a flat, tree or mesh topology (see Figure 2.3).

The simplest is the flat one, which provides that all but one nodes are

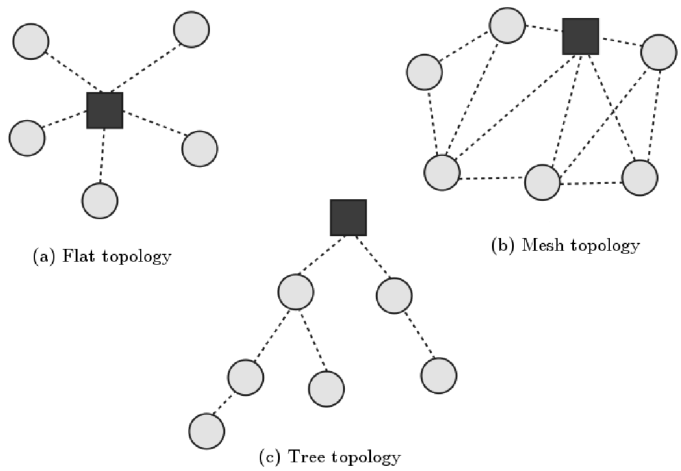


Figure 2.3: *Example of possible WSN topologies.*

equal, and there is a master node that acts as coordinator. It can coordinate the transmissions of the others motes, and has the assignment to transfer data from the WSN to the server. A very common configuration of this type is called *star network*, because all the nodes communicate directly with the master. We can observe that is impossible to create large size networks because they are constrained by the radio coverage range. Moreover networks are not very reliable because the coordinator is a single point of failure.

We can describe a mesh or Peer-to-peer (p2p) networks as structures in which each node can potentially communicate with every other node within its radio coverage area[4]. This topology increase network reliability due to the redundant paths available for the transmission of a message. It is possible, by using routing mechanisms, to determine what is the most energy efficient route, which is the shortest and so on, in order to raise up the network performance. The reliability and robustness provided by multiple paths among nodes requires however the implementation of more complex algorithms.

In the tree topology, as the name suggests, the nodes form a logical tree structure. The messages usually leave a node and climb the tree and reach

[4]If they are all interconnected among themselves the network is a full mesh.

the root[5], which is the data collector and coordinator of the network. For this reason, the nodes have a workload that increases with the decrease of their depth. Compared to the mesh, the advantage of this topology is the reduction of possible communication paths, enabling the development of less complex management systems.

2.5 Application fields

The great versatility of wireless sensor involves a large number of possible applications for WSN in many different scientific disciplines. Some of these applications can be grouped into the following categories:

Health Care Applications The use of sensor networks in this field are aimed to provide an interface for people with disabilities, monitoring physiological data, or for instance to help hospital administration. A well known example is the *CodeBlue* project of the Harvard University [44]. It is also possible to use sensors to identify allergies.

Military Surveillance Sensor networks were born in military research laboratories. The simple and fast deployment, the self-organization and fault tolerance capabilities made WSNs a promising technique for military applications. Possible applications range from monitoring of the allied forces, to the surveillance of the battlefield. Is possible to use a network of sensors in hostile places to recognize and to control the enemy movements, or recognize the type of suffered attacks thanks for instance to chemical, biological, and explosive vapor detection [21].

Environment control In this area, sensor networks could be used for some applications involving monitoring the movement of birds, small animals, insects and study their particular habitat. It can also possible for instance to monitor a forest fire or detect movement in the glaciers. Belong to this sector also the study of natural disaster events such as the volcanic eruptions. In agriculture one of the objectives can be for example to monitor the level of pesticides in the water or the air pollution. An example is

[5] The root is also called *sink*.

shown in Figure 2.4.

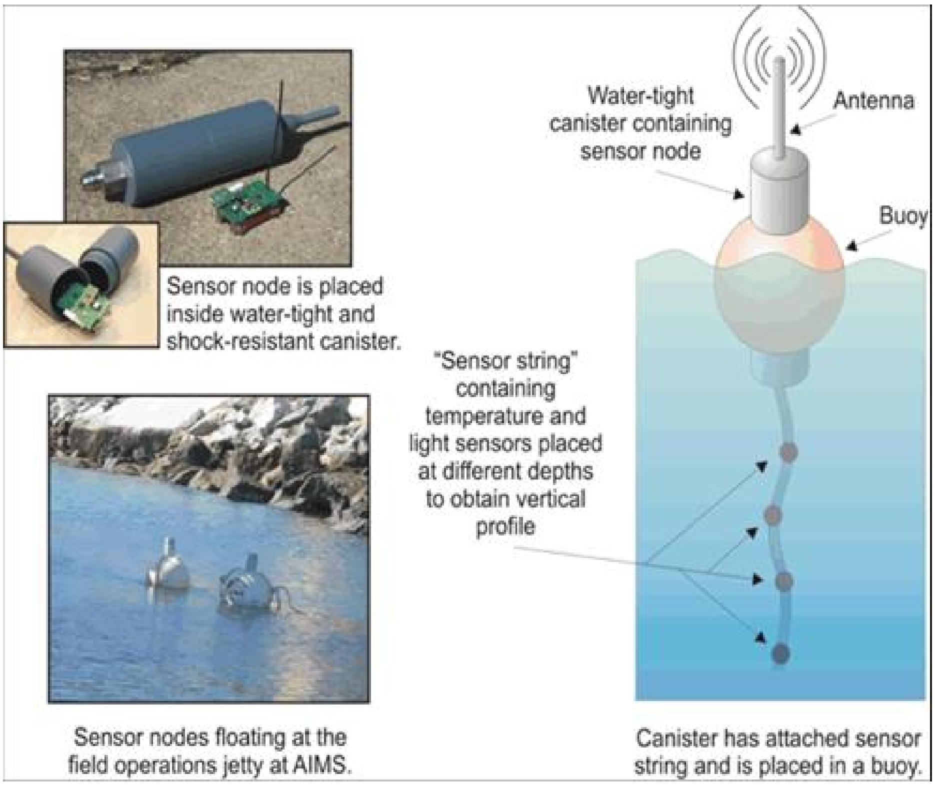


Figure 2.4: *WSN implemented in Nelly Bay, Magnetic Island to control the barrier reef [37].*

Indoor localization and tracking In particular, location-based applications are among the first and most popular applications of WSNs since they can be employed for tracking enemies in battlefield, locating moving objects in buildings (e.g. warehouses, hospitals), and tracking people inside buildings. An example of implemented systems can be found in [11].

Monitoring of industrial equipment The wireless sensors can be applied to industrial tools and machinery to analyze the behavior of components subjected to mechanical stress, improve performance and prevent breakdowns and failures [20].

Commercial Application All the applications with commercial aim belong to this group.

However we emphasize that the quality and potentiality of transmission among the sensors of a wireless network are strongly constrained by the environment conditions in which they are deployed. In particular, the factors that affect significantly the quality of the implementations may be the distance between nodes and the obstacles between them, the power transmission, the electromagnetic interference and finally the power supply problems.

Chapter 3

Tmote Sky, TinyOS and NesC language

3.1 The Tmote Sky

The mote platform Tmote Sky[26] (Figure 3.1) was designed by the developers of TinyOS of the University of California in Berkeley, and produced by MoteIV Corporation. Previous versions are the platform Telos, Telos Revision A and Revision B. Since 2007, MoteIV changed its name to Sentilla[38] and has stopped production and support for these wireless sensors in favor of a new hardware platform designed for Java applications. However, the new platform is backward compatible with Tmote Sky, and also we can still buy mote TelosB, that has the same functionality of the Tmote Sky, from Crossbow[39].

The module incorporates the 16-bit RISC MCU MSP430F1611 from Texas Instruments, which works at a frequency of 8MHz. This microcontroller has 48 KBytes of FLASH memory, 10 KBytes RAM, and 12-bit ADC/DAC.

The low-power, low voltage and low-cost radio chip used by the Tmote Sky for wireless communications is the CC2420 produced from Chipcom. The C2420 is compliant with the IEEE 802.15.4 physical layer and provid-

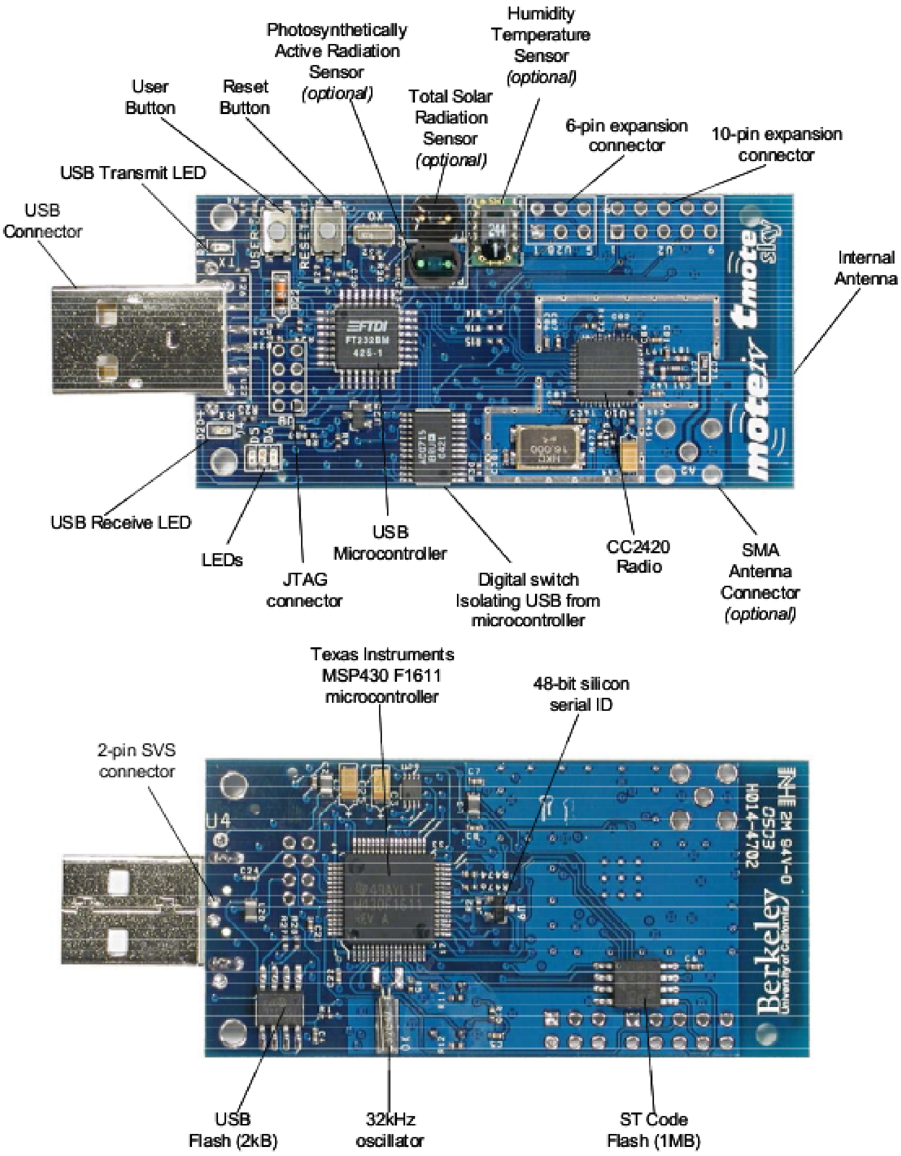


Figure 3.1: *Front and back of the Tmote Sky platform.*

ing the Media Access Control (MAC) layer dictated by the IEEE standard. The transmission is on the 2.4GHz band of IEEE 802.15.4 standard which allow to use channels from 11 to 26. The actual data rate is limited to 250 kbps. Not all features of IEEE 802.15.4 are implemented, and to achieve full compliance, the remaining functions must be implemented by software. The CC2420 provides extensive hardware support for packets handling, data buffering, burst transmissions[1], data encryption and authentication, clear channel assessment, link quality and packet time information.

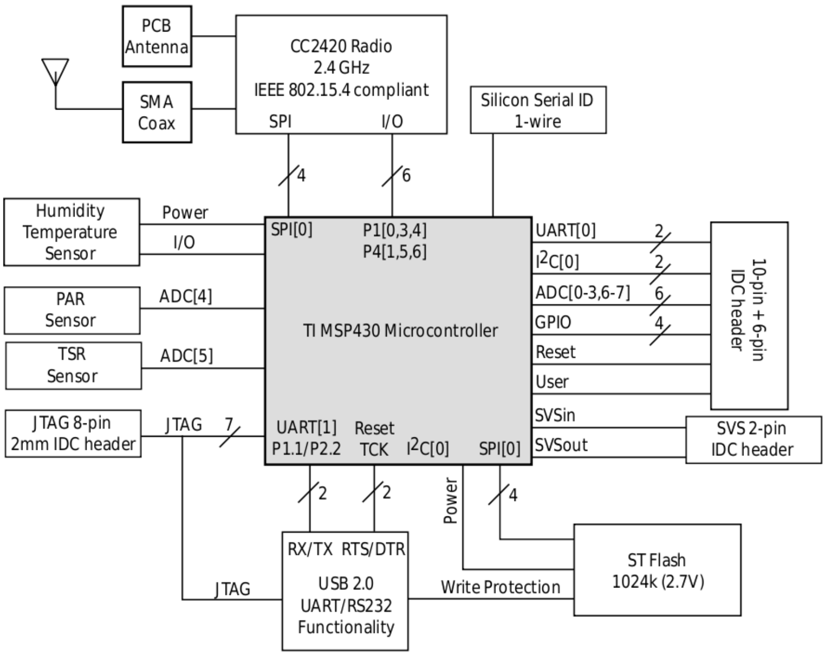


Figure 3.2: *Functional Block Diagram of the Tmote Sky module, its components, and buses.*

As shown in Figure 3.2 the chip is controlled by the MSP430 through the Serial Peripheral Interface (SPI) port, and a series of I/O lines and interrupt. Through the configuration registers can be programmed the reception and transmission approach (for example if Acknowledges are needed or not), the channel, the communication power, and other parameters. The default configuration provides compliance with IEEE 802.15.4. The capability to set the transmission power is a very desirable feature because, as just said, the consumption of the radio chip dominates the total consumption of the mote. It is also possible to know the Received Signal Strength Indicator (RSSI) of every received message; this feature is very useful for some kind of application as for instance localization.
The Tmote Sky can be powered by two AA batteries. The power should be

[1]They are characterized by short transmissions and long inactivity periods.

between 1.8 V and 3.6 V, but must be at least 2.7 V to program the flash memory of the microcontroller or the external flash. When the module is connected to a USB port of a PC can receive power from this interface, in this case the operating voltage is 3 V. Power can also be supplied via pins number 1 and 9 of the expansion connector, or through the terminals dedicated to the battery.
The antenna is an Inverted F Antenna (IFA) and although it has not a perfect omnidirectional pattern, may attain 50-meter range indoors and up to 125-meter range outdoors.
The EEPROM used in Tmote Sky is the M25P80 STMicroelettronics. It is a flash memory that can store 1024 KBytes of data, and is composed of 16 segments, each of 64 kBytes. The flash shares SPI communication lines with the CC2420 transceiver. So care must be taken when we want to read or write on the flash. Typically is implemented a software arbitration protocol for the SPI bus of the microcontroller. To get the energy savings should be limited as much as possible the memory usage.
Tmote Sky module can be equipped with a humidity and temperature sensors produced by Sensirion AG. They may be directly mounted on the Tmote module. The models used are SHT11 SHT15 different in the accuracy of the measurements. Even a light sensor can be mounted directly on the card and provides connections for two photodiodes.

3.2 TinyOS-2.x operating system

3.2.1 Versions

TinyOS-2.x is the natural evolution of TinyOS-1.x, the most popular OS for wireless sensor networks and embedded systems. The name comes from the abbreviation of Tiny Operating System, it is open source and it was developed, in cooperation with Intel Research, by the University of California in Berkeley. At the moment the latest version of the operating system is 2.1.1 that is not backward compatible with version 1.x. This is due to the fact that was made a complete rewrite of the operating system to improve organization and to optimize the use of the resources.

3.2.2 Hardware abstraction

The hardware abstraction of TinyOS 2 generally follow a three-level abstraction hierarchy[27, 34], called the Hardware Abstraction Architecture (HAA):

- Hardware Presentation Layer (HPL) is an abstraction layer placed immediately above the hardware platform that allows us to have the complete control on the underlying hardware such as I/O pins or system registers. This level is hardware-dependent and does not abstract any of the features of the platform, but only masks the control code.
- The Hardware Abstraction Layer (HAL) is placed above HPL and provides higher-level abstractions that are easier to use than the HPL but still provide the full functionality of the underlying hardware. HAL is still hardware-dependent.
- Hardware Independent Layer (HIL) is placed on top of HAL and provides abstractions that are hardware independent. This generalization means that the HIL usually does not provide all of the functionality that the HAL can. HIL components have no HW naming prefix, as they represent abstractions that applications can use and safely compile on multiple platforms. At this level, code optimization is not possible.

3.2.3 Component-base architecture

TinyOS architecture is based on entities called components, in fact it is composed of a lot of small components that application developers could reuse every time they desire. Component-Based Software Engineering (CBSE) is focused on the design and implementation of software systems using components already ready to use. These elements are standardized, independent, reusable, able to adapt to any architecture chosen for the application development.

The component-based systems are easy to assemble, change and enlarge, and so have lower production costs. The component architecture of TinyOS allows the minimization of the necessary code as required by the small memory of the wireless sensors. In fact, when an application is installed on a sensor even an image of TinyOS is compiled together, but it includes only those components of the OS that are strictly necessary for the application execution. For this reason, the software installed on a sensor take up only few Bytes of memory. In addition TinyOS is specifically designed to consider all the constraints concerning the resources of the wireless sensors, first of all the low power energy availability. The libraries of components included in this OS range over network protocols, distributed services, sensor drivers, and data acquisition tools. Obviously all the components can be modified to get customized implementations that are able to solve better specific tasks.

3.2.4 Traits of TinyOS

The main features of TinyOS-2.x are:

Scheduler

The scheduler implements a FIFO policy without preemption. Each task has its own reserved space in the queue and can not be queued more than once if it is already present in the FIFO structure. So to enqueue many instances of the same task, the code that implements this task must call the enqueue command during the execution of itself.
It is possible to develop another kind of scheduler and replace the FIFO one because in TinyOS it is a component and so we can modify it.

Virtualization of resources

In TinyOS for many components was introduced the concept of resource virtualization. This creates an instance of an object that provides the required interface every time it is necessary.
With this approach Virtual abstractions even hide multiple clients from each other through software virtualization. Every client of a virtualized resource

interacts with it as a dedicated resource. All the virtualized instances are then multiplexed on top of a single underlying resource. Because the virtualization is realized through software, there is no upper bound[2] on the number of clients using the abstraction.
This approach simplifies the resource management but it has some negative aspects. For example, a virtualized timer resource introduces CPU overhead from dispatching and maintaining each individual virtual timer, as well as introducing jitter whenever two timers are fired at the same time.

Power Management

All resources of the node, including the microcontroller and the radio chip, provide interfaces to manage their status. In particular TinyOS distinguishes microcontrollers power-management between the peripherals one. The microcontrollers in fact have different states of energy consumption, while the devices have only two states: on and off.

3.3 Network Embedded Systems C

3.3.1 Definition and principal characteristics

NesC is an extension to C language designed to embody the structuring concepts and execution model of TinyOS and optimized for the small amount of resources available in a wireless sensor.
When an application is compiled, the components of TinyOS are included with it and the result forms the entire software of the sensor. Furthermore it is not possible to install multiple independent applications in the same sensor[3]. In NESc there is neither dynamic memory allocation nor pointers to functions. This approach, is not very flexible, but allows a significant energy and memory saving and software robustness. Moreover, all the re-

[2]Except for the memory and the efficiency constraints.
[3]We must underline that in order to overcome this constraint, researchers of the University of Padua have designed a special protocol called *SYNAPSE* that is able to reprogram a WSN using Fountain Codes [22].

sources requests and the call graph are already known at compilation time. Finally, it is thus guaranteed a better generation and analysis of the code. The principles of nesC and TinyOS are similar, so the next paragraphs are concepts that are valid for both.

3.3.2 Interfaces and components

In CBSE each component is an independent part of the application software. Each component is defined by two parts: the first specifies the interfaces provided and used by the component while the second represents the internal implementation.
Interfaces are bidirectional structures used by components to communicate with each other. A single component may use or provide multiple interfaces or multiple instances of the same interface. The interfaces of a component are its access points.
Each interface specifies two type of functions supported by the component:

1. **Commands** are functions that must be implemented by the component that provides that interface.
2. **Events** are functions that must be implemented by the component that want to use the interface.

So a component that implements an interface must provide a set of implemented functions (commands) and requires that the component uses this interface implements another type of functions (events) that are invoked upon the occurrence of certain events.
In fact, the component that supplies an interface must only notify events, but what is necessary to do after the event must be implemented by which are using the interface.
The command *Signal* is used to notify an event.

Typically, the commands are called from “up to down” or to be more precise from an application component to a component closer to the hardware, while the events are reported upwards.

This structure is fixed for each component and highlights the relationship with the features of the physical components of the sensor node, so each component has some functionality and can generate events that must be managed.

3.3.3 Modules and configurations

The programs consist of components that are assembled together[4] to make up the whole application. So it can be represented as a graph of components. Each component consists of two elements, a module and a configuration.

The purpose of a module is to define the logic of a component, perform operations, implement interfaces, and use other components. Whereas the configuration aim is to assemble a component with other components it uses (wiring).
A NesC application is made up of two files, one for the module and one for the configuration. Each module or configuration file, has two different sections: one for the component specification and one for the implementation. The first of them[5] contains a list of elements, which can be an interface the component provides, or an instance of another used component. To utilize an element is used the keyword *uses*, while to provide an element is used the keyword *provides*.
The implementation section[6] of the module contain the real implementation of the component functionalities, while for the configuration it contains the wiring directives.
Every NesC application is always characterized by having a configuration component that serves as the root node of the program structure.

[4]The connections among elements of different components are also called *wiring*.

[5]The section for component specification is created with the construct *module [nome_mod] {...}* for modules, and *configuration [nome_conf] {...}* for configurations.

[6]It is created with the construct *implementation {...}* for both the module and the configuration.

3.3.4 Execution Model

The NesC code can be divided in two classes[28]:

- **Synchronous Code**, code (functions, commands, events, tasks) that is only reachable from tasks;
- **Asynchronous code**, code that is reachable from at least one interrupt handler.

A scheduler for NesC can execute tasks in any order, but must obey to the run-to-completion rule[7].
Instead, if a FIFO scheduler is executing any code, when the system signals an interrupt, the interrupt handler code is executed immediately suspending any synchronous code that was previously running.
To avoid that the execution of code is suspended, we must use the *atomic* statement. In fact this approach ensured the execution of all the operations contained in the atomic block.

The synchronous code can be the body of a **command**/synchronous **event** or code executed in tasks. Asynchronous code is instead the interrupt routines.
A task is an independent locus of control defined by a function of storage class task returning **void** and with no arguments [28]. It is posted (with the **post** statement) for a later execution of a portion of code. The **post** command programs task execution by inserting it into a FIFO queue[8] and then returns immediately. The Scheduler execute tasks in a particular order; the executing task can not be suspended by any other task. So tasks have all the same priority, and among them are non-preemptive. Because tasks are not-preempted and run to completion, they are atomic among themselves, but are not atomic if an interrupt occurs. A task is implemented when a component has to perform a job which does not have to be done at the moment of its invocation.

[7]The standard TinyOS scheduler follows a FIFO policy.
[8]If for example we are using the default scheduler.

The **function** is another NesC statement whose code is synchronous. It is defined within a module and can only be used by this module to perform internal operations. The difference between a function and a task is that when a function is invoked, its instructions are immediately executed without delay. The function is therefore a method to perform a short internal routine.

Although non-preemption eliminates data races among tasks, there are still potential *race condition*[9] and *data race*[10] between synchronous and asynchronous code. These problems are detected and reported when the software is compiled. The compiler also reports a compilation error for any synchronous *call* command, or synchronous *event* notification, within asynchronous code. This happens because any code that start from asynchronous code is also asynchronous.

3.3.5 Split-phase operations

All operations that has long latency are optimized with the split-phase technique. It is based on the separation between command that request something, and event that signal the satisfaction of a previously request (see Figure 3.3). Generally interface **commands** are requests to perform a task; if the **commands** is split-phase, the control returns immediately to the caller program. An event is raised (and signaled to the caller) only when the completion of this **command** is done. The split-phase code is often more verbose and complex than sequential one, however, has some advantages. For example this method reduce the use of the *execution stack*, and make the system more responsive.

[9] A race condition occur when the system's work depends on the order in which code sections are executed. A not valid execution order can involve a not consistent system.

[10] Is is a particular case of race condition that occur when data are read and written from two different entity without access control.

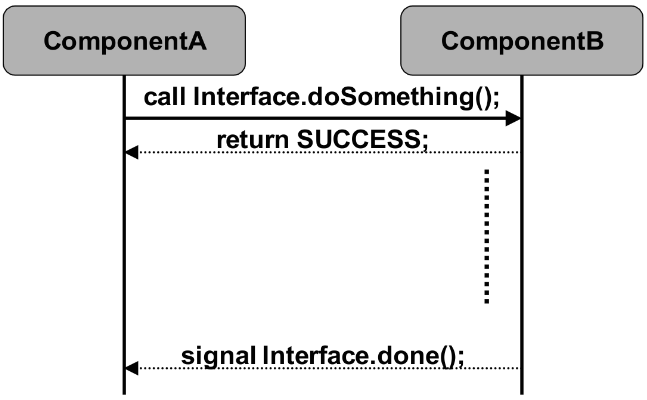


Figure 3.3: *Scheme of a split-phase operation.*

Chapter 4

The overlay-based synchronization algorithm

4.1 Clocks and synchronization

The synchronization is an important aspect of a DS like WSN. For example collect environmental data from a wireless sensor network without any time references typically does not carry real information. The clock of computers and other devices is based on a hardware oscillator. This autonomous component can generate a periodic pulse, with no input signal applied. Generally it uses crystal oscillators because they are stable and their costs are low.

Clock Model

A clock essentially measure time intervals. It consists of an ideal counter τ which is periodically incremented. Generally with $\tau(t)$ we intend a reading of this *local clock* made at the instant t. The counter is subject to an unpredictable deviation of the refresh rate. These variations may depend by many factors as for instance temperature, power supply, magnetic fields, voltage, aging, wear. However, alterations remain within certain small limits and can therefore be neglected.

So we can approximate the clock of the node i as:

$$\tau_i(t) = \alpha_i t + \beta_i \tag{4.1}$$

where α_i is the skew of the clock of the node i, and β_i is the offset. The Skew denotes the clock frequency, instead the offset is the distance from a referenced instant t.
Anyway nodes can't calculate the α_i and β_i values because they have not access to a reference timer. However, it is still possible to obtain indirect information about them by comparing the local clock of one node i with respect to another clock j. In fact, if we solve Equation 4.1 for t, in example $t = \frac{\tau_i - \beta_i}{\alpha_i}$ and we substitute it into the same equation for node j we get:

$$\tau_j = \frac{\alpha_j}{\alpha_i}\tau_i + (\beta_j - \frac{\alpha_j}{\alpha_i}\beta_i) = \alpha_{ij}\tau_i + \beta_{ij} \tag{4.2}$$

which is still linear (right side of Figure 4.1) and where α_{ij} and β_{ij} are respectively the relative skew and the relative offset between node i and j.

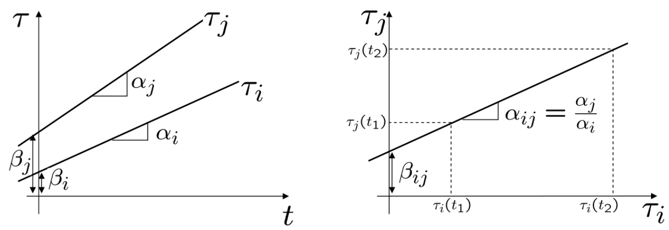


Figure 4.1: *Clocks dynamics as a function of absolute time t on the left, and relative to each other on the right.*

The synchronization of a network with n nodes can be global, but can also be local. In this second case only clocks of a subset of nodes[1] must match. There is another type of clock that is important to define: the software clock. A synchronization algorithm can adjust directly the local clock. However is possible to construct and modify a software clock $\widehat{\tau}$ based on the local clock. The software clock is a monotonic increasing function that transforms the local clock $\tau(t)$ into $\widehat{\tau}(t) = a\tau(t) + b$, with a and b generic parameters.

[1]Generally neighborhood nodes.

Existing algorithm

There is many algorithms in the literature that regard the synchronization aspects. It is due to the fact that time is a crucial subject for the networking sector, and even more for the WSN. We can mention the well known Network Time Protocol (NTP)[29, 9, 10] or the Precision Time Protocol (PTP)[40] algorithm designed for wired networks. But these algorithms are not suitable for wireless network because of their stiffness and even because they are designed with no power saving aims.
In the last years much R&D effort has been spent to develop algorithms for the WSNs. The most important are:

1. Reference Broadcast Synchronization (RBS) [9]
2. Tiny-Sync and Mini-Sync (TS/MS) [13]
3. Time-Sync Protocol for Sensor Network (TPSN) [14]
4. Lightweight Time Synchronization for Sensor Network (LTS) [15]
5. Flooding Time Syncronization Protocol (FTSP) [33]
6. Reachback Firefly Algorithm (RFA) [17]
7. Solis, Borkar, Kumar protocol [19]

Another algorithm that was designed and implemented in the University of Padua is the Average TimeSync [4, 30] one. It is fully distributed and asynchronous, and even has very poor memory and CPU requirements. Its strengths are the adaptability, reliability but over all the great precision that is able to reach. We underline the fact that the possibility to respond to network topology changes is very useful in the WSN world.
It is a consensus[2] algorithm and its principle is to converge to an average time among all the node of the network.
All nodes operate in the same way according to a peer-to-peer architecture, and every node is able to initiate a synchronization session. A node communicates to its neighborhood the local time-stamp with a message that is sent at a fixed rate (called *timesync* period). The smaller is the interval between synchronization messages, the better is the precision. For example

[2] Consensus is a problem in distributed computing that encapsulates the task of group agreement in the presence of faults [45].

ATS with *timesync* equal to 30 seconds is more precise than an ATS implementation with *timesync* equal to 90 seconds.
The ATS does not flood time information from a root node to the leaves. Information is contained in all the network nodes which exchange packets among neighborhoods[3]. Each node changes its software clock to the established consensus value. The main idea of the algorithm is to level all values of the different software clocks to their average.
The diffusion method can reach the global synchronization through the interconnection of synchronized parts of the network. After a few cycles of diffusion, all the node clocks have the same value.

4.2 Average TimeSync description

This algorithm wants to synchronize all the nodes of a network with respect to a *virtual reference clock* that we can represent as:

$$\overline{\tau_i}(t) = \overline{\alpha} t + \overline{\beta} \tag{4.3}$$

Every node estimate the virtual clock using a linear function of its own local clock:

$$\widehat{\tau_i}(t) = \widehat{\alpha}_i \tau_i(t) + \widehat{o}_i \tag{4.4}$$

The goal of ATS is to find the couple $\widehat{\alpha}_i$ and $\widehat{o}_i$ for all the node.

In necessary to underline that to implement ATS on a network, the wireless sensor device must support the MAC-layer time-stamping. In fact when a packet P is sent from i to j, it is assumed that the reading of the local clock $\tau_i(t_1)$ (when P is sent), the packet transmission and the reading of the local clock $\tau_j(t_2)$ (then P is received) are instantaneous. In other words that $t_1 = t_2$.

[3]Is important to notice that the synchronization is done locally under the node point of view.

4.2.1 Relative skew estimation and compensation

Every node i tries to estimate the relative skews α_{ij} with respect to its neighbor nodes j. This is accomplished by storing the current local time $\tau_j(t_1)$ of node j into a broadcast packet, then the node i that receives this packet immediately record its own local time $\tau_i(t_1)$. Therefore, node i records in its memory the pair $\tau_i(t_1), \tau_j(t_1)$. When a new packet from node j arrives to node i, the same procedure is applied to get the new pair $\tau_i(t_2), \tau_j(t_2)$. The estimate of the relative drift η_{ij} is:

$$\eta_{ij}^{+} = \rho_n \eta_{ij} + (1-\rho_n)\frac{\tau_j(t_2)-\tau_j(t_1)}{\tau_i(t_2)-\tau_i(t_1)} \tag{4.5}$$

where the symbol η_{ij}^{+} indicates the new value assumed by the variable η_{ij}, and $\rho_n \in (0,1)$ is a tuning parameter. The algorithm to compensate the skew is very simple, in fact every node stores its own virtual clock skew estimate $\widehat{\alpha}_i$, defined in Equation 4.4. As soon as it receives a packet from node j, it updates $\widehat{\alpha}_i$ as follows:

$$\widehat{\alpha}_i^{+} = \rho_v \widehat{\alpha}_i + (1-\rho_v)\eta_{ij}\widehat{\alpha}_j \tag{4.6}$$

where $\widehat{\alpha}_j$ is the virtual clock skew estimate of the neighbor node j. The initial condition for the virtual clock skews of all nodes are set to $\widehat{\alpha}_i(0) = 1$.

4.2.2 Relative offset estimation and compensation

According to the previous analysis, after the skew compensation algorithm is applied, the virtual clock estimators have all the same skew, and so they run at the same speed. At this point it is only necessary to compensate possible offset errors. Once again, we adopt a consensus algorithm to update the virtual clock offset, previously defined in Equation 4.4, as follows:

$$\widehat{o}_i^{+} = \widehat{o}_i + (1-\rho_o)(\widehat{\tau}_j - \widehat{\tau}_i) = \widehat{o}_i + (1-\rho_o)(\widehat{\alpha}_j\tau_j + \widehat{o}_j - \widehat{\alpha}_i\tau_i - \widehat{o}_i) \tag{4.7}$$

where τ_j and τ_i are computed in the same instant.

4.3 Offset Compensation Algorithm

The first step of this work concerns the implementation of a synchronization algorithm. This aspect is fundamental for the color therapy network that we want to realize. In fact in order to ensure that all the nodes show always the same color, the developed application makes the next mainly steps:

1. A software creates in real-time a sequence;
2. The sequence is sectioned, and these parts are sent over messages. Every message incorporates even the initial global time at which start to show the portion contained;
3. When a node receive this kind of message, it processes it, waits the initial global time inserted and then starts to show the sequence of colors through its RGB device.

We can understand that all the principal operation done with the purpose to produce the chromotherapy effect are very closely dependent on a common global time. For this reason a method to calculate a virtual reference clock is necessary.
In the selection of the algorithm to implement, the work made by F.Fiorentin [8] was used as foundation. In his thesis was presented all the weaknesses, the strengths, and the performance of the most important algorithms for WSN synchronization. Furthermore, in the analysis of the complexity and performance of the different synchronization methods, was proved that one of the most light is ATS.
Table 4.1 show a comparison between them. The Skew column indicates if the algorithm compensate the skew and the complexity column indicates the number of elaboration made in a network of n nodes by an algorithm that executes m synchronization cycles[4]. Instead the channel column displays

[4]For ATS k is the maximum number of neighborhood of a node. This must be considered because for every neighborhood j ATS needs to store an historical global time pair $(\tau_j(t_{old}), \tau_i(t_{old}))$ for the computation of the skew estimation.

	Skew	Complexity	Channel	Memory	Scalability	Topology
RSB	yes	(mn^2)	$(m + mn)$	$O(n)$	low	yes
TPSN	no	$(4m(n-1))$	$(m + mn)$	$O(1)$	sufficient	yes
TS/MS	yes	$(4m(n-1))$	$(m + mn)$	$O(1)$	sufficient	yes
LTS	yes	$(4m(n-1))$	$(m + mn)$	$O(1)$	sufficient	yes
FTSP	yes	$(2mn)$	(mn)	$O(1)$	high	yes
RFA	no	$(2mn)$	$(m + mn)$	$O(n)$	high	no
Solis et al	yes	$(2mn)$	$(mn(n-1))$	$O(n)$	high	no
ATS	yes	$(m(n+k))$	(mn)	$O(k)$	high	no

Table 4.1: *Comparison among synchronization algorithms.*

the amount of messages that pass through the channel while the memory column shows the memory usage. Finally the remaining two columns indicate if the method has a good scalability and if it is topology dependent.

We have to analyze these results and understand that algorithms with skew compensation are too precise and too complex for our purpose. On the other hand, algorithms that compensate only the offset are generally topology dependent. For these reasons we have decided to simplify the ATS method, that grants low memory and CPU usage, is not dependent by the network topology and is fully distributed.
Our goal is to be able to change the network color, and all the nodes must act together. The application must mask the fact that every node work individually.
Furthermore an individual that are seeing the network color sequence, should not see differences between the turning on of the same tint in two different RGB devices.
This important aspect was considered when we choose the way to synchronize the network. In fact if we suppose that the human eyes can see with a frequency of at about 40 Hz (25 ms), is sufficient, under the synchronization point of view, that our application has a millisecond precision.
The local clock of Tmote Sky is provided by the 32 KHz external crystal oscillator, which has a granularity of about 30μs per tic. As obtained in [4], ATS with a synchronization interval of 30 seconds can reach a precision of

± 10 ticks which are 600μs.
Our specification let us to be less precise. So we modify ATS trying to reduce the complexity even further.
When the nodes are showing the sequence, they should be as coordinate as possible among themselves, but it is difficult because they are load with work if the color rate is high. So the synchronization process should be as light as possible in order to obtain a fast system.

Starting from an implementation of ATS made in [8], we remove the skew estimation and compensation (expression 4.5 and 4.6) and we kept only the offset ones (expression 4.7). The precision of what we obtain is worse than the original method but is enough for our aims. Instead is fundamental that we have less operations to do when a node receive a synchronization message because now the skew computation is miss out. We have reduced the size of the synchronization message of 4 Bytes too, from 23 bytes, now it is made up by 19 Bytes: 17% less[5]. Finally Offset Compensation (OC) doesn't need historical information regarding neighborhood, so the memory usage is reduced from $O(k)$ to $O(1)$.
In summary, the only offset compensation involves a continuous resynchronization of the network nodes with a period proportional to the required accuracy. Unlike many other techniques that tend to the maximum precision, in OC the communication and computation needs for the synchronization of the single node were significantly reduced by taking advantage of the relaxation of the constraint of accuracy.

4.3.1 Convergence problems

In both the ATS and OC implementation, we have discover that in some particular cases, after the synchronization was started, the convergence of all the nodes to a virtual reference clock asks a very long time period which we can consider unacceptable.
In Figure 4.2 we show a test in which after at about 23 minutes of execu-

[5] In Subsection 6.1.2 we show that our implementation has removed another Byte from the synchronization message.

tion, OC algorithm has still a maximum pairwise synchronization difference among nodes of about 10.000 ticks.

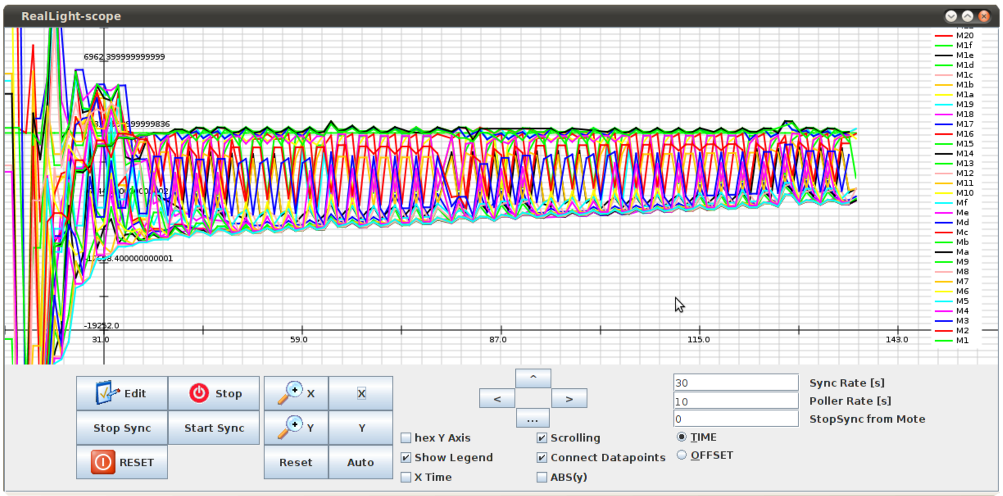


Figure 4.2: *An example of long initial convergence. The graph show a polling interval of about 23 minutes.*

As we can see, it seems that two portions of the network were synchronized locally at two different sub-global reference clocks. One of these network portion is visible on the topside of the graphic, while the other is on the downside. The nodes between them, that "jump" continuously from side to side, were not able to reduce the time gap. The situation presented can take even some hours to converge, and this is too much time. We cannot wait hundreds of minutes before start the chromotherapy effect because of the synchronization. Even more in an application that has commercial purposes it should be avoided.

We have just said that every node of the network sends a synchronization message every some seconds[6]. When a node is powered on, it begins to transmit this message starting from a randomly chosen instant t_i^{start}. For an entire WSN of N nodes we can define the sequence $T_{start} = [t_1^{start}, t_2^{start}, t_3^{start}, ..., t_N^{start}]$ as the set of all the t_i^{start} for $i = 1, ..., N$. A par-

[6]This interval can be defined by the user through a Java interface, and is called *timesync*.

ticular case of T_{start} can produce the situation in example. The algorithms OC and ATS as designed cannot prevent this scenario a priori.
So is not possible to forecast in which order the nodes will synchronize themselves, and is not hence possible to avoid that, in particular cases, some part of the network is going to synchronize locally without great influence[7] of other nodes.

[7] The low influence is caused by an unlucky sequence of starting times chosen by the different motes composing the WSN.

Chapter 5

Color sequence dissemination

The second part of the thesis concerns the design of a mechanism to permit the dissemination of a particular color sequence on the entire wireless sensor network. The fundamental peculiarity of this sequence is that it is not known a priori, but is generated contextually to its diffusion.
As next step we develop a method to manage the information regarding the colors to show, and the UART communication with the external RGB device.
All the mechanisms described are designed to run on an already synchronized WSN.

5.1 Sequence generation

The sequence that we want to compose is just a succession of colors that should create a particular visual effect. The various color shades change at a fixed rate that we call r_c. So, assuming a finite sequence, we can formalize it as

$$S = \{C_1^{t_1}, C_2^{t_2}, C_3^{t_3}, ..., C_n^{t_n}\} \qquad \text{with } n \text{ finite number} \tag{5.1}$$

where C_i is the i-th color tone of S that must be shown at the instant t_i.
The rate is therefore $r_c = \frac{1}{t_{i+1} - t_i}$ for i=1,...,n-1.
If we suppose that S is known a priori, is clear that is very trivial to create the color therapy system. In fact is only necessary to create S off-line and put

it in the software code before the compilation and the motes programming phases.
In our case, the list of colors is created run-time just as it must be shown. For this reason is possible to try to communicate every single color C_i to all the nodes of the network, but it is an inadequate method for our project aims. As a matter of fact, if r_c is high, the network traffic increases too much and the collisions can prevent the communication. Now, if we suppose for example that we entirely create S, and then we communicate it to all the nodes of the WSN, we obtain also a bad solution.
So the communication process necessarily introduce a time leg between the sequence generation and the instant in which S is displayed.
In order to limit this delay, the only reasonable possible way consists in the fragmentation of the sequence. As soon as a portion is created, it must be sent to all the nodes of the network. Finally they show the colors contained. So if the sequence is divided into f parts, Equation 5.1 becomes

$$S = \{S_1^{\widehat{t_1}}, S_2^{\widehat{t_2}}, S_3^{\widehat{t_2}}, ..., S_f^{\widehat{t_f}}\} \tag{5.2}$$

where $S_i^{\widehat{t_i}}$ is the i-th fragment of S whose first color must be shown at the instant $\widehat{t_i}$.
Because all the $S_i^{\widehat{t_i}}$ contain a fixed number m of colors, then:

$$S_i^{\widehat{t_i}} = \{C_{l+1}^{t_{l+1}}, C_{l+2}^{t_{l+2}}, C_{l+3}^{t_{l+3}}, ..., C_{l+m}^{t_{l+m}}\} \tag{5.3}$$

where $l = [(i-1)m]$. We observe that thanks to this method, when we communicate a sequence portions, we only need to know all the colors of $S_i^{\widehat{t_i}}$, the value of $\widehat{t_i}$, and the rate r_c. In fact the instant $t_{l+1} = \widehat{t_i}$ and all the others $t_{l+2}, ..., t_{l+m}$ can be calculated in this way:

$$t_{l+j} = \widehat{t_i} + \frac{1}{r_c}(j-1) \qquad \text{with } j = 2, ..., m \tag{5.4}$$

The next section wants to present how we can inform all the nodes of the WSN about the entire color sequence[1].

[1]a scheme of the behavior of the chromotherapy system is presented in Appendix B

5.2 The multi-hop sequence communication

Supposing to have a node M that knows all the $S_i^{\widehat{t_i}}$. If M sends all the S portions into different packets through its radio interface, only its neighbors can receive these information. In particular only the nodes that are located the radio range of mote M. For these reasons, if the network is large, and the M radio is not able to cover all the distances among nodes, we must use a multi-hop communication approach to flood the network with S.
Thanks to this method, communication between M and another node is carried out through a number of intermediate nodes whose function is to relay information from one point to another.

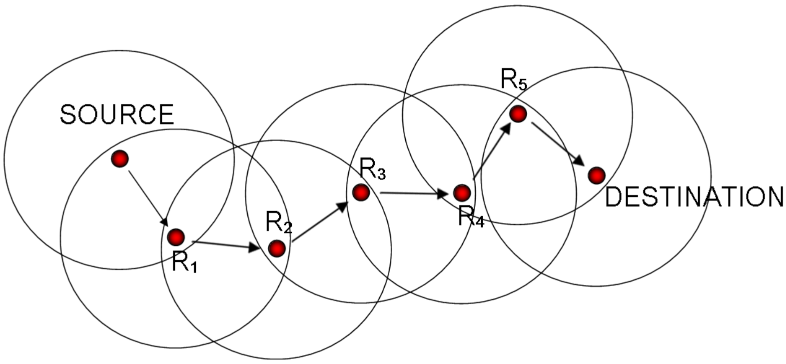


Figure 5.1: *Example of a multihop communication.*

In fact as shown in Figure 5.1 the source node cannot send directly a message to the destination, but the delivery is only possible passing through a path along some repeater nodes (R_i).
So if M wants that S arrives to all the motes of the network, every node that receive the packets must repeat it to its neighborhoods in order to forward the information. With this multi-hop methods we can distribute the sequence in all the WSN.
The flooding of the information must be done paying attention to one common threat: the network collapse. As a matter of fact, even if a node A receives the same message T more than once, T must be forwarded from A only ones. If this control is not implemented, the network communication inexorably crashes.

5.3 Communication through the UART pins

5.3.1 Description and configuration of the interface

At a r_c rate, all the colors of every $\widehat{S_i^{t_i}}$ are sent to an external device. This task are performed via UART interface. The Tmote Sky has two expansion connectors and a pair of on-board jumpers that may configured so that additional devices (analog sensors, LCD displays, and digital peripherals) may be controlled by the Tmote Sky module [26]. In Figure 5.2, we presented the expansion connector we used.

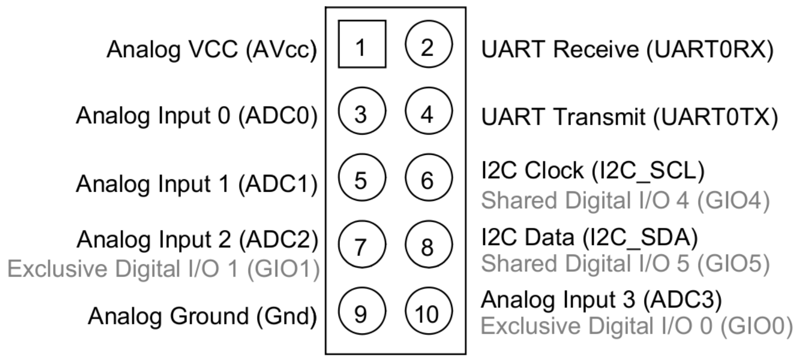


Figure 5.2: *Functionality of the 10-pin expansion connectors. Alternative pin uses are shown in gray.*

Through the PIN number 4 (and the ground of PIN 9) we send out the colors of the sequence S. In fact every colors of Equation 5.1 can be represented by

$$C_i^{t_i} = \{R_i, G_i, B_i\} \tag{5.5}$$

where R_i, G_i and B_i are respectively the value of the red, green and blue components of C_i. One byte is used for each one of these components.
To send the three byte via UART we only transmit each individual bits in a sequential fashion. At the destination, the RGB device re-assembles the bits into complete bytes. Each byte can be sent as a start bit, an amount of 8 data bits, an optional parity bit, and one or more stop bits. The start bit (a 0 bit) signals the receiver that a new character is coming. The next eight bit, represent the byte we want to send. Following the data bits may be a parity bit that we don't have used. The next one or two bits (in our

case two) are always in the mark (logic high, i.e., ‘1’) condition and called the stop bit(s). They signal the receiver that the byte is completed. (Figure 5.3)

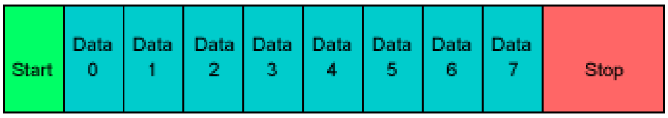


Figure 5.3: *Diagram of a serial byte encoding.*

The motes send the data bits starting from the least significant bit (lsb). The transmission of the data was realized using the *Msp430Uart0C()* components. We have choose to use a baud rate set to 19200 bps, no parity bit, and 2 stop bits.

5.3.2 The arbitration of the USART of the MSP430

Why arbitration?

The arbitration is a method that permits the multiple usage of a resource to different clients. In TinyOS there are three mechanisms (called *abstractions*) for managing shared resources [31]:

- An abstraction is **dedicated** if it is a resource which a subsystem needs exclusive access to at all times. In this class of resources, no sharing policy is needed since only a single component ever requires use of the resource. Examples of dedicated abstractions include interrupts.
- **Virtual** abstractions hide multiple clients from each other through software virtualization. Every client of a virtualized resource interacts with it as if it were a dedicated resource, with all virtualized instances being multiplexed on top of a single underlying resource. An example is the Timer resource. Because the virtualization is done in software, there is no upper bound on the number of clients using the abstraction, barring memory or efficiency constraints. Virtualization generally provides a very simple interface to its clients. This simplicity

comes at the cost of reduced efficiency and an inability to precisely control the underlying resource.

- A **sheared** resource is necessary when many clients need precise control of a resource. Clearly, they can not all have such control at the same time: some degree of multiplexing is needed. A motivating example of a shared resource is a bus.

In our chromotherapy project we need to access to both the radio (SPI mode) and the UART (UART mode) interface switching between them at a very fast frequency. This frequency depends from the r_c value. But as we can see in Figure 5.4 the two interfaces share the USART resources of the MCU.

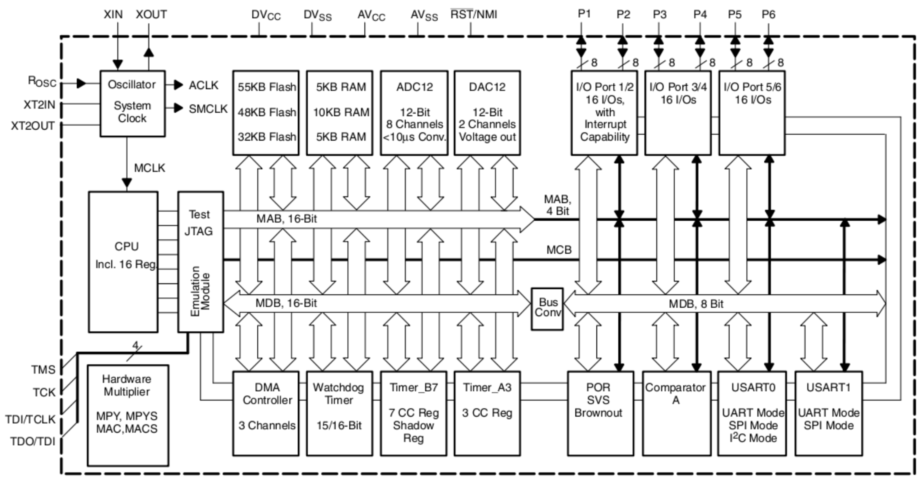


Figure 5.4: *Functional block diagram, of the MCU MSP430F161x series.*

More in detail, the MSP430F1611 microcontroller has two different USART: USART0 and USART1. Both of them are **sheared** abstraction resource. The USART1 is used by the USB interface that is very useful in debugging, so we choose to use USART0 to control the radio and the external device.

Implementation aspects

As consequence of what we have presented until now, we understand that when we use the UART, we can not access to the radio and vice-versa. But in order to be able to receive all the parts of the color sequence S, a node should listen the radio channel as much as possible. On the other hand, a node that must show S at a color frequency for instance equal to 5Hz (period 200ms), must access to the UART interface 5 times per second.
We have only a chance to implement:

- Request the USART0 only when we need to send a RGB color to the external device, and then release it as soon as possible
- The radio must obtain the USART0 resource as much as possible when is not used to send RGB colors.

But who does control and manage the resource access? This work is made in TinyOS by a resource arbiter that is responsible for multiplexing between the different clients of a shared resource, in this case the USART of the MSP430. It determines which client has access to the resource at which time. While a client holds a resource, it has complete and unfettered control. Arbiters assume that clients are cooperative, only acquiring the resource when needed and holding on to it no longer than necessary. Clients explicitly release resources: there is no way for an arbiter to forcibly reclaim it. So it is very important that every time a client need to send a color via UART, it must request the USART and immediately release it.
Furthermore, TinyOS offers even a helpful feature, that consist on the possibility to define a resource default owner. It is a specific client that needs to be given control of the resource whenever no one else is using it. By default the Radio is the default owner of the USART0 module.
In Figure 5.5 we have an example of how the USART0 resource is accessed by the clients. The steps are now explained a little bit in detail:

1. The resource is normally owned by the default owner (gray stripes)
2. When the client C needs the resource USART0, asks it with the *call Resource.request()* to the arbiter

3. When the resource is available for the client, the arbiter signals the happening with an event and reserves the USART0 to C (red stripe).
4. The client can now use the resource, for instance to send a byte through the UART interface
5. After the sending C must release the resource with the *call Resource.release()*
6. The USART0 is now used by the default owner

During the implementation, we had a lot of problems with the arbitration that is not a trivial procedure to realize. Anyway we have always find out solutions. For example, in order to communicate the $C_i^{t_i} = \{R_i, G_i, B_i\}$ to the external device, we must realize a "logic high" for the UART interface. But every time we release the USART0 resource, the UART interface was "turned off", and so its logic became low. This situation was misunderstood from the RGB device convinced in the receiving of a start bit of a new byte. The mistake was resolved with two directives inserted in the initialization of the components composing our code:

```
TOSH_MAKE_UTXD0_OUTPUT();
TOSH_SET_UTXD0_PIN();
```

Another problem happened when we sent the terns of bytes through the UART pins. In fact, after each sending, TinyOS rises the event *async event void UartStream.sendDone(uint8_t buf, uint16_t len, error_t error){. . .}* in which we release the USART resource to the arbiter. But a mistake occurs, in fact we have understood, using an oscilloscope connected to the UART pins, that the third byte was not sent entirely[2]. To solve this problem we force a microsecond wait interval between the rising of the *async UartStream.sendDone* event and the release of the USART. In this way all the three bytes are sent entirely via the UART interface.

[2] It was reported to the tinyos-help community too, but there is not still solution.

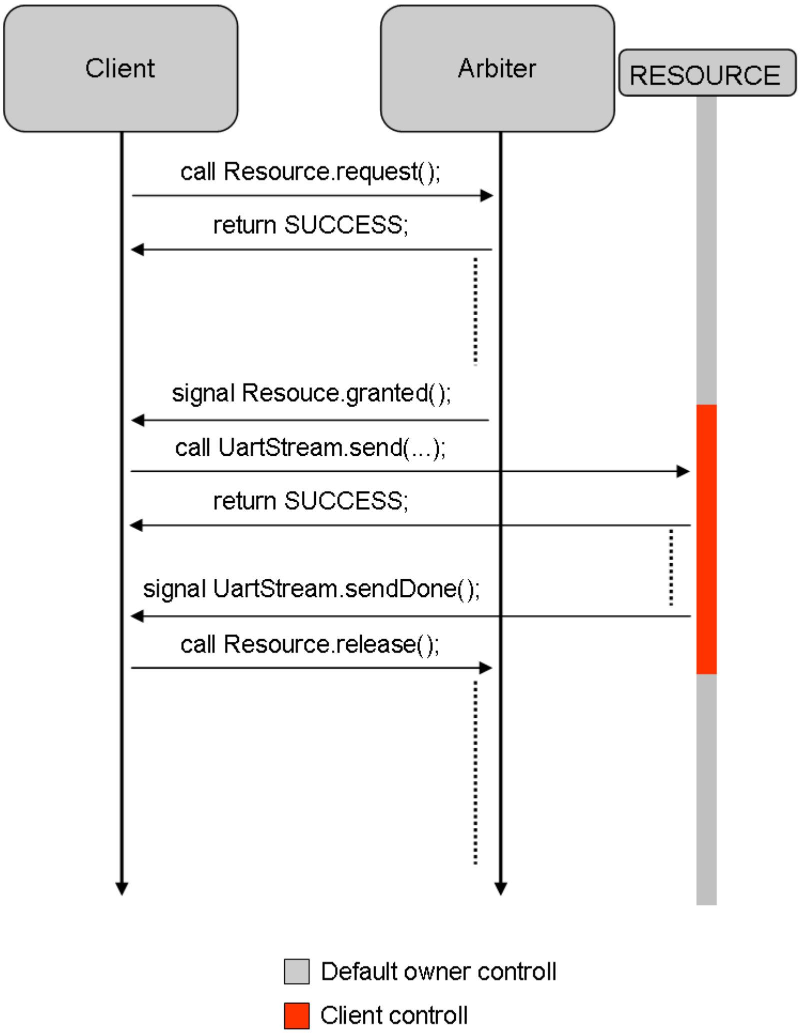


Figure 5.5: *Schematic description of how a client obtain and release a resource.*

Chapter 6

The software description

The system software consists in four NESc components which form the core of the project, however was necessary to build an entire suite of other Java applications to realize the synchronization of the nodes, the management of the overlay logical network and the realization of the chromotherapy effect. In fact the final implementation consist on a set of interdependent programs. Furthermore the software package allow also the collection of information necessary for monitoring and checking the correctness of the network behavior.
All the code was written for mote Tmote Sky, but with some changes can run even in other devices.
Thanks to the CBSE nature of NESc/TinyOS, we can present first the synchronization software, and then the chromotherapy components. This choice involve a clearer explanation of the code.

6.1 The synchronization software

The architecture (Figure 6.1) of the synchronization code are made up by three different actors:

- A node called poller, or base station, generates periodic radio requests to control the different parameters of the synchronization protocol. It must also receive the answers provided by each client. All these

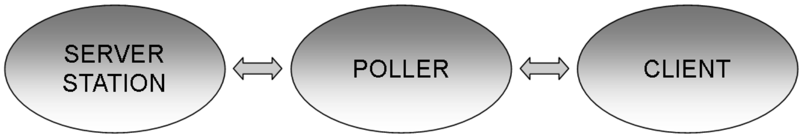


Figure 6.1: *Synchronization actors.*

information are sent via USB to a server. The poller has also to receive the overlay structure configuration parameters from the server and send this message to all the other nodes.

- An amount of nodes called Clients perform the synchronization protocol. The protocol was developed as an independent component that provides synchronization services through an appropriate interface. In each client run a software that allow:
  - to receive the requests generated by the Poller
  - to obtain the synchronization information by using the synchronization component
  - to transmit to the Poller the information collected
  - to receive the overlay structure parameters and modify itself as consequence.
- A PC running a Java application that has to retrieve, save and process all the information received from the poller. It let to the user to change the parameters modifying the overlay structure configuration.

6.1.1 Packets format

All the elements of the architecture described above communicate with each other exchanging packets. Each NesC component has to define the structure of the packets that want to handle. It is necessary to be able to identify the correct incoming packets and to be able to access to their fields (listed in the structure).
The written code uses a lot of different packets. So multiple services use the same radio to communicate. TinyOS provides the Active Message (AM) layer to multiplex access to the radio. The term “AM type” refers to the field used for multiplexing. AM types are similar in function to the Ethernet

frame type field, IP protocol field, and the UDP port in that all of them are used to multiplex access to a communication service [43].
To define a packet we can use parametrized interface where the parameter is the value of the field "AM Type" of the packet. An example is reported:

```
implementation {
    ...
    components new AMSenderC(AM_BLINKTORADIO);
    ...
}
```

This permit to write comprehensible code mode easily. In fact we avoid to use a single component for all the received and sent messages. In TinyOS 2.x, was introduced the standard message buffer *message_ t*. The message_t structure is defined in **tostypesmessage.h** as:

```
typedef nx_struct message_t {
    nx_uint8_t header[sizeof(message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(message_footer_t)];
    nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

The *headers*, *footers* and *metadata* fields cannot be accessed directly but through the appropriate interfaces. The data field of *message_ t* stores the packet payload. It is TOSH_DATA_LENGTH bytes long. The default size is 28 bytes. A TinyOS application can redefine TOSH_DATA_LENGTH at compile time with a command-line option to ncc:

```
-DTOSH\_DATA\_LENGTH=x
```

6.1.2 Code porting

The implementation of ATS from which we start to develop our system, was written in TinyOS version 2.0.2. In this moment the last tinyOS version is 2.1.1. A first problem found in our work was to be able to compile the ATS code written from Fiorentin's thesis [8]. In fact some very important

functionality offered by the old version are now deprecated.
Version 2.0.2 provided particular procedures to use MAC-layer time-stamp when messages are received and sent. The system had a mechanism that was able to report the event Start Frame Delimiter (SFD) and record the relative local time. This event corresponds to the transmission/reception of the first bit of an input/output packet. So for each message *M* received from a node, was possible to detect the local time when the first bit of the message was received. This value of time, called time-stamp, was stored automatically in a 16-bit field of the arrived message. Instead when a message was sent, tinyOS 2.0.2 offered the opportunity to perform a piece of code when the SFD event occurred. In this situation Fiorentin's code changed a field of the message to send, and in particular it included in the transmitted message the time value correspondent to the generation of the SFD event. This method was adopt to obtain the MAC time-stamp of inbound and outbound messages.
The developers of TinyOS understand that the SFD interrupt handler was exposed by the radio stack as an asynchronous event. This solution was problematic, because higher-level application components that wired the interface containing this event could break the timing of radio stack due to excessive computation in interrupt context. So with version 2.1.1 was introduced a new message component: the *CC2420TimeSyncMessageC*. This last one, through the interface *TimeSyncPacket*, provides two new command:

- **eventTime**: This command should be called by the receiver of a message. The time of the synchronization event[1] is returned as expressed in the local clock of the caller. This command must be called only on the receiver side and only for messages transmitted via the *TimeSyncSend*[2] interface. It is recommended that this command be called from the receive event handler. In other words this command permits to obtain the values of the local clocks of the sender and receiver referred to a particular event.
- **isValid**: It returns a boolean to be aware if the value returned from

[1] It is a parameter of the packet which holds the time of some event as expressed in the local clock of the sender.

[2] Even this interface is provided by the *CC2420TimeSyncMessageC* component.

the *eventTime* command is trusted. Under certain circumstances the received message cannot be properly time stamped, so the sender-receiver synchronization cannot be finished on the receiver side. In this case, this command returns FALSE. This command must be called only on the receiver side and only for messages transmitted via the *TimeSyncSend* interface. It is recommended that this command be called from the receive event handler.

With these commands we can benefit of the MAC time-stamp without the control of the SFD event as was in the previous version of the operating system. TinyOS 2.1.1 became even more stable and reliable under this point of view.
Furthermore, thanks to the new component *CC2420TimeSyncMessageC*, in our implementation of OC we removed from the synchronization message structure also a superfluous Byte used to menage the time-stamp information.
After some tests made when the entire chromotherapy project was implemented, we found that occasionally the *isValid* command does not work properly. So even if it return the *TRUE* value, saying that the received packet is ok, it was not so. The packet was probably malformed and the value returned from the *eventTime* command was abnormal. In this situation, the node which is processing the packet, is not synchronized any more. In order to solve this problem[3] a simple control is made even if the *isValid* command return positive: the value returned by the *eventTime* command can't differ too much from the local clock value, else the packet is ignored. This check drops the abnormal values returned from the *eventTime* command.

6.2 The color sequence control software

The architecture of this part of the software (Figure 6.2) is similar to the synchronization one. Anyway we must underline that the *master* node is totally independent from the *root* one. So they could be two different nodes

[3]It was reported to the tinyos-help community too, but there is not still solution.

connected to two different stations. This choice was done to improve the modularity and the flexibility of the entire system.
The component that realize the chromotherapy effect must be used only on a synchronized network.
In Appendix A is shown an example of how the color sequence is diffused in the entire WSN.

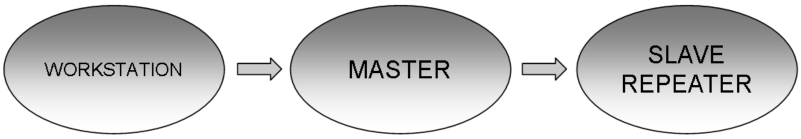


Figure 6.2: *Architecture of the colors sequence management software.*

Chapter 7

Testing of the developed system

The second experimental part of our work was made with the aim to observe the different comportments of the system under various workloads. All tests were performed on nodes Tmote Sky of Moteiv.

7.1 Performed tests

In order to understand the behavior of the system, every time the *master* sends a new RtLightMsg message containing a color sequence part, we gathered from each *slave-repeater* mote these information:

- The number of the received sequence portion. It is used to understand how many packets are lost from each mote.
- The local time instant of the message reception, called r instant.
- The global time estimation of r, called n instant.
- The global time instant at which the motes must start to show the portion contained, called t instant.
- The difference $t - n$, called d.
- The number of millisecond that a node wait before the message repetition, called w value. Its aim is to reduce the collision of the messages. This value is retrieved to understand more precisely the delay introduced from each hop in the flooding process.
- The local time instant referred to the *call TimeSyncAMSend.send()*

of the repetition of the RtLightMsg, called s instant.

- The local time instant of the *TimeSyncAMSend.sendDone* event referred to the repetition, called *sd* instant. The difference $sd - s$ is used to evaluate the responsiveness of the mote.
- The local clock value referred to the processing instant of the first color of the sequence part, called *d1*. The difference $d1 - (r + d)$ is used to evaluate the precision of the mote.

We collect the data thanks to the TinyOS *printf* library. It provides a terminal printing functionality to TinyOS applications through motes connected to a PC via their serial interface. Messages are printed by calling *printf* commands using a familiar syntax borrowed from the C programming language. An example of the obtained files follows:

```
# 5 r 54964499 t 55157523 n 55098447 d 59076 w 23 s 54965896 sd 54966714 d1 55023557
# 6 r 55099191 t 55291013 n 55233138 d 57875 w 20 s 55100454 sd 55101037 d1 55157030
# 7 r 55231549 t 55424050 n 55365496 d 58554 w 23 s 55232867 sd 55233580 d1 55290053
# 8 r 55363152 t 55556850 n 55497099 d 59751 w 7 s 55364212 sd 55365054 d1 55422886
# 9 r 55496163 t 55689849 n 55630110 d 59739 w 25 s 55497604 sd 55498605 d1 55555877
# 10 r 55630173 t 55822848 n 55764120 d 58728 w 26 s 55631641 sd 55632642 d1 55688870
# 11 r 55762786 t 55955584 n 55896732 d 58852 w 2 s 55763686 sd 55764528 d1 55821637
# 12 r 55898345 t 56089839 n 56032291 d 57548 w 16 s 55899515 sd 55900112 d1 55955878
# 13 r 56030225 t 56222632 n 56164171 d 58461 w 4 s 56031135 sd 56032182 d1 56088645
# 14 r 56162492 t 56355412 n 56296437 d 58975 w 21 s 56163707 sd 56164521 d1 56221414
```

The files was then processed with a Java parser that creates the relative Comma-Separated Values (CSV)-like files. This step was necessary in order to import easily the files in MATLAB. An example of the file obtained is:

```
5;54964499;55157523;55098447;59076;23;54965896;54966714;55023557
6;55099191;55291013;55233138;57875;20;55100454;55101037;55157030
7;55231549;55424050;55365496;58554;23;55232867;55233580;55290053
8;55363152;55556850;55497099;59751;7;55364212;55365054;55422886
9;55496163;55689849;55630110;59739;25;55497604;55498605;55555877
10;55630173;55822848;55764120;58728;26;55631641;55632642;55688870
11;55762786;55955584;55896732;58852;2;55763686;55764528;55821637
12;55898345;56089839;56032291;57548;16;55899515;55900112;55955878
13;56030225;56222632;56164171;58461;4;56031135;56032182;56088645
14;56162492;56355412;56296437;58975;21;56163707;56164521;56221414
```

The performed tests are several. For each experiment the parameters of the sequence management packet are set as summarized in Table 7.1.

	Colors per packet (N_c)	Color period (T_c) [ms]	Initial Delay (d_{TOT}) [ms]
test 1	10	100	700
test 2	10	150	750
test 3	10	200	1000
test 4	15	100	750
test 5	15	150	1100
test 6	15	200	1500
test 7	20	100	1000
test 8	20	150	1500
test 9	20	200	2000
test 10	20	300	500
test 11	20	300	1000
test 12	20	300	2000
test 13	20	400	2000

Table 7.1: *Parameters setups of the test performed on the developed chromotherapy system.*

All the 13 tests was done on 4 different chromotherapy system network configurations:

1. Linear array network without (W/O) the second retransmission of the sequence portion messages
2. Linear array network with (W) the second retransmission of the sequence portion messages
3. Grid network without the second retransmission of the sequence portion messages
4. Grid network with the second retransmission of the sequence portion messages.

The Linear array network is composed by only 11 motes (master plus one node per hop). The *master* node is placed in the head of the array. Instead the grid network of 35 motes is deployed as a mesh. The number of nodes per hops is:

Hop	1	2	3	4	5	6	7	8	9	10
Number of nodes	2	3	4	5	5	5	4	3	2	1

Table 7.2: *Number of nodes per hop in the grid network.*

We collect the data from only one node belonging to each hop. The node is randomly chosen because we suppose that the nodes at the same number of hops far from the master have similar behaviors.

7.2 Packet loss

7.2.1 Linear Array

We have investigated if the second retransmission of the sequence portion message brings benefits to our system or not. We analyze **test 9**[1] ($N_c = 20$, $T_c = 200$, $d_{TOT} = 2000$) on a linear array network without retransmission. We have seen that some packets were lost starting from the second hop (Figure 7.1). This poor performance is compared to the result obtained in the same linear array with the second retransmission of the sequence part messages. As shown in Table 7.3, the second experiment has no packet loss.

It is important to notice that even in the linear array without retransmission, some hops do not loose any packet. For example the 8-th and 9-th hops doesn't process 19 packets as the 7-th hop. But the loss is introduced by the 7-th one.

[1]This choice was made because our experiments have confirmed that test 9 parameters configuration is generally not problematic for the chromotherapy system.

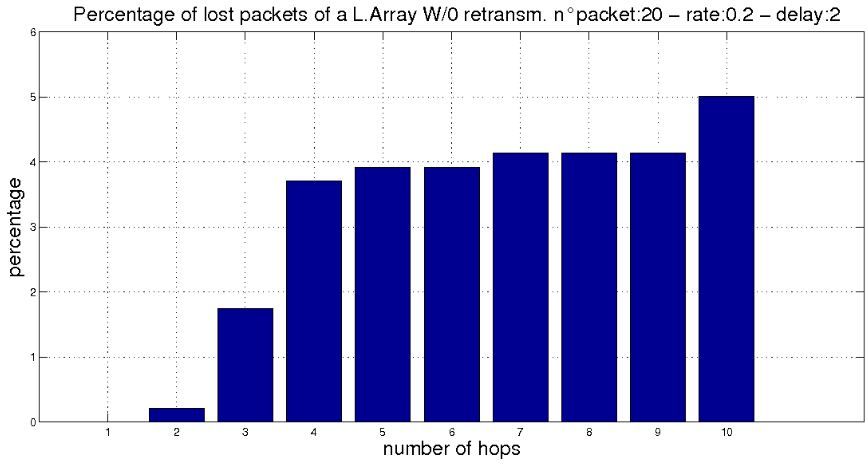


Figure 7.1: *Percentages of lost packets per hop on a linear array without the second retransmission of the sequence portions.*

	L.A. W/O retransm.	L.A. with retransm.
hop 1	0 (0%)	0 (0%)
hop 2	1 (0.22%)	0 (0%)
hop 3	8 (1.74%)	0 (0%)
hop 4	17 (3.70%)	0 (0%)
hop 5	18 (3.92%)	0 (0%)
hop 6	18 (3.92%)	0 (0%)
hop 7	19 (4.14%)	0 (0%)
hop 8	19 (4.14%)	0 (0%)
hop 9	19 (4.14%)	0 (0%)
hop 10	23 (5.01%)	0 (0%)

Table 7.3: *Lost packets on a linear array with and without the second retransmission of the sequence parts.*

Afterwards we studied the number of lost packets of **test 10** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 500$), **11** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 1000$) and **12** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 2000$) on a linear array with retransmission. These three setups have all 20 colors per packet, a color period of 300 ms, but different initial delays introduced from the *master*. Test 10 has only 500 ms of delay, while test 11 has 1 second and test 12 has 2 seconds. The obtained results are presented in Table 7.4.

	test 10	test 11	test 12
hop 1	0 (0%)	0 (0%)	0 (0%)
hop 2	0 (0%)	0 (0%)	0 (0%)
hop 3	0 (0%)	0 (0%)	0 (0%)
hop 4	0 (0%)	0 (0%)	0 (0%)
hop 5	0 (0%)	0 (0%)	0 (0%)
hop 6	0 (0%)	0 (0%)	0 (0%)
hop 7	0 (0%)	0 (0%)	0 (0%)
hop 8	44 (9.22%)	0 (0%)	0 (0%)
hop 9	396 (83.02%)	0 (0%)	0 (0%)
hop 10	477 (100.00%)	0 (0%)	0 (0%)

Table 7.4: *Lost packets on a linear array with retransmission. Comparison of test 10 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 500$), 11 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 1000$) and 12 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 2000$).*

The experiments demonstrate that in a reliable network configuration where no collision can occur, the length of the initial delay is fundamental to grant the sequence flooding in all the extension of the network. If this value is too small, the sequence messages can not reach in time the further hops. In fact for test 10 the mote at the 10-th hop does not receive any RtLightMsg packet.

7.2.2 Grid network

Then we studied the packet loss in the grid network of 35 motes too. In this topology we have more than one node per hop, so collisions can take place. This situation generally involve a greater packet loss in respect to a network where no collision can occur.

First of all we compare the **test 9** ($N_c = 20$, $T_c = 200$, $d_{TOT} = 2000$) results obtained first in a grid without the retransmission of the sequence portions, and second in a grid network that implements this feature. The result are summarize in Table 7.5.

	Grid W/O retransm.	Grid with retransm.
hop 1	0 (0%)	0 (0%)
hop 2	8 (1.75%)	0 (0%)
hop 3	9 (1.97%)	0 (0%)
hop 4	10 (2.19%)	0 (0%)
hop 5	10 (2.19%)	2 (0.43%)
hop 6	11 (2.40%)	3 (0.64%)
hop 7	12 (2.63%)	4 (0.86%)
hop 8	20 (4.38%)	6 (1.29%)
hop 9	24 (5.25%)	13 (2.80%)
hop 10	30 (6.56%)	16 (3.44%)

Table 7.5: *Lost packets on a grid network with and without the second retransmission of the sequence parts.*

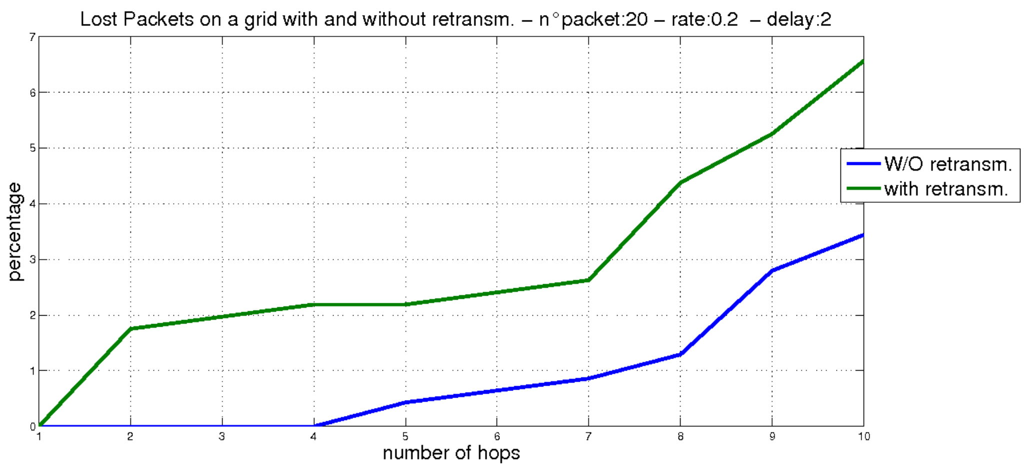


Figure 7.2: *Percentage of lost packets per hop on a grid network with and without the second retransmission of the sequence portions.*

In Figure 7.2 were displayed the behaviors of the two experiments and demonstrates that the retransmission involve a more reliable sequence flooding process. So hereafter we abandon the implementations without the retransmission of the sequence messages.

We can also observe that our system is able to perform a complete sequence displaying at 4 hops further from the root. As reported in the Tmote Sky datasheet [26], the antenna of this kind of devices may attain at about 50-meter range indoors. So if we are able to reach the 4-th hop without sequence packet loss, we can potentially realize a system with a 200-meter range extension. Moreover this result was obtained with a test configuration which realizes a fast color change, in fact the color frequency is 5Hz. 5 colors per second is more than enough for a chromotherapy system.

As for the linear array we try the sequence packet configurations of **test 10** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 500$), **11** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 1000$) and **12** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 2000$) even in the grid network (Figure 7.3).

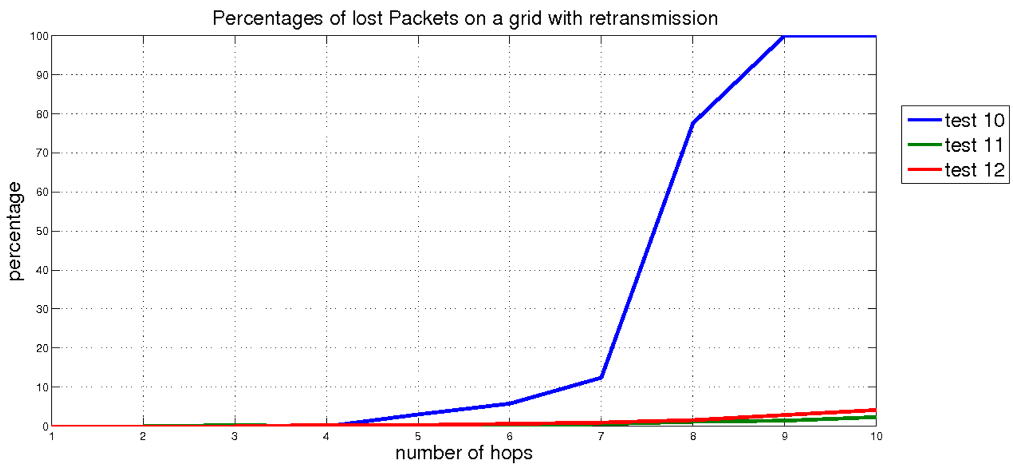


Figure 7.3: *Number of lost packets per hop on a grid network with retransmission. Comparison among tests with different initial delay values.*

Because of the collisions, in these cases more packets were lost if compared to the linear array. Furthermore when the delay is only 500 ms, no one sequence message is able to reach the 9-th hop (one less than the array), and only the 33% of them were received from nodes at the 8-th hop. The collected data are presented in Table 7.6.

	test 10	test 11	test 12
hop 1	0 (0%)	0 (0%)	0 (0%)
hop 2	0 (0%)	0 (0%)	0 (0%)
hop 3	0 (0%)	1 (0.29%)	0 (0%)
hop 4	0 (0%)	0 (0%)	1 (0.32%)
hop 5	11 (3.05%)	1 (0.29%)	1 (0.32%)
hop 6	21 (5.82%)	1 (0.29%)	2 (0.64%)
hop 7	45 (12.47%)	2 (0.59%)	3 (0.96%)
hop 8	280 (77.56%)	4 (1.17%)	5 (1.61%)
hop 9	361 (100.00%)	5 (1.47%)	8 (2.35%)
hop 10	361 (100.00%)	8 (2.35%)	13 (4.18%)

Table 7.6: *Lost packets on a grid with retransm. Comparison of test 10 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 500$), 11 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 1000$) and 12 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 2000$).*

We notice that in a grid network there can be more than one path from the master to a node. So is possible for instance that a node A at 4 hops can lose less packets than a node B at 3 hops far from the *master*. This is due to the fact that A receives the messages that B has lost from another neighborhood different from B. An example of this situation happened between hop 3 and 4 of test 11.

7.2.3 Rising of the packet frequency

There are two possibility to increase the packet frequency:

1. Reducing the value of the color frequency
2. Reducing the number of colors of each sequence portion

The experimental results of the two different approach are now presented.

Rising the color rate

In Figure 7.4 and Table 7.7 are compared **test 11** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 1000$) and **test 7** ($N_c = 20$, $T_c = 100$, $d_{TOT} = 1000$) performed on the grid. These configurations have both the number of colors per packet set to 20, and the delay equal to 1 second. The difference is that test 11 has

a color period set to 300 ms while for test 7 is 100 ms. The packet period from $T_p = 6$s becomes $T_p = 2$s.
Even if the frequency is tripled, test 7 looses in average only the 0.7824% of the packets more than test 11. So even if a lower rate is better, the system has a good response when the colors rate increases.

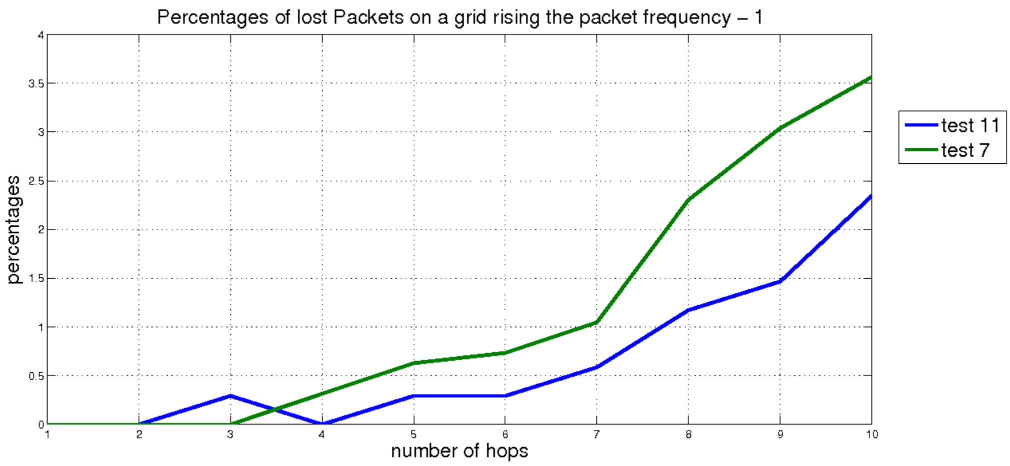


Figure 7.4: *Percentage of lost packets per hop on a grid network rising the color rate. Comparison among tests with different packet frequencies.*

	test 7	test 11
hop 1	0 (0%)	0 (0%)
hop 2	0 (0%)	0 (0%)
hop 3	0 (0%)	1 (0.2933%)
hop 4	3 (0.3145%)	0 (0%)
hop 5	6 (0.6289%)	1 (0.2933%)
hop 6	7 (0.7338%)	1 (0.2933%)
hop 7	10 (1.0482%)	2 (0.5865%)
hop 8	22 (2.3061%)	4 (1.173%)
hop 9	29 (3.0398%)	5 (1.4663%)
hop 10	34 (3.5639%)	8 (2.346%)

Table 7.7: *Lost packets on a grid network rising the color rate. Comparison of test 7 ($N_c = 20$, $T_c = 100$, $d_{TOT} = 1000$) and test 11 ($N_c = 20$, $T_c = 300$, $d_{TOT} = 2000$).*

Reducing the number of colors per packet

In this case we want to understand if the number of colors contained in the packets affects the behavior of the chromotherapy system. So we compare **test 3** ($N_c = 10$, $T_c = 200$, $d_{TOT} = 1000$), **6** ($N_c = 15$, $T_c = 200$, $d_{TOT} = 1500$) and **9** ($N_c = 20$, $T_c = 200$, $d_{TOT} = 2000$). The results are shown in Figure 7.5 and Table 7.8.

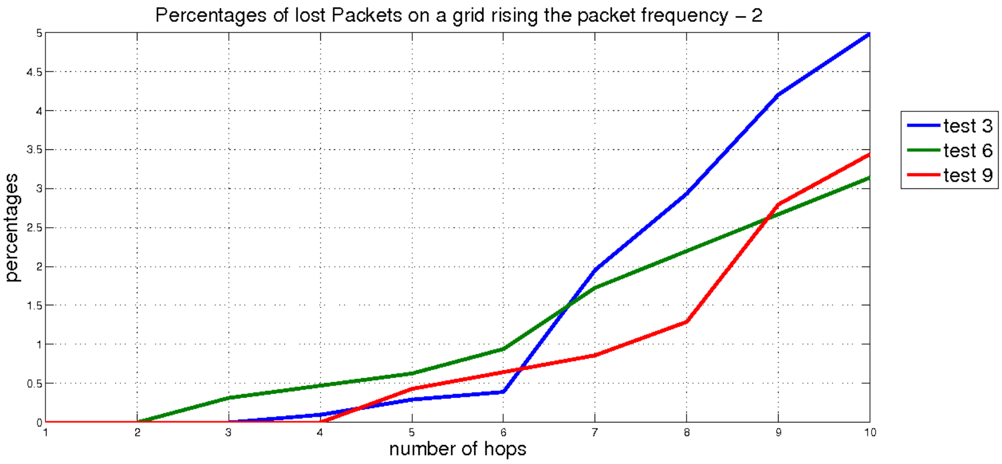


Figure 7.5: *Percentage of lost packets per hop on a grid network reducing the number of colors per packet. Comparison among tests with different packet frequencies.*

	test 3	test 6	test 9
hop 1	0 (0,00%)	0 (0,00%)	0 (0,00%)
hop 2	0 (0,00%)	0 (0,00%)	0 (0,00%)
hop 3	0 (0,00%)	2 (0,31%)	0 (0,00%)
hop 4	1 (0,10%)	3 (0,47%)	0 (0,00%)
hop 5	3 (0,29%)	4 (0,63%)	2 (0,43%)
hop 6	4 (0,39%)	6 (0,94%)	3 (0,65%)
hop 7	20 (1,96%)	11 (1,73%)	4 (0,86%)
hop 8	30 (2,93%)	14 (2,20%)	6 (1,29%)
hop 9	43 (4,20%)	17 (2,67%)	13 (2,80%)
hop 10	51 (4,99%)	20 (3,14%)	16 (3,44%)

Table 7.8: *Lost packets on a grid network reducing the number of colors per packet. Comparison of test 3, 6 and 9.*

These tests have the same r_c value but the number of colors contained into a RtLightMsg is 10, 15 and 20 respectively[2]. For this reason T_p is 2 seconds in test 3, 3 seconds in test 6 and 4 seconds in test 9.

All the tests have a packet loss percentage under the 1% until the 7-th hop. After this bound the test number 3, that has the higher number of packets per seconds, is a little bit less reliable than test 6 and 9. But it looses always less than the 5% of the total number of the sequence packets. We suppose that when the packets rate increases, a node must access to the UART interface most frequently. For this reason the USART is arbitrated a greater number of times per seconds and the radio resource can therefore listen the channel for less time. That is why the amount of lost packets increase with the increasing of the frequency of the RtLightMsg packets.

If we want to observe the behavior of the system under a higher workload, we can compare **test 1** ($N_c = 10$, $T_c = 100$, $d_{TOT} = 700$) and **test 4** ($N_c = 15$, $T_c = 100$, $d_{TOT} = 750$). The RtLightMsg packets for test 1 are sent every second, while for test 4 are sent every 1.5 seconds. The trends are shown in Figure 7.6.

Even in these experiments the system has the same behavior until the 7-th hop. After this limit what happens is not a significant fact because of the small initial delay values of test 1 and 4.
So we can underline that, even if the loss is greater than what we have obtained in the previous experiment of Table 7.8, for little variations of f_p the system remain stable. Although if the workload is high.

The data of this experiment is presented in Table 7.9:

[2]The value of the delay in this situation is negligible because even if it changes from one test to another, it is always greater than a second. So the flooding process is able to cover all the grid.

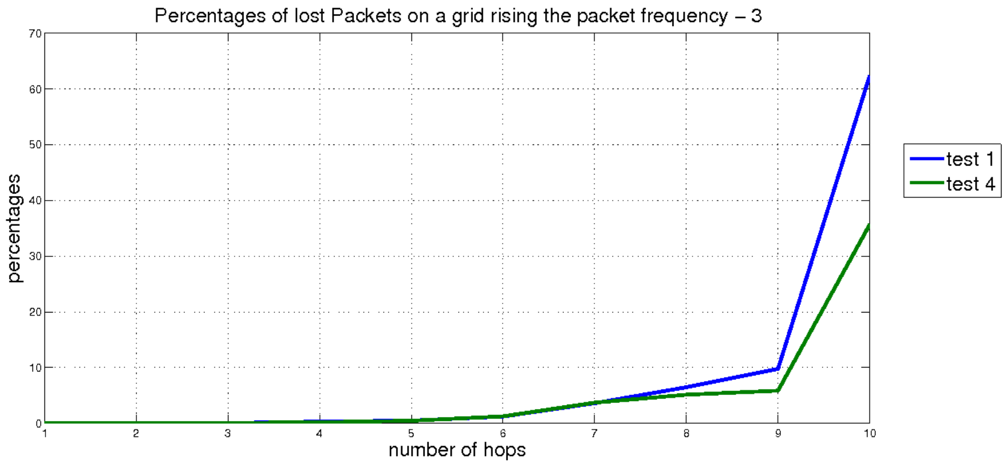


Figure 7.6: *Percentage of lost packets per hop on a busy grid network. Comparison among tests with different packet frequencies.*

	test 1	test 4
hop 1	0 (0,00%)	0 (0,00%)
hop 2	0 (0,00%)	1 (0,03%)
hop 3	1 (0,05%)	2 (0,07%)
hop 4	6 (0,32%)	6 (0,20%)
hop 5	9 (0,49%)	14 (0,46%)
hop 6	23 (1,24%)	38 (1,26%)
hop 7	67 (3,62%)	112 (3,72%)
hop 8	120 (6,48%)	155 (5,14%)
hop 9	181 (9,77%)	177 (5,87%)
hop 10	1155 (62,37%)	1075 (35,67%)

Table 7.9: *Lost packets on a grid network reducing the number of colors per packet. Comparison of test 1 ($N_c = 10$, $T_c = 100$, $d_{TOT} = 700$) and test 4 ($N_c = 15$, $T_c = 100$, $d_{TOT} = 750$).*

7.3 Precision of the nodes

In this section we want to understand the precision of the system in the different situations. We say that a node is precise if it processes the first color of the sequence portion in the exact global clock instant chosen from the *master* node and inserted into the field *turnOnTime* of the RtLightMsg packets.
To calculate the deviation between when the sequence parts should be shown, and when the node really display them, we consider the $d1-(r+d)$ values (Section 7.1).
During our analysis we notice that in the majority of the test the nodes are very precise at each hop. As presented in Table 7.10, the average deviation of each node is about 25 ticks (0.78 ms). We report the results of **test 3** ($N_c = 10$, $T_c = 200$, $d_{TOT} = 1000$), **5** ($N_c = 15$, $T_c = 150$, $d_{TOT} = 1100$), **6** ($N_c = 15$, $T_c = 200$, $d_{TOT} = 1500$), **7** ($N_c = 20$, $T_c = 100$, $d_{TOT} = 1000$), **9** ($N_c = 20$, $T_c = 200$, $d_{TOT} = 2000$), **12** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 2000$) and **13** ($N_c = 20$, $T_c = 400$, $d_{TOT} = 2000$).

	test 3	test 5	test 6	test 7	test 9	test 12	test 13
hop 1	25.56	24.85	25.30	26.73	23.47	25.83	24.35
hop 2	26.21	24.24	23.86	25.98	23.72	23.51	24.92
hop 3	25.24	24.76	24.59	25.75	24.98	23.28	24.21
hop 4	24.95	24.29	24.76	25.60	24.62	23.49	25.38
hop 5	25.17	24.94	25.77	26.11	25.38	24.70	24.86
hop 6	26.68	25.19	25.25	26.46	24.93	26.31	24.93
hop 7	26.10	24.76	26.27	26.75	25.15	24.79	25.45
hop 8	25.57	24.40	24.48	26.53	23.74	24.25	25.14
hop 9	26.46	24.25	24.01	26.42	24.73	25.80	23.78
hop 10	28.05	24.09	24.65	25.54	25.63	24.38	24.56

Table 7.10: *Precision of the system. Comparison of test 3, 5, 6, 7, 9, 12 and test 13.*

The global average per hop of these test are showed in Figure 7.7.

Instead we realize that in some other cases, like **test 1** ($N_c = 10$, $T_c = 100$, $d_{TOT} = 700$), **2** ($N_c = 10$, $T_c = 150$, $d_{TOT} = 750$), **4** ($N_c = 15$, $T_c = 100$, $d_{TOT} = 750$) and **10** ($N_c = 20$, $T_c = 300$, $d_{TOT} = 500$) the behavior of the

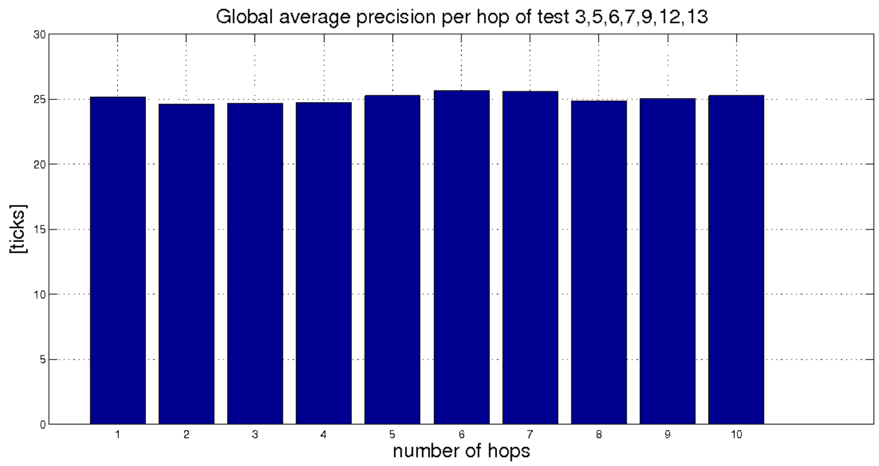


Figure 7.7: *Precision of the nodes per hop on a grid network. Comparison among several tests.*

chromotherapy system is different. During these tests all the nodes (even the motes closer to the *master*) have less precision in respect to the tests of Table 7.10. We notice that the configuration 1,2,4 and 10 have one common characteristic: the value of the initial delay (d_{TOT}) parameter lower than 750 ms. For this reason we find a motivation to their poor accuracy (see the experimental results in Table 7.11).

	test 1	test 2	test 4	test 10
hop 1	42,50	40,21	38,81	42,77
hop 2	41,50	39,82	38,62	41,57
hop 3	43,22	40,15	39,23	43,70
hop 4	42,08	40,14	39,37	42,69
hop 5	42,13	40,32	39,15	48,33
hop 6	44,85	39,84	41,06	117,99
hop 7	50,34	91,27	44,99	254,28
hop 8	102,74	52,03	62,99	-
hop 9	228,34	118,48	160,71	-
hop 10	349,43	228,05	283,93	-

Table 7.11: *Precision of the system in ticks. Comparison of test 1, 2, 4 and test 10.*

Because in our testbed the flooding process generally takes about 700 ms to cover all the grid network, every node is still busy from this activity when it should start the displaying of the first color of the message. So the less precision is due to the great number of messages that every node must menage during the flooding of the sequence. We can conclude that if the initial delay value is greater than the amount of time necessary to perform the flooding, the precision of the system increases.
In addition is possible to understand that if a node receives a RtLightMsg in the instant t very closer to the *turnOnTime* instant contained in the packet, the precision degrades more rapidly.

7.4 Delays introduced in the flooding process from each hop

This testing phase wants to observe which is the delay d_i introduced in the flooding process from each hop, and what aspects of the network influence these values.
In order to find out the delay introduced from the hop i, we calculate (using the nomenclature of Section 7.1) the value $d_{i+1} - d_i$ and we subtract also the random wait interval w in order to be more precise.
As we can see in Figure 7.8, the average of the delays introduced from each hop is not constant, but it depends from the number of nodes belonging to the hop (Table 7.2). So, if we have a small number of nodes at a certain hop, the introduced delay is low. On the other hand, a greater delay is involved from a hop which is populated with a lot of nodes.
In Figure 7.9 we present the trend of the introduced delay referred to the number of hop nodes. So, accordingly to Table 7.2 we have that:

- a single node in present only in the 10-th hop;
- the first and 9-th hop have 2 nodes each one;
- 3 nodes populate the second and also the 8-th hop;
- 4 nodes are in the third and in the 7-th hop;
- 5 nodes are in each the 4-th, 5-th and 6-th hop.

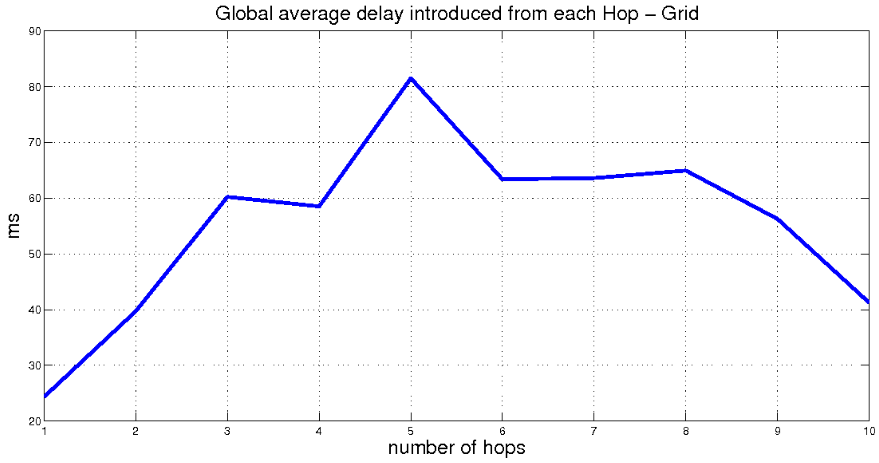


Figure 7.8: *Global average delay introduced from each hop in a grid network.*

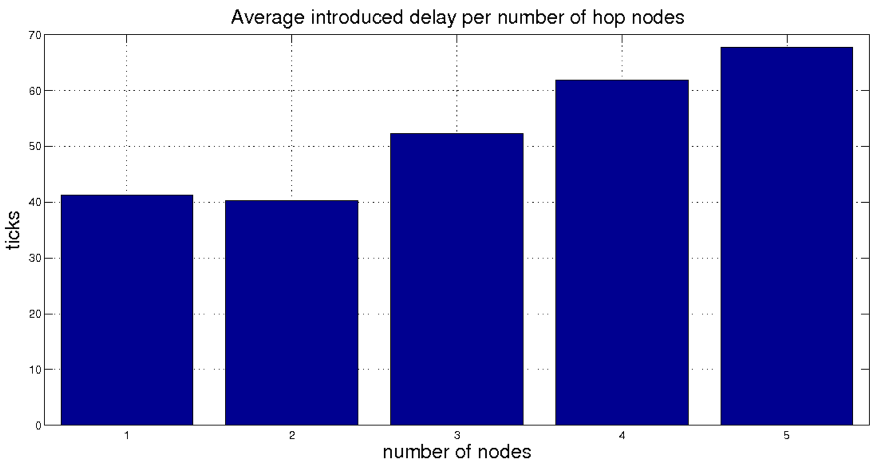


Figure 7.9: *Average introduced delay per number of hop nodes.*

Now we want to find out the relationship among the delay and the number of nodes of the hop. For this reason we found the equation of the regression line using the Ordinary Least Squares (OLS) method (Figure 7.10). The regression line of the introduced delays is:

$$y = 7.4761x + 30.2825$$

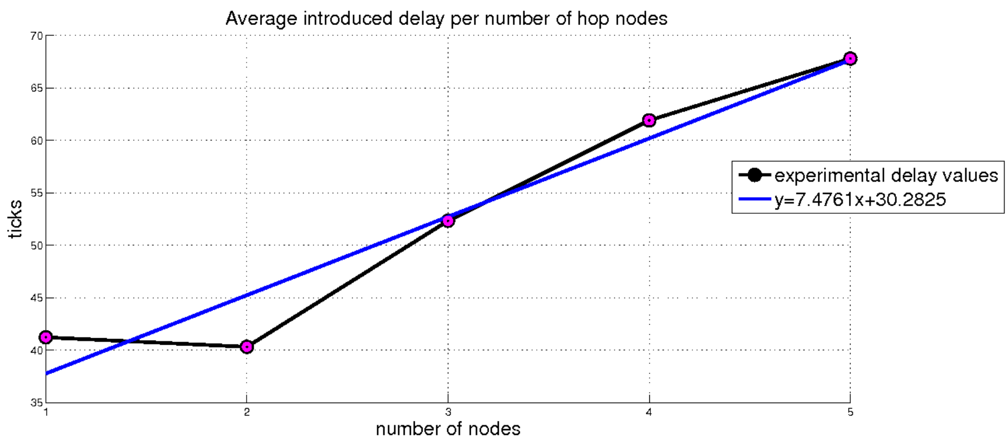


Figure 7.10: *Regression line for the estimations of the d_i values.*

Thanks to this equation we can estimate the delays d_i introduced from each hop in order to figure out the total delay d_{TOT} necessary for the communication.

Clearly, if we want to implement a chromotherapy system over a dynamic WSN in which the motes are not static, this process is not possible. The d_i values are not constant and so the d_{TOT} value changes every time a topology change occurs.

We have deduced that hop delays are strictly dependent from the number of nodes per hop and hence the collisions occurred. In our tests all the nodes are into the radio range of each others, but we suppose that in a WSN which have sparse nodes, the delay should remain constant.
To confirm our conclusion, we show in Figure 7.11 the delay introduced from the nodes in a linear array. This network topology in fact has only one node per hop. So each jump introduce in the flooding process the same delay of

about 47 milliseconds. Only the first hop is faster than the others, but it is due to the *master* implementation.

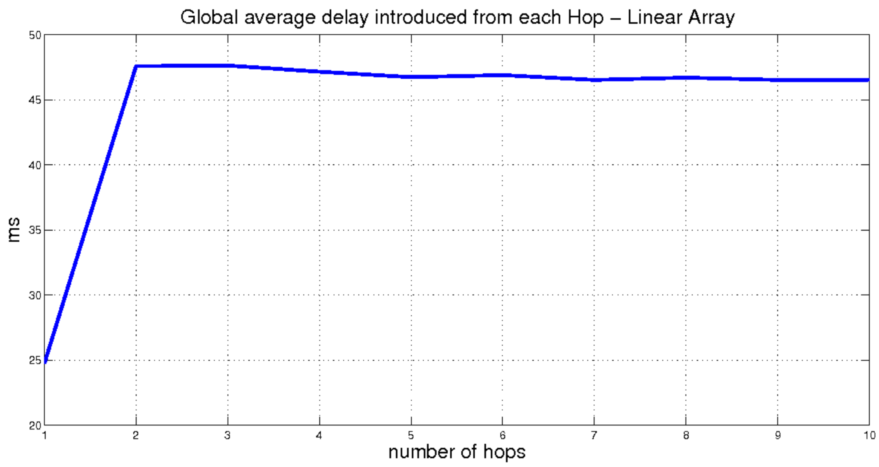


Figure 7.11: *Global average delay introduced from each hop in a linear array network.*

7.5 Variations of the responsiveness of the operating system

To understand if TinyOS is influenced from the chromotherapy system implementation we studied the variation of the amount of time that the OS needs to complete a message sending process. So the data processed were (using the nomenclature of Section 7.1) the values $sd - s$.
Our test has demonstrate that the operating system seems to be not affected from our application. All the sending activities were completed in average in 25.56 ms from when the application requests the forwarding (Figure 7.12).

In Table 7.12 are summarized the averages of the amounts of time employed in the sending processes and the standard deviation from the average. We observe that there is never great deviation from the average.

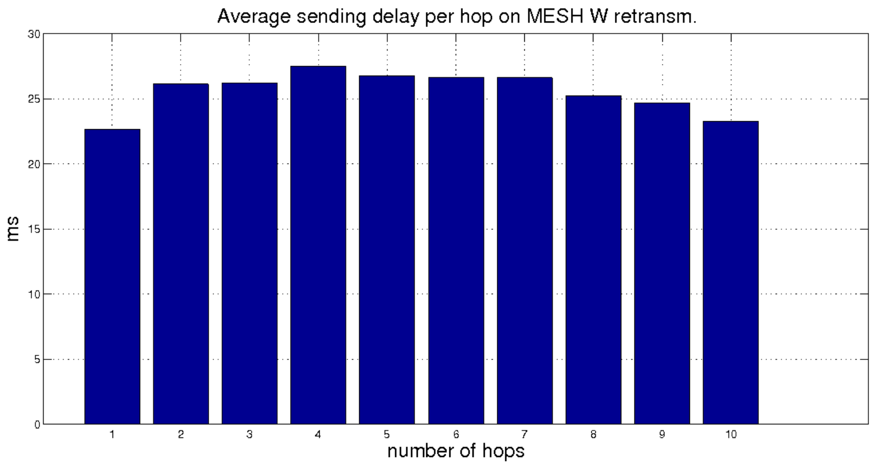


Figure 7.12: *Global average sending delay per hop on a grid network.*

	hop 1	hop 2	hop 3	hop 4	hop 5	hop 6	hop 7	hop 8	hop 9	hop 10
test 1	23,08	26,72	26,27	27,47	26,98	26,89	26,76	25,16	26,70	25,51
test 2	22,42	26,03	25,90	27,19	26,91	25,96	26,26	24,50	25,44	25,28
test 3	22,78	27,14	25,93	28,30	26,64	26,99	26,87	25,20	24,17	22,50
test 4	23,48	26,53	26,65	28,08	27,42	27,21	27,04	25,53	26,23	26,93
test 5	22,19	26,82	25,67	27,77	26,78	26,47	26,60	25,41	24,64	22,49
test 6	22,01	25,86	26,73	27,28	26,82	26,39	26,19	25,27	24,32	22,59
test 7	22,61	26,10	26,62	27,78	26,96	26,64	26,57	25,66	24,66	23,00
test 8	22,89	25,83	26,46	27,25	26,79	26,65	26,38	25,33	24,19	22,86
test 9	22,83	25,80	25,40	27,18	26,45	26,35	25,86	25,03	24,02	22,45
test 10	22,93	25,17	25,91	26,76	26,18	27,10	28,42	-	-	-
test 11	22,37	25,86	26,09	27,07	26,88	26,57	26,19	24,93	23,82	22,04
test 12	22,79	25,43	26,54	27,88	26,39	26,83	26,24	25,31	23,72	21,65
test 13	22,20	26,28	26,24	27,22	26,61	26,48	26,71	25,19	24,09	22,17
AVERAGE	22,66	26,12	26,18	27,48	26,75	26,66	26,62	25,21	24,67	23,29
STD DEV	0,41	0,56	0,41	0,45	0,31	0,35	0,63	0,30	0,96	1,66

Table 7.12: *Average of the amounts of time involved in a message sending (in milliseconds). Comparison of all the tests.*

Chapter 8

Conclusions

The objective of this thesis was to develop a chromotherapy system using WSN as infrastructure. It must perform the visualization, via external RGB devices, of a color sequence. An innovative aspect of the project is the possibility that our application inherits all the characteristics of a WSN as for instance the flexibility, the mobility and the multi-hop communication. The work also included the study of the architecture composed by Tmote Sky motes and the TinyOS operating system. We also made a great effort in the implementation of the entire project in Java and in the NesC programming language.

A key aspect of a chromotherapy system is the coordination that must exist among all the nodes. This specification can be satisfied on a synchronized network. So all the nodes must agree to a common reference global clock.

Therefore in the first phase of this work we have developed the network synchronization. After we studied all the algorithms in the literature for the WSN synchronization, it was decided to adopt the ATS algorithm [4, 30]. It is a very precise synchronization method. It is also independent from the network topology and fully distributed.

The characteristics of our application allow to relax the constraints of the algorithm accuracy. So it was carried out a simplification of ATS removing the skew compensation in the global clock estimation of the nodes, while maintaining the offset compensation. In this way, the lightness and dynamism of the original algorithm were maintained. Furthermore it was

possible to save computational resources which can be used in the management of the chromotherapy sequence. However, if compared to an algorithm that also compensates the drift of the clock, the only compensation of the offset requires more frequent synchronization messages in order to be precise. In our case was experimentally verified that with a synchronization message interval of 30 seconds we are able to obtain a millisecond accuracy. The project requires that there must not be chromatic differences between nodes while they are showing the color sequence through the RGB device. The millisecond accuracy ensures that this situation does not occur. In fact the human eyes are less precise than a system with a so great precision.
During the first testing phase of the algorithm, was verified that in some cases the convergence to a common reference clock took too long. To resolve this problem has been conceived and then implemented an overlay logical network. This last one builds a hierarchical structure over the WSN. A *root* node (chosen from the user) becomes the reference point for all the other nodes. Now the temporal information received from a node can have different importance. In fact each mote considers only information from neighborhoods that are closer to the root than itself. The overlay structure is also able to adapt itself if topology changes occur, or if the *root* falls. Our implemented system is able to decide what is the weight of the overlay structure in the synchronization algorithm. We have experimentally demonstrated that a particular configuration of this hybrid algorithm maximizes the performance.
The second testing phase of the Overlay-based synchronization algorithm has demonstrates also the robustness, the adaptability and the high speed of convergence. In addition a higher accuracy was noticed if compared to the previous offset compensation algorithm without overlay structure.

The second part of the thesis concerns the design of the method to create, manage and display the color therapy sequence. For this purpose, the succession of colors was divided into sections in order to make the process generation-reproduction of the sequence as real-time as possible. Each section consists of a fixed number of colors. A *master* node communicates each

portion to its neighborhood motes of the WSN. Afterwards the information are sent to all the others nodes thanks to a multi-hop flooding process.
The sequence communication requires a certain amount of time to cover the entire network. This time interval should be smaller than the delay generation-reproduction of the sequence (called *initial delay*) which is fixed from the user. If this constraint is not respected, the sequence is not able to reach the nodes further from the *master*.
The display phase of the sequence is made through an RGB device that is still under construction. The device must receive messages via the UART interface. In the Tmote sky the USART of the MCU is shared from the UART pins and the radio. Therefore it was necessary to implement the arbitration of this resource permitting to work to both the interfaces.
Another specification of the project was the possibility that the nodes could be grouped into independent systems. These systems must coexist in the same environment without interfering with each other. This feature has been realized thanks to a simple and effective idea. Every message contains a field called *groupMask* which permit to each node to diversify the group membership of the packet.
The final phase of the work tested the system when it is subjected to different workloads. We realize that any sequence portion message must be forwarded twice by each node. This is necessary to grant that the system reproduces the complete color sequence with high fidelity. Moreover, we found that even if the workload changes, the fidelity of the system remains satisfactory. We have also demonstrate that the more a node is far from the *master* and the more its sequence reproduction fidelity degrades.
Finally, we have understood that the parameter that defines the *initial delay* is very important. Obviously is better to have a short delay. But it is not possible to reduce too much this delay for several reasons. First because it is strongly dependent from the number of hops of the network (and hence its extension) and from the number of nodes at each hop. The second because if the delay is too short, the sequence is also not able to reach all the nodes of the WSN and the precision and the fidelity of the system decrease.

Further developments of this work of thesis are possible. It is possible

for example to implement a *root* election system for the overlay structure. A mechanism that elects the node with the lowest ID (as presented in FTSP [33]) makes the network more independent from the user. On the other hand a greater computation and memory usage are required to the nodes.
It is also possible to design a cluster-based chromotherapy system for wide networks in which the nodes are very concentrate. In this situation the communication becomes difficult because of the collisions of the messages[1]. For this reason with a cluster-based network approach, in which only some nodes perform the flooding of the color sequence and all the others only receive and display the sequence parts, we can reduce the collisions and realize a high fidelity chromotherapy system in high density networks.
In addition, the actual developed system can be used for many others activities which require remote coordination of light sources or in applications which create colored sequences in relation to external events as for instance movements or sounds.
Modifying the RtLightControlC component is also possible to create light effects based on environmental changes, as for example the ambient brightness.
One limit of our current project implementation is the energy consumption. In fact this aspect is not handled in any way. It is mainly because for a compute-intensive application the realization of this feature became impossible.
Supposing that we are able to know some information of how the generator creates the colors sequence. So for instance we are able to know that among color c_x and c_y there are always the intermediate colors $c_{x+1}, c_{x+2}, ..., c_{y-2}$, c_{y-1}. If this happens, we can implement a *compression of the sequence*. The *master* could send only a packet containing the two colors c_x and c_y, then the *slaves-repeaters* nodes that receive this couple of colors already know that the sequence they must show is composed by all the colors $c_x, c_{x+1}, c_{x+2}, ...,$ c_{y-2}, c_{y-1}, c_y. Another possible improvement is to remove the constraint that imposes the synchronization of the *master* node. In this case the reference time in the field *turnOnTime* of the RtLightMsg package could

[1]In our work we have also understood that collisions are the greatest weak point of the project.

be calculated and filled from one of the first nodes that receive the message from the *master*. Because this last one can have more than one neighborhood, some control mechanisms must be implemented to avoid the multiple calculation of the *turnOnTime* value.
Finally some code changes can easily adapt our work to all the applications that need to realize a coordinated succession of actions (through external devices or not) at a specific frequency over the entire extension of a wireless network.

Bibliography

[1] A.S.Tanenbaum, M.V.Steen. *"Distributed Systems: Principles and Paradigms"*. PEARSON Prentice Hall, 2007. pages 1-30.

[2] U.Hansmann, L.Merk, M.S.Nicklous, T. Stober. *"Pervasive Computing: The Mobile World"*. Springer, 2003. pages 15-21.

[3] R.N.Murty, G.Mainland, I.Rose, A.R.Chowdhury, A.Gosain, J.Bers, M.Welsh. *"CitySense: An UrbanScale Wireless Sensor Network and Testbed"*. 2008. School of Engineering and Applied Sciences, Harvard University BBN Technologies, Inc. 2008 IEEE International Conference on Technologies for Homeland Security.
Web site: `http://www.citysense.net/`

[4] L.Schenato, F.Fiorentin. *"Average TimeSynch: a consensus-based protocol for time synchronization in wireless sensor networks"*. 2009. Proceedings of 1st IFAC Workshop on Estimation and Control of Networked Systems (NecSys'09).

[5] P.Casari, A.P.Castellani, A.Cenedese, C.Lora, M.Rossi, L.Schenato, M.Zorzi. *"The Wireless Sensor Networks for City-Wide Ambient Intelligence (WISE-WAI) Project"*. 2009. SENSORS volume 9-2009 pages 4056-4082.
Web site *CaRiPaRo* project: `http://cariparo.dei.unipd.it/`

[6] D.Gay, P.Levis. *"TinyOS Programming"*. 2009 Cambridge University Press.

[7] B.W.Kernighan, D.M.Ritchie. *"The ANSI C Programming Language - 2nd edition"*. 1988 Prentice Hall.

[8] F.Fiorentin. *"Implementazione di sincronizzazione temporale distribuita in reti di sensori wireless"*. 2007-2008

[9] D.L.Mills. *"Improved algorithms for synchronizing computer network clocks"*. 1994. Proceedings of ACM Conference on Communication Architectures (ACM SIGCOMM 1994). London, UK.

[10] D.L.Mills. *"Network Time Protocol Version 4 Reference and Implementation Guide"*. 2006.
Web site: `http://www.eecis.udel.edu/%7emills/database/`
`/reports/ntp4/ntp4.pdf`

[11] M.Bertinato, G.Ortolan, F.Maran, R.Marcon, A.Marcassa, F.Zanella, P.Zambotto, L.Schenato, A.Cenedese. *"RF Localization and tracking of mobile nodes in Wireless Sensors Networks: Architectures, Algorithms and Experiments"*. 2007. Proceedings of the 5th European Conference on Wireless Sensor Networks (EWSN'08), 2008.

[12] J. Elson, L. Girod, D. Estrin. *"Fine-grained network time synchronization using reference broadcasts"*. 2002. Proceedings of the 5th symposium on Operating systems design and implementation (OSDI'02), pages 147-163.

[13] S.Yoon, C.Veerarittiphan, M.L.Sichitiu. *"Tiny-sync: Tight time synchronization for wireless sensor networks"*. 2007. ACM Journal of Sensor Networks, 3(2), 2007.

[14] S.Ganeriwal, R.Kumar, M.B.Srivastava. *"Timing-sync protocol for sensor networks"*. 2003. Proceedings of the first international conference on Embedded networked sensor systems (SenSys'03), pages 138-149, 2003.

[15] J.v.Greunen, J.Rabaey. *"Lightweight time synchronization for sensor networks"*. 2nd ACM International Workshop on Wireless Sensor Networks and Applications, pages 11-19, September 2003.

[16] Q.Li, D.Rus 2006. *"Global Clock Synchronization in Sensor Networks"*. IEEE Transactions on computer, vol. 55, no.2, February 2006.

[17] G.Werner-Allen, G.Tewari, A.Patel, M.Welsh, R.Nagpal. *"Firefly-inspired sensor network synchronicity with realistic radio effects"*. ACM Conference on Embedded Networked Sensor Systems (SenSys'05), November 2005.

[18] O.Simeone, U.Spagnolini. *"Distributed time synchronization in wireless sensor networks with coupled discrete-time oscillators"*. EURASIP Journal on Wireless Communications and Networking, 2007: Article ID 57054, 13 pages, 2007. doi:10.1155/2007/57054.

[19] R.Solis, V.Borkar, P.R.Kumar. *"A new distributed time synchronization protocol for multihop wireless networks"*. 45th IEEE Conference on Decision and Control (CDC'06), December 2006.

[20] K.S.Low, W.N.N.Win, M.J.Er. *"Wireless Sensor Networks for Industrial Environments"*. 2005. Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05).

[21] Md.A.Hussain, P.khan, K.K.Sup. *"Md.Asdaque Hussain, Pervez khan, Kwak kyung Sup"*. 2009. Proceedings of the 11th international conference on Advanced Communication Technology (ICACT'09). 2009.

[22] M.Rossi, G.Zanca, L.Stabellini, R.Crepaldi, A.F.Harris III, M.Zorzi. *"SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks using Fountain Codes"*. 2008. 5th Annual IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON), 2008.

[23] H.Garcia-Molina. *"Elections in a Distributed Computing System"*. 1982. IEEE Transactions on Computers, vol. 31, no. 1, pages 48-59, Jan. 1982, doi:10.1109/TC.1982.1675885.

[24] E.Chang, R.Roberts. *"An improved algorithm for decentralized extrema-finding in circular configurations of processes"*. 1979. Communications of the ACM (ACM) 22 (5): pages 281283, doi:10.1145/359104.359108.

[25] S.Vasudevan, J.Kurose, D.Towsley. *"Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks"*. 2004. Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04), pages 350-360, doi:10.1109/ICNP.2004.1348124.

[26] From Moteiv Corporation: *"Tmote Sky: Datasheet"*. 2004-2006 Moteiv Corporation.
Web site: `http://sentilla.com/files/pdf/`
`/eol/tmote-sky-datasheet.pdf`

[27] P. Levis. *"TinyOS 2.0 Overview"*.
Web site:`http://www.tinyos.net/tinyos-2.x/doc/`
`/html/overview.html`

[28] E.Brewer, D.Culler, D.Gay, P.Levis. *"nesC 1.2 Language Reference Manual"*. 2005.
Web site: `http://www.tinyos.net/dist-2.0.0/`
`/tinyos-2.0.0beta1/doc/nesc/ref.pdf`

[29] D.L.Mills. *"Internet time synchronization: the network time protocol"*. 1991 IEEE Trans. Communications 39, 10 (Oct.), 1482-1493.
Web site: `http://www.eecis.udel.edu/ mills/ntp.html`

[30] L.Schenato, G.Gamba. *"A distributed consensus protocol for clock synchronization in wireless sensor network"*. 2007. 46th IEEE Conference on Decision and Control.

[31] K.Klues, P.Levis, D.Gay, D.Culler, V,Handziski. *"TEP 108 (TinyOS Enhancement Proposals) - Resource Arbitration"*. 2009.
Web site: `http://www.tinyos.net/tinyos-2.x/doc/`
`/html/tep108.html`

[32] Texas Instruments Incorporated. *"MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller manual (Rev. F)"*. 2009.

Web site: `http://www.cs.jhu.edu/ cliang4/public/ /datasheets/msp430f1611.pdf`

[33] M.Maroti, B.Kusy, G.Simon, A.Ledeczi. *"The flooding time synchronization protocol"*. 2004. Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys 2004). 2004 - ACM Press, pages 39-49.

[34] D.Culler, D.Gay, V.Handziski, J.H.Hauer, J.Polastre, C.Sharp, A.Wolisz. *"TEP 2: Hardware Abstraction Architecture"*. 2007. Web site: `http://www.tinyos.net/dist-2.0.0/tinyos-2.x/doc/ /html/tep2.html`

[35] Web site *SIMEA*: `http://automatica.dei.unipd.it/people/ /cenedese/research/simea.html`

[36] Web site *OPTICONTROL*: `http://www.opticontrol.ethz.ch/index.html`

[37] Web site Nelly Bay-Magnetic Island WSN: `http://www.science.org.au/nova/110/110key.htm`

[38] Web site *SENTILLA*: `http://www.sentilla.com/`

[39] Web site *CROSSBOW*: `http://www.xbow.com/`

[40] Web site: National Institute of Standards and Technology, IEEE 1588: `http://ieee1588.nist.gov/`

[41] Web site dedicated to various overlay technologies: `http://www.overlay-networks.info/`

[42] Wikipedia definition: `http://en.wikipedia.org/wiki/Best_effort_delivery`

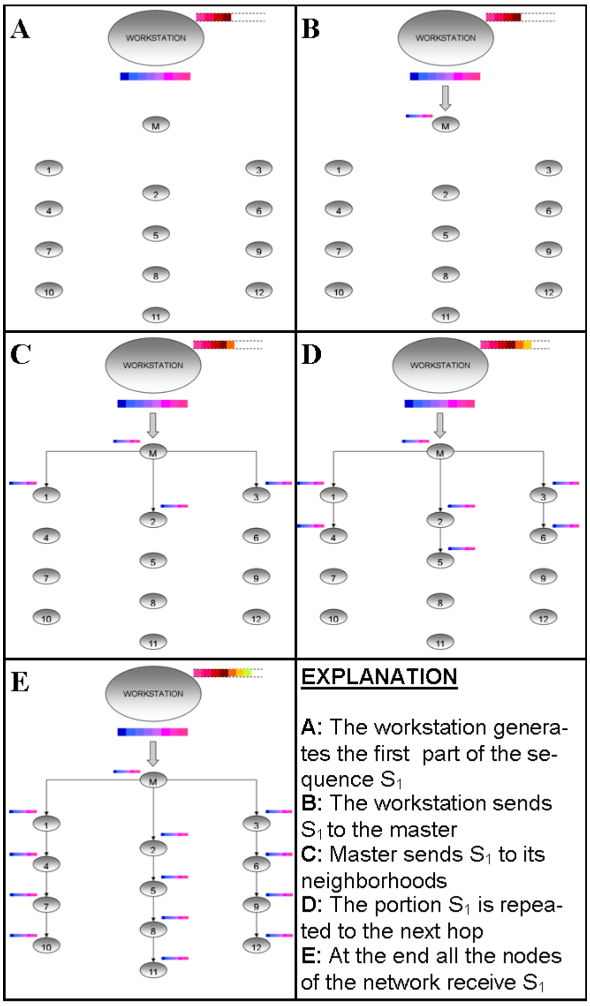
[43] TinyOS tutorial web page: `http://docs.tinyos.net/index.php/ /Mote-mote_radio_communication`

[44] *CodeBlue: Wireless Sensors for Medical Care* web site:
http://fiji.eecs.harvard.edu/CodeBlue

[45] Wikipedia definition:
http://en.wikipedia.org/wiki/Consensus_%28computer_science%29

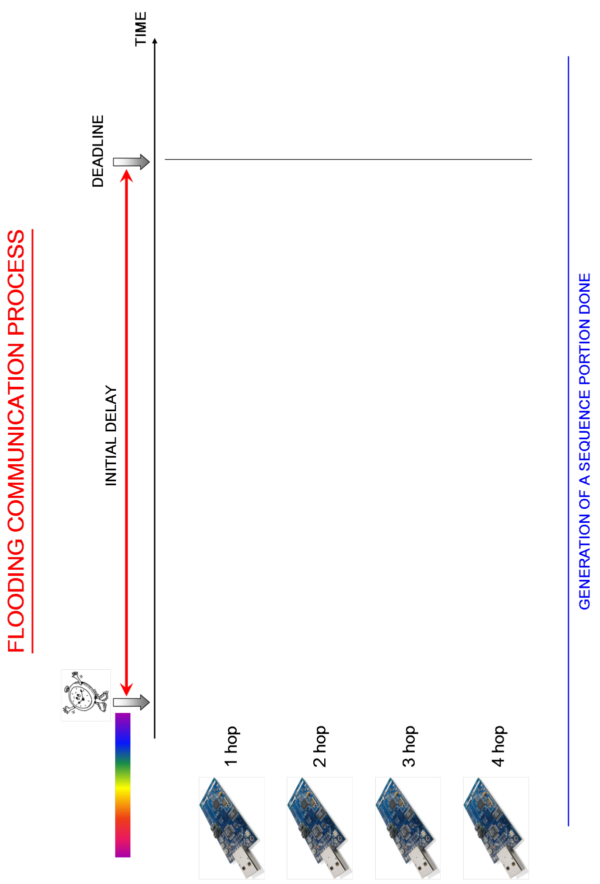
Appendix A

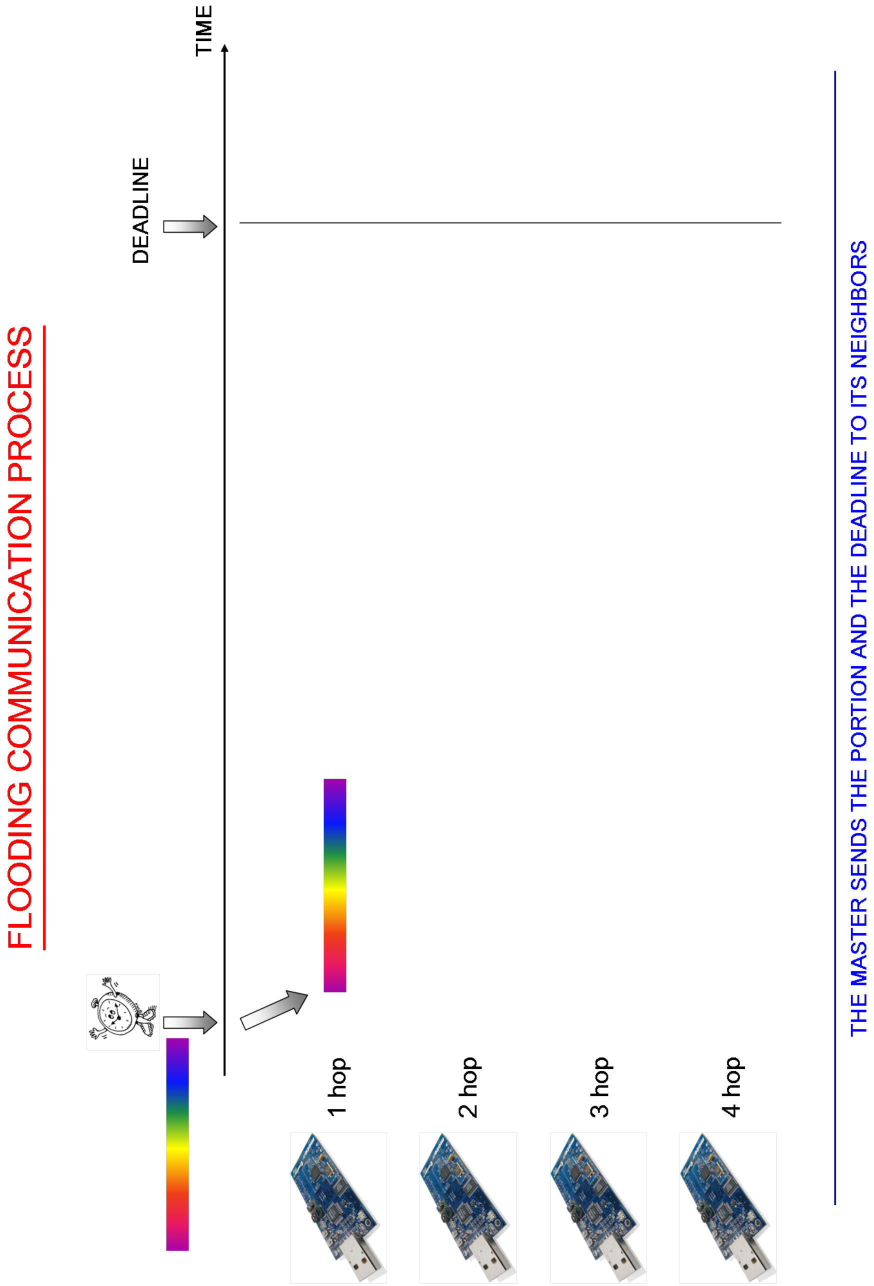
Example of sequence diffusion

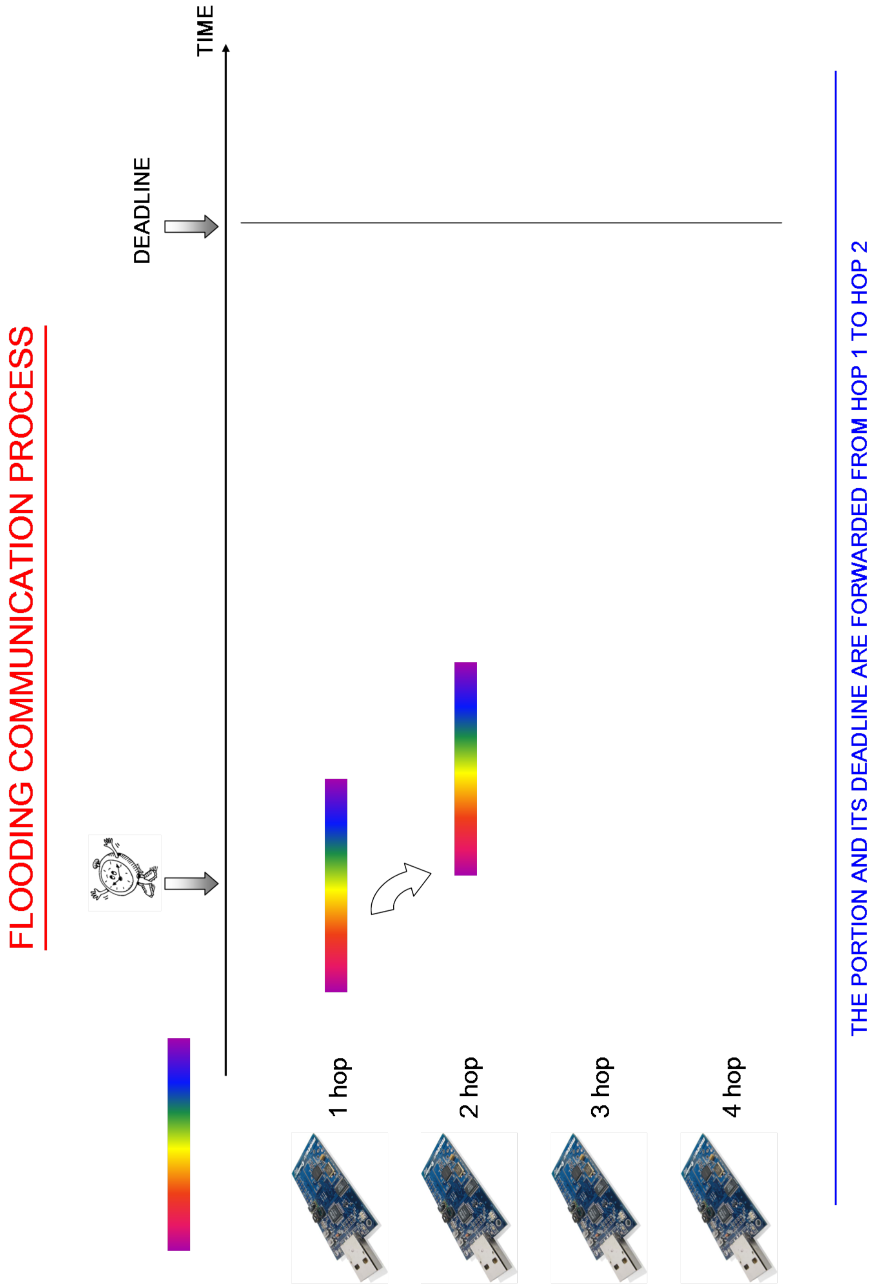


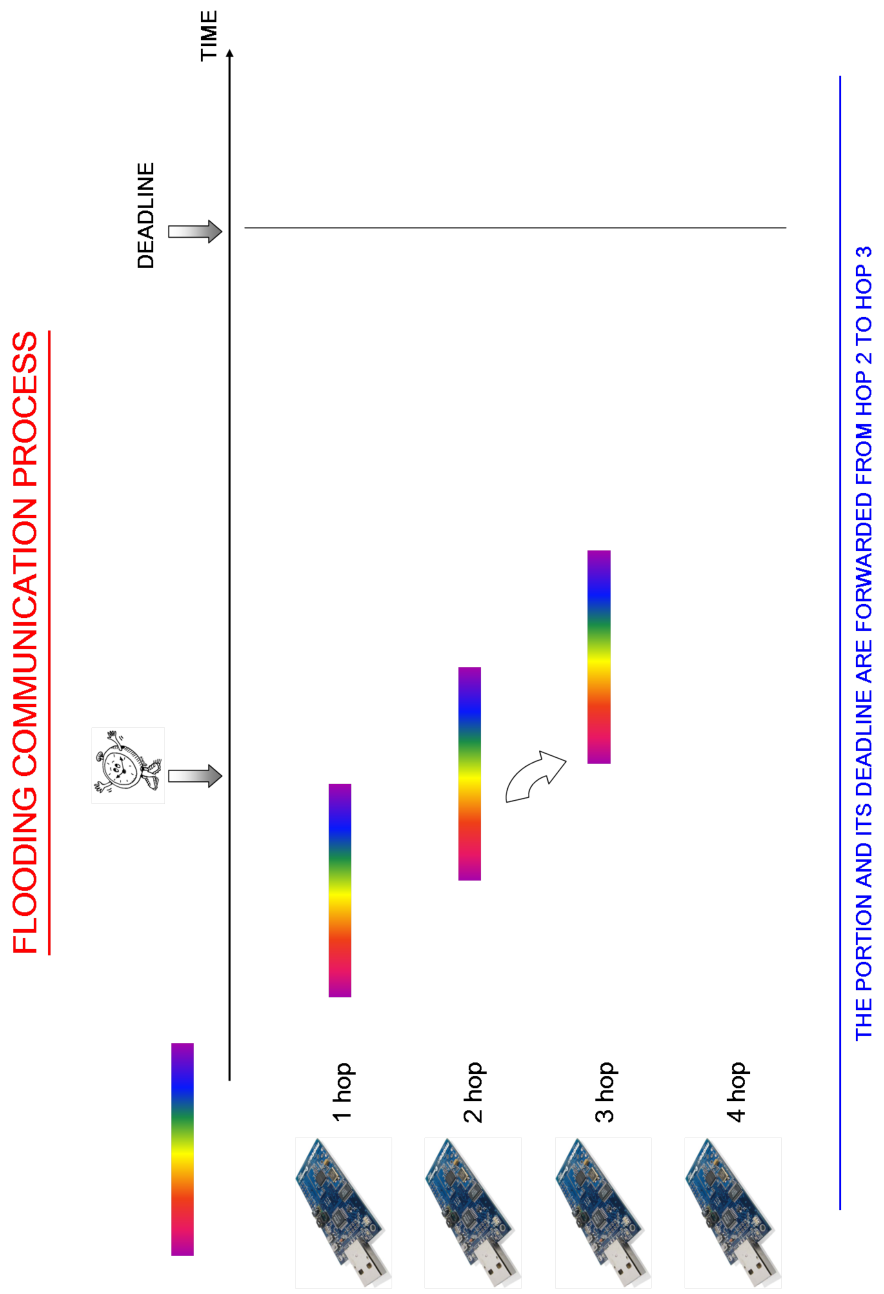
Appendix B

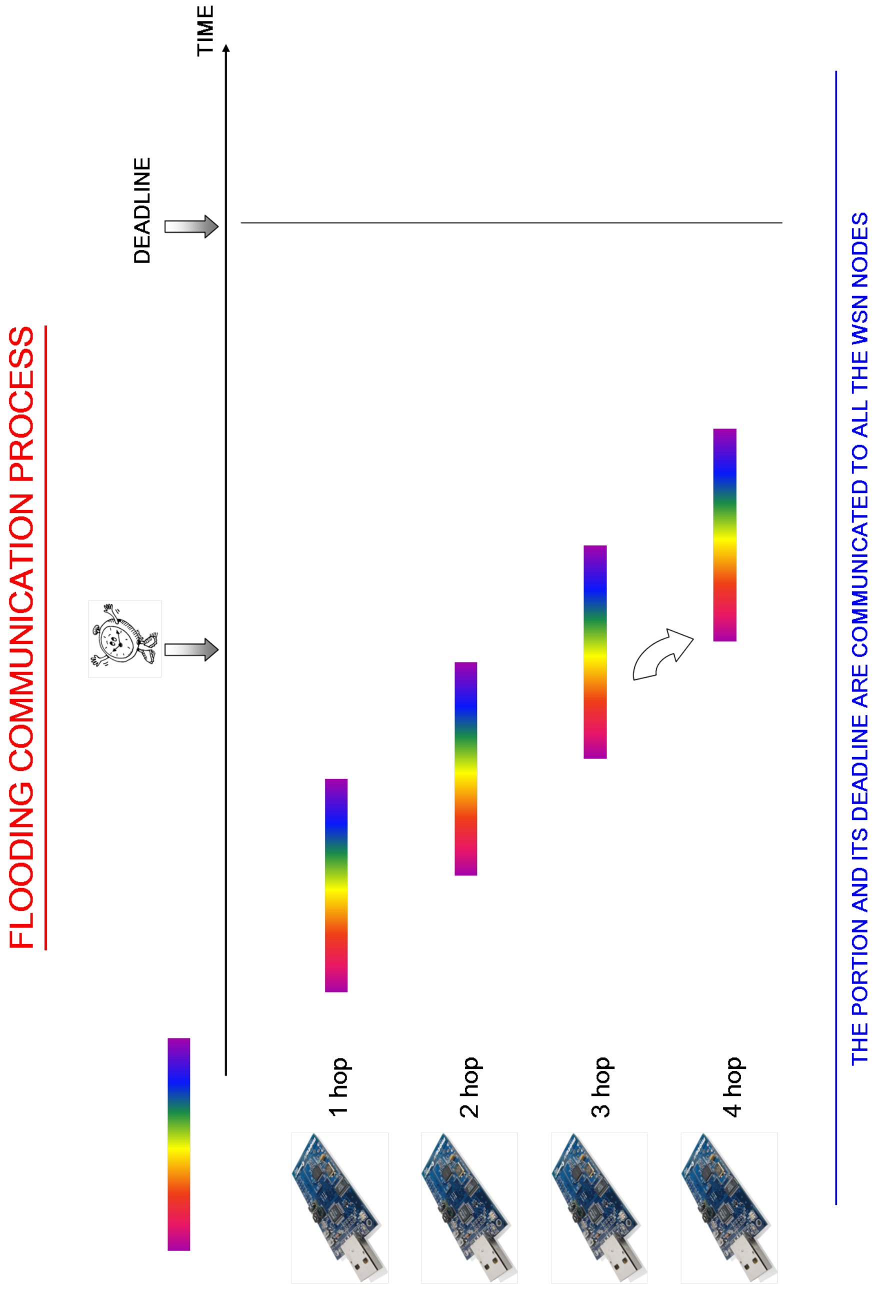
Behavior of the chromotherapy system

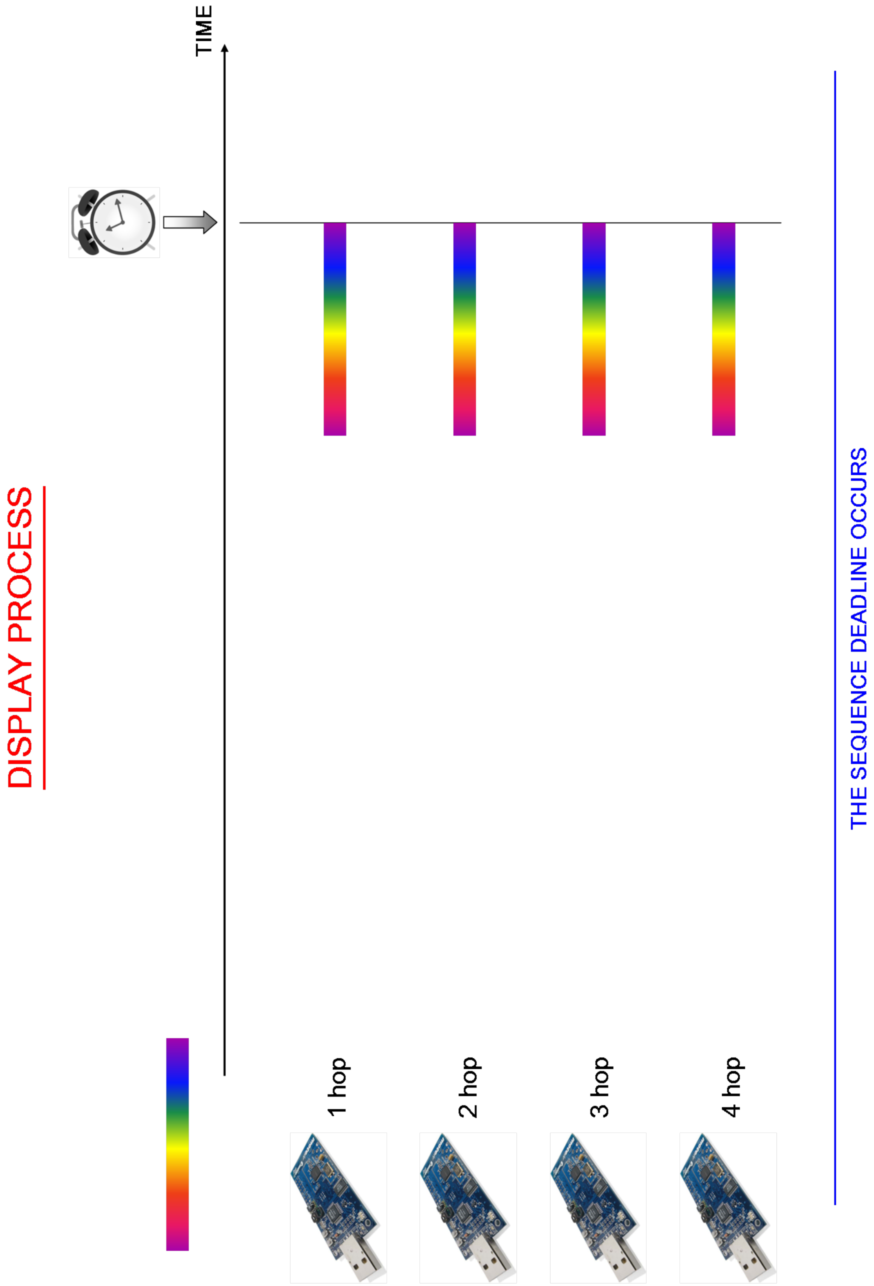


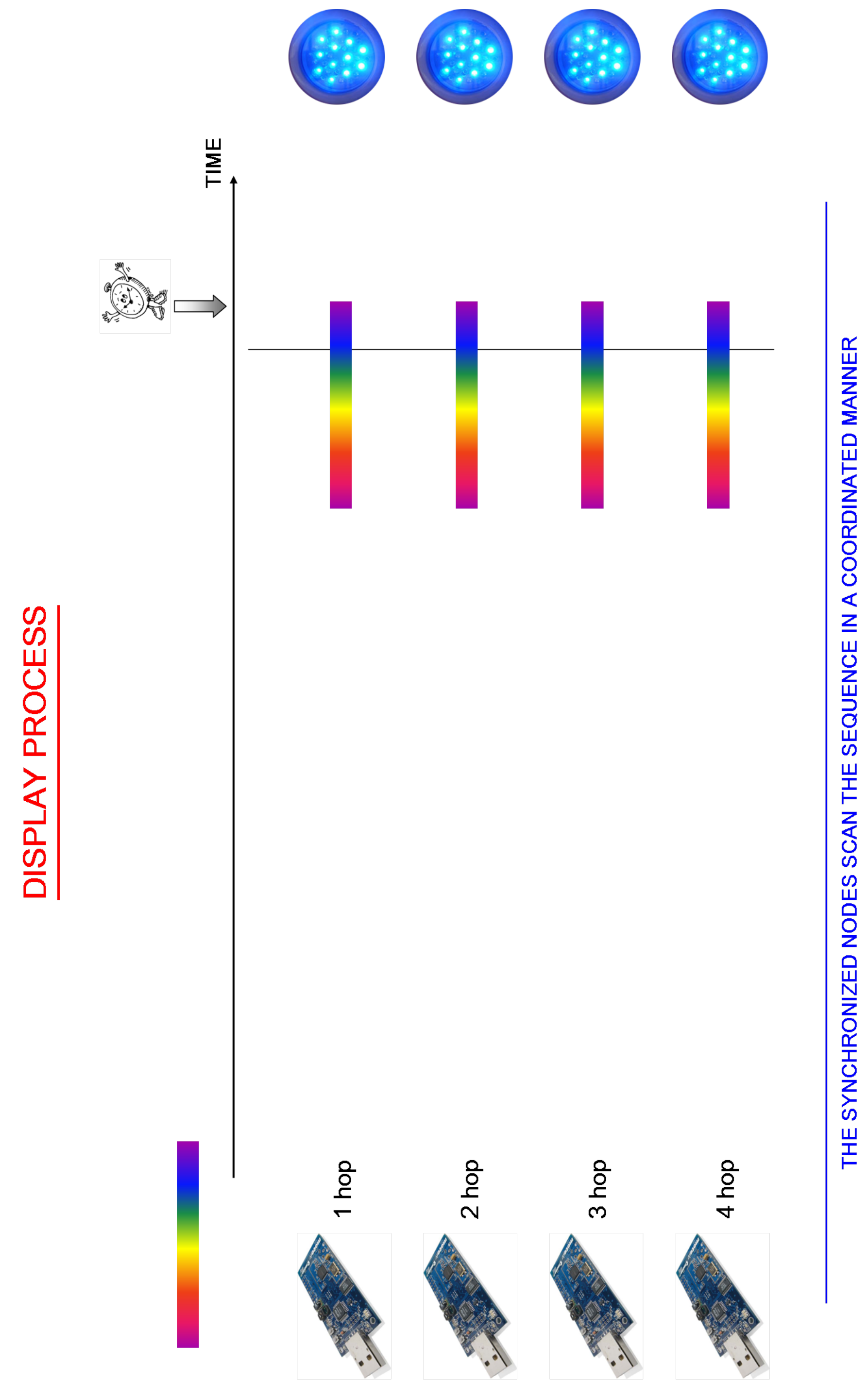

FLOODING COMMUNICATION PROCESS
TIME
DEADLINE
1 hop
2 hop
3 hop
4 hop
THE MASTER SENDS THE PORTION AND THE DEADLINE TO ITS NEIGHBORS

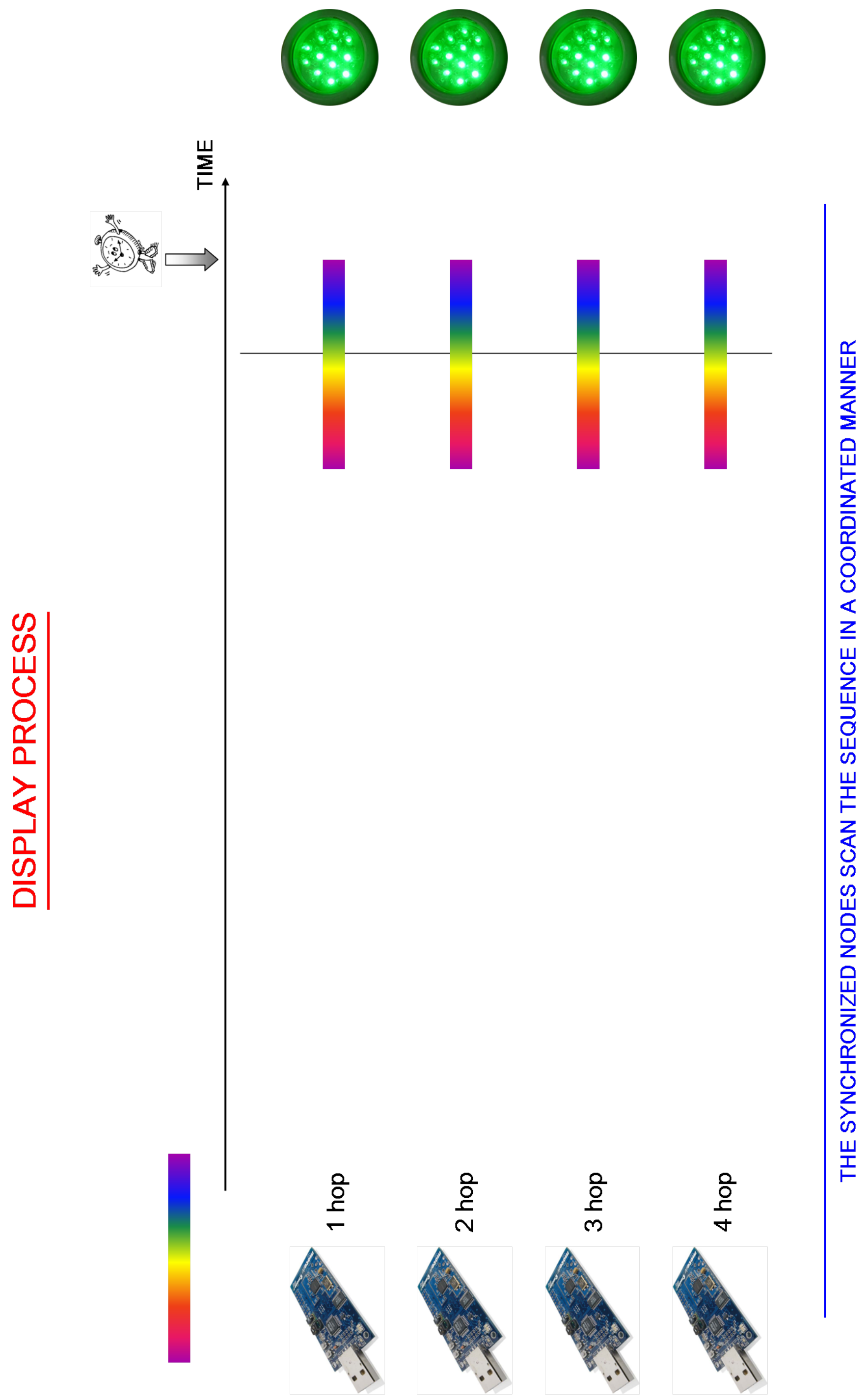

FLOODING COMMUNICATION PROCESS
DEADLINE
TIME
1 hop
2 hop
3 hop
4 hop
THE PORTION AND ITS DEADLINE ARE FORWARDED FROM HOP 1 TO HOP 2

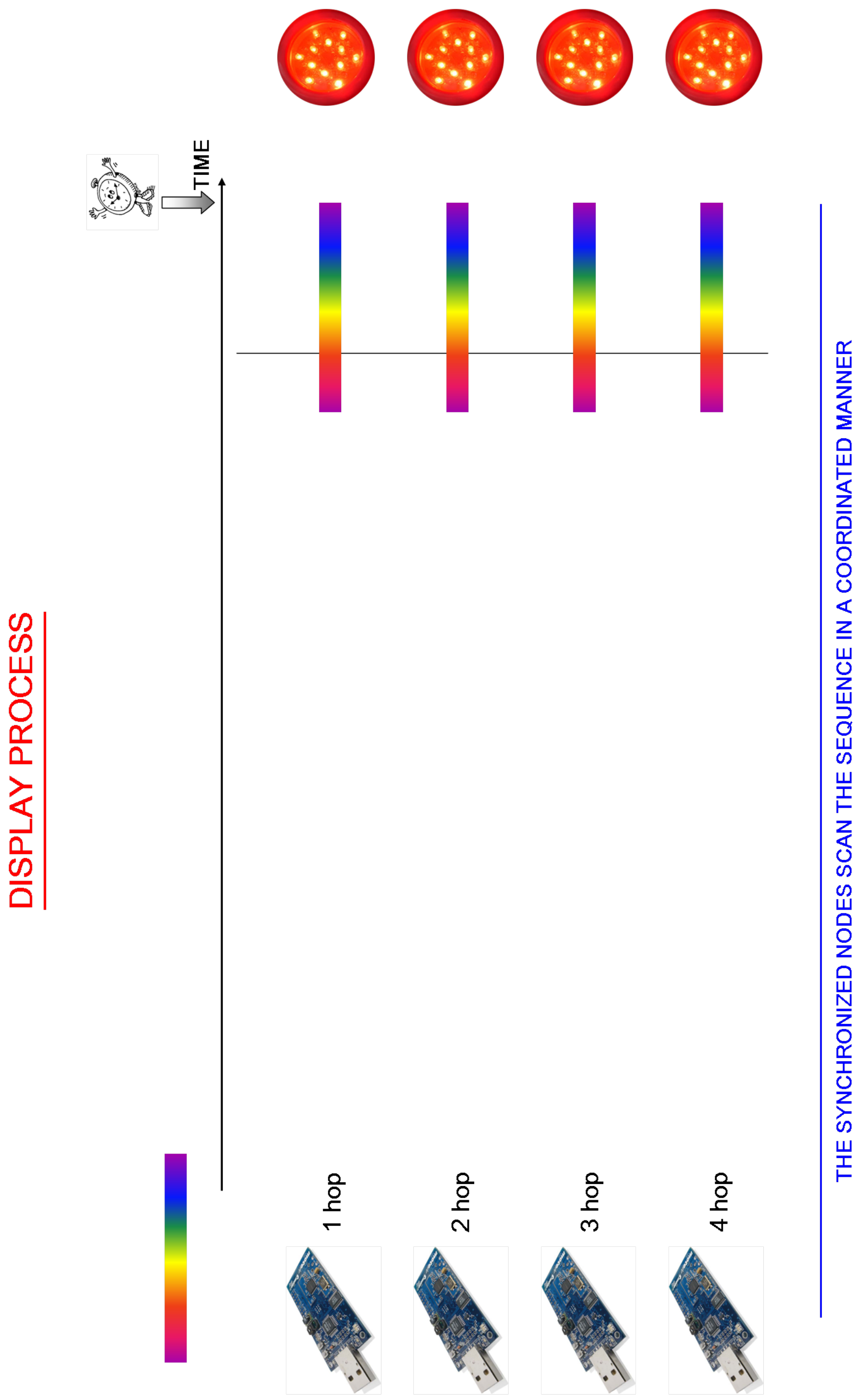

FLOODING COMMUNICATION PROCESS
DEADLINE
TIME
1 hop
2 hop
3 hop
4 hop
THE PORTION AND ITS DEADLINE ARE FORWARDED FROM HOP 2 TO HOP 3


FLOODING COMMUNICATION PROCESS
DEADLINE
TIME
1 hop
2 hop
3 hop
4 hop
THE PORTION AND ITS DEADLINE ARE COMMUNICATED TO ALL THE WSN NODES


DISPLAY PROCESS
TIME
THE SEQUENCE DEADLINE OCCURS
1 hop
2 hop
3 hop
4 hop


DISPLAY PROCESS
TIME
1 hop
2 hop
3 hop
4 hop
THE SYNCHRONIZED NODES SCAN THE SEQUENCE IN A COORDINATED MANNER


DISPLAY PROCESS
TIME
1 hop
2 hop
3 hop
4 hop
THE SYNCHRONIZED NODES SCAN THE SEQUENCE IN A COORDINATED MANNER


DISPLAY PROCESS
TIME
1 hop
2 hop
3 hop
4 hop
THE SYNCHRONIZED NODES SCAN THE SEQUENCE IN A COORDINATED MANNER

Acknowledgements

First of all I would like to express my sincere gratitude to my supervisor, Prof. Luca Schenato. He provided me with many helpful suggestions and constant encouragement during the course of this work.
My special appreciation goes to my parents, Danilo and Renata, for the support they provided me through my entire life. Particular thanks to my sister Silvia, my brother Marco, Tania and my little nephews Matteo and Edoardo. My family has always supported and encouraged me to do my best in all matters of life.
I want to express my gratitude to my uncle Daniele Burattin for his precious help and sustain. A thanks also to those who have believed in me.
I would like to thank my fellow students and friends Vincenzo Maria Cappelleri, Carlo Alberto Cazzuffi and Riccardo Levorato for putting up with me all these years. Thanks also to my friend Francesco Roveron for all the days spent together in navlab, his companionship revitalizes me.
I wish to thank even my friend for life Michele Costola. I hope that our friendship will continue forever.
Lastly, and most importantly, I wish to thank my love Marta for the very special person she is. Without her love, her help and her incredible amount of patience, my studies would not have been completed. A special thanks also to her lovely family.