Data-Centric Architecture for Wireless Sensor Networks

Stefan Octavian Dulman

Printed by Ipskamp PrintPartners, Enschede, the Netherlands.
ISBN 90-365-2262-5

DATA-CENTRIC ARCHITECTURE FOR WIRELESS SENSOR NETWORKS

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday 6 October 2005 at 16.45

by

Stefan Octavian Dulman

born on 11 September 1977

in Iasi, Romania

This dissertation is approved by:

Prof. Dr. Ir. Thijs Krol (promotor)

Acknowledgments

Before going into the details of the research I have performed in the last three years, it is my pleasure to write down a few lines about the people who were near me and supported me during this time. I would like to thank them for helping me with my work and for bringing joy into my life. They certainly made my PhD a nice and interesting experience.

I dedicate this thesis to my parents. They were there to listen to me each time I needed them. We shared together the best moments as well as the not so good ones. They always had a piece of good advice for me (even if I complained and ignored it for the moment - just to come back to it later). I thank them for motivating me and keeping me on the right track. I hope that we will be able to spend more time together instead of the weekly phone call and the pictures taken in the exotic parts of this world.

I would like also to thank Paul in the first place for his tremendous patience with me (my strict and early office hours, postponing a few important emails in order to finish computing some useless coefficient, my usual delays to the ”deadline tomorrow” tasks and not only, etc. would have driven anyone else crazy). Not only that he guided me in this new field of research and helped me gather a tremendous amount of information, but he knew how to make our working environment a friendly and fun place. He appreciated my ideas and encouraged me to continue with them - this thesis would have never happened if it wasn’t for him. The same holds for my promotor Thijs Krol, who, in the first place, gave me the possibility of completing my studies in his research group.

An important part of my research activities were triggered and sustained by the long discussions with Pierre. The data centric architecture as a concept was born from the operating system we developed together on top of his scheduling schemes. I learned a lot from him about operating systems (although I am still not fully convinced yet of that linux-windows issue). I cannot pass over without mentioning Michelle and the nice research group he belongs to in Ferrara. The second part of the thesis, my current interests and certainly my future interests were heavily influenced by our discussions! A lot of other people gave me feedback on my work or joined me in my research or just simply helped me with the

daily tasks (thanks Marlous and Nicole!).

The working environment picture is far from being complete if I do not mention EYES. While this acronym may represent for some just the name of yet another European Project, it brings lots of common memories for me and the few ones known as the EYES group in our department - one can identify us by observing our positive reaction to the motivating call ”EYES meeting!” (in the form of ”nothing to report”). If still uncertain try the keywords: ”deliverable” or ”review a paper”.

I will briefly mention the other few guys with whom I shared the EYES room(s) for half of the time (and Paul’s office during the other half): Tim is one of the oldest in the ”EYES business” (I will still ask for his opinion on things I prove by simulations). Lodewijk came afterwards. He patiently listened and tried to answer to all our complaints about the Netherlands (he could actually rewrite ”The Undutchables” from the perspectives of a foreigner). Jian gave us the first and most important Chinese lesson. Supriyo carefully supervised me with my experiments involving plants and bellow-zero temperatures (shall we write a project proposal about them?). Tjerk survived to my extreme supervising skills and decided to join us in the new larger ex-EYES room (and is abusing of our privacy, placing all over the place small devices that are ”pervasive without being intrusive”). Together we had (and will still have!) a nice time together. I will remember our dart games (despite our continuously dropping skills and some guy always cheating), the trips to exotic places (thanks Paul for teaching us the important ”your tax money at work” concept!), the various group outings and so on.

Studies were only part of the life in Twente. I cannot forget the continuously changing group of international students with whom I spent many afternoons and weekends. Tennis table, skating, soccer, basketball, swimming, volleyball, outings, parties, trips, movies, games, etc. - just enough things to consume all the available spare time I had and make me wish I had more. The names are simply too many to list all of them in here, and even then I would not be sure if I didn’t miss anyone.

A special place in my social life is taken by the Romanian group of friends, all over this world. Mihai, Raluca, Diana, Andrei, Geo, Angela, Eugen, Codrin to name a few, I am so lucky to have met you and have you as my friends. Thank you Ileana for being such a wonderful person and for showing me that special things do exist in this world!

Unfortunately, I have to stop here with the acknowledgments. It can easily take a book to describe the past three years that led to the completion of this work only. My PhD training was a very interesting and rewarding experience especially due to the extraordinary people that surrounded me. Thank you all!

Abstract

Ad-hoc and sensor networks have gained a lot of attention lately. Due to technological advances, building small-sized, energy-efficient reliable devices, capable of communicating with each other and organizing themselves in ad-hoc networks has become possible. These devices have brought a new perspective to the world of computers as we know it, pushing us into what can be called the third era of computing: intelligent devices can be embedded into the environment, assisting the user in performing various tasks while being invisible to him. The need for reconfiguration and maintenance disappears as the networks organize themselves to adapt to the continuously changing environment and requirements.

Wireless sensor networks is the generic name under which a broad range of devices hide. Basically, any collection of devices equipped with a processor, having sensing and communication capabilities and being able to organize themselves into a network created in an ad-hoc manner falls into this category. The addition of the wireless communication capabilities to sensors increased their functionality dramatically. Wireless sensor networks bring monitoring capabilities that will forever change the way in which data is collected from the ambient environment.

This thesis focuses on the topic of finding the right architecture for building a wireless sensor network in an efficient manner. The choice for a specific architecture is usually one of the most debated issues at the beginning of each research project and the various trade-offs decided upon at this early step have important consequences on the performances of the final system.

Although a lot of effort goes into defining the right architecture to be adopted, the shortcomings of the initial decision will make themselves visible during any project. Unfortunately, changing the characteristics of the architecture after a project has started costs a lot of resources and is often not possible. The reason for this is because the new characteristics affect or completely change the hypotheses on which most of the building blocks of that system are built, changing also the parameters of their output and in general asking for a complete new design.

As the sensor networks is a new field, the initial research has the characteristics of the related fields from which the researchers emerged. Approaches specific

to theoretical computer science, telecommunications, computer networks architectures, databases, etc. have been employed.

Currently, none of the solutions we are aware of offers flexibility and easy adaptation of the sensor network to the continuously changing environment. In the vast majority of the scenarios, a layered protocol stack is adopted and a lot of effort is spent in each project to enable the cross layer communication and decision taking needed by this category of systems.

Another major disadvantage of the existing systems is that all of the architectures are adopted with the generalization in mind (the architecture should be as general as possible and fit many scenarios) but often the projects end offering a modified version of it that fits only the particular application of the demonstrator. This is hardly reusable and thus contains various unnecessary trade-offs that limit the performances of the system.

Contributions of this thesis

The major contribution of this thesis is the design of *a new flexible data centric architecture* specially tailored to fit the needs of wireless sensor networks. It allows the reconfiguration at any time of the internal structure at the cost of a insignificant overhead in terms of memory consumption and execution time overhead. The sensor nodes can be reprogrammed and can reconfigure themselves to adapt to the requirements of the environment in which they are deployed.

The new architecture is just a theoretical concept. To show that it is also feasible we have designed *a new operating system for wireless sensor networks* on top of it. We have shown that the often completely different concepts of operating system and system architecture can be combined in a single entity, reducing a lot of overhead and allowing implementation of new features otherwise thought not feasible with the amount of resources available. In parallel, we developed also *a simulator* for a distributed network in close relationship with the operating system, built also on top of the data centric architecture.

Having the support for building new applications ready, we focused on the particularities of a building block necessary in most of the wireless sensor networks applications: the positioning protocols. We have implemented some already existing algorithms in our framework, showing that the concept of dynamic configuration holds. Out of the experiences we had with the *localization protocols* and due to the tremendous amount of interest in this topic from the industry side, we conducted further research into it, being able to significantly improve the existing protocols and develop new ones. We have dedicated a separate chapter to this issue, presenting three of our new contributions.

The field of wireless networked sensors is still a young, new field. There are a lot of questions to be answered at all the levels. Theoretical research and fancy application domains will answer questions such as which is the best routing protocol to be used, what trade-offs need to be made to acquire a smaller latency, how to combine and present the pieces of data such that the big picture makes sense, etc.

Contents

Chapter 1

Introduction

Embedded systems have gained a lot of attention lately. Due to technological advances, building small-sized, energy-efficient reliable devices, capable of communicating with each other and organizing themselves in ad-hoc networks has become possible. These devices have brought a new perspective to the world of computers as we know it, pushing us into what can be called the third era of computing: intelligent devices can be embedded into the environment, assisting the user in performing various tasks while being invisible to him. The need for reconfiguration and maintenance disappears as the networks organize themselves to adapt to the continuously changing environment and requirements.

This thesis is dedicated to finding the right architecture for the class of embedded systems known as *wireless sensor networks*. While having the simple task of gathering the most basic information, these systems can pose very complex design challenges because of the limited quantity of resources available. The proposed architecture was the foundation for a novel operating system, a simulation framework for sensor networks and was shown to be able to support basic building blocks for such systems (as position finding algorithms). The theoretical results are confirmed in practice by several working prototypes.

The work presented in this thesis has been supported by a number of research projects focusing on various design challenges in wireless sensor networks and embedded systems in general. The list of projects includes: EYES [13], Consensus [11], Smart Surroundings [14], Cobis [10] and Embedded Wisents [12].

This chapter will give a brief overview of the area of ubiquitous computing (Section 1.1), focusing on embedded systems and especially on the particular topic of wireless sensor networks (Section 1.2). Then, a number of applications as well as possible typical scenarios will be presented in order to better understand the field of application of this new emerging technology (Section 1.3).

Wireless sensor networks bring lots of challenges and often contradictory de-

mands from the design point of view. The last part of the chapter is dedicated to highlighting the main directions of research involved in this field (Section 1.4). The particular topics described in the thesis and the contribution of this thesis are detailed in Section 1.5 while Section 1.6 contains an overview of the way the information is structured through the chapters of the book.

1.1 Ubiquitous computing

Things are continuously changing in the world of computers. The evolution of computing systems is interesting and surprisingly unpredictable, with new devices and technologies emerging and replacing the old settled-down conception.

When the first usable computer became available, the mainframe era was born: some thirty years ago, these huge devices were widely deployed within universities. Lots of users made use of a single mainframe computer which they had to share among themselves. The computation power came together with a high cost and a huge machine requiring quite a lot of maintenance.

Technology advanced according the the lines predicted by Moore's law (*the number of transistors on chip doubles each 18 months*) and we stepped into the second era of computers. It is a period that is still present today, but which is slowly approaching its final part. It is the era of the personal computers, cheaper and smaller, and increasingly affordable. Quite often, the average user has access to and makes use of more than one computer, these machines being present now in almost any home and work place.

But, in this familiar environment things are starting to change and the third era of computing gains more and more terrain each day. Let us take a look at the main trends today. The technology advancements cause the personal computers to become smaller and smaller. The desktop computers tend to be replaced by laptops and other portable devices.

The main factor that is influencing the new transition is the availability of wireless communication technology. People are getting rapidly used to wireless communicating devices due to their independence from fixed machines. The success and availability of the Internet brought even more independence to the user: the data could now be available regardless of the physical location of its owner.

The advancements in technology did not stop here: the processors became small and cheap enough to be found now in almost any familiar device around us, starting with an every-day watch and ending with (almost) any home appliance we own. The new efforts nowadays are to make these devices ”talk” to each-other and organize themselves into ad-hoc networks to accomplish their design goal as fast and reliably as possible.

This is, in fact, the third computer age envisioned two decades ago by Mark Weiser. He was writing in *The computer for the 21st century* [22]:

> *My colleagues and I at PARC think that the idea of a "personal" computer itself is misplaced, and that the vision of laptop machines, dynabooks and "knowledge navigators" is only a transitional step toward achieving the real potential of information technology. Such machines cannot truly make computing an integral, invisible part of the way people live their lives. Therefore we are trying to conceive a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background.*

Several names such as: *ubiquitous computing, pervasive computing, ambient intelligence, invisible computing, disappearing computer, smart surroundings* etc. were created to indicate different aspects of the new computing age. The ubiquitous computing world brings a reversed view on the usage of computing power: instead of having lots of users gathered around the mainframe computer, now, each user will be using the services of several embedded networks. The user will be in the middle of the whole system, surrounded by an invisible intelligent infrastructure. The original functionality of the objects and application will be enhanced, and a continuous interaction will be present in a large variety of areas of daily life.

1.2 What are wireless sensor networks?

So what are wireless sensor networks and where is their place in this new environment that starts "growing" around us?

Wireless sensor networks is the generic name under which a broad range of devices hide. Basically, any collection of devices equipped with a processor, having sensing and communication capabilities and being able to organize themselves into a network created in an ad-hoc manner falls into this category.

The research in the field of wireless sensor networks increased tremendously after the year 2000. The initial project was Smart Dust [16] that has shown that building cheap smart sensors that can network together is possible. A large number of projects was initiated, the main support coming initially from the DARPA Agency. Initially results were borrowed from the ad-hoc computing community but soon it was realized that new approaches and algorithms were needed.

The addition of the wireless communication capabilities to sensors increased their functionality dramatically. Wireless sensor networks bring monitoring ca-

pabilities that will forever change the way in which data is collected from the ambient environment. Let us take, for example, the traditional monitoring approach of a remote location for a given phenomenon such as recording the geological activity, monitoring the chemical or biological properties of a region, or even monitoring the weather at a certain place.

The old approach was the following: rather big and robust devices needed to be built. They should have contained besides the sensor pack itself, a big power supply and local data storage capabilities. A team of technicians would have to travel together to the destination to be monitored, place these expensive devices at predefined positions and calibrate all the sensors. Then, they would come back after a certain amount of time in order to collect the sensed data. If by misfortune some hardware would fail, then nothing could be done for it, as the information about the phenomenon itself would be lost.

The new approach is to construct inexpensive, small sized, energy-efficient sensing devices. As hundreds, thousands or even more of these devices will be deployed, the reliability constraints for them will be diminished. No local data storage is needed anymore as they will process locally and then transmit by wireless means the observed characteristic of the phenomenon to one or more access points connected to a computer network. Individual calibration of each sensor node is no longer needed as it can be performed by localized algorithms [23]. The deployment will also be easier, by randomly placing the nodes (e.g. simply throwing them from a plane) onto the monitored region.

Having this example in mind, we can give a general description of a sensor node. Throughout this thesis, the name *sensor node* will be used to describe a tiny device that has a short range wireless communication capability, a small processor and several sensors attached to it. It may be powered by batteries and its main function is to collect data from a phenomenon, collaborate with its neighbors and forward its observations (pre-processed version of the data or even decisions) to the endpoint if requested. This is possible because its processor additionally contains the code that enables inter-node communication and setting-up, maintenance and reconfiguration of the wireless network. When referring to wireless communication, we have in mind mainly radio communication (other means as ultrasound, visible or infrared light, etc. are also being used [21]). A *sensor network* is a network made up of large numbers of sensor nodes. By a large number we understand at this moment hundreds or thousands of nodes but there are no exact limits for the upper bound of the number of sensors deployed.

Wireless sensor networks are one of the most important tools of the third era of computing. They are the simplest intelligent devices around, having as their main purpose monitoring the environment surrounding us and alerting us of the main events happening. Based on the observation reported by these instruments,

Project name	**Research area**
CoSense [4]	collaborative sense making
Consensus [11]	collaborative sensor networks
CoBis [10]	collaborative business items
EYES [13]	self-organizing, energy-efficient sensor networks
PicoRadio [8]	develop low cost, energy-efficient transceivers
SensoNet [15]	protocols for sensor networks
Smart Dust [16]	cubic millimeter sensor nodes
Smart Surroundings [14]	architectures and frameworks for future ambient systems
TinyDB [20]	query processing system
WINS [24]	distributed network access to sensors and processors
Wisents [12]	cooperative embedded systems for exploration and control

Table 1.1: List of some research projects related to sensor networks

humans and machines can make decisions and act on them.

1.3 Typical scenarios and applications

At this moment a large variety of sensors exists. Sensors have been developed to monitor almost every aspect of the ambient world: lighting conditions, temperature, humidity, pressure, the presence of absence of various chemical or biological products, detection of presence and movement, etc. By networking large number of sensors and deploying them inside the phenomenon to be studied we obtain a sensing tool capable of constructing a tridimensional map of the sensed phenomenon, thus more powerful than a single punctual sensor.

The sensor networks field is still rapidly evolving. Although a large number of sensor network prototypes exists at this moment, the possible application areas are still being explored. The typical application one can think of has as the main goal some sort of monitoring (the most common one is environmental monitoring). Nevertheless, stand-alone applications such as position finding or collaborative business items have been suggested and explored [10].

In the following we will present two classifications that have been already proposed in the literature. The first classification of wireless sensor networks takes into consideration the complexity of the network involved, this being related more or less to the surface covered by the deployed network. Thus, we can distinguish three main categories (adapted from [6]):

- *Small scale applications: intelligent warehouse*

  Each item contained inside the warehouse will have a tag attached, that will

be monitored by the sensor nodes embedded into the walls and shelves. Based on the read data, knowledge of the spatial positioning of the sensors and time information, the sensor network will offer information about the traffic of goods inside the building, create automatic inventories, and even perform long term correlations between the read data. The need of manual product scanning thus disappears. In this category we can include the scenario of the modern supermarket, where the selected products of the customers will automatically be identified at the exit of the supermarket. This scenario also has the minimum complexity. The sensor nodes are placed at fixed positions, in a more or less random manner. The deployment area is easy accessible and some infrastructure (e.g. power supplies and computers) already exists. At the same time, the nodes are operating in a ”safe” environment meaning that there are no major external factors that can influence or destroy them.

- *Medium-large scale applications: environmental monitoring*

  Environmental monitoring is the widest area of applications envisioned up to now. A particular application in this category is disaster monitoring. The sensor nodes deployed in the affected areas can help humans estimate the effects of the disaster, build maps of the safe areas and direct the human actions towards the affected regions. A large number of applications in this category address monitoring of the wild life. This scenario has an increased complexity. The area of deployment is no longer accessible in an easy manner and no longer safe for the sensor nodes. There is hardly any infrastructure present, nodes have to be scattered around in a random manner and the network might contain moving nodes. Also a larger number of nodes will have to be deployed.

- *Very large scale sensor networks applications*

  The scenario of a large city where all the cars have integrated sensors. These sensor nodes will communicate with each other collecting information about the traffic, routes and special traffic conditions. On one hand, new information will be available to the driver of each car. On the other hand, a global view of the whole picture will also be available. The two main constraints that characterize this scenario are the large number of nodes and their high mobility. The algorithms employed will have to scale well and deal with a network with a continuously changing topology.

On the other hand, the authors of [1] present a classification of sensor networks based on their area of application. It takes into consideration only the military, environment, health, home and other commercial areas and can be extended with

additional categories such as space exploration, chemical processing and disaster relief.

- *Military applications*

  Factors as rapid deployment, self-organization and increased fault tolerance make wireless sensor networks a very good candidate for usage in the military field. They are suited to deployment in battlefield scenarios due to the large size of the network and the automatic self-reconfiguration at the moment of the destruction/unavailability of some sensor nodes [3]. Typical applications are: the monitoring of friendly forces, equipment and ammunition; battlefield surveillance; reconnaissance of opposing forces and terrain, targeting, battle damage assessment; and nuclear, biological and chemical attack detection and reconnaissance. A large number of projects have already been sponsored by The Defense Advanced Research Projects Agency (DARPA) [5].

- *Environmental applications*

  Several aspects of the wildlife are being studied with the help of sensor networks. Existing applications include the following: monitoring the presence and the movement of birds, animals and even insects; agricultural related projects observing the conditions of crops and livestock; environmental monitoring of soil, water and atmosphere contexts and pollution studies; etc. Other particular examples include forest fire monitoring, bio-complexity mapping of the environment and flood detection. Ongoing projects at this moment include the monitoring of birds on Great Duck Island [9], the zebras in Kenya [7] or the redwoods in California [25]. The number of these applications is continuously increasing as the first deployed sensor network show the benefits of easy remote monitoring.

- *Health-care applications*

  An increasing interest is being shown to the elder population [17]. Sensor networks can help in several areas of the health-care field. The monitoring can take place both at home and in hospitals. At home, patients can be under permanent monitoring and the sensor networks will trigger alerts whenever there is a change in the state of the patient. Systems that can detect their movement behavior at home, detect any fall or remind them to take their prescriptions are being studied. Also inside the hospitals sensor networks can be used in order to track the position of doctors and patients (their status or even errors in the medication), expensive hardware, etc. [2].

- *Home applications*

Company name	**Web address**
Ambient Systems (NL)	http://www.ambient-systems.net
CrossBow (USA)	http://www.xbow.com
Digital Sun (USA)	http://www.digitalsun.com
Dust Networks (USA)	http://dust-inc.com
Ember (USA)	http://www.ember.com
Invocon Inc. (USA)	http://www.invocon.com
MicroStrain (USA)	http://www.microstrain.com
Millennial Net (USA)	http://www.millennial.net
Sensite Solutions (NL)	http://www.sensite-solutions.com
Sensoria Corporation (USA)	http://www.sensoria.com
Xsilogy (USA)	http://www.xsilogy.com

Table 1.2: Current sensor networks companies list

The home is the perfect application domain for the pervasive computing field. Imagine all the electronic appliances forming a network and cooperating together to fulfill the needs of the inhabitants [19]. They will have to identify each user correctly, remember their preferences and their habits and, at the same time, monitor the entire house for unexpected events. The sensor networks have also an important role here, being the *eyes and the ears* that will trigger the actuator systems.

- *Other commercial applications*

  This category includes all the other commercial applications envisioned or already built that do not fit in the previous categories. Basically they range from simple systems as environmental monitoring within an office to more complex applications such as managing inventory control and vehicle tracking and detection. Other examples include incorporating sensors into toys and thus detecting the position of the children in ”smart” kindergartens [18]; monitoring the material fatigue and the tensions inside the walls of a building, etc.

The number of research projects dedicated to wireless sensor networks has increased dramatically over the last years. A lot of effort has been invested in studying all possible aspects of wireless sensor networks. Please refer to Table 1.1 for a few examples. Also, a number of companies were created, most of them start-ups from the universities that perform research in the field. Some of the names in the field, valid at the date of writing this document, are listed in Table 1.2.

We synthesized the most common applications in Table 1.3. For each application, its characteristics are described and a list of needed services is provided.

Application	**Characteristics**	**Required services**
Green-house monitoring	small scale network, resource poor nodes, static scenarios, fine grained deployment, star networks, easy to maintain	inaccurate localization and timing, low quality sensors, fixed sampling data rates, low in-network processing
Wild life monitoring	medium scale network, resource poor nodes, mobile scenarios, coarse grained deployment, multihop networks, difficult to maintain	precise localization and timing, local storage, fixed sampling rates, medium in-network processing
Home and office monitoring	small scale network, resource poor nodes, static scenarios, coarse grained deployment, multihop network, easy to maintain	inaccurate localization and timing, low quality sensors, fixed sampling data rates, low in-network processing, security and privacy
Monitoring as military application	large scale network, resource rich nodes, static scenarios, coarse grained deployment, multihop networks, difficult to maintain	very precise localization and timing, high quality sensors, dynamic sampling rates, intense in-network processing, local storage systems, security and privacy
Warehouse monitoring	large scale network, resource poor nodes, mobile scenarios, fine grained deployment, multihop networks, easy to maintain	very precise localization and timing, low quality sensors, high in-network processing, security and privacy

Table 1.3: Sensor networks specific applications

If we take a look at the number of sensors deployed with respect to the area covered, we can give the following categorization:

- *Coarse grained sensor networks*

  In this category usually fall the sensor networks made up of devices, each covering a large area. These devices are usually large and expensive, because they are equipped with high quality sensors. The network topology is usually a star topology. The sensor nodes themselves are fixed.

- *Fine grained sensor networks*

  This category comprises the networks made up of large number of cheap devices, equipped with low quality sensors having small amounts of resources available. The network topology is usually a multihop network. The large number of sensors and the dense deployment compensates the low quality

of the sensors, the network as a whole producing high quality results.

1.4 Directions of research

The field of sensor networks is a relatively new one. Scientists from various communities approached this research area with enthusiasm and brought together knowledge from various domains of computer science, electrical engineering, telecommunications, etc. The initial directions of research were specific to each of these fields, everyone trying to adapt their knowledge to make wireless sensor networks a reality.

Several approaches were taken, but research focused on a few main directions recognized by the community as being the key issues in the sensor network field. We can name the following major topics of interest, based on the amount of effort and publications devoted to them:

- *Hardware platforms*

  The vision of sensor networks states that large numbers of energy-efficient devices, able to communicate by wireless means will be deployed and left to work unattended for years. This imposes a large range of challenges from hardware design point of view. First of all, the availability of low power wireless communication means (most often radios but also ultrasound, light or magnetic induction based devices) with low transmission ranges and fast switching times between the various possible states is a must. Additionally what is needed is a low power and low cost processor and a set of efficient sensors. Advances in all the fields related to developing and improving these devices are needed and followed closely by the wireless sensor networks community.

- *Networking protocols*

  To be of any use, the sensor network needs a set of networking protocols (media access control, routing, clustering, etc.), robust in front of the multiple existing sources of failure. Various degrees of redundancy (often dynamically decided upon) need to exist to assure the correct functioning of the network. Intelligent network protocols can save a lot of energy and can prolong the lifetime of the network at various degrees. The first approach taken in this field was to emulate the existing algorithms from the wired computer networks but it has been seen that new approaches needed to be taken.

- *Specific protocols*

In general, large arrays of data make no sense to anyone if they are not stamped with the (more or less exact) time and location of the sensing process. This simple observation brings into discussion two new basic protocols, that are needed in the vast majority of applications: a timing and synchronization protocol and a localization protocol. Soon after these blocks gained attention it was also noticed that the networking protocols can be redesigned and transformed into more efficient ones if the notions of time and position were locally known for each sensor node (e.g. superior time based media access control protocols can be designed, efficient geographical based routing can be employed, etc.). The availability of localization algorithms brought into attention localization as a stand alone application provided by sensor networks as an alternative to the not everywhere available and often expensive Global Positioning System.

- *Data acquisition, manipulation and storage*

  The main goal of the sensor networks is to monitor particular features of the environment they are deployed in. This requirement translates to huge quantities of data being sensed (produced) by each node, that reach the limits of the storage systems and available bandwidth in the network. The only way for the data dissemination to work is to dynamically adapt the sensing periods, to locally process the data and to aggregate the pieces of data traveling to a certain destination (sensor fusion). Querying a distributed network in an efficient way and specifying the best way to aggregate data on the way back to the requesting gateway is a research question still being opened.

- *High level dissemination*

  Once the network is setup, the question arises on how to make use of the available data. A layer of so called *distributed services* is added on top of the networking layer dealing with issues such as informing the user what is the network capable of doing (services discovery). This management layer is compulsory and opens the spectrum of a large range of applications closer to the business domain. The availability of sensing (and actuating on) the environment and of connections to the high speed networks makes possible applications such as the collaborative business items.

- *Application domain*

  The application domain is also a topic of research as well in this new research field. The first prototypes have shown environmental monitoring and military surveillance as the specific applications for wireless sensor networks. Soon afterward it was realized that the application class of sensor networks can be extended to more than basic sensing (see Section 1.3 for

possible scenarios). Completely new applications (such as a localization devices) came into play as well. At this moment, the range of possible applications is largely unexplored and it gets larger and larger with the availability of more prototypes and interaction with the industry and business community.

Our research focused mainly on two of the previous categories: the networking layers and the specific protocols. The other ones were also in attention as the cross layer design and communication is one of the basic requirements of sensor networks.

1.5 Topics and contributions of this thesis

This thesis focuses on the topic of finding the right architecture for building a wireless sensor network in an efficient manner. The choice for a specific architecture is usually one of the most debated issues at the beginning of each research project and the various trade-offs decided upon at this early step have important consequences on the performances of the final system.

Although a lot of effort goes into defining the right architecture to be adopted, the shortcomings of the initial decision will make themselves visible during the project. Unfortunately, changing the characteristics of the architecture after the project has started costs a lot of resources and is often not possible. This because the new characteristics affect or completely change the hypotheses on which most of the building blocks of that system are built, changing also the parameters of their output and in general asks for a complete new design.

As the sensor networks is a new field, the initial research has the characteristics of the related fields from which the researchers emerged: approaches specific to theoretical computer science, telecommunications, computer networks architectures, databases, etc. have been employed when the architectures of the large number of prototypes already deployed have been adopted (see Chapter 3).

Currently, none of the solutions we are aware of offers flexibility and easy adaptation of the sensor network to the continuously changing environment. In the vast majority of the scenarios, a layered protocol stack is adopted and a lot of effort is spent in each project to enable the cross layer communication and decision taking needed by this category of systems. Current applications are rather static: they are designed to serve on particular purpose (they are made up of a small number of nodes, nodes are placed at fixed positions, etc.). For most of these, we cannot talk basically about a clear architecture.

Another major disadvantage of the existing systems is that all of the architectures are adopted with the generalization in mind (the architecture should be as

general as possible and fit many scenarios) but often the projects end offering a modified version of it that fits only the particular application of the demonstrator. This is hardly reusable and thus contains various unnecessary trade-offs that limit the performances of the system.

The major contribution of this thesis is the design of a new flexible data centric architecture special tailored to fit the needs of wireless sensor networks. It allows the reconfiguration at any time of the internal structure at the cost of a insignificant overhead in terms of memory consumption and execution time overhead. The sensor nodes can be reprogrammed and can reconfigure themselves to adapt to the requirements of the environment in which they are deployed (see Chapter 3).

The new architecture is just a theoretical concept. To show that it is also feasible we have designed a new operating system for wireless sensor networks on top of it. We have shown that the often completely different concepts of *operating system* and *system architecture* can be combined in a single entity, reducing a lot of overhead and allowing implementation of new features otherwise thought not feasible with the amount of resources available. In parallel, we developed also a simulator for a distributed network in close relationship with the operating system, built also on top of the data centric architecture.

Having the support for building new applications ready, we focused on the particularities of a building block necessary in most of the wireless sensor networks applications: the positioning protocols. We have implemented some already existing algorithms in our framework, showing that the concept of dynamic configuration holds. Out of the experiences we had with the localization protocols and due to the tremendous amount of interest in this topic from the industry side, we conducted further research into it, being able to significantly improve the existing protocols and develop new ones. We have dedicated a separate chapter to this issue, presenting three of our new contributions.

1.6 Thesis organization

The thesis is organized in the following manner: Chapter 2 presents the status of the work performed in the field of sensor networks with a focus on the existing architectures. The main challenges will be described and the way in which various existing systems try to solve those. Chapter 3 presents the new data centric architecture we propose. We will show how the flexible flexible architecture can be used to avoid some of the shortcomings of the existing systems.

The concept of the data centric architecture was the starting point for a new simulator and a novel operating systems, both of them targeting wireless sensor networks. They will be described and analyzed in detail in Chapter 4. As we will

show, both tools provide features that no other currently existing system has and allow fast design, testing and implementation of complex protocols for wireless sensor networks.

Chapter 5 describes one of the most important building blocks of wireless sensor networks: positioning algorithms.Three new contributions are described and the advantages and shortcomings of the new algorithms are shown and compared against already existing algorithms.

The thesis ends with conclusions and the description of a future roadmap for designers of wireless sensor networks.

Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Elsevier) Journal*, 38(4):393–422, March 2002.

[2] H. Baldus, K. Klabunde, and G. Muesch. Reliable set-up of medical body-sensor networks. In *Proc. Wireless Sensor Networks, First European Workshop (EWSN 2004)*, Berlin, Germany, January 2004.

[3] R.R. Brooks, P. Ramanathan, and A.M. Sayeed. Distributed Target Classification and Tracking in Sensor Networks. *Proceedings of the IEEE*, 91(8):1163–1171, August 2003.

[4] CoSense. http://www2.parc.com/spl/projects/ecca.

[5] DARPA. http://www.darpa.mil/body/off_programs.html.

[6] D. Estrin, R. Govindan, J.S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.

[7] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *ASPLOS-X*, San Jose, California, October 2002.

[8] Picoradio. http://bwrc.eecs.berkeley.edu/Research/Pico_Radio.

[9] J. Polastre, R. Szewczyk, and D. Culler. *Wireless sensor networks*, chapter Analysis of Wireless Sensor Networks for Habitat Monitoring. Editors: C.S. Ragavendra, K.M. Sivalingam and T. Znati, Kluwer Academic Publishers, 2004.

[10] Cobis European Project. http://www.cobis-online.de.

[11] Consensus Dutch Project. http://www.consensus.tudelft.nl.

[12] Embedded Wisents European Project. http://www.embedded-wisents.org.

[13] European EYES Project. http://www.eyes.eu.org.

[14] Smart Surroundings European Project. http://www.smart-surroundings.org.

[15] SensoNet. http://users.ece.gatech.edu/w̃eilian/Sensor/index.html.

[16] SmartDust. http://robotics.eecs.berkeley.edu/ pister/SmartDust.

[17] IEEE Computer Science Society. Pervasive computing vol. 3 nr. 2 - successfull aging, April-June 2004.

[18] M. Srivastava, R. Muntz, and M. Potkonjak. Smart kindergarten: Sensor-based wireless networks for smart developmental problem-solving environments (challenge paper). In *Proc. 7th Ann. Intl. Conf. on Mobile Computing and Networking*, pages 132–138, Rome, Italy, July 2001. ACM.

[19] T.Basten, M.Geilen, and H.D. Groot. *Ambient Intelligence: Impact on embedded system design*, chapter Omnia fieri possent, pages 1–8. Editors: T. Basten, M. Geilen and H.D. Groot, Kluwer Academic Publishers, 2003.

[20] TinyDB. http://telegraph.cs.berkeley.edu/tinydb.

[21] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *Acm Transactions on Information Systems, vol. 10, no. 1*, pages 91–102, January 1992.

[22] M. Weisser. The computer for the 21st century. *Scientific American*, 1991.

[23] K. Whitehouse and D. Culler. Calibration as parameter estimation in sensor networks. In *Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, Georgia, September 2002.

[24] Wins. http://www.janet.ucla.edu/WINS.

[25] S. Yang. Redwoods go hightech: researchers use wireless sensors to study california's state tree. UCBerkeley News, July 2003.

Chapter 2

Current state of research

This chapter provides a brief overview of the state of the art of the wireless sensor network field. The reader is offered a global view of the research area, including a description of the influences of the related fields of science. The chapter also addresses the premises on which this technology is built and presents the major existing research trends in the area.

We have started working in the field of wireless sensor networks in a moment when sensor networks were just a little more than a concept. The first energy-efficient hardware device containing an integrated radio and a processor was produced (as part of the SmartDust set of projects in 2001) and the idea of having thousands of miniaturized versions of this sensor node connected in a network and monitoring the environment around came alive.

The last four years have brought major advancements in the field. The initial knowledge, mere adaptation of theories already established in related fields of science, has evolved to the point where protocols targeting only wireless sensor networks are developed, major universities are offering specialized courses on this new technology and a large number of workshops and conferences appeared. For example, a search on an online book vending site (amazon.com) revealed over one hundred book titles and other publications related to this field.

In this chapter, we will focus on the current state of the art of the research in the wireless sensor networks field. As the amount of related results achieved lately is impressive, we would prefer to give the reader a global picture pointing out the major components and trends rather than a long list of references that might be considered irrelevant in a few years.

We will start this chapter by talking about the challenges the wireless sensor

networks pose. In this way, we fix the major premises of any application running on top of this technology and give a subjective characterization of the tandem available resources - expected goals. Then, we will proceed with splitting the wireless sensor networks into their building blocks and describe them, as well as the major influences from various fields of research and show the existing overlapping. The chapter ends with conclusions and directions for future research.

2.1 Challenges

When designing a wireless sensor network one faces, on one hand, the simplicity of the underlying hardware and, on the other hand, the requirements that have to be met. In order to satisfy them, new strategies and new sets of protocols have to be developed [13, 1, 32]. In the following we will address the main challenges that are present in the wireless sensor network field. The research directions involved and the open questions that still need to be answered will be presented as well.

To begin with, a high-level description of our ultimate constraints for the sensor networks can be synthesized as:

- **Long life** - The sensor node should be able to ”live” as long as possible using its own batteries or other sources of energy. This constraint can be translated to a power consumption less than one hundred micro-watts (meaning a lifetime period of months to years for one device). The condition arises from the assumption that the sensor nodes will be deployed in a harsh environment where maintenance is either impossible or has a prohibitively high price. It makes sense to maximize the battery lifetime (unless the sensor nodes use some form of energy scavenging). The targeted lifetime of a node powered by two AA batteries is a couple of years. This goal can be achieved only by applying a strict energy policy which will make use of power-saving modes.

- **Small size** - The size of the device should be less than one cubic millimeter to make possible its attachment to various sorts of objects. This constraint gave the sensor nodes the name of *smart dust*, a name which gives a very intuitive idea about the final design. The processor and the radio were integrated in a chip having a size of approximately one cubic millimeter. What was left was the antenna, the sensors themselves and the battery. Advances are required in each of these three fields in order to be able to meet this design constraint.

- **Inexpensive** - The third high level design constraint is about the price of these devices. In order to encourage large scale deployment, this technology must be really cheap, meaning that the targeted prices should be in the range of a couple of cents (having a powerful impact on the available resources at each node).

2.1.1 Locally available resources

Wireless sensor networks can consist of thousands of devices working together. Their small size comes also with the disadvantage of very limited resource availability (limited processing power - in the form of a 8 or 16 bits microcontroller, low-rate unreliable wireless communication over a range of a few tens of meters, small memory footprint - in the range of few hundred bytes to a few kilobytes of memory and low energy). This raises the issue of designing a new set of protocols across the whole system.

Energy receives special attention and can by far be considered the most important design constraint. The sensor nodes will be mainly powered by batteries. In most of the scenarios, due to the environment where they will be deployed, it will be impossible to have a human change their batteries. In some designs energy scavenging techniques will also be employed. Still, the amount of energy available to the nodes can be considered limited and this is why the nodes will have to employ energy-efficient algorithms to maximize their lifetime.

By taking a look at the characteristics of the sensor nodes, we notice that the energy is spent for three main functions: environment sensing, wireless communication and local processing. Each of these three components will have to be optimized in order to obtain minimum energy consumption. For the sensing of the environment component, the most energy efficient available sensors have to be used. From this point of view, we can regard this component as a function of a specific application and a given sensor technology.

The energy needed for transmitting data over the wireless channel dominates by far the energy consumption inside a sensor node. More than that, it was previously shown that it is more efficient to use a short-range multihop transmission scheme than sending data over large distances [2]. A new strategy characteristic to the sensor networks was developed based on a trade-off between the last two components and is in fact one of the main characteristics of the sensor networks (see for example techniques developed in: [10, 37]). Instead of blindly routing packets through the network, the sensor nodes will act based on the content of the packet [18].

Let us suppose that a certain event took place. All nodes that sensed it will characterize the event with some piece of data that needs to be sent to the inter-

ested nodes. There will be many similar data packets, or at least, some redundancy will exist in the packets to be forwarded. In order to reduce the traffic, each node on the communication path will examine the contents of the packet it has to forward. Then it will aggregate all the data related to a particular event into one single packet, eliminating the redundant information. The reduction of traffic by using this mechanism is substantial. Another consequence of this mechanism is that the user will not receive any raw data, but only high level characterizations of the events. This makes us think of the sensor network as a self-contained tool, a distributed network that collects and processes information.

From an algorithmic point of view, the local strategies employed by sensor nodes have as a global goal the extension the overall lifetime of the network. The notion of lifetime of the network usually hides one of the following interpretations: one can refer to it as the time passed since power on and a particular event such as: the energy depletion of the first node or of thirty percent of the nodes, or even the moment when the network is split in several sub-networks. No matter which of these concepts will be used, the nodes will choose to participate in the collaborative protocols following a strategy that will maximize the overall network lifetime.

To be able to meet the goal of prolonged lifetime, each sensor node should:

- spend all the idle time in a deep power down mode, thus using an insignificant amount of energy;
- when active, employ scheduling schemes that take into consideration voltage and frequency scaling.

It is interesting to notice at the same time, the contradictory wireless industry trends and the requirements for the wireless sensor nodes. The industry focuses at the moment in acquiring more bits/second/Hz while the sensor nodes need more bits/euro/nJ. From the transmission range point of view, the sensor nodes need only limited transmission range to be able to use an optimal calculated energy consumption, while the industry is interested in delivering higher transmission ranges for the radios. Nevertheless, the radios designed nowadays tend to be as reliable as possible, while a wireless sensor network is based on the assumption that failures are regarded as a regular event.

Energy is not the only resource the sensor nodes have to worry about. The *processing power* and *memory* are also limited. Usually the nodes are equipped with a resource poor 8 or 16 bits microcontroller having at best a few kilobytes of memory available. Large local data storage can not be employed, so strategies need to be developed in order to store the most important data in a distributed fashion and to report the important events to the outside world. A feature that helps dealing with these issues is the heterogeneity of the network. There might be several types of devices deployed. Resource poor nodes will be able to ask

more powerful nodes to perform complicated computations. At the same time, several nodes could associate themselves in order to perform the computations in a distributed fashion.

Bandwidth is also a constraint when dealing with sensor networks. The low power communication devices used (most of the time radio transceivers) can only work in simplex mode. They offer low data rates due also to the fact that they are functioning in the free unlicensed bands where traffic is strictly regulated.

2.1.2 Diversity and dynamics

As we already suggested, there may be several kinds of sensor nodes present inside a single sensor network. We could talk of heterogeneous sensor nodes from the point of view of hardware and software. From the point of view of hardware, it seems reasonable to assume that the number of a certain kind of devices will be in an inversely proportional relationship to the capabilities offered. We can assist to a tiered architecture design, where the resource poor nodes will ask more powerful or specialized nodes to make more accurate measurements of a certain detected phenomenon, to perform resource intensive operations or even to help in transmitting data at a higher distance.

Diversity can also refer to sensing several parameters and then combine them in a single decision, or in other words to perform data-fusion. We are talking about assembling together information from different kinds of sensors like: light, temperature, sound, smoke, etc. to detect for example that a fire has started.

Sensor nodes will be deployed in the real world, most probably in harsh environments. This puts them in contact with an environment that is dynamic in many senses and has a big influence on the algorithms that the sensor nodes should execute. First of all, the nodes will be deployed in a random fashion in the environment and in some cases, some of them will be mobile. Secondly, the nodes will be subject to failures at random times and they will also be allowed to change their transmission range to better suit their energy budget. This leads to the full picture of a network topology in a continuous change. The algorithms for the wireless sensor networks have as one of their characteristic the fact that they do not require a predefined well-known topology.

One more consequence of the real world deployment is that there will be many factors influencing the sensors in contact with the phenomenon. Individual calibration of each sensor node will not be feasible and probably will not help much as the external conditions will be in a continuous change. The sensor network will calibrate itself as a reply to the changes in the environment conditions. More than this, the network will be capable of self-configuration and self-maintenance.

Another issue we need to talk about is the dynamic nature of the wireless

communication medium. Wireless links between nodes can periodically appear or disappear due to the particular position of each node. Bidirectional links will coexist with unidirectional ones and this is a fact that the algorithms for wireless sensor networks need to consider.

2.1.3 Needed algorithms

For a sensor network to work as a whole, some building blocks need to be developed and deployed in the vast majority of applications. Basically, they are: a localization mechanism, a time synchronization mechanism and some sort of distributed signal processing. A simple justification can be that data hardly has any meaning if some position and time values are not available with it. Full, complex signal processing done separately at each node will not be feasible due to the resource constraints.

The self-localization of sensor nodes gained a lot of attention lately [5, 12, 21, 14]. It came as a response to the fact that global positioning systems are not a solution due to high cost (in terms of money and resources) and it is not available or provides imprecise positioning information in special environments as indoors, etc. Information such as connectivity, distance estimation based on radio signal strength, sound intensity, time of flight, angle of arrival, etc. were used with success in determining the position of each node within degrees of accuracy using only localized computation.

The position information once obtained was not only used for characterizing the data, but also in designing the networking protocols, for example, leading to more efficient routing schemes based on the estimated position of the nodes [47].

The second important building block is the timing and synchronization block. Nodes will be allowed to function in a sleep mode for long periods of time, so periodic waking up intervals need to be computed within a certain precision. However, the notion of local time and synchronization with the neighbors is needed for the communication protocols to perform well. Light-weight algorithms have been developed that allow fast synchronization between neighboring nodes using a limited number of messages. Loose synchronization will be used, meaning that each pair of neighbor nodes are synchronized within a certain bound, while nodes situated multiple hops away might not be synchronized at all.

Global timing notion might not be needed at all in most of the applications. Due to the fact that many applications measure natural phenomenon such as temperature, where delays up to the order of seconds or even minutes can be tolerated, the trade-off between latency and energy is preferred.

The last important block is the signal processing unit. A new class of algorithms has to be developed due to the distributed nature of wireless sensor net-

works. In their vast majority the signal processing algorithms are centralized algorithms that require a large computation power and the availability of all the data at the same time. Transmitting all the recorded data to all nodes is impossible in a dense network even from theoretical point of view, not to mention the needed energy for such an operation. The new distributed signal processing algorithms have to take into account the distributed nature of the network, the possible unavailability of data from certain regions due to failures and the time delays that might be involved.

2.1.4 Dependability

More than any other sort of computer network, the wireless sensor networks are subject to failures. Unavailability of services will be considered "a feature" of these networks or "regular events" rather than some sporadic and highly improbable events. The probability for something going wrong is at least several orders of magnitude higher than in all the other computer networks.

All the algorithms have to employ some form of robustness in front of the failures that might affect them. On the other hand, this comes at the cost of energy, memory and computation power, so it has to be kept at a minimum. An interesting issue is the one of the system architecture from the protocols point of view. In traditional computer networks, each protocol stack is designed for the worst case scenario. This scenario hardly ever happens simultaneously for all the layers and a combination of lower layer protocols could eliminate such a scenario. This leads to a lot of redundancy in the sensor node, redundancy that costs important resources. The preferred approach is that of cross-layer designing and studying of the sensor node as a whole object rather than separate building blocks. This opens for discussion the topic of what is a right architecture for all the sensor networks and if a solution that fits all the scenarios makes sense at all.

Let us summarize the sources of errors the designer will be facing: nodes will stop functioning starting with even the (rough) deployment phase. The harsh environment will continuously degrade the performances of the nodes making them unavailable as the time passes. Then, the wireless communication medium will be an important factor to disturb the message communication and to affect the links and implicitly the network topology. Even with a perfect environment, collisions will occur due to the imprecise local time estimates and lack of synchronization. Nevertheless, the probabilistic scheduling policies and protocol implementations can be considered sources of errors.

Another issue that can be addressed as a dependability attribute is the security. The communication channel is opened and cannot be protected. This means that others are able to intercept and to disrupt the transmissions or even to transmit

their own data. In addition to accessing private information, a third party could also act as an attacker that wants to disrupt the correct functionality of the network. The security in a sensor network is a hard problem that still needs to be solved. Like almost any other protocol in this sort of networks it has contradictory requirements: the schemes employed should be as light as possible while achieving the best results possible. The usual protection schemes require too much memory and too much computation power to be employed (the keys themselves are sometimes too big to fit into the limited available memory - if we refer to the data privacy scenarios).

A real problem is how to control the sensor network itself. The sensor nodes will be too many to be individually accessible to a single user and might also be deployed in an inaccessible environment. By control we understand issues as deployment and installation, configuration, calibration and tuning, maintenance, discovery and reconfiguration. Debugging the code running in the network is completely infeasible, as at any point inside, the user has access only to the high level-aggregated results. The only real debugging and testing can be done with simulators that prove to be invaluable resources in the design and analysis of the sensor networks.

2.2 Overview of the covered topics

When building a wireless sensor network from scratch, the designer is faced with questions whose answers span across various research fields. This section will decompose the concept of wireless sensor network in its basic building blocks and take a look at the influences coming from the other related disciplines.

At a high level we can talk about the following building blocks: hardware platform (including radio, processor, sensors, power source), wireless communication (as the energy efficient media access control protocols, routing schemes, etc.), data handling (here we can talk about the process of gathering and pre-processing the data, data aggregation and dissemination, data storage and querying, etc.) and other high level services (for example service discovery and subscription issues, specific classes of applications, etc.).

2.2.1 Hardware platforms

Sensor network solutions should work unattended for large periods of time. This implies that the nodes of the network must preserve their limited amounts of energy for as long as possible. One compulsory step to achieve this is to use energy efficient pieces of hardware to build the device itself:

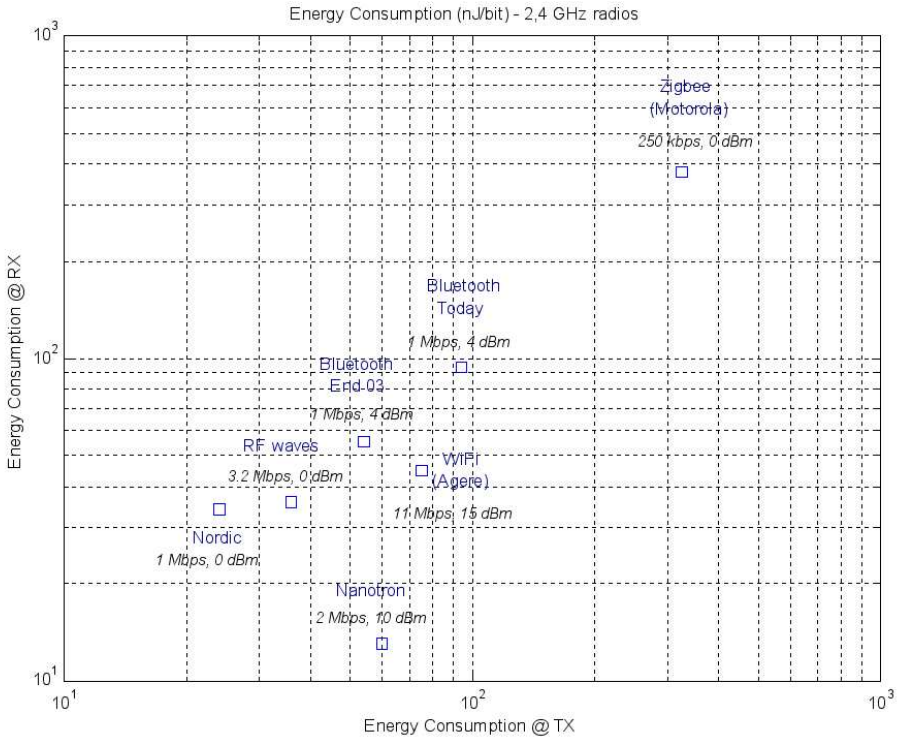

Figure 2.1: Energy consumption (nJ/bit) for 2.4GHz transceivers

- *Radio*

  The most power hungry component on the sensor node platform is the wireless communication device. In the vast majority of the situations radio communication is the preferred mean - although applications using infrared, ultrasound or magnetic induction communication have been developed. Various sorts of radio transceivers are available on the market right now (see Table 2.1). The most important parameters of the radio devices from a wireless sensor network applications are the power consumption levels used during transmission, reception, and low power mode state. Lower interest parameters (also very important) are the switching times from one state to another.

  The table also mentions the power consumed during standby and power down modes, as well as the time required by the chip to move to an active mode (receiving or transmitting) from these respective low-power modes. Standby mode usually constitutes an intermediate step between receiving

or transmitting and complete shutdown of the radio, in order to insure fast wake-up while still saving power if non active. In that case, the crystal is most of the time the only component still alive.

Figure 2.1 compares the power consumption of the Bluetooth transceiver with other $2.4Ghz$ transceivers in terms of energy consumed per bit, that is the power consumption divided by the bit rate, for both receive and transmit modes. As can be seen, current best Bluetooth transceivers feature higher figures compared to Nanonet or recent Nordic chips featuring similar transmission characteristics.

At the extremes can be found IEEE 802.15.4 - for which the figures are based on Motorola's preliminary data sheets. There isn't a huge difference in terms of power consumption in the active mode between the first generation IEEE 802.15.4 and the current Bluetooth transceivers. The major difference resides in fact on the protocol controller side: while Bluetooth stack requires an ARM like microprocessor, IEEE 802.15.4 protocol can easily be implemented on a 8051 microcontroller consuming an order of magnitude less than the Bluetooth controller.

- *Processor*

  Various functionality of the communication task, sensing and pre-processing of data and application task will be supported by an on-board processor. The choice for a processor is based on factors such as: power consumption and available memory. Currently, low power 8-bit and 16-bit microcontrollers are in use. The memory available in each processor is of special interest. Usually, each processor is equipped with some non-volatile *program memory* (nowadays using the Flash technology) and some volatile *data memory*. The real restriction is given by the availability of data memory (the amounts available are less than $10kBytes$ at this moment). This restriction limits the available memory for buffering and data storage and at the same time reduces quite a lot the complexity and number of the high level applications (usually only a simple application is being run on top of a sensor network).

  The frequency the embedded processors use is of a secondary concern as their processing power is more than enough for most of the applications. See Table 2.2 for an indication of the processors used in the typical available testbeds. Usually there is a large choice in terms of tools used to program these devices and specially tailored compilers exist, so the availability of software is not a problem.

Table 2.1: Low Power RF radio transceivers

Vendor	Model Ref.	RX power (mW)	TX power (mW)	Pout dBm	IC technology	Voltage (V)	Freq. (MHz)	Quick startup standby	Slow startup power down	Data rate (Kbits/s)	Modulation type, Receiver arch.
Melexis	TH72015	-	3.5	-12	CMOS	1.9-5.5	433	-	0.8ms/0.1uA	40/40	ASK/FSK
Microchip	RFPIC 12C509	-	5	-12			433			20/40	ASK/FSK ?
Micrel	MICRF007	3	-	-		5	433	-	2.5ms/0.5uA	2	ASK Superheter
Chipcon	CC1000	7.4	5.3	-20	0.35 CMOS	2.1-3.6	433	250us/30uA	2ms/0.2uA	76.8	FSK Superheter
Chipcon	CC1020	16.9	13.7	0	0.35 CMOS	2.3-3.6	402-470	4.8ms/77uA	5.8ms/0.2uA	153.6	FSK Superheter
AMIS	ASTRX2	7.5	25	6	CMOS	2.7-3.3	433	65us/0.5uA	-	20	ASK Low-IF
XEMICS	XE1201A	6	13.5	+5	BICMOS	2.4-3.6	315-433	60us/55uA	/0.2uA	64	FSK Zero-IF
Melexis	TH72035	-	4	-12	CMOS	1.9-5.5	868/900	-	0.8ms/0.1uA	40/40	ASK/FSK
Chipcon	CC1000	9.6	8.6	-20	0.35 CMOS	2.1-3.6	868/900	250us/30uA	2ms/0.2uA	76.8	FSK Superheter

Continued on next page

Vendor	**Model Ref.**	**RX power (mW)**	**TX power (mW)**	**Pout dBm**	**IC technology**	**Voltage (V)**	**Freq. (MHz)**	**Quick startup standby**	**Slow startup power down**	**Data rate (Kbits/s)**	**Modulation type, Receiver arch.**
Chipcon	CC1020	17.3	21.9	0	0.35 CMOS	2.3-3.6	804-940	2.5ms/77uA	3.5ms/0.2uA	153.6	FSK Superheter
AMIS (IEEE 802.15.4)	ASTRX1	36	25	0	CMOS	2.7-3.3	868/900	-/4 uA		20 (EU) 40 (US)	BPSK, DSSS Zero-IF
XEMICS	XE1203	14	33	+5	BICMOS	2.4-3.6	868/900	200us/850uA	/0.2uA	153.2	FSK Zero-IF
Philips (Bluetooth)	UAA3559 (Transc)	40	33	+4	QUBIC CMOS	2.7-3.4	2400	160us/60uA	?	1000	GFSK, FH Low IF
TI (Bluetooth)	BRF6100 (module)	37	25	+4	0.13 CMOS	1.7-3.6	2400	?	/30uA	1000	GFSK, FH ? - Digital RF
Motorola (IEEE 802.15.4)	MC13192	35	30	0	CMOS	2.0-3.6	2400	150us/1mA 500us/40uA	18.5ms/3uA 23.5ms/<1uA	250	O-QPSK, DSSS Low-IF
Philips (IEEE 802.15.4)	Not yet released	40 (MAC incl)	40 (MAC incl)	0	RF CMOS18	2.4-3.6	2400	/20uA	-	250	O-QPSK, DSSS Low-IF
Cypress	CYWUSB6932	58	62	0	0.25 CMOS	2.7-3.6	2400	300us/3mA	0.8ms/2uA	62.5	GFSK DSSS Low-IF

Continued on next page

Vendor	Model Ref.	RX power (mW)	TX power (mW)	Pout dBm	IC technology	Voltage (V)	Freq. (MHz)	Quick startup standby	Slow startup power down	Data rate (Kbits/s)	Modulation type, Receiver arch.
RFMD (WiFi)	RF2958 RF3002	52 ?	68 ?	+3 -	BICMOS SiGe	2.7-3.6	2400			11000	CCK Superheter
Agere (WiFi)	WL1141	150	250	+15	BICMOS	3.0-3.6	2400	/9mA	/1mA	11000	CCK Zero-IF
Nordic	nRF2401	18	13	0	0.18 CMOS	1.9-3.6	2400	200us/12uA	3 ms/1uA	1000	GFSK Low IF
Chipcon	CC2400	23	19	0	0.18 CMOS	1.6-2.0	2400	100us /1.2mA	1 ms/1.5 uA	10,250, 1000	(G)FSK Low IF
Nanotron	Nanonet	17	50	+10	0.35 BIC-MOS	2.4-3.6	2400	11us (Tx)/2uA 6us (Rx)/2uA	-	2000	Chirp ?
RFWaves	RFW302	43	43	0		2.7-3.6	2400	20us/6uA	-	3200	ASK, DSSS Superheter
RFMD (802.11a)		350 (system)	500 (system)	+14		3.3	5000			54000	OFDM
Xtreme Spectrun	Trinity Chipset	60	60		0.18CMOS & SiGe	3.3	3000-10000	?	?	75000 (demonst)	UWB ?

It is interesting to see that *all* the current applications are using serial processors and try to emulate the parallel behavior by adopting various scheduling schemes. Recent advancements in technology made available true parallel processors for embedded systems, and more than this, specially designed for driving the communication with a wireless transceiver. An example is *XiNc* processor produced by Eleven Engineering (CA) that allows eight threads of executions and has an instruction set specially tailored for communication purposes.

- *Sensors and actuators*

  The task of sensor networks is to perform some sort of sensing or actuating. For this, the microcontroller is connected to some specialized sensors (and actuators) and if needed to specialized digital signal processing processors. The choice of sensors depends on the particular application. There are large number of technologies developed for sensing any particular feature of the surrounding environment. The results obtained in the form of a spatial map of sensed values make it possible to use lower quality cheaper sensors to perform the task. Table 2.2 presents also some of the sensor types used in various projects. The indication "extensible" refers to the fact that specially designed sensor boards can be attached to the device. Please refer to Table 2.3 for some examples of sensor boards.

- *Power source*

  The most critical parameter of a sensor node is the amount of energy it can spend during its lifetime. The power source of the node is thus of special interest and a lot of attention has been dedicated to it. The main approach of using some sort of a energy reservoir (e.g. a battery) can be combined with techniques of environmental energy scavenging to provide a long lasting power source. The big disadvantage with regular batteries is the relative low quantity of energy they can store in a given volume (zinc-air based batteries offer $3780J/cm^3$, lithium $2880J/cm^3$ and alkaline $1200J/cm^3$). Rechargeable batteries come as a second choice as the amount of energy they can store is even smaller and it is often impossible to have a setup in which to recharge them (lithium technology offers $1080J/cm^3$, NiMHd $860J/cm^3$ and NiCd $560J/cm^3$). Micro-batteries and micro fuel-cells come as additional choices, but their technology needs major improvements until it would be usable.

  As the advancements in the battery technology increase at best linearly with time (and thus it hardly can keep up with the semiconductor industry), the only choice to have additional available energy in the sensor node

Platform	Processor	RAM	ROM	Radio	Actuators	Sensors
Mica2 based	Atmel ATMega 128L	4kB	128kB	CC1000	extensible	extensible
MicaZ	Atmel ATMega 128L	4kB	128kB	CC1000	3 LEDs, speaker	extensible
BT-Nodes	Atmel ATMega 128L	4kB	128kB	ZV4002	4LEDs	extensible
Ambient pico node	TI MSP 430	2kB	60kB	CC1010	4 LEDs, LCD	temperature, extensible
Telos	TI MSP 430	10kB	48kB	CC2420	3 LEDs	humidity, temperature, light, 2 buttons
EYES Nedap	TI MSP 430	2kB	60kB	TR1001	2 LEDs, LCD, 5 ub. LEDs, Buzzer	Compass, 4 buttons, light, accelerometer
EYES IFX	TI MSP 430	10kB	48kB	Infineon TDA5250	4 LEDs	1 button, light, temperature
Particle Communication	PIC 18F6720	4kB, 1kB	128kB	—	—	—
Particle Sensor	PIC 18F452	1.5kB, 256B	32kB	—	2 LEDs, speaker, LCD	temperature, light, accelerometer, microphone
Sensicast STAR 100, RTD, EMS	—	—	—	IEEE 802.15.4	—	temperature, humidity
Sensoria WINS 3.0	PXA 255, TMS 320VC5502	64MB	32 MB	802.11	—	GPS
Cricket	Atmel ATMega 128L	4kB	128kB	CC1000	3 LEDs	ultrasound
PushPin	Cygnal C8051F016	2kB	32kB	Infrared	—	extensible
Soapbox	PIC 16LF877	—	—	TR 1001	—	accelerometer, light, magnetic, temperature
Microstain, MillenialNet	—	—	—	IEEE 802.15.4	—	—
Piconodes	Strong ARM 1100, Xilinx XC4020XLA	4MB	4MB	—	—	temperature, humidity, light, sound, acceleration, magnetic

Table 2.2: Sensor network platforms

is to use energy scavenging techniques from the environment. Solutions for power distribution as electromagnetic power distribution (RFID like devices), acoustic or laser based are highly inefficient and require most often direct line of sight so are not preferred in the ad-hoc deployment scenarios.

On the other hand, energy scavenging offers promising results. The most used devices are the solar cells. They can be used to recharge a secondary battery or directly power the sensor node. It should be kept in mind that the amount of energy available varies between $100mW/cm^2$ in direct sunlight and $100\mu W/cm^2$ in an office setting on the surface of a desk while the solar cells exhibit efficiencies between 10-20% depending on the technology used.

Converting vibrations into electrical energy using piezoelectric power converters is a growing area of interest. It is attractive due to the fact that this sort of energy can be scavenged in almost any type of environment (the harsher, the better). Prototypes have already been built that prove the technology works [36]. Additional electrical energy can be obtained by exploiting temperature gradients (see for example the products of Applied Digital Solutions), wind or air flow and even human power (piezoelectric devices hidden in the shoes for example). However, the usage of these "exotic" sources is conditioned by the specific applications in which they can be deployed and their availability in the form of off-the-shelf components.

Recent research has revealed nuclear power as possible source of power for wireless sensor network. While certainly a little bit frightening topic at the first view, it should be known that the isotopes used in the actual prototypes penetrate no more than 25 micrometers in most solids and liquids, so in a battery they could be safely contained by a simple plastic package (most smoke detectors and some emergency exit signs already contain radioactive material). The huge amount of energy these devices can produce is shown by the next figures: the energy density measured in milliwatt-hours/milligram is 0.3 for a lithium-ion battery, 3 for a methanol based fuel cell, 850 for a tritium based nuclear battery and 57.000 for a polonium-210 nuclear battery. The current efficiency of a nuclear battery is around 4% and current research projects (e.g. as part of the a new Defense Advanced Research Project Agency program called RadioIsotope Micro-power Sources) aim at 20%. To make a little more sense out of these figures, for example, with 10 milligrams of polonium-210 (contained in about 1 cubic millimeter) a nuclear battery could produce 50 milliwatts of electric power for more than four months. With this level of power, it would be possible to run a microcontroller and a handful of sensors (for instance for environmental control) for all those months.

Model	Characteristics
MTS101	precision thermistor, light sensor, and general prototyping area
MTS300 MTS310	supports a variety of sensor modalities for the MICA and MICA2
MDA500	sensor and data acquisition board provides a flexible user-interface for connecting external signals to the MICA2DOT mote
MTS400 MTS420	supports environmental monitoring for the MICA2 with built-in sensors and an optional GPS
MDA300	supports data acquisition and environmental monitoring for the MICA2
MTS510	light, accelerometer, microphone sensor board for MICA2DOT
MEP401	light, photo-active light, humidity, barometric pressure, and temperature module with built-in MICA2
MEP500	sealed temperature and humidity module with built-in MICA2DOT

Table 2.3: Sensor boards provided by CrossBow Technology Inc.

2.2.2 Wireless communication

In most of the envisioned scenarios, the sensor nodes need to communicate to each other in an ad-hoc fashion, without the help of any external infrastructure. Moreover, the communication technology must be robust, scale well and must efficiently use the limited energy of the device. However, so far no single technology has established itself in the field: many current wireless communication technologies lack robustness, consume too much energy or require an infrastructure to be viable candidates.

Several wireless communication systems have been developed and are in use at the moment. Let us take a look at the ones that could constitute possible candidates for wireless sensor networks:

- *Bluetooth*

  Bluetooth is a very popular standard nowadays that allows automatic setup and configuration of a short range ad-hoc network. Almost all the portable devices on the market have Bluetooth capability incorporated, this making it an attractive choice from the point of view of standardization and availability of ready to use hardware and software. Bluetooth allows networks of up to eight devices separated by distances up to ten meters with a total bandwidth of $1Mbps$. Unfortunately, there are several disadvantages that

make it unusable in the wireless sensor networks applications: first of all, the *piconet concept* on which it is built. If more than eight devices need to be connected, then nodes must be placed in park mode. All the communication involves the master nodes so direct slave to slave communication is not possible (substantial overhead can solve this problem). A second disadvantage is the concept of *scatternet*. Even if nodes can be in more than one piconet at a time, they can be active only in one of them. Another disadvantage is related to the connection establishment that takes several seconds to complete - which is not suited for applications involving moving nodes. Additionally, when taking in consideration the amount of power needed ($50mW$ when active based on Ericsson PBA31301/2) and the high computation resources asked (it demands multithreading environment and large amounts of memory) we can rule out Bluetooth as a feasible solution at least at this stage of its development.

- *Wireless LAN*

  Wireless LAN (represented by both IEEE 802.11 set of standards and the HiperLAN/2 competitor) represents an elegant and reliable solution to connect wireless devices to each other. The network is usually set up around a dedicated access point that takes care of many features including quality of service and power saving. Networks can be setup in ad-hoc manner as well in the absence of a dedicated access point. The main drawback of using such a technology for wireless sensor networks is the energy consumption of the device, problem that is hardly solved even for laptops and personal digital assistants (PDAs). Still, wireless LAN can be part of a wireless sensor network scenario where specially equipped nodes can act as gateways between the sensor network and the wireless LAN, becoming bridges between the distributed sensing tool and networks such as Internet.

- *ZigBee*

  ZigBee is another approach to develop a standard for a small range wireless network. The effort incorporates physical and media access control layers, providing thus the developers with the lower layers of the protocol stack. The focus of the standard is on low power and low cost at the expense of data rates (the achievable $200Kbps$ is anyhow enough for most of the command and control applications). The standard is quite new, as the final specifications have been released in December 2004. Currently, it is not deployed at a large scale.

- *Ultra wide band*

  Ultra wide band is a wireless technology that aims at achieving extremely

large bandwidths for short distances between devices. It uses as basic idea the spread spectrum modulation technique, allowing very small amounts of energy to be spread across a large spectrum (several GHz). Ultra wide band technology comes at the expense of complex modulation and demodulation hardware and limited transmission power (thus limited distance between terminals). A secondary advantage of this technology with respect to sensor networks is the tight time synchronization between terminals, allowing very precise distance measurements. The technology is still under development, at the moment of writing this thesis, no off-the-shelf integrated circuits are available (with the exception of few very expensive cumbersome development kits).

- *Radio frequency identifiers (RFID)*

  RFID technology is used in tagging, identifying and tracking individual objects on the whole path from the manufacturing for to the end user. Each object is equipped with a tag that stores unique information. The tags can be read and their information matched with a database in order to access the full description of a particular item. The RFID technology was thought as a passive technology: the tags have no batteries, they just collect energy from the reader and send back their information (limiting in this way the distance between the reader and the tags). New advancement in the technology allowed the development of enhanced tags (active RFID) whose function fills the gap between the RFID traditional field and wireless sensor networks field.

All of these technologies are designed to overcome the high degrees of unreliability associated with wireless links. There are a number of contradictory requirements for a wireless network to work. The various already proposed solutions select different trade-offs between them. For example, to increase the reliability of a link, one should try to maximize the signal to noise ratio. If the chosen solution is to increase the power at the emitter, the possible length of a link increases causing more interference for the other members of a large distributed network. On the other hand, energy is a hard constraint, so one should try to minimize the transmitted power at each terminal (it has been proved that multihop communication is more efficient than large distance, single hop communication). This action increases the number of errors introduced by the wireless link. At the same time, additional energy will be spent on redundancy, retransmissions, etc. so a certain equilibrium point needs to be found (or decided upon).

2.2.3 Data handling

We focused our research mainly on the needs of the lower layers of the protocol stack in wireless sensor networks. But as it was soon noticed in this field of research, the main application in the form of data collection and dissemination cannot be ignored at this low level. More than this, the most efficient protocols should consider from the beginning the targeted data traffic, acceptable delay, possible operations of data pre-processing and data aggregation, data generation and traffic patterns, etc. In this section we will address the issue of *data handling* in general and give a brief description of the problems that arise.

As we stated before, the initial and main application of wireless sensor networks was data collection and dissemination. Having a large number of sensors collecting data and sending it to the collection points raises the problem of scalability: the capacity of the network can easily become insufficient for the amounts of generated data. For wireless sensor networks, data dissemination is embedded and studied together with the routing protocols.

In the basic scenario including all the surveillance applications, some parameters of the environment need to be monitored and alarms must be triggered if they go out of their normal scope. The availability of all data makes hardly any sense. The concept of *the network is the tool* comes into play. The network must collect and *pre-process* the data, reporting only the most important events. The communication load is tremendously diminished and the lifetime of the network is extended. A first example of such a mechanism was Directed Diffusion [18] that adapted the routing and media access control protocol to the needs of the data traffic pattern. Although this was one of the first dedicated integrated set of protocols for wireless sensor networks, various other mechanisms have been studied.

The next step is specifying the thresholds for the alarms. The most simple queries (*report if temperature is larger than a value*) can become more complex, by combination of different rules and end up in the typical abstract query (e.g. *does anyone see the pink elephant in the area (0,0,100,100)?*) [15]. Queries can suffer transformations while they travel the network and various specified conditions are met [3]. The mechanism for efficient query dissemination can be seen as a second stage of complexity and evolution of the data dissemination algorithms.

The last step of complexity of this class of algorithms regards the sensor network as a distributed database [4, 38, 34]. This scenario fits the application domains of environmental monitoring where the scientists are interested in subsets of data, depending on the evolution of the observed phenomenon. Mechanisms for efficient distribution of queries [24] and availability of *data streams* for the end user have been developed (an example being the TinyDb application [25]).

It is interesting to notice that in the large majority of sensor network scenarios

the efficient retrieval of data is also the main application of the network. This means that usually what is understood by *routing mechanism* contains the routing protocol, the query dissemination, the data retrieval and aggregation and also a small *application* (as a set of a few conditions put on top of the data retrieval). This justifies the large interest allocated to the routing mechanisms and the huge amount of research papers and journals dealing with this topic.

2.2.4 Timing and localization

Data collected by a large number of sensor randomly deployed in the environment has usually no meaning if it is not stamped with the location of the sensor and the time of collection. Thus, these two basic building bocks are required by virtually any application for wireless sensor networks.

The *timing and synchronization* protocols are not only needed for stamping data with the time of acquisition. A sense of time is needed more or less in all the building blocks of the protocol stack of a sensor network. The media access control can be based on a time division multiple access scheme [41] or can make use of a accurate timer to know when to listen or talk on the channel [45]. Localization techniques need timers in order to estimate distances (e.g. by the time difference of arrival technique), routing protocols need time to know when to update routing tables or which messages to ignore, data dissemination and data fusion need timing information to know how to handle queries and data streams, etc.

Timing issues in a distributed environment have been studied for a long time [19, 39]. Precise global time is a luxury that usually is not affordable in wireless sensor networks. The reduced amounts of resources render traditional protocols [20, 33] useless in all the application scenarios. New specific algorithms that give acceptable resolutions have been developed [35]. They usually have to achieve a common notion of time across the network facing problems such as the imperfections of the hardware clocks used, dynamic topology of the network leading to messages arriving in wrong order, communication failures, etc.

Localization protocols allow the sensor nodes to get information about their placement in space. Finding the position of objects has always been a subject of interest, starting even with ancestral times - for example the navigation of ships by sea. Traditionally, localization is linked with time measurement. The development during the 18th century of *portable clocks* (the Harrison clocks) made navigation at sea way easier by solving the *longitude problem*. At that moment, the precision of clocks was measured in degrees of longitude rather than seconds.

Nowadays localization is still closely related to clocks. Localization schemes using range measurements compute the time it takes a signal to travel from one place to another (e.g. the Global Positioning System could not exist without ac-

curate notion of time, distances are very often estimated from the time of flight of sound/radio signal, etc.). Localization in sensor networks is a very important problem to be solved; the availability of position information allows a series of efficient *location-aware protocols* to be developed (e.g. geographic routing [47]).

We have dedicated a special chapter to localization protocols, so, for a detailed state of the art of existing work in the field, please refer to Chapter 5. A nice classification of existing localization protocols is presented in [29].

What can be pointed out here is that a large number of prototypes have already been built and deployed. Unfortunately, the number of theoretical algorithms sustaining these prototypes is quite limited, many of them being based on simulation results. The approach can still be considered valid as a number of important observations have been made and build the starting hypothesis for the algorithms currently under study.

2.2.5 Other bits and pieces

Wireless sensor networks raise a large number of problems and at the same time a lot of interesting challenges. It would be simplistic to reduce a vast field only to the problems described in the previous sections. We consider that the issues described in this chapter are the main building blocks of any general application involving wireless sensor networks. The current research must converge to general accepted robust solutions to all of the previous problems. On top of those, a number of beautiful problems could be discussed and studied from feasibility point of view.

In the following we will point out some other related problems that caught the attention of the research community. The large majority of them assumes one or more of the previous issues solved already.

- *System software*

  The system software running inside the sensor nodes must be small, simple and robust. It has to provide the basic functionality needed by software developers and specific features for the protocols running on the nodes. Just a general purpose operating system will not suffice, it makes things more difficult instead of simplifying them (see Chapter 4 for a detailed descriptions and state of the art of operating systems for sensor networks). These considerations have lead to the appearance of operating systems such as TinyOS [30] and AmbientRT [17].

- *Simulators and emulators*

  Development and testing of distributed protocols on actual hardware is hardly

possible from a practical point of view. Emulators of the actual hardware and simulators that connect the virtual devices in a network have been under continuous development. Dedicated network simulators such as ns2 [40] as well as adapted ones (Omnet++ and the frameworks for wireless sensor networks [11, 16]) or the specially designed ones (e.g. the TinyOs related tools [30]) are now in use. Attention should be paid to the unrealistic assumptions on top of which the protocols are developed and to software bugs. The results obtained from such tools should be used only as guidance and should be backed up by theory and tested afterward in practice to claim that a (possible) correct algorithm has been developed.

- *Security and privacy aspects*

  Sensor networks raise difficult problems at the level of security and privacy. Unfortunately, existing research from other fields can hardly apply and completely new approaches must be made. An interesting description could be the following: imagine the analogy between a wireless sensor network and a medieval castle. Traditional approaches would be the equivalent of building strong walls and motes around the castle and thus delay as long as possible its fall. Wouldn't it be more interesting to have an *active* defense, where the walls still exist but are thinner and a reactive mechanism exists (in our analogy, archers and trebuchets must be placed on the walls to defend it, a strong cavalry must also be present, means of alerting nearby friendly forces, etc.). From the sensor network point of view, this story translates to automatic mechanisms that identify and isolate the malicious nodes, and means of alerting the human operator about the attack. As a start point the work presented in [31, 22] could be consulted.

- *Other research problems*

  Many more things can be imagined. For example (and in no particular order): automatic calibration of the sensors [43, 6], system integrity [26], in network data processing [23, 8], coordinated actuation, programming abstractions for querying [9], topology control [7, 44], etc. The list of interesting unsolved problems can be extended even further, the practical deployments being an endless source of inspiration.

2.3 Conclusions

In this section we have presented a general state of the art of the research being done in the field of sensor networks. This overview is not extensive by any means, it just points out the main contributions and advances, and highlights some of the

most important theories in the various domains that come together when it comes to sensor networks.

As a general remark, we can say that at this moment a huge amount of work is based on simulations *only*. A large number of algorithms have been developed just on top of the simulators and it has already been the case that the discovery of a bug in the simulators raised question marks about many algorithms. This is, for example, the case in [46] where the authors have shown that the implementation of Random Way Point Movement in one of the most important simulators [40] contains a bug that makes the simulation of a dynamic network slowly converge to the the simulation of a static network (it is a pity that several papers in the same Mobicom 2003 conference had results based exclusively on that simulation tool).

Another issue is that a lot of theory is developed on top of *widely believed to be true* working hypothesis that are false in practice. An example is the series of articles entitled *Top five myths about the energy consumption of wireless communication* [28, 27] that give a more accurate model of the energy model of the real radio transceivers.

The end point of this argument is that sensor networks have been approached from a practical point of view rather than theoretical. At this moment we can say that *there is hardly any theory* when comparing the number of research papers that have a strong theoretical foundation to the ones based on empirical results or, even worst, on simulations. The main reason for this is the unavailability of cheap hardware and of mature software tools for programming it.

This situation started to change lately. Various hardware and software platforms are already available; open source system software such as TinyOS has been developed is in a stable form and sustained by a huge community of developers, etc. A large number of research projects with participation from both academia and industry are currently running and business journals consider wireless sensor networks as “the next big thing” [42].

From the research point of view, there is a slight shifting towards creating generally accepted theories of wireless sensor networks, proved theoretically correct and shown to be built on realistic assumptions (showing that a protocol is working on several prototypes is seen as a big step further from simulations). The appearance of a set of standards will draw more attention and support from the industry side and push the field even further.

Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Elsevier) Journal*, 38(4):393–422, March 2002.

[3] R. Avnur and J.M. Hellerstein. Eddies: continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272, 2000.

[4] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. In *IEEE Personal Communications, vol. 7*, pages 10–15, 2000.

[5] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor localization for very small devices. In *IEEE personal communications*, pages 28–34, 2000.

[6] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. Colibration: A collaborative approach to in-place sensor calibration. In *2nd International Workshop on Information Processing in Sensor Networks (IPSN'03)*, 2003.

[7] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. SPAN: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *ACM Wireless Networks Journal, nr.8(5)*, 2002.

[8] J.C. Chen, K. Yao, and R.E. Hudson. Acoustic source localization and beamforming: theory and practice. In *EURASIP Journal on Applied Signal Processing nr.3*, 2003.

[9] E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A programming model for event-driven embedded systems. In *ACM Symposium on Applied Computing*, 2003.

[10] I. Chlamtac, C. Petrioli, and J. Redi. Energy-conserving access protocols for identification networks. *IEEE/ACM Transactions on Networking*, 7(1):51–59, February 1999.

[11] OMNet++ discrete event simulator. http://www.omnetpp.org/.

[12] L. Doherty, K. Pister, and L.El Ghaoui. Convex position estimation in wireless sensor networks. In *IEEE INFOCOM, Anchorage, AK*, 2001.

[13] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, Jan-Mar 2002.

[14] L. Evers, S. Dulman, and P.J.M. Havinga. A distributed precision based localization algorithm for ad-hoc networks. In *Proceedings of Pervasive Computing (PERVASIVE 2004)*, 2004.

[15] A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A probability space ADT for representing and querying the physical world. In *ICDE*, 2002.

[16] Mobility framework for OMNet++ simulator. http://mobility-fw.sourceforge.net/.

[17] T.J. Hofmeijer, S.O. Dulman, P.G. Jansen, and P.J.M. Havinga. AmbientRT - real time system software support for data centric sensor networks. In *Proceedings of ISSNIP 2004, Australia*, 2004.

[18] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 11(1), February 2003.

[19] L. Lamport. Time, clocks, and ordering of events in a distributed system. In *Communications of the ACM, nr.21(4)*, pages 558–565, 1978.

[20] L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. In *Journal of ACM, nr.32(1)*, 1985.

[21] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: A quantitative comparison. In *Computer Networks (Elsevier), special issue on Wireless Sensor Networks*, 2003.

[22] Y.W. Law, S. Dulman, S. Etalle, and P. Havinga. Assessing security-critical energy-efficient sensor networks. In *Department of Computer Science, University of Twente, Technical Report TR-CTIT-02-18*, 2002.

[23] D. Li, K. Wong, Y.H. Hu, and A. Sayeed. Detection, classification and tracking of targets in distributed sensor networks. In *IEEE Signal Processing Magazine, nr.19(2)*, 2002.

[24] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proceeding of OSDI Conference, Boston*, 2002.

[25] S. Madden, M. Franklin, J. Hellerstein, and Wei Hong. TinyDB: An acqusitional query processing system for sensor networks. In *ACM Transactions on Database Systems*, 2005.

[26] K. Marzullo. Tolerating failures of continuous-valued sensors. In *ACM Transactions on Computer Systems, nr.8(4)*, pages 283–304, 1990.

[27] R. Min and A. Chandrakasan. Energy-efficient communication for high density networks. In *Ambient intelligence: impact on embedded system design, Kluwer Academic Publishers, Norwell, MA*, 2003.

[28] R. Min and A. Chandrakasan. Top five myths about the energy consumption of wireless communication. In *ACM SIGMOBILE Mobile Computing and Communications Review archive, vol.7(1)*, 2003.

[29] D. Niculescu. Positioning in ad hoc sensor networks. In *IEEE Network Magazine*, 2004.

[30] TinyOS operating system. http://www.tinyos.net/.

[31] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of MOBICOM*, 2001.

[32] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[33] P. Ramanathan, K.G. Shin, and R.W. Butler. Fault-tolerant clock synchronization in distributed systems. In *C.J.Walter, M.M.Hugue and N. Suri, editors, Advances in ultra-dependable distributed systems, IEEE Computer Society, USA*, 1995.

[34] S. Ratnasamy, B. Karp, Li Yin, Fang Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage. In *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[35] K. Romer, P. Blum, and L. Meier. Time synchronization and calibration in wireless sensor networks. In *I. Stojmenovic (Ed.), Wireless Sensor Networks, Wiley and Sons*, 2005.

[36] S. Roundy, P.K. Wright, and J.M. Rabaey. *Energy scavenging for wireless sensor networks: with special focus on vibrations*. Springer, 2003.

[37] C. Schurgers, V. Raghunathan, and M.B. Srivastava. Power management for energy-aware communication systems. *ACM Transactions on Embedded Computing Systems*, 2(3):431–447, August 2003.

[38] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. In *First Workshop on Hot Topics in Networks (HotNets-I)*, 2002.

[39] B. Simons, J.Welch, and N. Lynch. An overview of clock synchronization. In *Technical report RJ 6505, IBM Almaden Research Center*, 1988.

[40] NS2 Simulator. http://www.isi.edu/nsnam/ns/.

[41] L.F.W. van Hoesel and P.J.M. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. In *Proceedings of INSS, Japan*, June 2004.

[42] Bussiness Week. Tech wave 2: The sensor revolution. In *The future of Tech*, 2003.

[43] K. Whitehouse and D. Culler. Calibration as parameter estimation in sensor networks. In *Proceedings of the First ACM International Workshop on Sensor Networks and Applications (WSNA 2002)*, 2002.

[44] Y. Xu, S. Bien, Y. Mori, J. Heidemann, D. Estrin, and A. Cerpa. Topology control protocols to conserve energy in wireless ad hoc networks. In *CENS Technical Report 0006*, 2003.

[45] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of INFOCOM*, 2002.

[46] J. Yoon, M. Liu, and B. Noble. Sound mobility models. In *Proceedings of MOBICOM 2003*, 2003.

[47] M. Zorzi and R.R. Rao. Geographic random forwarding GeRaF for ad hoc and sensor networks: energy and latency performance. In *IEEE Transactions on Mobile Computing*, Oct.-Dec. 2003.

Chapter 3

Data-centric architecture

This chapter presents an overview of the existing architectures for wireless sensor networks and introduces the concept of data-centric architecture. The new architecture will be presented in detail and we will analyze the way in which it can remove some of the disadvantages of the already existing frameworks and what additional overhead is required.

The vision of ubiquitous computing requires the development of devices and technologies that can be pervasive without being intrusive. The basic component of such a smart environment will be a small node with sensing and wireless communications capabilities, able to organize itself flexibly into a network for data collection and delivery. Building such a sensor network presents many significant challenges, especially at the architectural, protocol, and operating system level.

Although sensor nodes might be equipped with a power supply or energy scavenging means and an embedded processor that makes them autonomous and self-aware, their functionality and capabilities will be very limited. Therefore, collaboration between nodes is essential to deliver smart services in a ubiquitous setting. New algorithms for networking and distributed collaboration need to be developed. These algorithms will be key for building self-organizing and collaborative sensor networks that show emergent behavior and can operate in a challenging environment where nodes move, fail, and energy is a scarce resource.

The question that rises is how to organize the internal software and hardware components in a manner that will allow them to work properly and be able to adapt dynamically to new environments, requirements and applications. At the same time the solution should be general enough to be suited for as many applications as possible. Architecture definition also includes, at the higher level, a global view

of the whole network. The topology, placement of base stations, beacons, etc. is also of interest.

In this chapter we will present and analyze some of the characteristics of the architectures for wireless sensor networks. Then, we will propose a new data-flow based architecture that allows, as a new feature, the dynamic reconfiguration of the sensor nodes software at run time.

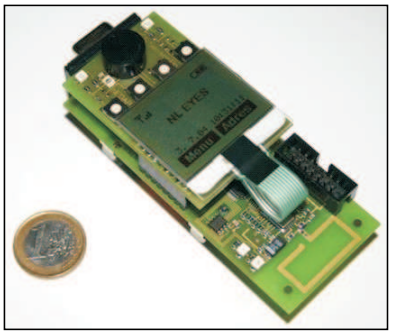

Figure 3.1: EYES sensor node [23]

3.1 Sensor node architecture

Current existing technology already allows integration of functionality for information gathering, processing and communication in a tight packaging or even in a single chip (e.g. Figure 3.1 presents the EYES sensor node [23]). The four basic blocks needed to construct a sensor node are (see Figure 3.2):

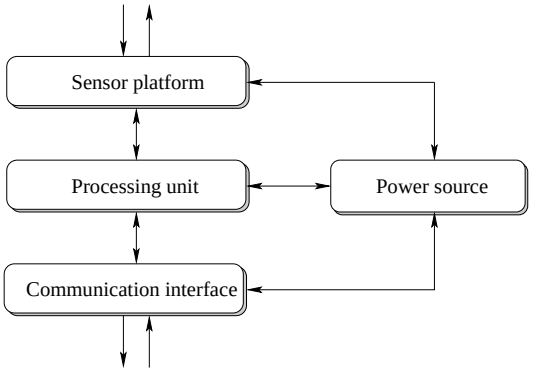

Figure 3.2: Sensor node components

- *Sensor platform*

  The sensors are the interfaces to the real world. They collect the necessary information and have to be monitored by the central processing unit. The platforms may be built in a modular way such that a variety of sensors can be used in the same network. The utilization of a very wide range of sensors (monitoring characteristics of the environment such as light, temperature, air pollution, pressure, etc.) is envisioned. The sensing unit can also be extended to contain one or more actuation units (e.g. to give the node the possibility to re-position itself).

- *Processing unit*

  The processing unit is the central intelligence of the sensor node. It will not only collect the information detected by the sensor but will also be used in the communication with the network. The level of intelligence in the sensor node will strongly depend on the type of information that is gathered by its sensors and by the way in which the network operates. The sensed information will be pre-processed to reduce the amount of data to be transmitted via the wireless interface. The processing unit will also have to execute some networking protocols in order to forward the results of the sensing operation through the network to the requesting user.

- *Communication interface*

  The communication interface is the link of each node to the sensor network itself. The focus relies on a wireless communication link, in particular on the radio communication, although visible or infrared light, ultrasound, etc. means of communications have already been used [8]. The used radio transceivers can usually function in simplex mode only, and can be completely turned off, in order to save energy.

- *Power source*

  Due to the application areas of the sensor networks, autonomy is an important issue. Sensor nodes are usually equipped with a power supply in the form of one or more batteries. Current studies focus on reducing the energy consumption by using low power hardware components and advanced networking and data management algorithms. The usage of such energy scavenging techniques for sensor nodes might make possible for the sensor nodes to be self-powered. No matter which form of power source is used, energy is still a scarce resource and a series of trade-offs will be employed during the design phase to minimize its usage.

Sensor networks will be heterogeneous from the point of view of the types of nodes deployed. Moreover, whether or not a any specific sensor node can be considered as being part of the network only depends on the correct usage and participation in the sensor network suite of protocols and not on the node's specific way of implementing software or hardware. An intuitive description given in [13] envisions a sea of sensor nodes, some of them being mobile and some of them being static, occasionally containing tiny isles of relatively resource-rich devices. Some nodes in the system may execute autonomously (e.g. forming the backbone of the network by executing network and system services, controlling various information retrieval and dissemination functions, etc.), while others will have less functionality (e.g. just gathering data and relaying it to a more powerful node).

Thus, from the sensor node architecture point of view we can distinguish between several kinds of sensor nodes. A simple yet sufficient in the majority of the cases approach would be to have two kind of nodes: *high-end sensor nodes* (nodes that have plenty of resources or superior capabilities; the best candidate for such a node would probably be a fully equipped PDA device or even a laptop) and *low-end nodes* (nodes that have only the basic functionality of the system and have very limited processing capabilities).

The architecture of a sensor node consists of two main components: defining the needed functionality and how to join various elements to form a coherent sensor node. In other words, sensor node architecture means defining the exact way in which the selected hardware and software components connect to each other, how they communicate and how they interact with the central processing unit, etc.

A large variety of sensor node architectures have been built up to this moment. As a general design rule, all of them have targeted the following three objectives: energy efficiency, small size and low cost. Energy efficiency is by far the most important design constraint because energy consumption depends on the lifetime of the sensor nodes. As the typical scenario of sensor networks deployment assumes that the power supplies of nodes will be limited and not rechargeable, a series of trade-offs need to be made to decrease the amount of consumed energy. Small size of the nodes leads to the ability of deploying lots of them to study a certain phenomenon. The ideal size is suggested by the name of one of the first research projects in the area: Smart Dust [25]. Very cheap sensor nodes will lead to rapid deployment of such networks and large scale usage.

3.2 Wireless sensor network architectures

A sensor network is a very powerful tool when compared to a single sensing device. It consists of a large number of nodes, equipped with a variety of sensors that are able to monitor different characteristics of a phenomenon. A dense network of such small devices, will give the researcher the opportunity to have a spatial view over the phenomenon and, at the same time, it will produce results based on a combination of various sorts of sensed data.

Each sensor node will have two basic operation modes: initialization phase and operation phase. But, the network as a whole will function in a smooth way, with the majority of the nodes in the operation mode and only a subset of nodes in the initialization phase. The two modes of operation for the sensor nodes have the following characteristics:

- *Initialization mode*

  A node can be considered in initialization mode if it tries to integrate itself in the network and is not performing its routine function. A node can be in initialization mode for example at power on or when it detects a change in the environment and needs to configure itself. During initialization, the node can pass through different phases such as detecting its neighbors and the network topology, synchronizing with its neighbors, determining its own position or even performing configuration operations on its own hardware and software. At a higher abstraction level, a node can be considered in initialization mode if it tries to determine which services are already present in the network, which services it needs to provide or can use.

- *Operation mode*

  After the initialization phase the node enters a stable state, the regular operation state. It will function based on the conditions determined in the initialization phase. The node can exit the operation mode and pass through an initialization mode if either the physical conditions around it or the conditions related to the network or to itself have changed. The operation mode is characterized by small bursts of node activity (such as reading sensors values, performing computations or participating in networking protocols) and periods spent in an energy-saving low power mode.

3.2.1 Protocol stack approach

A first approach to building a wireless sensor network will be to use a layered protocol stack as a starting point, as in the case of traditional computer network.

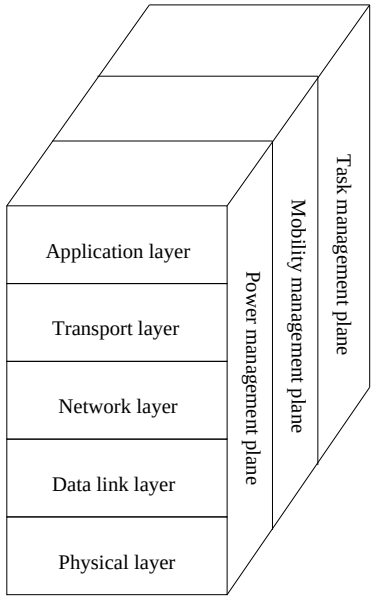

Figure 3.3: Protocol stack representation of the architecture [1]

In the following we will give a description of the main building blocks needed to setup a sensor network. The description will follow the OSI model. This should not imply that this is the right structure for these networks, but should be rather taken as a reference point:

- *Physical layer*

  The physical layer is responsible for the management of the wireless interface. For a given communication task, it defines a series of characteristics as: operating frequency, modulation type, data coding, interface between hardware/software, etc.

  The large majority of already built sensor networks prototypes and most of the envisioned application scenarios assume the use of a radio transceiver as the means for communication. The unlicensed industrial, scientific and medical band (ISM) is preferred because it is a free band designed for short range devices using low power radios and requiring low data-transmission rates. The modulation scheme used is another important parameter to decide upon. Complex modulation schemes might not be preferred because they require important resources (in the form of energy, memory, computation power).

  In the future, the advancements of the integrating circuits technology (e.g.

ASIC, FPGA) will allow the use of modulation techniques such as ultra-wide band (UWB) or impulse radio (IR), while if the sensor node is built using off-the-shelf components the choice comes down mainly to schemes such as amplitude shift keying (ASK) or frequency shift keying (FSK). Based on the modulation type and on the hardware used, a specific data encoding scheme will be chosen to assure both the synchronization required by the hardware component and a first level of error correction. At the same time, the data frame will also include some carefully chosen initial bytes needed for the conditioning of the receiver circuitry and clock recovery.

It is worth mentioning that the minimum output power required to transmit a radio signal over a certain distance is directly proportional to the distance raised to a power between two and four (the coefficient depends on the type of the antenna used and its placement relative to the ground, indoor-outdoor deployment, etc.). In these conditions, it is more efficient to transmit a signal using a multihop network composed of short range radios rather than using a (power consuming) long range link [1].

The communication subsystem usually needs a controller hierarchy to create the abstraction for the other layers in the protocol stack (we are referring to the device hardware characteristics and the strict timing requirements). If a simple transceiver is used, some of these capabilities will need to be provided by the main processing unit of the sensor node (this can require a substantial amount of resources for exact timing execution synchronization, cross-layer distribution of the received data, etc.). The use of more advanced specialized communication controllers is not preferred as they will hide important low-level details of information.

- *Data link layer*

  The data link layer is responsible for managing most of the communication tasks within one hop (both point-to-point and multicasting communication patterns). The main research issues here are the media access control protocols (MAC), the error control strategies and the power consumption control.

  The media access control protocols make the communication between several devices over a shared channel possible by coordinating the sending-receiving actions function of time or frequency. Several strategies have already been studied and implemented for the mobile telephony networks and for the mobile ad-hoc networks but unfortunately, none of them is directly applicable. Still, ideas can be borrowed from the existing standards and applications and new MAC protocols can be derived - this can be proved by the large number of new schemes that target specifically the wireless sensor networks.

As the radio component is probably the main energy consumer in each sensor node, the MAC protocol must be very efficient. To achieve this, the protocol must, first of all, make use of the power down state of the transceiver (turn the radio off) as much as possible because the energy consumption is negligible in this state. The most important problem comes from the scheduling of the sleep, receive and transmit states. The transitions between these states also need to be taken into account as they consume energy and sometimes take large time intervals. Message collisions, overhearing and idle listening are direct implications of the scheduling used inside the MAC protocol which, in addition, influences the bandwidth lost due to the control packet overheads.

A second function of the data link layer is to perform error control of the received data packets. The existent techniques include automatic repeat-request (ARQ) and forward error correction codes (FEC). The choice of a specific technique comes down to the trade-off between the energy consumed to transmit redundant information over the channel and the energy and high computation power needed at both the coder/decoder sides.

Additional functions of the data-link layer are creating and maintaining a list of the neighbor nodes (all nodes situated within the direct transmission range of the node in discussion); extracting and advertising the source and destination as well as the data content of the overheard packets; supplying information related to the amount of energy spent on transmitting, receiving, coding and decoding the packets, the amount of errors detected, the status of the channel, etc.

- *Network layer*

  The network layer is responsible for routing of the packets inside the sensor network. It is one of the most studied topics in the area of wireless sensor networks and it received a lot of attention lately. The main design constraint for this layer is, as in all the previous cases, the energy efficiency.

  The main function of wireless sensor networks is to transfer requests and deliver sensed data from and to a base station. The concept of data-centric routing has been used to address this problem in an energy-efficient manner, minimizing the amount of traffic in the network. In data-centric routing, each node is assigned a specific task based on the interests of the base stations. In the second phase of the algorithm, the collected data is returned to the requesting nodes. Interest dissemination can be done in two different ways, depending on the expected amount of traffic and level of events in the sensor network: the base stations can broadcast the interest to the whole

network or the sensor nodes themselves can advertise their capabilities and the base stations will subscribe to that.

Based on the previous considerations, the network layer needs to be optimized mainly for two operations: spreading the user queries (generated at one or more base stations) around the whole network and then retrieving the sensed data to the requesting node. Individual addressing of each sensor node is not important in the majority of the applications.

Due to the high density of the sensor nodes, a lot of redundant information will be available inside the sensor network. Retrieving all this information to a certain base station might easily exceed the available bandwidth, making the sensor network unusable. The solution to this problem is the data aggregation technique which requires each sensor node to inspect the content of the packets it has to route and aggregate the contained information, reducing the high redundancy of the multiple sensed data. This technique was proved to substantially reduce the overall traffic and make the sensor network behave as an instrument for analyzing data rather than just a transport infrastructure for raw data [16].

- *Transport layer*

  This layer appears also from the need to connect the wireless sensor network to an external network such as the Internet in order to disseminate its data readings to a larger community [21]. Usually the protocols needed for such interconnections require significant resources and they will not be present in all the sensor nodes. The envisioned scenario is to allow a small subset of nodes to behave as gateways between the sensor network and some external networks. These nodes will be equipped with superior resources and computation capabilities and will be able to run the needed protocols to interconnect the networks.

- *Application layer*

  The application layer usually links the user's applications with the underlying layers in the protocol stack. Sensor networks are designed to fulfill one single application scenario for each particular case. The whole protocol stack is designed for a special application and the whole network is seen as an instrument. These make the application layer to be distributed along the whole protocol stack, and not appear explicitly. Still, for the sake of classification we can consider an explicit application layer that could have one of the following functionality [1]: sensor management protocol, task management and data advertisement protocol and sensor query and data dissemination protocol.

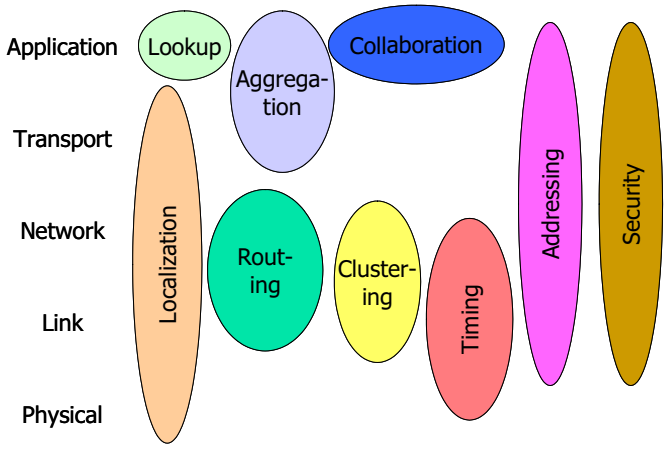

Figure 3.4: Relationship building blocks / OSI layers

The main difference between the two kinds of networks is that the blocks needed to build the sensor network usually span themselves over multiple layers, while depending on each-others. This characteristic of sensor networks comes from the fact that they have to provide functionality that is not present in traditional networks. Figure 3.4 presents an approximate mapping of the main blocks onto the traditional OSI protocol layers [13].

The authors of [1] propose an architecture based on the five OSI layers together with three management planes that go throughout the whole protocol stack (see Figure 3.3). A brief description of the layers included can be: the physical layer addresses mainly the hardware details of the wireless communication mechanism: the modulation type, the transmission and receiving techniques, etc. The data link layer is concerned with the Media Access Control (MAC) protocol that manages communication over the noisy shared channel. Routing the data between the nodes is managed by the network layer, while the transport layer helps to maintain the data flow. Finally, the application layer contains (very often) only one single user application.

In addition to the five network layers, the three management planes have the following functionality: the power management plane coordinates the energy consumption inside the sensor node. It can, for example, based on the available amount of energy, allow the node to take part in certain distributed algorithms or to control the amount of traffic it wants to forward. The mobility management plane will manage all the information regarding the physical neighbors and their movement patterns as well as its own moving pattern. The task management plane coordinates sensing in a certain region based on the number of nodes and their

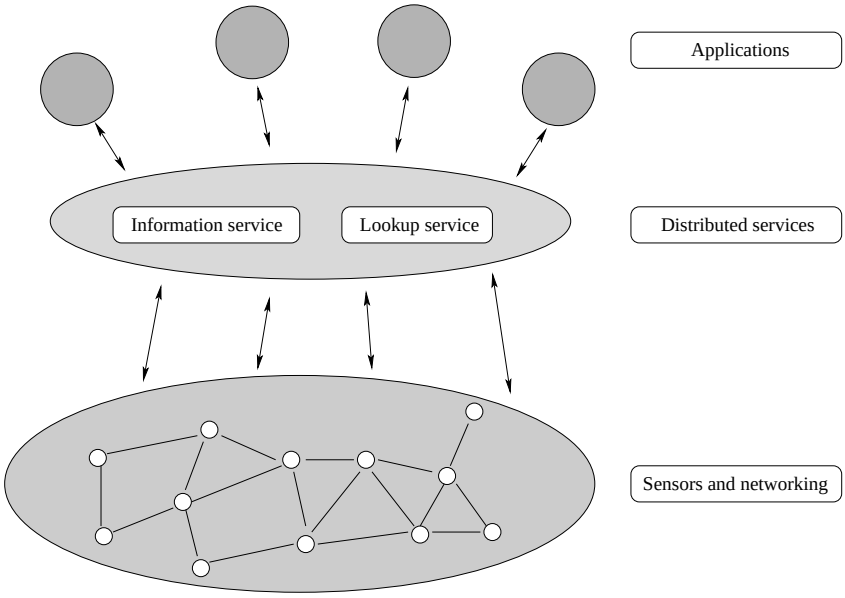

Figure 3.5: EYES project architecture description

placement (in very densely deployed sensor networks, energy might be saved by turning certain sensors off to reduce the amount of redundant information sensed).

3.2.2 EYES project approach

The approach taken in the EYES project [23] consists of only two key system abstraction layers: the sensor and networking layer and the distributed services layer (see Figure 3.5). Each layer provides services that may be dynamically specified and reconfigured.

- **Sensors and networking layer** - This layer contains the sensor nodes (the physical sensor and wireless transmission modules) and the network protocols. Ad-hoc routing protocols allow messages to be forwarded through multiple sensor nodes taking into account the mobility of nodes and the dynamic change of topology. Communication protocols must be energy-efficient since sensor nodes have very limited energy supplies. To provide more efficient dissemination of data, some sensors may process data streams, and provide replication and caching.

- **Distributed services layer** - This layer contains distributed services for supporting mobile sensor applications. Distributed services coordinate with

each other to perform decentralized services. These distributed servers may be replicated for higher availability, efficiency and robustness. We have identified two major services. The look-up service supports mobility, instantiation, and reconfiguration. The information service deals with aspects of collecting data. This service allows vast quantities of data to be easily and reliably accessed, manipulated, disseminated, and used in a customized fashion by applications.

On top of this architecture applications can be built using the sensor network and distributed services. Communication in a sensor network is data-centric since the identity of the numerous sensor nodes is not important, only the sensed data (together with the time and the location information) counts. The three main functions of the nodes within a sensor network are directly related to this:

- **Data discovery** - Data will be collected using the several classes of sensors employed in the network. Specialized sensors can monitor climatic parameters (humidity, temperature, etc.), motion detection, vision sensors and so on. A first step of data pre-processing can also be included in this task.
- **Data processing and aggregation** - This task is directly related to performing distributed computations on the sensed data and also aggregating several observations into a single one. The goal of this operation is the reduction of energy consumption. Data processing influences it by the fact that the transmission of one (raw sensed) data packet is equivalent to many thousands of computation cycles in the current architectures. Data aggregation keeps the overall traffic low by inspecting the contents of the routed packets, and in general, reducing the redundancy of the data in traffic by combining several similar packets into a single one.
- **Data dissemination** - This task includes the networking functionality comprising routing, multicasting, broadcasting, addressing, etc.

The existing network scenarios contain both static and mobile nodes. In some cases, the static nodes can be considered to form a back-bone of the network and are more likely to be preferred in certain distributed protocols. Both mobile and static nodes will have to perform data dissemination, so the protocols should be designed to be invariant to node mobility. The particular hardware capabilities of each kind of sensor node will determine how the previously described tasks will be mapped onto them (in principle all the nodes could provide all the previous functionality). During the initialization phase of the network, the functionality of every node will be decided based on both the hardware configurations and the particular environmental conditions.

For a large sensor network to be able to function correctly, a tiered architecture is needed [10]. This means that nodes will have to organize themselves into clusters based on certain conditions. The nodes in each cluster will elect a leader - the best fitted node to perform coordination inside the cluster (this can be for example the node with the highest amount of energy, or the node having the most advanced hardware architecture, or just a random node). The cluster leader will be responsible for scheduling the node operations, managing the resources and the cluster structure and maintaining communication with the other clusters.

We can talk about several types of clusters that can coexist in a single network:

- **Geographical clustering** - The basic mode of organizing the sensor network. The clusters are built based on the geographical proximity. Neighboring nodes (nodes that are in transmission range of each-other) will organize themselves into groups. This operation can be handled in a completely distributed manner and it is a necessity for the networking protocols to work even when the network scales up.

- **Information clustering** - The sensor nodes can be grouped into information clusters based on the services they can provide. This clustering structure belongs to the distributed services layer and is built on top of the geographical clustering. Nodes using this clustering scheme need not be direct neighbors from the physical point of view.

- **Security clustering** - An even higher order hierarchy appears if security is taken into consideration. Nodes can be grouped based on their trust levels or based on the actions they are allowed to perform or resources they are allowed to use in the network.

Besides offering increased capabilities to the sensor network, clustering is considered one of the principal building blocks for the sensor networks also from the point of view of energy consumption. The overhead given by the energy spent for creating and organizing the sensor network is easily recovered in the long term due to the reduced traffic it leads to.

3.2.3 Distributed services layer examples

This section focuses on the distributed services that are required to support applications for wireless sensor networks. We discuss the requirements of the foundation necessary to run these distributed services and describe how various research projects approach this problem area from a multitude of perspectives. A comparison of the projects is also carried out.

One of the primary issues of concern in wireless sensor networks is to ensure that every node in the network is able to utilize energy in a highly efficient manner so as to extend the total network lifetime to a maximum [1, 10, 11]. As such, researchers have been looking at ways to minimize energy usage at every layer of the network stack, starting from the physical layer right up to the application layer.

While there are a wide range of methods that can be employed to reduce energy consumption, architectures designed for distributed services generally focus on one primary area - how to reduce the amount of communication required and yet get the main job done without any significant negative impact by observing and manipulating the data that flows through the network [5, 16, 19]. This leads us to looking at the problem at hand from a *data-centric* perspective.

In conventional IP-style communication networks, such as on the Internet for instance, nodes are identified by their end-points and inter-node communication is layered on an end-to-end delivery service that is provided within the network. At the communication level, the main focus is to get connected to a particular node within the network, thus the addresses of the source and destination nodes are of paramount importance [20]. The precise *data* that actually flows through the network is irrelevant to IP.

Sensor networks however, have a fundamental difference compared to the conventional communication networks described above as they are *application-specific networks*. Thus instead of concentrating on which particular node a certain data message is originating from, a greater interest lies in the data message itself - what is the data in the data message and what can be done with it? This is where the concept of a data-centric network architecture comes into play.

As sensor nodes are envisioned to be deployed by the hundreds and potentially even thousands [11], specific sensor nodes are not usually of any interest (unless of course a particular sensor needs to have its software patched or a failure needs to be corrected). This means that instead of a sensor network application asking for the temperature of a particular node with ID 0.3.1.5, it might pose a query asking: *What is the temperature in sector D of the forest?*

Such a framework ensures that the acquired results are not only dependent on a single sensor. Thus other nodes in sector D can respond to the query even if the node with ID 0.3.1.5 dies. The outcome is not only a more robust network but due to the high density of nodes [9], the user of the network is also able to obtain results of a higher fidelity (or resolution). Additionally, as nodes within the network are able to comprehend the meaning of the data passing through them, it is possible for them to carry out application-specific processing within the network thus resulting in the reduction of data that needs to be transmitted [14]. In-network processing is particularly important as local computation is significantly cheaper

than radio communication [22].

Let us take a look at three of the most well known schemes dealing with this problem:

- *Directed Diffusion*

  Directed Diffusion is one of the pioneering data-centric communication paradigms developed specifically for wireless sensor networks [16]. Diffusion is based on a publish/subscribe API where the details of how published data is delivered to subscribers is hidden from the data producers (sources) and publishers (sinks). The transmission and arrival of events (*interest* or *data* messages) occur asynchronously. Interests describe tasks that are expressed using a list of attribute-value pairs as shown bellow:

  ```
  // detect location of seagull
  type = seagull
  // send back results every 20ms
  interval = 20ms
  // for the next 15 seconds
  duration = 15s
  // from sensors within rectangle
  rect = [-100,100,200,400]
  ```

  A node that receives a data message sends it to its Filter API which subsequently performs a matching operation according to a list of attributes and their corresponding values. If a match is established between the received data message and the filter residing on the node, the diffusion substrate passes the event to the appropriate application module. Thus the Filter API is able to influence the data which propagates through the network from the source to the sink node as an application module may perform some application-specific processing on the received event, e.g. it may decide to aggregate the data. For example, consider a scenario in an environmental monitoring project where the user needs to be notified when the light intensity in a certain area goes beyond a specified threshold. As the density of deployed nodes may be very high, it is likely that a large number of sensors would respond to an increase in light intensity simultaneously. Instead of having every sensor relaying this notification to the user, intermediate nodes in the region could aggregate the readings from their neighboring nodes and return only the Boolean result thus greatly reducing the number of radio transmissions.

  Apart from aggregating data by simply suppressing duplicate messages, application-specific filters can also take advantage of named-data to decide

how to relay data messages back towards the sink node and what data to cache in order to route future interest messages in a more intelligent and energy-saving manner. Filters also help save energy by ensuring that nodes react appropriately to incoming events only if the attribute matching process has proved to be successful.

Diffusion also supports a more complex form of in-network aggregation. Filters allow *nested queries* such that one sensor is able to trigger other sensors in its vicinity if the attribute-value matching operation is successful. It is not necessary for a user to directly query all the relevant sensors. Instead the user only queries a certain sensor which in turn eventually queries the other relevant sensors around it if certain conditions are met. In this case, energy savings are obtained from two aspects. Firstly, since the user may be geographically distant from the observed phenomenon, the energy spent transmitting data can be reduced drastically using a triggering sensor. Secondly, if sampling the triggered (or secondary) sensor consumes a lot more energy than the triggering (initial) sensor, then energy consumption can be reduced greatly by reducing the duty cycle of the secondary sensor to only periods when certain conditions are met at the initial sensor.

- *COUGAR*

  Building up on same concept, that processing data within the network would result in significant energy savings, but deviating from the library-based lower level approach that is used by Directed Diffusion, the COUGAR project [5, 6] envisions the sensor network as an extension of a conventional database thus viewing it as a device database system. It makes the usage of the network more user-friendly by suggesting the use of a high-level declarative language similar to SQL. Using a declarative language ensures that queries are formulated independent of the physical structure and organization of the sensor network.

  Conventional database systems use a *warehousing* approach [4] where every sensor that gathers data from an environment subsequently relays that data back to a central site where this data is then logged for future processing. While this framework is suitable for historical queries and snapshot queries, it cannot be used to service *long-running* queries [4]. For instance, consider the following query:

  > *Retrieve the rainfall level for all sensors in sector A every 30 seconds if it is greater than 60mm.*

  Using the warehousing approach, every sensor would relay its reading back to a central database every 30 seconds regardless of whether it is in sector

A or its rainfall level reading is greater than 60mm. Upon receiving all the readings from the sensors, the database would then carry out the required processing to extract all the relevant data. The primary problem in this approach is that excessive resources are consumed at each and every sensor node as large amounts of raw data need to be transmitted through the network.

As the COUGAR approach is modeled around the concept of a database, the system generally proceeds as follows. It accepts a query from the user, produces a query execution plan (which contains detailed instructions of how exactly a query needs to be serviced), executes this plan against the device database system and produces the answer. The query optimizer generates a number of query execution plans and selects the plan that minimizes a given cost function. The cost function is based on two metrics, namely resource usage (expressed in Joules) and reaction time.

In this case, the COUGAR approach selects the most appropriate query execution plan which pushes the selection (rainfall level $>$ 60mm) onto the sensor nodes. Only the nodes that meet this condition send their readings back to the central node. Thus just like in Directed Diffusion, the key idea here is to transfer part of the processing to the nodes themselves which in turn would reduce the amount of data that needs to be transmitted.

- *TinyDB*

  Following the steps of Directed Diffusion and the COUGAR project, TinyDB [19] also proclaims the use of some form of in-network processing to increase the efficiency of the network and thus improving network lifetime. However, while TinyDB views the sensor network from the database perspective just like COUGAR, it goes a step further by pushing not only selection operations to the sensor nodes but also basic aggregation operations that are common in databases, such as MIN, MAX, SUM, COUNT and AVERAGE.

  Figure 3.6 illustrates the obvious advantage that performing such in-network aggregation operations have compared to transmitting just raw data. Without aggregation, every node in the network needs to transmit not only its own reading but also those of all its children. This not only causes a bottleneck close to the root node but also results in unequal consumption of energy, i.e. the closer a node is to the root node, the larger the number of messages it needs to transmit which naturally results in higher energy consumption. Thus nodes closer to the root node die earlier. Losing nodes closer to the root node can have disastrous consequences on the network due to network partitioning. Using in-network aggregation however, every

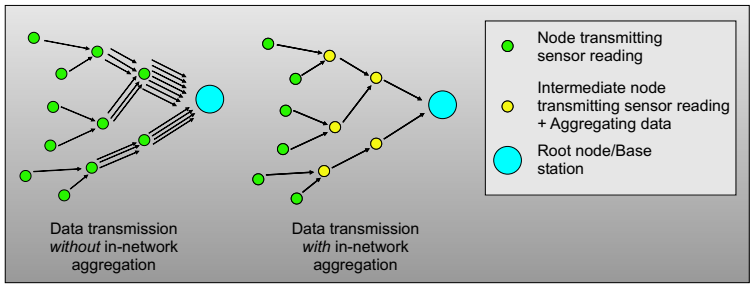

Figure 3.6: The effect of using in-network aggregation

intermediate node aggregates its own reading with that of its children and eventually transmits only one combined result.

Additionally, TinyDB has numerous other features such as communication scheduling, hypothesis testing and acquisition query processing which makes it one of the most feature-rich distributed query processing frameworks for wireless sensor networks at the moment.

TinyDB requires users to specify queries injected into the sensor network using an SQL-like language. This language describes what data needs to be collected and how it should be processed upon collection as it propagates through the network towards the sink node. The language used by TinyDB varies from traditional SQL in the sense that the semantics supports queries that are continuous and periodic. For example, a query could state:

> *Return the temperature reading of all the sensors on Level 4 of the building every five minutes over a period of ten hours".*

The period of time between every successive sample is known as an *epoch* (in this example it is five minutes).

Just like in SQL, TinyDB queries follow the "SELECT - FROM - WHERE - GROUPBY - HAVING" format which supports *selection*, *joining*, *projection*, *aggregation* and *grouping*. Just like in COUGAR, sensor data is viewed as a single virtual table with one column per sensor type. Tuples are appended to the table at every epoch. Epochs also allow computation to be scheduled such that power is minimized. For example, the following query specifies that each sensor should report its own identifier and temperature readings once every sixty seconds for a duration of three hundred seconds:

> *SELECT nodeid, temp*
> *FROM sensors*

SAMPLE PERIOD 60s FOR 300s

The virtual table sensors is conceptually an unbounded, continuous data stream of values that contains one column for every attribute and one row for every possible instant in time. The table is not actually stored in any device (meaning that it is not materialized), sensor nodes generating only the attributes and rows that are referenced in active queries. Apart from the standard query shown above, TinyDB also supports event-based queries and lifetime queries [18]. Event-based queries reduce energy consumption by allowing nodes to remain dormant until some triggering event is detected. Lifetime queries are useful when users are not particularly interested in the specific rate of incoming readings but more on the required lifetime of the network. So the basic idea is to send out a query saying that sensor readings are required for say sixty days. The nodes then decide on the best possible rate at which readings can be sent given the specified network lifetime.

Queries are disseminated into the network via a routing tree rooted at the base station that is formed as nodes forward the received query to other nodes in the network. Every parent node can have multiple child nodes but every child node can only have a single parent node. Every node also keeps track of its distance from the root node in terms of the number of hops. This form of communication topology is commonly known as tree-based routing.

Upon receiving a query, each node begins processing it. A special acquisition operator at each node acquires readings from sensors corresponding to the fields or attributes referenced in the query. Similar to the concept of nested queries in Directed Diffusion where sensors with a low sampling cost are sampled first, TinyDB performs the ordering of sampling and predicates. Consider the following query as an example where a user wishes to obtain readings from an accelerometer and a magnetometer provided certain conditions are met:

SELECT accel, mag
FROM sensors
WHERE accel $>$ *c1*
AND mag $>$ *c2*
SAMPLE INTERVAL 1s FOR 60s

Depending on the cost of sampling the accelerometer and the magnetometer sensors, the optimizer will first sample the cheaper sensor to see if its condition is met. It will only proceed to the more costly second sensor if the first condition has been met.

Next we describe how the sampled data is processed within the nodes and is subsequently propagated up the network towards the root node. Consider the following query:

> *Report the average temperature of the fourth floor of the building every 30 seconds.*

To service the above query, the query plan has three operators: a data acquisition operator, a select operator that checks if the value of floor equals 4 and the aggregate operator that computes the average temperature from not only the current node but also its children located on the fourth floor. Each sensor node applies the plan once per epoch and the data stream produced at the root node is the answer to the query. The partial computation of averages is represented as $\{sum, count\}$ pairs, which are merged at each intermediate node in the query plan to compute a running average as data flows up the tree.

TinyDB uses a *slotted* scheduling protocol to collect data where parent and child nodes receive and send (respectively) data in the tree-based communication protocol. Each node is assumed to produce exactly one result per epoch, which must be forwarded all the way to the base station. Every epoch is divided into a number of fixed-length intervals which is dependent on the depth of the tree. The intervals are numbered in reverse order such that interval 1 is the last interval in the epoch. Every node in the network is assigned to a specific interval which correlates to its depth in the routing tree. Thus for instance if a particular node is two hops away from the root node, it is assigned the second interval. During its own interval, a node performs the necessary computation, transmits its result and goes back to sleep. In the interval preceding its own, a node sets its radio to ”listen” mode collecting results from its child nodes. Thus data flows up the tree in a staggered manner eventually reaching the root node during interval 1 as shown in Figure 3.7.

In the following we do a comparison of the various projects described above and highlight some of their drawbacks. We also mention some other work in the literature that has contributed further improvements to some of these existing projects. Table 3.1 shows a list comparing some of the key features of the various projects.

As mentioned earlier, Directed Diffusion was a pioneering project in the sense that it introduced the fundamental concept of improving network efficiency by processing data within the sensor network. However, unlike COUGAR and TinyDB it does not offer a particularly simple interface, flexible naming system, or any

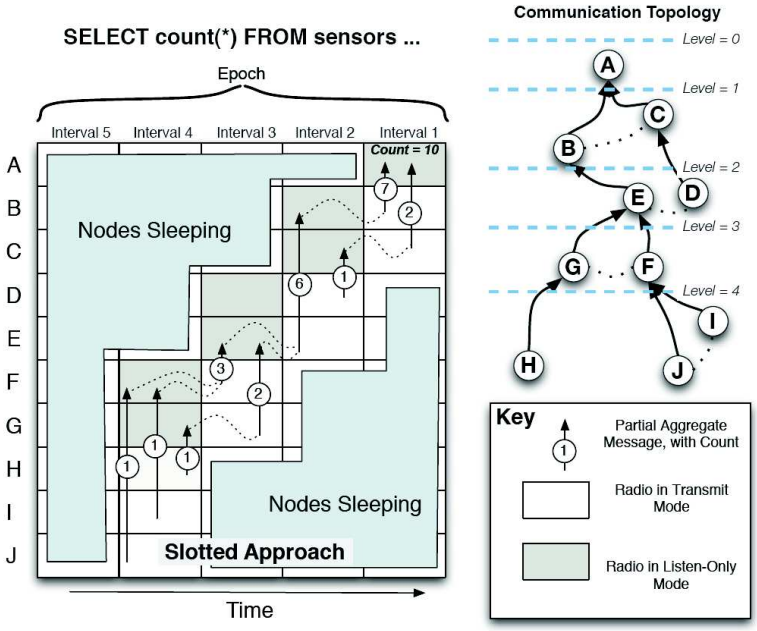

Figure 3.7: Communicating scheduling in TinyDB using the slotted approach [17]

generic aggregation and join operators. Such operators are considered as application specific operators and must always be coded in a low-level language. A drawback of this approach is that query optimizers are unable to deal with such user-defined operators as there are no fixed semantics. This is because query operators are unable to make the necessary cost comparisons between various user-defined operators. A direct consequence of this is that since the system is not able to handle optimization tasks autonomously, the arduous responsibility of placement and ordering of operators is placed on the user. This naturally would be a great hindrance to users of the system (e.g. environmentalists) who are only concerned with injecting queries into the network and obtaining the results - not figuring out the intricacies of energy-efficient mechanisms to extend network lifetime!

While the COUGAR project specifically claims to target wireless sensor networks [28, 29], apart from the feature of pushing down selection operations into the device network, it does not demonstrate any other novel design characteristics that would allow it to run on sensor networks. In fact, the COUGAR project has simulations and implementations using Linux-based iPAQ class hardware which has made them take certain design decisions that would be unsuitable for sensor networks. For instance, unlike Directed Diffusion [14] and TinyDB [18],

	Directed Diffusion	**COUGAR**	**TinyDB**
Type	Non-database	Database	Database
Platform	iPAQ class (Mote class for Micro-diffusion)	iPAQ class	Mote class
Query language	Application specific, dependent on Filter API	SQL-based	SQL-based
Type of in network aggregation	Suppression of identical data messages from different sources	Selection operators	Selection, aggregation operators and limited optimization
Cross layer features	Routing integrated with in-network aggregation	None	Routing integrated with in-network aggregation; communication scheduling also decreases burden on the MAC layer
Caching of data for routing	Yes	No	Yes
Power saving while sampling sensors	Yes - Nested queries	None	Yes - Acquisition query processing
Type of optimization	None	Centralized	Mostly centralized - Metadata is occasionally copied to catalog.

Table 3.1: Comparison of data management strategies

COUGAR does not take the cost incurred by sampling sensors into consideration during the generation of query execution plans. It also does not take advantage of certain inherent properties of radio communication, e.g. snooping and also fails to suggest any methods which could link queries to communication scheduling. Additionally, the usage of XML to encode messages and tuples makes it inappropriate for sensor networks given their limited bandwidth and high cost of transmission per bit.

Among the various query processing systems currently stated in the literature, TinyDB seems to be the one which is the most feature packed. The TinyDB software has been deployed using Mica2 motes in the Berkeley Botanical Garden to monitor the micro climate in the garden's redwood grove [12]. However, the initial deployment only relays raw readings and does not currently make use of any of the aggregation techniques introduced in the TinyDB literature. While it may have approached the problem of improving energy efficiency from several angles it does have a number of inherent drawbacks the most significant being the lack of adaptability. Firstly, the communication scheduling mentioned above is highly dependent on the depth of the network which is assumed to be fixed. This is makes it unable to react to changes in the topology in the network on-the-fly which could easily happen if new nodes are added or certain nodes die. Secondly, the communication scheduling is also directly dependent on the epoch that is specified in every query injected into the network. With networks expected to span say hundreds or even thousands of nodes, it is unlikely that environmentalists using a particular network would only inject one query into the node at any one time. Imagine that the Internet was designed in a way such that only one person was allowed to use it at any instant! Thus methods need to be devised to enable multiple queries to run simultaneously in a sensor network.

Although TinyDB reduces the number of transmissions greatly by carrying out in-network aggregation for every long-running query, it keeps on transmitting data during the entire duration of the active query disregarding the temporal correlation in a sequence of sensor readings. [2] takes advantage of this property and ensures that nodes only transmit data when there is a significant enough change between successive readings. In other words, sensors may refrain from transmitting data if the readings remain constant.

Another area related to the lack of adaptability affecting both COUGAR and TinyDB has to do with the generation of query execution plans. In both projects the systems assume a global view of the network when it comes to query optimization. Thus network meta data is periodically copied from every node within the network to the root node. This information is subsequently used to work out the best possible query optimization plan. Obviously the cost of extracting network meta data from every node is highly prohibitive. Also query execution plans

generated centrally may be outdated by the time they reach the designated nodes as conditions in a sensor network can be highly volatile, e.g. the node delegated to carry out a certain task may have run out of power and died by the time instructions arrive from the root node. In this regard, it is necessary to investigate methods where query optimizations are carried out using only local information. While they may not be as optimal as plans generated based on global network meta data, it will result in significant savings in term of the number of radio transmissions. [3] looks into creating an adaptive and decentralized algorithm that places operators optimally within a sensor network. However, the preliminary simulation results are questionable since the overhead incurred during the *neighbor exploration* phase is not considered. Also there is no mention of how fast the algorithm responds to changes in network dynamics.

3.3 Data centric architecture

As we previously stated, the layered protocol stack description of the system architecture for a sensing node cannot cover all the aspects involved (such as cross-layer communication, dynamic update, etc.). In this section we address the problem of describing the system architecture in a more suited way and its implications in the application design.

3.3.1 Motivation

The sensor networks are dynamic from many points of view. Continuously changing behaviors can be noticed in several aspects of sensor networks, some of them being:

- **Sensing process** - The natural environment is dynamic by all means (the basic purpose of sensor networks is to detect, measure and alert the user of the changing of its environment). The sensor modules themselves can become less accurate, need calibration or even break down.
- **Network topology** - One of the features of the sensor networks is their continuously changing topology. There are a lot of factors contributing to this, such as: failures of nodes or the unreliable communication channel, mobility of the nodes, variations of the transmission ranges, clusters reconfiguration, addition/removal of sensor nodes, etc. Related to this aspect, the algorithms designed for sensor networks need to have two main characteristics: they need to be independent on the network topology and need to scale well with the network size.

- **Available services** - Mobility of nodes, failures or availability of certain kinds of nodes might trigger reconfigurations inside the sensor network. The functionality of nodes may depend on existing services at certain moments and when they are no longer available, the nodes will either reconfigure themselves or try to provide these services themselves.

- **Network structure** - New kinds of nodes may be added to the network. Their different and increased capabilities will bring changes to the regular way in which the network functions. Software modules might be improved or completely new software functionality might be implemented and deployed in the sensor nodes.

Most wireless sensor network architectures currently use a fixed layered structure for the protocol stack in each node. This approach has certain disadvantages for wireless sensor networks. Some of them are:

- **Dynamic environment** - Sensor nodes address a dynamic environment where nodes have to reconfigure themselves to adapt to the changes. Since resources are very limited, reconfiguration is also needed in order to establish an efficient system (a totally new functionality might have to be used if energy levels drop under certain values). The network can adapt its functionality to a new situation, in order to lower the use of the scarce energy and memory resources, while maintaining the integrity of its operation.

- **Error control** - Error control normally resides in all protocol layers so that for all layers the worst case scenario is covered. For a wireless sensor network this redundancy might be too expensive. Adopting a central view on how error control is performed and cross-layer design will reduce the resources spent for error control.

- **Power control** - Power control is traditionally done only at the physical layer, but since energy consumption in sensor nodes is a major design constraint, it is found in all layers (physical, data-link, network, transport and application layer).

- **Protocol place in the sensor node architecture** - An issue arises when trying to place certain layers in the protocol stack. Examples may include: timing and synchronization, localization and calibration. These protocols might shift their place in the protocol stack as soon as their transient phase is over. The data produced by some of these algorithms might make a different protocol stack more suited for the sensor node (e.g. a localization algorithm for static sensor networks might enable a better routing algorithm that uses information about the location of the routed data destination).

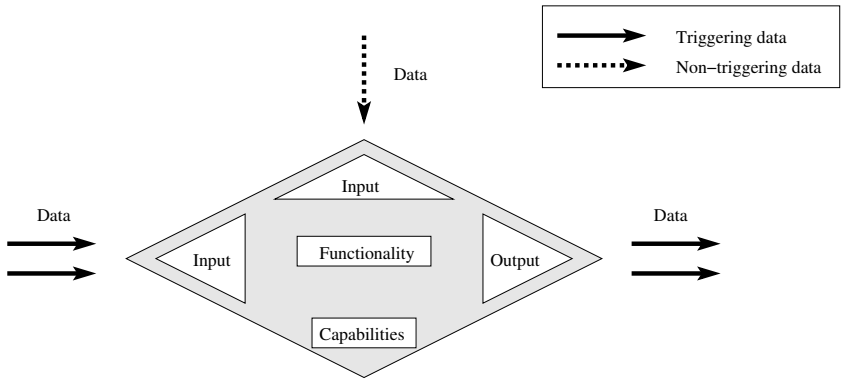

Figure 3.8: Entity description

- **Protocol availability** - New protocols might become available after the network deployment or at certain moments, in specific conditions, some of the sensor nodes might use a different protocol stack that better suits their goal and the environment.

It is clear from these examples that dynamic reconfiguration of each protocol as well as dynamic reconfiguration of the active protocol stack is needed.

3.3.2 Architecture description

The system we are trying to model is an event-driven system, meaning that it reacts and processes the incoming events and afterward, in the absence of these stimuli, it spends its time in the sleep state (the software components running inside the sensor node are not allowed to perform blocking waiting).

Let us name a higher level of abstraction for the event class as *data*. Data may encapsulate the information provided by one or more events, have a unique name and contain additional information such as deadlines, identity of producer, etc. Data will be the means used by the internal mechanisms of the architecture to exchange information components.

In the following we will address any protocol or algorithm that can run inside a sensor node with the term *entity* (see Figure 3.8). An entity is a software component that will be triggered by the availability of one or more data types. While running, each entity is allowed to read available data types (but not wait for additional data types becoming available). As a result of the processing, each software component can produce one or more types of data (usually on their exit).

An entity is also characterized by some *functionality*, meaning the sort of op-

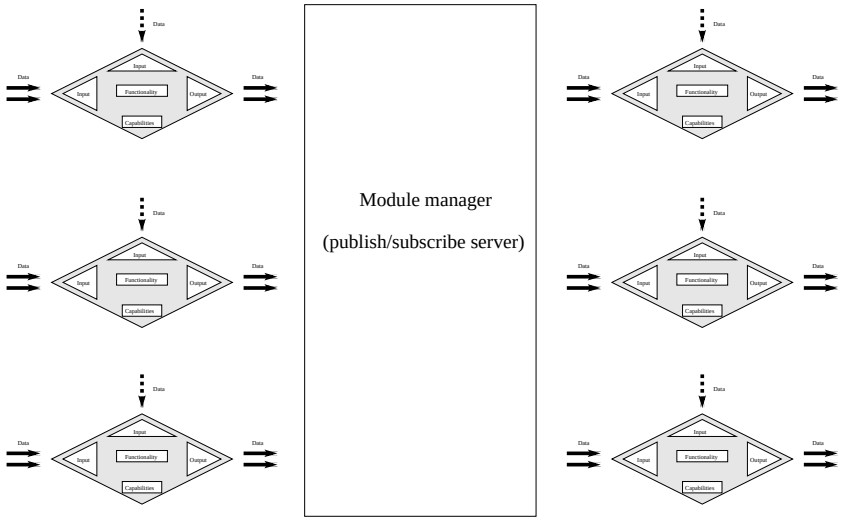

Figure 3.9: Data-centric architecture

eration it can produce on the input data. Based on their functionality, the entities can be classified as being part of a certain protocol layer as in the previous description. For one given functionality, several entities might exist inside a sensor node; to discern among them, one should take into consideration their *capabilities*. By capability we understand high-level description containing the cost for a specific entity to perform its functionality (as energy, resources, time, etc.) and some characteristics indicating the estimated performance and quality of the algorithm.

In order for a set of components to work together, the way in which they have to be interconnected should be specified. The existent architectures in the wireless sensor network field, assume a fixed way in which these components can be connected, which is defined at compile time (except for the architectures that for example allow execution of agents). To change the protocol stack in such an architecture, the user should download the whole compiled code into the sensor node (via the wireless interface) and then make use of some boot code to replace the old running code in it. In the proposed architecture we are allowing this interconnection to be changed at run time, thus making on-line update of the code possible, the selection of a more suited entity to perform some functionality based on the changes in the environment, etc. (in one word allowing the architecture to become dynamically reconfigurable).

To make this mechanism work, a new entity needs to be implemented; we call this the *data manager*. The data manager will monitor the different kinds of data being available and will coordinate the data flow inside the sensor node. At the

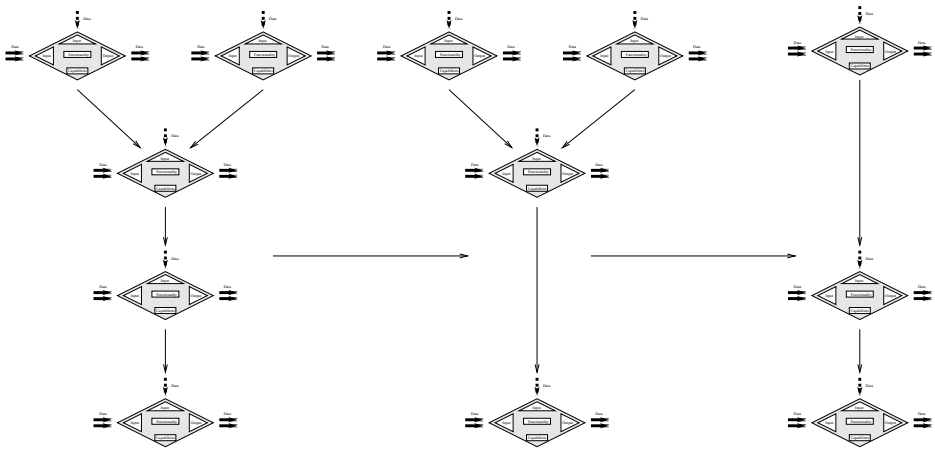

Figure 3.10: Architecture transitions

same time it will select the most fitted entities to perform the work and it will even be allowed to change the whole functionality of the sensor node based on the available entities and external environment (see Figure 3.10).

The implementation of these concepts can not make an abstraction of the small amount of resources each sensor node has (as energy, memory, computation power, etc.). Going down from the abstraction level to the point where the device is actually working, a compulsory step is implementing the envisioned architecture in a particular operating system (in this case maybe a better term is system software). A large range of operating systems exist for embedded systems in general [24, 27]. Scaled down versions with simple schedulers and limited functionality have been developed especially for wireless sensor networks [15].

Usually, the issues of system architecture and operating system are treated separately, both of them trying to be as general as possible and to cover all the possible application cases. A simplistic view of a running operating system is a scheduler that manages the available resources and coordinates the execution of a set of tasks. This operation is centralized from the point of view of the scheduler that is allowed to take all the decisions. Our architecture can also be regarded as a centralized system, with the data manager coordinating the data flow of the other entities. To obtain the smallest overhead possible there should be a correlation between the function of the central nucleus from our architecture and the function of the scheduler from the operating system. This is why we propose a close relationship between the two concepts by extending the functionality of the scheduler with the functionality of the data manager. The main challenges that arise are keeping the size of the code low and the context-switching time.

3.3.3 Additional requirements

As we mentioned earlier, the general concept of *data* is used rather than the *event* one. For the decision based on data to work, there are some additional requirements to be met.

First of all, all the modules need to declare the name of the data that will trigger their action, the name of the data they will need to read during their action (this can generically incorporate all the shared resources in the system) and the name of the data they will produce. The scheduler needs all this information to take the decisions.

From the point of view of the operating system, a new component that takes care of all the data exchange needs to be implemented. This would in fact be an extended message passing mechanism, with the added feature of notifying the scheduler when new data types become available. The mapping of this module in the architecture is the constraint imposed to the protocols to send/receive data via, for example, a publish/subscribe mechanism to the central scheduler.

An efficient naming system for the entities and the data is needed. Downloading new entities to a sensor node involves issues similar to services discovery. Several entities with the same functionality but with different requirements and capabilities might co-exist. The data centric scheduler has to make the decision which one is the best.

3.3.4 Extension of the architecture

The architecture presented earlier might be extended to groups of sensor nodes. Several Data Centric Schedulers together with a small, fixed number of protocols can communicate with each other and form a virtual backbone of the network.

Entities running inside sensor nodes can be activated using data types that become available at other sensor nodes (for example, imagine one node using his neighbor routing entity because it needs the memory to process some other data).

Of course, this approach raises new challenges. A naming system for the functionality and data types, reliability issues of the system (for factors such as mobility, communication failures, node failures, security attacks) are just a few examples. Related work on these topics already exist (for example: [7, 26]).

3.4 Example

In order to better understand the flexibility offered by this architecture, we will proceed with an example. Figure 3.11 illustrates the case of a dynamic architec-

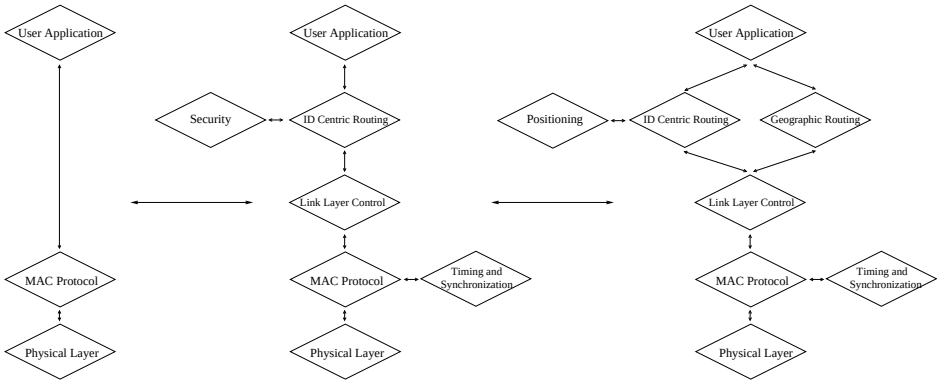

Figure 3.11: Example of architecture transitions

ture for a sensor node.

Let us assume that the goal of the sensor network is to monitor and analyze a certain feature of the environment - as the quality of perishable merchandise stored in containers in a large warehouse. The nodes are attached to containers, so they are static for most of the time; this scenario involves a limited amount of mobility.

The nodes will start running a basic Media Access Control (MAC) protocol which makes communication possible between each other. The monitoring application will run directly on top of the MAC layer, basically storing sampled data from the actual sensors. As soon as the MAC layer has gathered enough information about the neighbors of the node, an ID based routing block can be loaded into memory to allow the dissemination of the sensed data.

In parallel, a security module could be loaded for example to determine the keys needed for each node to encrypt their data. At the same time, a link layer module could also run in order to make communication more reliable and to save energy.

As the routing protocol gathers even more data about the neighborhood, a timing and synchronization protocol can run, in order to give all the nodes the same notion of time.

A new transition towards a more efficient architecture is possible right now. As the set of needed keys for encrypting the data have been gathered, there is no need for the security module to occupy memory any longer. It can be unloaded and, for example, a localization protocol can start running. Its results will enable the employment of a geographic routing block, known for its increased performances and reduced memory usage when compared to an ID based routing scheme.

As position is determined, the localization block can be unloaded from mem-

ory as it is not useful anymore (or at least until the position of the node changes). Running an efficient protocol stack, having notion of time and position, being capable of encrypting data, the node is ready for continuous monitoring of the environment and has the resources free in order to load data processing algorithms or to participate in efficient data dissemination protocols.

In the case of failures, mobility, availability of new algorithms, low energy levels, etc. the architecture of the node can change again on the fly, suiting the current running situation without the need of a human operator. Loading new blocks into the memory is easy due to the publish/subscribe mechanism that does not require a fixed protocol stack and enables easy sharing of data between any number of protocols (no matter where they might be placed in the protocol stack).

3.5 Conclusions

In this chapter we have outlined the characteristics of wireless sensor networks from an architectural point of view. As sensor networks are designed for specific applications, there is no precise architecture to fit them all but rather a common set of characteristics that can be taken as a starting point.

The combination of the data centric features of sensor networks and the need to have a dynamic reconfigurable structure has led to a new architecture that provides enhanced capabilities than the existing ones. The new architecture characteristics and implementation issues have been discussed, laying the foundations for future work.

This area of research is currently in its infancy and major steps are required in the fields of communication protocols, data processing and application support to make the vision of Mark Weiser a reality.

Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.

[2] J. Beaver, M.A. Sharaf, A. Labrinidis, and P.K. Chrysanthis. Power aware in-network query processing for sensor data. In *Proceedings of the 2nd Hellenic Data Management Symposium*, 2002.

[3] B.J. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. In *Proceedings of the Second International Workshop on Information Processing in Sensor Networks (IPSN)*, volume 2634 of *Lecture Notes in Computer Science*, pages 47–62. Springer–Verlag Berlin Heidelberg, 2003.

[4] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world, 2000.

[5] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14. Springer-Verlag, 2001.

[6] P. Bonnet and P. Seshadri. Device database systems. In *Proceedings of the International Conference on Data Engineering*, 2000.

[7] E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: a programming model for event-driven embedded systems. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 698–704. ACM Press, 2003.

[8] P.B. Chu, N.R. Lo, E. Berg, and K.S.J. Pister. Optical communication using micro corner cuber reflectors. In *MEMS97*, pages 350–355, Nagoya, Japan, January 1997.

[9] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks, 2001.

[10] D. Estrin, R. Govindan, J.S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.

[11] D. Ganesan, A. Cerpa, W. Ye, Y. Yu, J. Zhao, , and D. Estrin. Networking issues in wireless sensor networks. *Journal of Parallel and Distributed Computing (JPDC), Special issue on Frontiers in Distributed Sensor Networks*, 2004.

[12] J. Gehrke and Samuel Madden. Query processing in sensor networks. In *Pervasive Computing*, 2004.

[13] P. Havinga. Eyes deliverable 1.1 - system architecture specification.

[14] John S. Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Symposium on Operating Systems Principles*, pages 146–159, 2001.

[15] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[16] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 11(1), February 2003.

[17] S. Madden. The design and evaluation of a query processing architecture for sensor networks, phd thesis, university of california, berkley, 2003.

[18] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 491–502. ACM Press, 2003.

[19] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks, 2002.

[20] J. Postel. Internet protocol, rfc 791, September 1981.

[21] G. J. Pottie and W. J. Kaiser. Embedding the internet: Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[22] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[23] European EYES Project. http://www.eyes.eu.org.

[24] Salvo. Pumpkin Incorporated, http://www.pumpkininc.com.

[25] SmartDust. http://robotics.eecs.berkeley.edu/ pister/SmartDust.

[26] P. Verissimo and A. Casimiro. Event-driven support of real-time sentient objects. In *Proceedings of WORDS 2003*, 2003.

[27] VxWorks. Wind River, http://www.windriver.com.

[28] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks, sigmod record, vol. 31, nr.1, march 2002.

[29] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proceedings of the Conference on Innovative Data Systems Research*, 2003.

Chapter 4

System support

This chapter presents two tools developed for wireless sensor networks: a new operating system and a simulation environment. Both tools were designed using the concepts of data centric architecture and are a proof that the concept is feasible in practice. We will present their main characteristics and compare them against the other already existing tools.

In this chapter we describe two tools designed for the development of wireless sensor network applications: a novel operating system and a simulator environment. Both tools are built on top of the data centric architecture described in Chapter 3 proving that the concept works and can be easily used in practice.

One key issues that brings wireless sensor networks one step closer to reality is the system software running inside each sensor node. Ideally, this software component should allow the user to write the applications directly, without spending additional time on programming the low level hardware or the basic functionality of the node. The most convenient framework would be an operating system like system software that offers a hardware abstraction layer and drivers for all the present hardware components.

Additionally, real time scheduling, memory management and resource management would be desirable (as they allow the implementation of superior capabilities protocol stack components). These additional features are thought to be a too large overhead for the resource poor microcontrollers on which the software needs to run. In Section 4.1 we will show that designing the system software based on the data centric architecture solved these issues and had as an end result a fully working operating system.

Debugging a protocol designed for a distributed network involving unreliable

wireless communication can be quite a challenge and consumes a lot of time. An easy way to do everything is by using a dedicated simulator to test the protocol in simulated environments where the degrees of errors can be controlled. After that, the protocol can be ported to the targeted hardware and its parameters can be altered to match the reality.

Usually implementing the protocol both in the simulator and in the targeted hardware involves two different implementation; not only the programming languages differ but also the targeted architectures. To ease the programming and debugging effort we have designed a simulator based on the data centric architecture. The architecture being the same, the porting problem between the two platforms becomes an issue of mere syntax changing.

The major goals for developing the system support in the form of an operating system and a simulator can be synthesized as:

- add flexibility and dynamics to the sensor network application
- enable power control mechanisms leading to energy efficient designs
- add real time capabilities allowing for example the implementation of superior scheduling schemes for radio communication

In Section 4.1 we describe the basic functionality of the operating system highlighting the advantages it has over the existing approaches. Section 4.2 presents some features of the simulator and a small guide on how to use it. The chapter ends with conclusions and ideas for future work.

4.1 Data centric operating system

In Chapter 3 we introduced the ideas for a new data centric architecture. The main motivation for exploring a new architecture stood in the fact that we needed to combine somehow the concept of a dynamic architecture with the concept of a light weight operating system. The resulting system must be event driven and very light - in order to allow both the best usage of resources and be suited for wireless sensor network applications.

We have developed such a system proving that the concept of data centric architecture made sense. At the same time we have incorporated in the new *Data centric operating system* (DCOS) issues as real time, automatic mutual exclusions, memory management, etc. - in one word almost all the things that were believed to be impossible to achieve on a microprocessor equipped with two kilobytes of data memory.

4.1.1 Targeted hardware

Before going into the description of the operating system issues it is better to take a look at the targeted hardware. This will help the reader understand exactly what amounts of resources were available for the system we designed.

The initial platform we considered was the prototype developed by a dutch company (Nedap BV) for the EYES European Project. The software was adapted for the updated versions of the hardware that were developed. The software was also extended to work on the prototypes made available by Infineon for the same project. The DCOS operating system was even further extended to support the products of the Ambient Systems company [6].

The boards have a very similar architecture and use the same or comparable components with the exception of the radio transceiver. The first and older boards developed by Nedap are equipped with a TI MSP430F149 CPU and an RFM TR1001 radio transceiver. The second and newer board developed by Infineon uses also the TI MSP430F149 but has an Infineon TDA5250 radio transceiver. Both boards have a serial port, some LED's and a serial EEPROM. For programming the CPU each board has a JTAG interface and a power supply in the form of two AA Alkaline batteries [23].

The TI MSP430 micro-controller family is designed for ultra-low-power applications. The MSP430 incorporates a 16-bit RISC micro-controller, peripherals and flexible clock system that interconnect using a von-Neumann common memory address bus and memory data bus [40]. The MSP430F149 comes with a collection of on-chip digital and analog devices like a 12-bit ADC, 2 USART interfaces and a hardware multiplier. It has 60kB of self programmable flash memory and 2kB of RAM. Furthermore it has a watchdog timer, one timer with 3 capture/compare registers and one timer with 6 capture/compare registers. For interfacing with external devices the MSP430F149 comes with numerous digital I/O pins.

The RFM TR1001 is a very small size, very low-power consumption single chip OOK/ASK radio transceiver with a maximum data rate of 115.2kb/s. Signal power regulation is done through an external digital potentiometer. The TDA5250 is a low-power consumption single chip ASK/FSK transceiver with a maximum data rate of 64kb/s. Signal power regulation is done by the on-chip power amplifier. Both transceivers supply a RSSI pin that can be measured by the ADC of the MSP430.

The serial EEPROM for the NEDAP board is an ST M25P20 which is a 2 Mbit, low voltage, serial flash memory that can be programmed through an SPI bus interface. Because an MSP430 USART can be put into SPI mode, interfacing the chip with the CPU is rather easy. The memory of the M25P20 is divided into

four 64kB sectors which are divided in 256 byte pages. The smallest erasable unit is a sector. The Infineon board has an ST M25P05 EEPROM that has the same features of the M25P20 but has a memory size of only 512Kbit that is divided into two 32kB sectors which are divided in 128 byte pages.

4.1.2 Operating system alternatives

The advance in low power embedded micro-controller design has lead to the development of operating systems targeting these new devices. This section will describe some of these operating systems. First a description of the Salvo operating system will be given and after that the popular TinyOS will be introduced. We will end this section by presenting PEEROS (*Preemptive EYES Real Time Operating System*) - our first approach to real time operating systems (RTOS) targeting resource poor microcontrollers and mention some other existing alternatives.

- *Salvo RTOS*

  Salvo is a commercially available RTOS developed by Pumpkin, Inc. It is designed to run on microprocessors and micro-controllers with severely limited resources. It requires little memory and no stack. Salvo is mainly written in ANSI C, with limited processor specific extensions. It is available for a wide range of processors and controllers including the TI MSP430 series.

  Salvo is a cooperative multitasking RTOS with full support for event and timer services. The multitasking is priority based and fifteen different priority levels are supported. If tasks share the same priority level they are scheduled in a round robin fashion. Salvo uses semaphores, messages and message queues for interprocess communication and resource management. A full complement of RTOS functions (context switch, wait, process control,etc.) is supported. Timer functions, including delays and timeouts, are also supported [21].

  Salvo uses cooperative task scheduling which means that the application programmer must explicitly manage task switches. If a running task fails to cooperate, then no other tasks will execute. The arrival of an interrupt will suspend the current running task and invoke the interrupt service routine for that interrupt.

  The Salvo version for the MSP430 has the following characteristics [22]: a context switch takes 25μs at an 8MHz clock, it uses a maximum of 14 bytes for a task control block, it uses a maximum of 6 bytes for an event control block, it uses 400 to 1700 bytes of ROM depending on the version used.

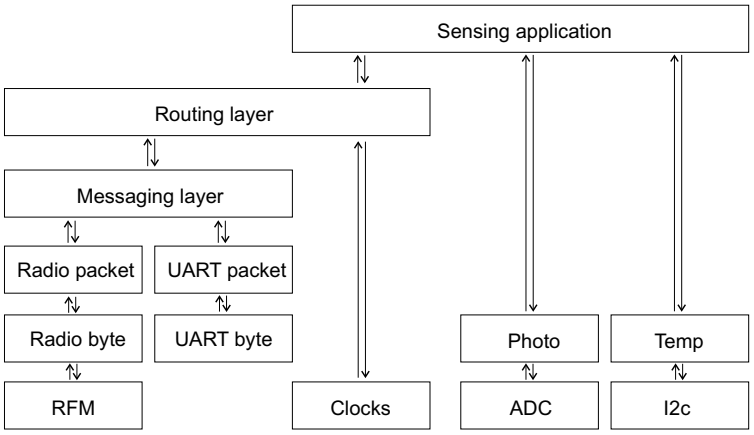

Figure 4.1: TinyOS component graph example

The advantages of using Salvo (from the point of view of our targeted applications) can be summarized as:

- Salvo uses very little memory resources;
- It requires no general purpose stack;
- Due to the cooperative scheduling, less context switches than in a preemptive scheme may occur (it is less intrusive), which might save energy.

Some of its disadvantages are:

- Cooperative scheduling often leads to a worse task response time than when using preemptive scheduling because it depends on the longest task;
- The user must provide the context switches, making the system more user error prone;
- Having no general purpose stack leads to the usage of global data for the local variables of a function. Local data normally exist on the stack and for the duration of that function only. This leads to an inneficient usage of the memory resources;
- Salvo is a commercial product and therefore not free available (except for the functionally challenged 'lite' version).

- *TinyOS*

  TinyOS is the operating system developed for wireless sensor networks at the University of California at Berkeley. It is one of the first operating systems addressing small, low cost, sensor nodes with limited energy and

processing resources. TinyOS was developed for a prototype sensor node that is very similar to the nodes used in the EYES project. It uses a 4MHz ATMEL micro-controller and an RFM TR1000 radio transceiver. The available program memory and RAM is 8KB and 512 bytes respectively. At the moment TinyOS is available for a wider range of sensor boards including the two prototype EYES boards.

TinyOS is a component based runtime environment (see Figure 4.1). The system configuration consists of a tiny scheduler and a graph of *components* (software blocks implementing a specific functionality). A component consists of a set of *command handlers*, a set of *event handlers*, an encapsulated fixed-size *frame*, and a set of simple *tasks*. A command is issued by the higher components in the stack downto the lower components and is executed immediately (basically it is a function call). Events propagate from the lower components upto the higher ones and are scheduled in a queue.

The tasks, commands and the handlers execute in the context of the frame and operate on its state. To facilitate modularity, each component also declares the commands it uses and the events it signals. These declarations are used to compose the modular components in a per-application configuration. The composition process creates layers of components where higher level components issue commands to lower level components and lower level components signal events to the higher level components. Physical hardware represents the lowest level of components [14].

Commands are blocking requests made to lower level components. Event handlers are invoked to deal with hardware events, either directly or indirectly. An event handler can deposit information into its frame, post tasks, signal higher level events or call lower level commands. Tasks perform the primary work. They are atomic with respect to other tasks and run to completion. Tasks can call lower level commands, signal higher level events, and schedule other tasks within a component. The task scheduler is a simple FIFO scheduler, utilizing a bounded size scheduling data structure.

Some of the advantages of using TinyOS are:

- It is modular and portable, making it easy to construct configurations dedicated for a specific application and target hardware;
- It uses little energy and processing resources;
- It is supported by a large user community.

And some of its disadvantages are:

- An application is configured at compile time and after that is unable to alter its configuration dynamically;

- Reaction to hardware events can be very late due to the FIFO mechanism of the scheduler. This is especially bad for radio interrupts in TDMA based MAC protocols;
- It is not possible to determine the maximum possible load of the system, making it impossible to give real-time guarantees.

- *PEEROS*

  PEEROS was our first approach to defining an operating system for wireless sensor networks and was written specifically for the Nedap sensor node. It was rather unique because of its preemptive nature, something very uncommon in operating systems that target the small embedded devices like the sensor nodes [28].

  PEEROS uses a preemptive task scheduler with fixed priorities that can handle a maximum of 16 tasks. Tasks can wait on events before continuing. A task can respond to at most 2 different events. The scheduler uses a single stack allowing only higher priority tasks to preempt the current running or waiting task.

  Interprocess communication is supported through an internal messaging system. Tasks can send very short messages to other tasks. Receiving tasks can either perform a blocking or a non-blocking message read. This introduces a threat for deadlock situations: a task that waits on a message that will be send by a task of the same or lower priority, will do so indefinitely.

  PEEROS also provides an interface for communicating through the radio transceiver and serial port, and it has a generic shell that can interpret commands coming either from the serial port, radio, or another task.

  PEEROS provides a great service. It has successfully hosted several applications developed during master students assignments. However, for the data centric architecture PEEROS either lacks functionality or is too abstracted from that architecture. Furthermore, PEEROS gives only soft real-time guarantees.

  Although it was a simple system, PEEROS was a good tool to explore the maximum capabilities that the MSP430 processor could offer. It helped us understand some of the practical problems related to operating system design and was the first step towards designing DCOS.

Apart from these operating systems, there is a large range of others developed for embedded systems in general (VxWorks [43], RedHat eCos [13], PalmOS [32], Microsoft Windows CE [29], etc.). These systems are not suited for our platform due to the fact that the resources needed often exceed with at least several

orders of magnitude what is available. Some of these operating systems claim to offer *real time capabilities*. From our experience, all of them offer priority based or cooperative scheduling leaving the real time feature to the programmer: the shortest the tasks and the better event planning, the better the system responds. Unfortunately none of them offers real time guarantees.

4.1.3 EDFI protocol

In this section we present the basics of the scheduling protocol we have chosen for DCOS (EDFI - *Early Deadline First with Inheritance*). This specific choice was made due to the facilities offered by this scheduling scheme and also due to the small overhead involved. This section is an excerpt from [20] briefly describing the theory behind this scheme.

EDFI is a lightweight real-time scheduling protocol that combines EDF with deadline inheritance over shared resources. We will show that EDFI is flexible during a task's admission control, efficient with scheduling and dispatching, and straightforward in feasibility analysis. The application programmer only needs to specify a task's timing constraints (deadline, period, runtime) and resource needs, after which EDFI can execute admission control, scheduling, dispatching and resource synchronization automatically. EDFI avoids gratuitous task switching and its programming overhead, as well as runtime overhead is very low, which makes it ideal for lightweight and featherweight kernels.

A *task set* Ω consists of a set of preemptable tasks $\tau_i(i = 1..n)$. Each task τ_i is specified by a minimum *period* T_i, a *deadline* D_i, a *cost* C_i, and a *resources specification* ρ_i. τ_i is released every T_i time units and must be able to consume at most C_i seconds of CPU time before reaching its deadline D_i seconds after release $(C_i \leq D_i \leq T_i)$. We use capital letters for time intervals (e.g., T, D, C) and lower case for absolute 'points in time': r for the next release time, d for the next deadline.

The utilization U of Ω is defined as $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$. For Ω to be schedulable, $U \leq 1$ must hold. We define two functions: *processor* demand $H(t)$ introduced in [4], and *workload* $W(t)$ introduced in [1]. $H(t)$ represents the total amount of CPU time that must be available between 0 and t for Ω to be schedulable. $W(t)$ represents the cumulative amount of CPU time that is consumable by all task releases between time 0 and t.

$$H(t) = \sum_{i=1}^{n} \left\lfloor \frac{t - D_i + T_i}{T_i} \right\rfloor C_i \tag{4.1}$$

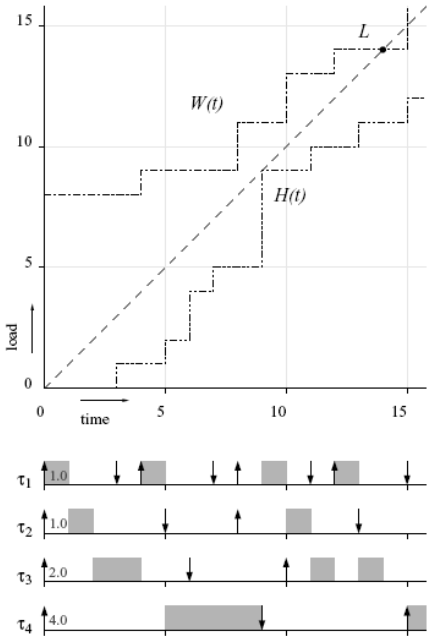

Figure 4.2: Example task set and its EDF schedule with processor demand $H(t)$ and workload functions $W(t)$.

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{4.2}$$

Figure 4.2 illustrates the functions for an example task set. Our feasibility analysis is based on the behavior of H(t) and W(t) and on the observations earlier proved by Baruah *et al* [4]:

> *If for any interval with length L, all work load offered during $[0, L]$ can be resolved before or at L, then this can be concluded for any arbitrary time interval $[t, t + L]$.*

Therefore all tasks in Ω are released simultaneously at $t = 0$, in which case they will produce the largest response time. If the tasks in Ω can make their deadlines from $t = 0$, they can make their deadlines from any point in time.

Figure 4.2 shows the functions $H(t)$ and $W(t)$. Both are used for schedulability analysis of the task set Ω. Note that the vertical distance between $W(t)$ and the

Ω_1	D_i	T_i	C_i
τ_1	3	4	1
τ_2	5	8	1
τ_3	6	10	2
τ_4	9	15	4

Table 4.1: Specification of Ω_1

diagonal in the graph represents the amount of work still to do in released tasks. At point L, there is no more work to do and the system becomes idle. $H(t)$ represents the amount of work that must be finished. If $H(t)$ crosses the diagonal, then more work would have to be finished than there is time available. The schedulability analysis tracks $W(t)$ and $H(t)$ until either $W(t)$ touches the diagonal or $H(t)$ crosses it. If $H(t)$ crosses the diagonal, the task set is not schedulable. If $W(t)$ touches before $H(t)$ could cross, the task set is schedulable. The example task set Ω is thus schedulable. Task sets can be constructed in which neither $W(t)$ nor $H(t)$ reaches the diagonal. The schedulability analysis, therefore, traces these functions for only a predetermined maximum number of steps and rejects a task set if this maximum is reached.

The scheduler manages the set of admitted tasks using two queues and a stack. The *Wait Queue*, holds tasks awaiting their release. When a task gives up the processor or reaches its deadline, it is put on this queue, from which it will be transferred to the next queue when it is released. The *Released Queue* holds processes that have been released but have not yet run. This queue is maintained in deadline order, earliest deadline first. The *Preemption Stack* holds the tasks that run but have been preempted; the currently running task is at the top of the stack and the tasks below it were preempted by the tasks immediately above them.

When a task needs to be released (due to an external event or to an internal signal for example from a timer), the task is transfered from the Wait Queue to the Released Queue. When a task gets to the front of the Released Queue or when a task is popped from the Preemption Stack, the deadlines of the task τ_h at the head of the Released Queue and the running task at the top of the Stack τ_r are compared. If $d_h < d_r$, $\tau_h$is removed from its queue and pushed onto the Preemption Stack. If both Preemption Stack and Released Queue are empty, best effort processes are scheduled.

Nested Critical Sections (NCSs) [11] can also be used in tasks for the use of shared resources. Their advantage is that they can express situations where a task makes use of specific combination of resources. What we need to do is to specify NCSs and their durations. NCSs in combination with inheritance have been used in other protocols such as the well-known Priority Ceiling (PC) protocol

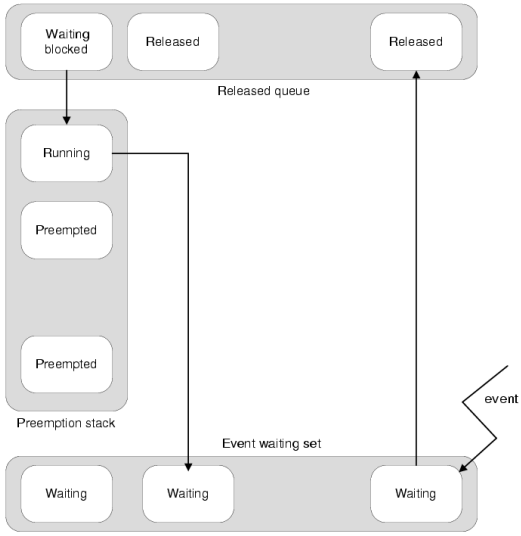

Figure 4.3: Transaction system: released queue, preemption stack and event waiting queue

[37] and the Stack Resource (SR) protocol [2]. An in-depth overview is given by Rajkumar [35]. Sha *et al.* [38] give an overview of how to generalize PC for deadline monotonic under blocking and they present how to use this protocol for a practical system implementation. The protocol we present in this chapter has similarities with PC and SR. PC is from the class of fixed priority protocols while SR belongs to the class of dynamic priority protocols.

A *resource specification* ρ of a task τ is specified according to the following syntax:

ρ_{list} : $float$ '{' R_{list} ρ_{list} '}' | $\rho_{list}\varepsilon$
R_{list} : R_{list} R | ε
R : 'a'...'z' | 'A'...'Z'
$float$: floating-point number

in which the non-capital resource R indicates a read access to a shared resource, while a capital resource R indicates an exclusive-access to it. An example of a task set with a resource specification is given in Table 4.2.

Task 1 has a period T_1 of 5 seconds, a deadline D_1 of 4 seconds (if it is released at t, its deadline is at $t+4$ and its next release is at $t+5$); it needs at most 1 second of CPU time (C_1) between release and deadline. Resource a is shared by tasks 1, 2 and 4. All tasks only require read access to the resource, so no restrictions on

Ω_2	D_i	T_i	C_i	ρ_i
τ_1	4	5	1	0.9{a B}
τ_2	5	8	1	0.8{a 0.2{B 0.1{C}}}
τ_3	6	10	2	0.2{b}1.7{c 1.3{b}}
τ_4	9	9	3	1.8{a c}

Table 4.2: Specification of Ω_2

the schedulability of these tasks exist. Resource b/B is shared by tasks 1, 2 and 3. Task 1 needs exclusive access to B for 0.9 time units, it also holds read resource a. Task 3 needs shared-read access to resource b for 0.2 time units and again for 0.13 time units while holding resource c for 1.7 time units.

The principle behind scheduling a task set with shared resources is that a released tasks stays on the Released Queue if it needs resources that are already in use by one of the task in the Run Stack, even if such a released task has a shorter deadline. Therefore, once a task τ_r is on the Preemption Stack, it will never claim a resource already held by another, preempted, task. Such a task τ_r would simply not have been scheduled.

We enforce this by *deadline inheritance*, which is similar to Priority Inheritance, introduced by Sha *et al.* [37]. Every resource, R is assigned an inherited deadline $D_R = \min_{\rho_i \in \Omega}\{D_i | R \in \rho_i\}$, the minimum of the deadline of all tasks using R, where ρ_i denotes the set of tasks in use by task τ_i. If $\rho_i^{'} \subseteq \rho_i$ denotes the subset of resources in use by τ_i, then the inherited deadline of τ_i is $\Delta_i^{'} = \min_{\{R \in \rho_i^{'}\}}\{\Delta_R\}$. The minimal inherited deadline of τ_i is reached if all resources are used: $\Delta_i = \min_{\{R \in \rho_i\}}\{\Delta_R\}$. A task's $\Delta^{'}$ thus changes as the task acquires and releases resources.

Each released task is now characterized by the triple $(d, D, \Delta^{'})$, where d is the current *absolute deadline*.

Earlier, we presented the EDF scheduling rule that the task τ_h at the head of the Released Queue would move to the top of the Preemption Stack if its d_h was less than d_r of the task τ_r on top of the Preemption Stack. A released task with an earliest deadline will preempt the currently running task. Now we modify that rule to:

$$\tau_h \text{ preempts } \tau_r \text{ iff } d_h < d_r \wedge D_h < \Delta_r^{'}$$

Figure 4.4 shows an example Preemption Stack (rectangles) and Released Queue (ellipses). At this time, the task at the head of the Released Queue may not preempt the one on top of the Run Stack because ($9 < 7 \wedge 3 < 4$ is false). For every task $\tau_i, \Delta_i^{'} \leq D_i$ and, because of the scheduling rule, for a task τ_h higher on

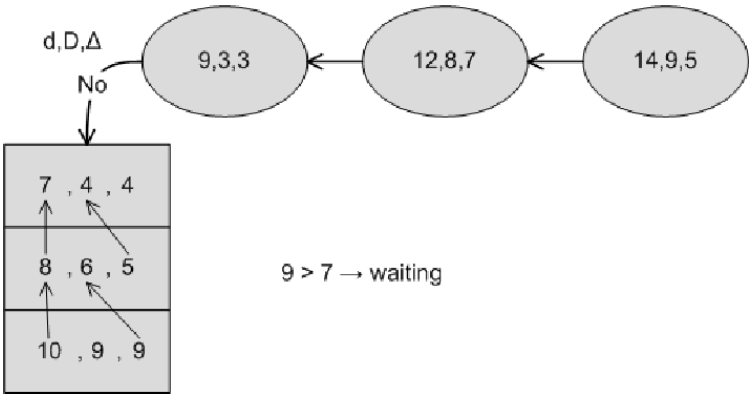

Figure 4.4: Example Preemption Stack (rectangle) and Released Queue (ellipses); the arrows indicate the partial order between the parameters. No preemption is allowed because $D_h < \Delta_r^{'}$ $(9 < 7)$ is not true.

the Preemption Stack than another task $\tau_l, D_h < \Delta_l^{'}$. There is, therefore, a total ordering from D to $\Delta^{'}$ to D, etc. up and down the Run Stack. This is indicated by the arrows in Figure 4.4. This ordering, plus the definition of Δ, establishes the property that the currently running task, which is at the top of the Preemption Stack, will not attempt to acquire any resources held by preempted tasks, which are further down in the Preemption Stack. This is because, if they held such resources, they would be less than or equal to the D of the running task and this the scheduler does not allow.

A second property is that there is no transitive blocking, because a process that is blocked due to shared resource usage only has to wait for this only blocker to release the resource. Stated more formally, task τ_h at the head of the Released Queue is blocked by τ_s on the Preemption Stack despite having a higher priority $(d_h < d_s)$ because $D_h = D_s$ prevents the preemption of τ_s. It can be proved that under these circumstances $\Delta_s^{'} \leq D_h < D_s$. Due to the full ordering property of D's and Δ's on the stack the number of blockers has a maximum of 1. This is also a property of the Priority Ceiling protocol [37], the first protocol that introduces static priority inheritance, similar to our static deadline inheritance.

The schedulability analysis is only moderately more complex when considering resource sharing as well. The processor demand and workload functions do not change, because the work that needs to be done and when it needs to be done is the same. But we do have to take into account now that *one* task, not more, may *block* another to access to the CPU. To illustrate the process, we use the same specification as the one given before, this time in a more convenient mathematical

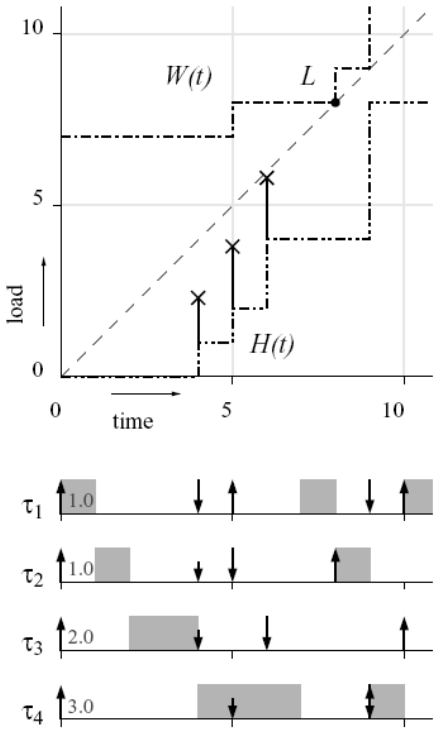

Figure 4.5: Ω_2 is feasible under EDF with nested critical sections

notation, in Table 4.3. Blocking can be represented graphically by adding spikes at time t to the processor demand function, as illustrated in Figure 4.5.

The height of a spike is the result of calculating the *blocking times* of a maximum blocker from the resource specification: at $t = 4$, τ_1 reaches its deadline. Before reaching the deadline, it may have been prevented from being scheduled by a task with a longer deadline, but holding a resource that τ_1 might need. For τ_1, the amount of slack in the schedule needed is 1.3 time units, because that is how long τ_3 might hold resource b. Similarly, at $t = 5$, τ_2 needs 1.8 time units of slack to compensate for τ_4, which might hold resource c, preventing τ_2 from being scheduled. The maximum potential blocking is given by $C_B(t) = \max_\Omega\{C'_{\tau'}|\Delta'_{\tau'} \leq t < D_t\}$ where t' is a nested critical section, C' its cost, $\Delta'_{\tau'}$ its inherited level and t is the length of the interval over which blocking has to be computed. The new admission rule calculates these potential blocking as spikes on the processor demand function at the expiration times of deadlines and declares a task set inadmissible

Ω	$\mathbf{D_i}$	$\mathbf{T_i}$	$\mathbf{C_i}$	***resources*** $\rightarrow \mathbf{\Delta_s}$
τ_1	4	5	1	0.9{a B} $\rightarrow$ (4,0.9)
τ_2	5	8	1	0.8{a 0.2{B 0.1{C}}} $\rightarrow$ (∞,0.8)(4,0.2)(5,0.1)
τ_3	6	10	2	0.2{b}1.7{c 1.3{b}} $\rightarrow$ (4,0.2)(5,1.7)(4,1.3)
τ_4	9	9	3	1.8{a c}$\rightarrow$ (5,1.8)

Table 4.3: The Δ_s are converted to tuples consisting of *inherited deadline* and *usage time*

if one of the spikes crosses the diagonal. If there are no shared resources, there is no blocking (there are no spikes), and the schedulability test reduces to the normal preemptive-EDF schedulability test. If there is one resource, shared full-time by all tasks, the schedulability test reduces to the non-preemptive schedulability test. This schedulability test spans the range between the extremes of completely preemptive and completely non-preemptive scheduling.

A more formal consideration for feasibility analysis under EDF with deadline inheritance is given in Jansen and Laan [19].

4.1.4 DCOS requirements

This section describes some of the design issues of the DCOS operating system, supporting the data centric architecture.

DCOS was designed having in mind two sets of requirements: the requirements of any general operating system to which we added the requirements needed by the data centric architecture.

From the point of view of any operating system design issues we faced, we can enumerate:

- **Hardware abstraction layer** - DCOS must abstract the hardware for its client by creating a hardware abstraction layer (HAL). The HAL is normally made up of drivers that each provide an abstracted access method to specific hardware components (in our case we created drivers for all the peripherals MSP430 processor provides).

- **System calls availability** - User-space software like the modules must be able to do system calls. Because user-space software is compiled separately from the DCOS kernel, the entry points of the system calls aren't know. To solve this a translation scheme was employed.

- **Command shell** - There should be a service of user interaction. The way to do this is to create a shell in which users can enter commands to perform

system tasks. The system gives feed back using the serial line for communication, an externally connected display or the set of LEDs available on the development board.

The employment of the data centric architecture raised new requirements. Some of them are:

- **Real time characteristics** - DCOS needs a scheduler for determining which task may run on the processor at a certain time. Further it should ensure that for a given task-set the real-time constraints for each task are met. Bellow are some more consideration regarding this topic.

- **Data centric architecture** - DCOS is designed to enable a data-centric architecture so the extended message passing mechanism is required. The component responsible for the mechanism is called the *data manager* and the main challenge was designing it by not using too many resources (as memory and processor time).

- **Local file system** - DCOS must provide a way of working with modules. Modules are blocks of software that aren't part of the operating system itself. The goal is that the kernel can load and run modules dynamically. With module support, DCOS will be able to support reconfiguration based on entities becoming available after the network was deployed.

- **Memory management** - One of the goals of DCOS is that it must be able to dynamically reconfigure task and data sets. A task or data set can grow or shrink in size. A solution could be to accept a maximum size for such a set and reserve the memory needed to store the maximum set size. With our severe memory limitation this would mean a waste of space if less then the maximum set size is needed. So a memory allocation scheme has to be designed that can reserve and free memory space while taking up as least as possible space itself.

As mentioned earlier, DCOS was built on top of the data centric architecture concept. This implies that the general concept of *data* is used rather than the *event* one. In order for the decision based on data to work, there are some additional requirements to be met. First of all, all the modules need to declare the name of the data that will *trigger* their action, the name of the data they will need to *read* during their action (this can generically incorporate all the shared resources in the system) and the name of the data they will *produce*. All this information is available for the scheduler to make the decisions.

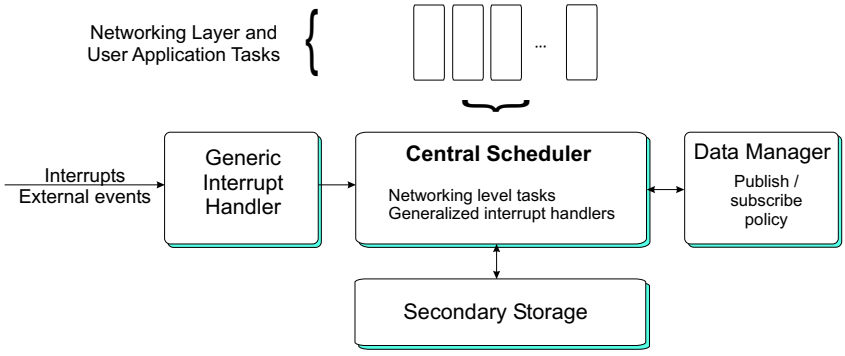

Figure 4.6: DCOS building blocks

From the point of view of the operating system, a new component that takes care of all the data exchange needs to be implemented. This would be in fact an extended message passing mechanism, with the added feature of notifying the scheduler when new data types become available. The mapping of this module in the architecture is the constraint imposed to the protocols to send/receive data via a publish/subscribe mechanism to the central scheduler. An efficient naming system for the entities and the data is needed. Downloading new entities to a sensor node involves issues similar to services discovery. Several entities with the same functionality but with different requirements and capabilities might co-exist. The data centric scheduler has to make the decision which one is the best.

4.1.5 DCOS design

In the following we will take a look at the various mechanisms involved in the operating system we designed (see Figure 4.6). This is a mere highlighting of the system's characteristics, the exact description of the data structures, tools, code examples, etc. are available in the master's thesis [15].

- **Real time scheduler** - The scheduler uses the EDFI mechanism described in Section 4.1.3. The scheduler needs each task to declare some parameters as minimum period, maximum cost, the deadline before which the computation must be performed and a resource declaration list. Due to the limited amount of resources, the implementation of nested critical sections was not feasible. Nevertheless, real time transactions have been used and it has been seen in practice that they provided the needed functionality for the targeted applications. The scheduler offers mutual resource exclusion at system level, eliminating the need of defining and implementing another mutual exclusion algorithm at user level.

In the data centric architecture however, entities respond to data produced by other entities becoming available. Translating this to real-time scheduling means that the release of a task can depend on the finishing of others. Such a task set is said to have precedence constraints. The impact of these constraints on the EDFI scheduling technique is that the presented feasibility analysis does not reflect reality since the order in which tasks are executed is not taken in consideration. In general the finding of an optimal schedule for a set of tasks with precedence constraints is NP-hard [5]. The EDFI analysis can still be used as such, but regards the system as a more complex one; thus the set of tasks might be reported unfeasible, while under the precedence constraints is feasible.

To still determine the feasibility of a schedule with data centric entities we proposed and implemented the following simple approach:

- A task-set consists of one or more directed task-dependency graphs that contain no cycles. The graphs may start with one entity only which implies that entities in the same graph share the same period.
- The graphs are transformed into a schedule where release times and deadlines are adjusted as described in [9]. The resulting schedule can than be analyzed using the normal EDFI analysis.
- The resulting schedules of all graphs are independent with the exception of the resources used. The schedules can now be seen as separate tasks that have their own worst computation time (the sum of the C's of each task in the graph), period (the period of the first task in the graph), deadline (the deadline of the last task in the graph relative to the start of the first task), and a resource usage list (The union of the tasks NCSs). The feasibility analysis can then be concluded by treating the graph schedules as tasks in an EDFI task-set and performing the normal feasibility analysis on them.

- **Queue's** - With EDFI, tasks can reside either in the waiting queue, in the ready queue or on the preemption stack. The waiting list is an unordered collection of tasks which can be transferred to the ready queue when a certain event occurs. When a task is released to the ready queue its absolute deadline is calculated by adding its relative deadline to its release time. The ready queue is sorted on absolute deadline in ascending order. Since the ready queue is initially empty its ordering can be maintained by inserting the task at the right position when it is released which makes the sorting algorithm a linear search $O(n)$ where n is the number of tasks in the ready queue. When the head of the ready queue changes the scheduling condition is calculated and if the condition is true the head of the ready queue will be

pushed on the preemption stack. Because of the scheduling condition and the fact that the preemption stack is empty initially, the preemption stack will be automatically sorted on absolute deadline in ascending order. When a task is completed it will be popped from the preemption stack and added to the waiting list.

In the actual implementation the deadline of a task entry is a 16-bit value. The reason for this approach is that, doing arithmetic on 16-bit values is faster than on 32-bit values on our target platform. This value is only decreased, preventing that the integer value will overflow and represent a moment in the past. It also prevents the need of a global clock value for testing the deadlines, since a deadline of zero always means that the deadline is due.

Using the absolute deadline for the tasks in the running queue is disadvantageous from the point of view of storing and updating this value. By storing the deadline difference with respect to the task that is in front of a task, instead of storing the absolute value, only the deadline of the first task of the linked list representing the ready queue has to be decreased. When the head of a list is removed the remaining value of its absolute deadline is added to the new head so that the integrity of the deadlines is maintained. Attention is paid to the insertion and extraction of tasks from the queue.

- **Context Switching** - EDFI uses a single stack which has the advantage that the context of currently running tasks doesn't have to be saved somewhere else but that it can be left on the stack itself. The new context will be created on top of it. It is not possible that a preempted task (it has a context on the stack) preempts the task it was preempted by or any task that has a context higher on the stack. A task with a context on the stack cannot be rescheduled until it is removed from the stack. The context itself consists only in saving the set of CPU registers, a pointer to the exit routine and the local stack of the task.

  The stack-base is a pointer to the start of the stack space for the task and is saved in the task descriptor. This value can be used to remove an entire context from the stack with the exception of the CPU registers. This way, when a context is removed (due for example to the fact that it did not finish before its deadline), the state of the previous task is restored by restoring the registers. When a task finishes on time then, because of the structure of the context, it will return to the exit routine.

  When a new task must run the context can be created by pushing the CPU registers, the address of the exit routine, the address of the new task and a clean status register on the stack and performing a return from interrupt

(*RETI*) instruction. The CPU will then continue execution from the new task with a correct context on the stack.

- **The scheduler** - The scheduling system is data driven meaning that availability of new data will trigger its execution when such a piece of data causes a task to be released. The scheduling system will also come in effect when a tasks exits to the exit routine.

  When an interrupt occurs and its interrupt service routine releases a task the *schedule* primitive is called causing the scheduler to run. The first thing the scheduler does is updating the time of its queues by subtracting the number of clock cycles since its last execution from the absolute deadline values of the queues. When a task exits it will return to the exit routine. In this routine the queue time is also updated first but it will also remove the finished task from the top of the preemption stack. In both cases the scheduler will then continue with the update of the states of the queues.

  In this state the scheduler will check the release queue and the preemption stack for tasks that have a relative deadline of zero. These tasks missed their deadline and *are moved back to the waiting queue*. If a task on the stack missed its deadline then also the task's context is removed from the stack. This is done by updating the stack pointer with the stack base of the task that was removed last. The scheduler does not take care of any resource that might be in use by the task that is removed. Then, when all the queue's are updated, the scheduler calculates the EDFI schedule condition using the head of the ready queue and the top of the preemption stack. If the condition indicates that the the head of the ready queue must run, then a new context is created on the stack. Otherwise the stack is left alone. The scheduler will then set its deadline timer to the earliest deadline, which is either the deadline of the head of the ready queue or the top of the preemption stack, and continue execution with the task which has the context on the top of the stack by performing a return from interrupt instruction.

  Because of the existence of a deadline timer which generates an interrupt when reaching zero, the scheduler will always be activated on time.

- **Data manager** - The main goal of the DCOS operating system is to support a data centric architecture. In such an architecture data and the validity of that data, is considered the most important. Data is what is produced and consumed by tasks and what causes tasks to respond. The data manager is the entity that regulates the flow of the data between the tasks. The data manager and real-time scheduler combined will act as the proposed data centric scheduler.

Each piece of data has an associated data type. Data types have a fixed size and have a unique identifier. The operating system maintains a table that holds a descriptor for each data type. Tasks can produce and consume a subset of the available data types.

For tasks to have access to data types, the data manager provides a pointer to the memory area of the needed data type. To prevent a task from being preempted while accessing the memory of a data type, data types are declared as resources. This list is then combined with the existing resource list and then the needed inherited deadline is calculated.

The power of the data centric architecture is not having a shared collection of data types, but how the system behaves when something happened to a data type. Through a publish/subscribe system the data manager enables tasks to respond (subscribe) when another task altered a certain data type (publish). Combining this with the real-time scheduler, the tasks will only run when necessary with a guarantee on the real-time execution.

When a data type is published the data manager has to know which of the entries in the subscriber table have to be released. One could do this by adding a data type ID to the subscriber entry and traverse through the subscriber table releasing every entry found that has the same ID as the type being published. The disadvantage is that it would be quite slow. A task may publish several data types which would result in the same amount of searches through the subscriber table as types being published. To overcome this, the method described in [24] is used. Sacrificing a small amount of memory, the scheme allows subscribers to be indexed directly with a constant search time per subscriber, considerably increasing the access time.

The publish/subscribe mechanism is also used for system interrupts. When for example a timer goes off, its interrupt service routine will publish a system defined data type. That way tasks can respond to system events by subscribing to the special data types.

- **Dynamic Loadable Modules** - Modules are tasks that are compiled separately. By having module support in the operating system, modifications to an application can be done more efficient. Instead of updating the complete application only a subset of the modules has to be changed, resulting in less data traffic and thus less energy consumed. Another advantage is that nodes can be more heterogeneous in the software point of view which results in less occupied memory space and better dedicated operation possibilities.

  The normal task information has to be added to the module. This consists in: the real-time properties (deadline, period and worst case CPU time),

the resources the task uses and the access type, the specification of the data types it uses, the data types to which it can subscribe.

Modules must be dynamically loadable meaning that the kernel can load a module in its code memory at an arbitrary location and execute it from there. The difficulty with dynamic load-ability is that compiled code has absolute reference to subroutines, variables, etc. Therefore relocation information has to be added to the module. Relocation information tells which addresses of the code refer to a memory location and what that memory location is relative to the start of the code section. Having that, the operating system can load the module code at any location and using the relocation information, alter each address in the code so that the memory location referenced is correct in respect to the location the kernel loaded the code. The extraction of relocation information must be provided by the compiler and linker for the method to be possible. Post processing is performed on the executable file generated by the compiler to extract the information needed by our system software.

- **Module Transfer** - A dynamically loadable module (DLM) is transferred to a node over the radio or the serial port where on arrival it is stored in the EEPROM. For the communication a packet protocol called *GOOSE* was developed. GOOSE is a simple protocol in which the DLM is divided into small packets which are uploaded individually so that the node can store it in RAM temporary before writing it to the EEPROM. When the node is ready with a packet, it will acknowledge the sender so that the next one can be sent. The protocol can be used to upload to the EEPROM any block of binary data. In GOOSE a binary file is divided into 64-byte blocks. Each block get its own number to indicate its position in the original file, and a field containing the total number of blocks. For each block a 16-bit checksum is also computed. After receiving a block, the receiver node can perform two integrity checks: calculating the checksum based on the received data and comparing it to the received checksum, and whether the received block was expected based on its number and the number of the blocks already received.

  The communication is initiated by the sending party that will send a request (REQ) control-code and wait for a response. The receiver will respond with a ready to receive (RDY) message. The sender will then start sending packets one at a time. Each packet is started with a start of packet (PTS) code and ended with an end of packet code (PTE) to which the receiver must respond with either a packet okay (ACK) or packet corrupt (NAK) message. If the packet was good the sender will send the next packet or an end of file

(EOF) message if it just sent the last packet. If the packet was corrupt the sender will send the same packet again. After sending the end of file marker the sender will wait for the receiver to send a save and sound (SAS) code to indicate that everything was received correctly.

- **Module Storage** - On receiving a module the communication protocol must be able to store it in the EEPROM. A simple file system is used to help writing files and to keep track of where each file resides in the EEPROM. The file descriptor table is ordered meaning that the order of the file descriptors is also the order of the files in the data space. Creating a file means setting the name of the first free descriptor and setting the offset of the next descriptor in the table to the offset of the new file plus the size of the file. In this way the size of file can be deducted by subtracting its offset from the offset of the file descriptor directly after it in the table. The descriptor table is 4080 bytes large and with a descriptor size of 16 bytes this means that the file system can hold a maximum of 255 files.

  The file system allows the user to perform some basic operations. The user can for example format the file system, create a file, search for a file, read and write a file.

  The used EEPROM has the following characteristic: a bit set to 1 can always be changed to a 0 but not the other way around. For a 0 to become a 1 again the sector the bit is in must be erased. The only way to erase a file without erasing files also in the same sector, is copying the entire sector to an unused sector and erase the sector the file was in, then rewrite the sector without adding the file to be erased. This means that with only 4 sectors available on the target hardware, 25% of the memory would be wasted. No erase function is supplied but only a file system format. This is because the expected lifetime of the node, however as long as possible, is shorter than the time it will take to fill the entire EEPROM with software updates considering the imagined application. When the EEPROM becomes full nevertheless, it is always possible to format the file system and upload only the latest versions of the modules.

- **Dynamic Task-Set Alteration** - When a node has several modules stored in its EEPROM, it is possible to have them form a new task-set to be executed. Every module defines the data types it uses in its local data type table. From these local type definitions the global data type table has to be build. The task-set load and merge procedure begins with the reservation of flash memory for the global data type table. This table is first filled with the data types that represent the system events. After that each module in the task-set is loaded using the following steps:

- Every data type in the module's local data type table that does not exist in the global data type table is added there;
- The module's data type usage list is altered so that its values are referencing the data types in the global table instead of the local one, and its stored in flash;
- The module's resource list is copied to the flash memory;
- The module is loaded. First the RAM and flash memory needed for the module's data and code section are allocated. After that the module's code section is written into the allocated flash memory area and it is patched using the module's relocation information.

When all modules are loaded the task specification table can be created by using the values obtained in the module loading process. Finally the subscriber table is created by searching for each data type if there are modules subscribed to it. The data manager and the scheduler can now be initialized with the new data type, subscriber and task specification tables, and the the system is put back online. The system will then execute the new task-set.

- **Heap Management** - The operating system does heap allocation using what we call the *3FS* algorithm. The allocation algorithm searches for the first free block of memory that is large enough to fit the requested size and shrinks the free block by that size. The chopped off top of the block is then marked allocated. When an allocated block is freed it is marked free and if the block just before and/or after it is also a free block then the blocks are merged into a single free block so that fragmentation is reduced.

  The first advantage of the algorithm is that it only needs a single value per allocated block and two values for a free one on overhead. These values are stored in the block itself. It is no problem to use a bit more memory overhead on a free block than a allocated block because it is unused memory anyway. Allocating the block frees up its overhead. The other advantage is that it only keeps track of the free blocks and in that way reduces the length of the linked list and thereby the search time.

  To describe a free or an allocated block the algorithm uses two types of descriptors. The descriptor of an allocated block consists of a single value: the size of the block. The descriptor of a free block uses two values: the amount of free space in the block and a pointer to the next free block . The descriptor of each block is stored inside the block itself, at the beginning of the block.

  When a client requests the allocation of a memory block with size M the algorithm will traverse through the linked list starting with the free block descriptor at the start of the memory space and will compare the *size* value

with M. If $size >= M$ the free block is large enough to hold the requested size and an allocated block descriptor is created at memory address $memorystart + (size - (M + 2))$. The *size* member of the allocated block descriptor is set to $M + 2$ and the *size* value of the free block descriptor is set to $size - (M + 2)$. If the free block is not large enough to hold the requested block size, the algorithm will try again with the next free block in the chain. If there exists no contiguous block of free space large enough to hold the requested size and the algorithm will return *NULL* indicating the failure of the allocation.

When a client wishes to free a previously allocated block it will pass the pointer to the allocated block (as received after allocation), to the heap management algorithm. The algorithm will free this block and then go through the linked list of free space to see if the newly freed block is can be merged with another neighboring free block.

- **Drivers** - Drivers are software modules that provide an interface to the services a device can offer. For the target hardware drivers are needed for the radio transceiver, RS232 communication and external EEPROM. These are all character devices meaning that a minimum of a single character can be read from or written to them. We adopted the *POSIX* standard for accessing devices. With POSIX each device can be accessed through the file system using the normal file operations while keeping the amount of API functions to a minimum.

  The standard API calls for character and block devices are: *open, close, read, write, init, lseek* and *ioctl*. Devices can be used through these operations as follows. The device has an own unique name within the system. To setup the device so that it becomes operational the *init* operation is called using the device name to identify the correct device. This is most of the time done only once during the initialization process of the complete system. When the device is operational a client can try to obtain a handle to the device through the *open* function. If this call returns a valid handle the client is allowed to do other operations on the device, otherwise not. The client can then *read* a stream of bytes from the device and *write* a stream of bytes to the device. If the device supports random access the client is able to move the read/write pointer with the *lseek* call. Any device specific control, for example setting the signal strength of the radio transceiver, can be done by using the *ioctl* call. When the client is done with the device it calls the *close* functions to release its device handle.

  The operating system does not have a file system and for accessing the devices it uses a different naming method than POSIX suggests. Each device

in DCOS has a unique identifier that is a pointer into a handler table. A handler is located inside the driver and performs the translation of the different API calls. When a client for example calls the *open* function it uses the device identifier to name the device and the kernel implementation of *open* will call the handler of the identified device to complete the operation.

Added advantage for this method is that drivers can be made replaceable, provided that the handler table can be rewritten, because replacing a driver only means altering the address of the driver handler routine in the handler table.

- **Library** - For software that is compiled separately from the kernel, a library is provided so that this software can access the kernel's system calls. To call a function residing in kernel space from user space is normally done through a trap. With a trap the program in user space generates a software interrupt that has its handler in kernel space. Based on the state of the stack, registers or the software vector called, the handler will perform the appropriate system call in kernel space. The main advantage of this is that the user space program doesn't need to know the exact location of the system call in memory. A library can be implemented by creating a small function that executes the trap for each system call available.

  However, the target hardware for the operating system lacks this trap functionality. To provide access to the system calls even then, the trap method can be emulated. The address of the equivalent of the kernel trap handler will be stored at a fixed location in memory called the *Kernel Vector*. For each system call a small front end function is created that has the same definition as the system call, and this front end pushes a function number on the stack and will direct the execution to the address stored at "Kernel Vector". The handler comes in effect and looks up the address of the system call indicated by the pushed function number. The handler removes the function number from the stack, pushes the address of the system call and returns. The system call is now executed as if it were called directly by the application. The library is implemented by the set of small front end functions.

- **The Shell** - The operating system comes with a shell task that can be used to execute commands on the node. When connecting the node to the serial port of a PC and running a terminal emulation program the shell provides a command line interface.

  Operations are executed by entering two-character commands followed by an optional unsigned integer value. The shell will read characters from the

Component	**Code size**	**Global RAM**	**Local RAM**
Necessary components	*(bytes)*	*(bytes)*	*(bytes)*
Clock	516	4	8
Data manager	924	6	32
Heap allocator	240	1034*	6
Impulse handler	212	4	2
Scheduler	1642	18	26
System init	266	0	2
Sub Totals	3800	1066	76
Optional components			
Console	344	84	14
Device mapper	282	2	4
EEPROM driver	1246	4	26
EEPROM FS	1000	0	28
Flash FS	412	0	16
Goose receiver	622	14	28
I^2C driver	428	16	10
IO Expander driver	158	1	6
LCD driver	1158	8	28
LED driver	226	0	6
List template	1228	0	16
Module routines	1018	0	160
Radio driver	2510	72	16
Serial driver	492	24	14
Shell	1226	12	32
Symbol table	58	0	0
Task-set reconfigurator	1068	0	124
Timers	500	20	10
Totals	17776	1317	614

*Including 1024 bytes heap space

Table 4.4: DCOS memory usage list

Criteria	**DCOS**	**TinyOS**	**PEEROS**	**Salvo**
Scheduling method	Preemptive	FIFO buffer	Preemptive	Cooperative
Task switches per second	12500	40000	4000	40000[1]
Maximum number of tasks	$\frac{Heapspace}{14}$	unknown	256	unknown
Maximum number of scheduled tasks	$\frac{Heapspace}{14}$	unknown	10	unknown
Maximum task runtime	∞[2]	unknown	∞	∞
Available ROM	61440	8192	61440	61440
Available RAM	2048	512	2048	2048
Kernel ROM	3800	432	4344	1550[3]
Kernel RAM	32[4]	46	78	48[3]
Dynamic task priorities	Yes	No	No	No
Resource access protocol	Yes	No	No	No
Dynamic memory allocation	Yes	No	No	No
Dynamic loadable modules	Yes	No	No	No

[1] Using a MSP430 running at 8 MHz
[2] If the scheduler quantum guarding option in enabled this figure would be 2s
[3] Using Salvo tu6pro
[4] Excluding the heap space

Table 4.5: DCOS compared to other sensor network operating systems

serial input into its command line buffer until it receives a carriage return. When the carriage return is received it parses the command line buffer to check if it is correctly formatted. If so, the shell will look up if the command value is a valid command. The shell executes the command if it is valid or returns an error message otherwise.

The current available commands allow the user to inspect the currently active tasks, to perform operations on the modules available in the EEPROM, etc. The command set can be easily extended.

4.1.6 Conclusions

The prototype operating system that we developed fulfills the goals of supporting a data centric architecture and meeting real-time constraints at the same time. All of these are done using the limited resources of our development platform, leaving enough space for user applications.

The operating system uses a minimum of memory resources - see Table 4.4. The global RAM indication shows in fact the needed RAM for the local variables while the local RAM indicates the maximum stack size needed.

In Table 4.5 a comparison between DCOS, TinyOS, PEEROS and Salvo is given.

Looking at the comparison table it can be concluded that DCOS has a higher context switch latency and higher ROM usage than TinyOS and Salvo, but offers more functionality. PEEROS has an even higher latency than DCOS while using about the same amount of ROM and more RAM.

The kernel supports the loading of executable modules at arbitrary memory locations and offers the application programmer dynamic memory allocation, an easy to use list template, and drivers for the devices on the sensor node. The operating system is robust as well as user-friendly.

The conclusion of this section is that by using the concept of data centric architecture in combination with the featherlight scheduling scheme proposed in [20], it is possible to create an operating system suited for wireless sensor networks. Its capabilities outperform by far the other best existing operating systems by using a comparable amount of resources. DCOS was the starting point for the commercial product named AmbientRT [16] and is already used in several research projects.

4.2 Wireless sensor networks simulator

In this section we introduce a new simulation template for wireless sensor networks. It is built upon the OMNet++ [42] networks simulator and it contains the main features needed for simulating protocols designed for large ad-hoc networks of autonomous nodes with sensing capabilities.

The template provides besides the easy integration of new code a graphical interface useful for debugging or illustration purposes and a collection of tools that help modifying or implementing new features in an easy fashion.

4.2.1 Related work

Before describing the main features of our template, let us first take a look at the other existing alternatives and explain why we chose to develop a new tool. The authors of [17] present an overview of some of the tools used for simulating distributed wireless networks. We will remind here the most important ones:

- **ns-2** [30] - imposed itself as a standard for network simulations. It is a very trusted simulator, supported by both US Defense Advanced Research Projects Agency and US National Science Foundation. *ns-2* is a discrete event simulator organized according the OSI model and primarily designed to simulate wired networks. Several implementations brought extensions for simulating wireless networks with mobile nodes [25, 27]. *ns-2* is written in C++ and can be configured by using OTCL (a version of object oriented TCL) scripts. While configuring a simulation using already implemented algorithms is as simple as creating a scenario file, not the same can be said about implementing a new algorithm and adding it to the simulator. This is quite a challenge for the beginner (the lack of proper documentation making it even harder). The simulator needs a lot of resources to run. It has a problem of scalability as networks of thousand of nodes are hard to simulate.

- **GloMoSim** [45] - is another open source simulator. It is designed to handle the simulation of tens of thousands of sensor nodes. It is written in Parsec and hence can be run on shared memory symmetric processor computers. To solve the scalability problem, GloMoSim partitions the network in different balanced sub-networks and runs each one on a different processor. Then it uses *node aggregation* by implementing models of the networks layers that nodes are able to share (they do not create an instance at each node, saving this way important resources). The communication between the various entities is done by using time stamped messages (is it a discrete time event simulator).

- **OPNet** [10] - is the most widely used commercial simulation environment. It is a discrete event simulator that was first proposed by MIT in 1986 and converged to a commercial tool containing an extensive list of protocols for network simulation. The accuracy of ad hoc networks simulation has been already questioned [8], as it shows completely different results from the previously presented two simulators.

- **OMNet++** [42] - is a general purpose discrete event based simulator. It can be used to simulate any sort of network and in particular wireless, ad hoc networks. Although it is designed in a clean manner and has extensive documentation and examples, the extensions for wireless sensor networks are still in their infancy. Several extensions (such as [12]) exist allowing wireless sensor networks to be simulated. It is written in C++ and has extensive visual debugging capabilities. Although it is an open source project sustained by a large community, lately, a commercial version has been released [41].

Apart from the previous examples, a lot of other tools have been developed. From the most well known ones (often extensions of wired networks simulators) such as: QualNet [31], NAB [18], J-Sim [26], SWANS [3], etc. and to the specific frameworks developed in the major projects dealing with sensor networks (e.g. [39]) there is a wide range to choose from. The major disadvantages of all these simulators are that they are either too general, such that the implementation of a specific protocol requires the user to write also models for the layers it makes use of, or too specific, making the implementation of a specific protocol very difficult. Probably the wireless sensor networks simulators will converge to a small subset when it will be decided upon a standardized set of protocols to be used.

4.2.2 Simulation template characteristics

The initial code for the mobility framework was developed for the EYES project, in order to have a unified simulation platform to compare and integrate the algorithms developed by the various project participants. The original framework has been extended to a more general framework, solving some of the problems raised during the early stages. In the following we present some of its most important features:

- **Wireless communication emulation** - OMNet++ does not contain native support for wireless communication. The messages are sent on links connected to the ports of the various entities. We emulated the wireless connectivity by dynamically creating/destroying of the communication links

between the entities based on their position in a two dimensional plane and of the concept of *transmission range*. This model is easy to implement and runs very fast. A full channel model can also be easily implemented, but unfortunately that will make the simulation more slow. Approximating the packet error rate is under investigations and several models (based on the transmission range concept have been proposed [36]). At the simulator level, a central component holds the connectivity matrix of the entire system. This global view is available to the user for statistics purposes, etc.

- **Mobility framework** - The nodes of the network are assumed by default to be mobile. Their exact trajectories can be specified in a configuration file, using one of the tools associated with the simulator. We implemented the Random Waypoint Model, taking care of the speed decay problem as suggested in [44]. Based on the granularity of the simulation time, the mobility file is generated before the actual simulation and can be reused, saving important simulation time (its computation is resource intensive especially when it is done based on a messaging system). The mobility generation tool will be enhanced to include some other well known mobility patterns [7].

- **Protocol sessions and statistics collection** - A lot of effort goes into the description of a distributed algorithm for the identification of the session of the protocol. Usually the data structures are duplicated for each instance of the same protocol running in the network and each component redefines the management of the local database. We proposed an approach where the user can choose for the simulation software to take care of this issue automatically. Each instance of a protocol is uniquely identified and the database maintenance (adding the data structure of one session, deleting the expired sessions, etc.) are done automatically. Closely related to this issue is the problems related to the statistics data collection. Specialized functions are provided to the user to make this operation easy and to automatically manage the database of collected data.

- **Library of components** - One of the issues with the OMNet++ simulator is the fact that there is no standard library of components. A lot of work goes into combining modules belonging to different applications. Designing the modules according to the data centric architectures implies that all the modules will have the same set of connections and will declare which data types they make use of or they produce. This brings the concept of building library of protocols one step closer to reality.

- **Processor model** - We have implemented a module that can be extended to emulate the used processors on the development boards. This makes it

possible to compile the protocol code with the compiler associated with a specific processor and then input the executable in the OMNet++ simulation framework. This feature adds a lot of advantages (nevertheless with the cost of some additional processing power): the synchronization between sensor nodes can be accurately modeled, the costs of the tasks can be estimated, the code needs to be written only once for the simulator and for the hardware platform, etc.

- **Energy management** - The energy management function is integrated in the simulation code. Modules can declare an estimate of the energy they consumed for performing a certain operation and the simulation framework monitors the traffic over the wireless interface. These two mechanisms combined with a model of the power source plus optionally power scavenging techniques make possible to have at all times an estimate of the energy resources inside each sensor node and thus to design protocols that make use of this information.

- **Failures** - The wireless nodes are subject to different kinds of failures. The simulator allows to define failures for individual nodes, failures that affect regions of the map, etc. using a simple map definition mechanism. This mechanism can be used also to define interferences, sources of signal for the sensing simulation, etc.

4.2.3 Conclusions

The simulation frameworks that we wrote is an important tool for the development of protocols for wireless sensor networks. It aids the developed to easily detect the bugs in the protocol design by the use of an extensive, user friendly, graphical interface and the support of a powerful logging mechanism.

We have already used this simulation template to develop and test the large majority of protocols developed by the Twente University during the EYES project [33]. The second version of the tool is being used in a variety of research projects including the Smart Surroundings project [34].

4.3 Conclusions and future work

In this chapter we have described two ways in which the data centric architecture can be used in the field of wireless sensor networks, by designing an operating system and a simulator based on it.

In the first part of the chapter we have described DCOS, a new operating system targeting sensor networks. It offers superior capabilities to the already existing operating systems at a comparable resource cost. DCOS includes also a novel scheduling scheme allowing users to make use of real time features included in the system. It is interesting to notice that the system is designed having the event-driven architecture in mind and is optimized to be energy efficient (small running time overhead, efficient mechanisms offered allowing superior protocols to be implemented, idle processor time spent in power down modes).

The second part of the chapter described a simulation framework we have developed on top of the OMNet++ simulator. The simulator framework allows the user to test the behavior of a large network of mobile wireless nodes, reducing the time spent for designing and testing of distributed algorithms.

The future work consists in improving the operating system by correcting the scheduling mechanism to include the influence of the scheduler itself in it. The current work is focused in the integration of the two tools, such that the operating system behavior can be simulated also in our framework.

Bibliography

[1] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, May 1991.

[2] T.P. Baker. Stack-based scheduling of real-time processes. *The journal of real-time systems*, 3(1):67–99, 1991.

[3] R. Barr. An efficient, unifying approach to simulation using virtual machines. In *PhD Thesis*, May 2004.

[4] S.K. Baruah, A.K. Mok, and L. Rosier. Preemptive scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the Real-Time Systems Symposium*, pages 182–190, Dec 1990.

[5] G.C. Buttazzo. *Hard Real-Time Computing System, Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 2000.

[6] Ambient Systems BV. http://ambient-systems.net.

[7] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. In *Wireless communication and mobile computing (WCMC)*, 2002.

[8] D. Cavin, Y. Sasson, and A. Schipher. On the accuracy of manet simulators. In *Proceedings of the second ACM international workshop on principles of mobile computing*, pages 38–43, 2002.

[9] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Journal of Real-Time Systems*, 2, 1990.

[10] F. Desbrandes, S. Bertolotti, and L. Dunand. Opnet 2.4: an environment for communication network modeling and simulation. In *Proceedings of European Simulation Symposium, Society for Computer Simulation*, pages 64–74, 1993.

[11] E.W. Dijkstra. *Cooperating sequential processes*, pages 43–112. Academic Press, 1968.

[12] W. Drytkiewicz, S. Sroka, V. Handzinski, and A. Koepke. A mobililty framework for OMNet++. In *Proceedings of the 3rd International OMNet++ Workshop, Budapest*, 2003.

[13] RedHat ecos. http://ecos.sourceware.org.

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *ASPLOS 2000*, Nov 2000.

[15] T.J. Hofmeijer. The development of system software to support a data centric real-time architecture for sensor networks. Master's thesis, Twente University, july 2004.

[16] T.J. Hofmeijer, S.O. Dulman, P.G. Jansen, and P.J.M. Havinga. AmbientRT - real time system software support for data centric sensor networks. In *Proceedings of ISSNIP 2004, Australia*, 2004.

[17] L. Hogie, P. Bouvry, and F. Guinand. An overview of MANETs simulation. In *Proceedings of First International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*, 2005.

[18] EPFL Information Sciences Institute. The nab (network in a box) wireless network simulator, 2004.

[19] P.G. Jansen and R. Laan. The stack resource protocol based on Real-Time transactions. *IEE Proc.-Software*, 146(2):112–119, Apr 1999.

[20] P.G. Jansen, S.J. Mullender, P.J.M. Havinga, and H. Scholten. Lightweight edf scheduling with deadline inheritance. Technical report, Twente University, may 2003.

[21] A.E. Kalman. *Salvo User Manual*. Pumpkin, Inc, 2003.

[22] A.E. Kalman. Using the salvo rtos on ti's msp430. Presented at TI's second Annual MSP430 Advanced Technical Conference, Ft. Worth, Texas, November 2003.

[23] H. Kip, T. Lentsch, and P. Havinga. *Energy Efficient Sensor Architecture, EYES Project Deliverable 2.1*, August 30 2003.

[24] A. Köpke, V. Handziski, J.H. Hauer, and H. Karl. Structuring the information flow in component-based protocol implementations for wireless sensor nodes. In *Proc. Work-in-Progress Session of the 1st European Workshop on Wireless Sensor Networks (EWSN)*, Technical Report TKN-04-001 of Technical University Berlin, Telecommunication Networks Group, pages 41–45, Berlin, Germany, January 2004.

[25] IBM India Research Lab. Bluehoc.

[26] Newcastle University Computing Laboratory. Javasim's users guide.

[27] CMU Monarch. The CMU Monarch Project's wireless and mobility extensions to NS, 1998.

[28] J. Mulder. Peeros preemptive eyes real-time operating system. Master's thesis, Twente University, april 2003.

[29] J. Murray. Inside microsoft windows ce (microsoft programming series), 1998.

[30] The network simulator ns 2. http://www.isi.edu/nsnam/ns, 2004.

[31] Scalable Networks. Qualnet user manual, 2004.

[32] PalmOS. http://www.almsource.com/palmos.

[33] Eyes European Project. http://www.eyes.eu.org.

[34] Smart Surroundings European project. http://www.smart-surroundings.nl.

[35] R. Rajkumar. *Synchronization in Real-Time Systems, A priority Inheritance Approach*. Kluwer Academic Press, 1991.

[36] N. Reijers, G. Halkes, and K. Langendoen. Link layer measurements in sensor networks. In *First IEEE Conference on Mobile AdHoc and Sensor Systems (MASS 2004), USA*, 2004.

[37] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sep 1990.

[38] L. Sha, R. Rajkumar, and S. Sathaye. Generalised rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, Jan 1994.

[39] TinyOS Community Site. http://www.tinyos.net.

[40] Texas Instruments. *MSP430x1xx Family, User's Guide*, 2003.

[41] Omnest the OPEN simulator. http://www.omnest.com.

[42] A. Varga. Omnet++. In *Software tools for networking, IEEE Network Interactive, v.16(4)*, July 2004.

[43] VxWorks. Wind River, http://www.windriver.com.

[44] J. Yoon, M. Liu, and B. Noble. Sound mobility models. In *Proceedings of MOBICOM 2003*, 2003.

[45] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Proceedings of Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

Chapter 5

Localization techniques

This chapter focuses on the localization problem in distributed wireless sensor networks. In the first part, the current state of the art of the field is presented (algorithms, prototypes and invention patents). Next, three related research topics are presented: a precision based localization algorithm for static networks, the statistics associated with random deployment of nodes in the one dimensional space and two statistically enhanced localization schemes for static networks.

For a sensor network to work as a whole, some building blocks need to be developed and deployed in the vast majority of applications. Basically, they are: a localization mechanism, a time synchronization mechanism and some sort of distributed signal processing. A simple justification can be that sensed data hardly has any meaning if some position and time values are not available with it. In the following we will focus on the localization problem in sensor networks.

The self-localization of sensor nodes gained a lot of attention lately [8, 22, 30, 10]. It came as a response to the fact that Global Positioning System (GPS) is not a solution due to high cost (in terms of money and resources) and it is not available or provides imprecise positioning information in special environments as indoors, etc. Information such as connectivity, distance estimation based on radio signal strength information (RSSI), sound intensity, time of flight, angle of arrival, etc. were used with success in determining the position of each node within degrees of accuracy using only localized computations.

The position information once obtained is not used only for characterizing the data, but also in designing better networking protocols, for example, leading to more efficient routing schemes based on the estimated position of the nodes [57]. At the same time, obtaining the position information at each node can be seen as

an application as such.

In this chapter we will focus on various aspects of the localization problem. Section 5.1 presents the state of the art of the field. We will briefly describe the available theoretical results, the most well known existing prototypes and also the registered (most relevant) invention patents related to this topic.

The following three sections are dedicated to the improvements of existing techniques. Section 5.2 focuses on distance-based algorithms applicable in the situation when something is known about the precision of the distance measurements. Section 5.3 focuses on randomly deployed sensor networks and builds on top of this simple assumption - the one dimensional deployment case is studied and the already existing results are discussed for the two dimensional case. General theoretical results are derived and then applied to two existing algorithms producing significantly improved results (Section 5.4). The chapter ends with conclusions, directions for future work and our personal vision about what should be the ultimate localization protocol.

5.1 State of the art

The localization problem has been studied for a long time and recently a shift of interest towards it has been initiated. The most well known system in use is the GPS [42]. Both military and civil parties can make use of it, and its success has already been verified: hardly can anyone envision a modern car that is not equipped with it. A second system is currently being tested and made ready for deployment. The first two satellites of Galileo, the European version of GPS, are being under test and will be launched next year (2006).

Localization in indoor environment is still a problem to be solved (take for example a large airport or a conference hall). Basic solutions include deployment of large numbers of guiding signs, interactive maps and information points. As Wireless Local Area Networks (WLANs) gained more importance nowadays, a new system has been envisioned and started being commercialized lately. Users carrying laptops or Personal Digital Assistants (PDAs) can compute their position by analyzing the characteristics of received radio signal from the base stations and use these estimated distances together with the map of the base stations. This system it is still in its infancy (although commercial products are already available), the precision of localization being quite low but probably the technology will become more mature in the future [12].

Another alternative for finding ones position is by the usage of the mobile phones. By combining information from several base stations, the position of a person carrying a mobile phone can be estimated with certain degrees of precision

if the device is online. Unfortunately, privacy and security issues raise, stopping this technology from being used in applications available to regular users.

The localization problem from the point of view of wireless sensor networks is very attractive and interesting to study. These networks have as main goal monitoring features of the environment, so, collecting data is their basic motivation to exist. As data has hardly any meaning if information on location and time is not provided with it, localization schemes are one of the basic blocks of this technology. Security and privacy are also addressed, mainly by using smart lightweight encryption schemes or completely passive environments (the user can locate himself using the ad-hoc infrastructure, while the infrastructure can be designed in such a way to be restricted from knowing anything about the users position).

5.1.1 Prototypes and invention patents

The most well known positioning system is GPS [42]. It works very well in open environments where the direct line of sight between the user and the satellites exists. Before its time, systems such as MIT's Loran positioning system [42] (using time difference of arrival of radio signals) and Transit [42] (first operational satellite based navigation system using the measurements of the Doppler shift of signals received from the satellites) have been used.

Unfortunately, devices incorporating GPS receivers do not work inside indoors environments because of poor radio reception, they consume a lot of energy and require a substantial amount of time to determine position with high accuracy. These are just a few reasons that make GPS unpractical for wireless sensor networks scenarios.

Wireless sensor networks and distributed embedded systems in general gave researchers lots of ideas about how to solve the localization problem. In the following we give a short list with existing prototypes (please refer to [36] for the full survey):

- **GPS-less system** [8] - designed by Bulusu et al. The system is suited for small, low cost devices deployed in an outdoor environment. The system uses RF-based signal technology and proximity based position estimation. It is expected to work indoors as well but with a decrease in accuracy.

- **Active Badge** [56] - is one of the early centralized indoor personal location systems that makes use of infrared technology. Each person in the office wears a badge that emits infrared signals captured by the infrastructure and routed towards a central master server. An example application was routing the phone calls towards the closest terminal and it was a success.

- **Active Bat** [24] - came as a response to the fact that the Active Badge system is not able to provide fine-grained tridimensional location information which is needed by many applications. The Active Bat system makes use of ultrasound technology to estimate distances and to compute the exact position of the *bat*. An accuracy of $3cm$ has been reported.

- **Cricket** [45] - is a proximity based localization technique. The *listeners* are able to determine to which subspace of the environment they belong based on the information received from the infrastructure and from the distance estimates (these are computed from the difference of arrival times of a radio and ultrasound signal).

- **RADAR** [3] - makes use of the characteristics of the radio signal to locate the user indoors. A centralized system gathers signal strength from multiple receivers and performs triangulation to compute the position of the user.

- **Pinpoint 3D-iD** [44] - is similar to the RADAR system but it is more expensive. It makes use of proprietary base stations and tag hardware. The accuracy is between one and three meters.

- **SpotOn** [25] - the distance between tags is estimated using RSSI. The system is designed to make use of the wireless devices to get a relative position rather than using some fixed infrastructure.

- **SmartLOCUS** [27] - uses differential time of flight of radio signals and ultrasound pulses to determine the distance between the nodes. It yields to accuracies between $2 - 15cm$.

- **SmartFloor** [41] - identifies people based on the characteristics of their footsteps. The system tries to recognize a user as accurate as possible. The hardware infrastructure brings nevertheless a tremendous cost.

- **Ubisense** [53] - uses ultrawideband technology to locate objects and people targeting an accuracy of $15cm$.

- **Landmarc** [37] - is a system that uses the *radio frequency identification* (RFID) technology to locate objects within buildings. The system comprises RFID tags and RFID readers.

- **Easy Living** [29] - is a system developed by Microsoft and uses Digiclops real time 3D cameras to provide stereovision-positioning capability in a home environment. The scalability of such an environment is limited (as in most of the previous cases) by the needed infrastructure.

There has been a large number of inventions published in the area of localization systems. For example, the work described in [15] presents a distributed positioning system, based on possible directions of the coordinate system but does not provide performance results and is inherently complex. Other systems try to avoid these limitations.

We can classify these systems in two categories. The first category includes the systems where the infrastructure detects the object and computes its position. As examples we can cite an electromagnetic system for detection of a passive tag within a room (described in [19]), a multiple pass location processing method [20] and a system for monitoring the position of at least one portion of an object based on magnetic fields detection [14].

The second category includes systems where the object itself determines its own position by inspecting the existing infrastructure. Examples include a system with uniquely identifiable beacons placed in the environment [18] and an infrared beacon positioning system [21].

Radio systems for locating mobile devices are described as well. The infrastructure determines the position of an object based on the time-difference of the received signals [16] or by interpreting the characteristics of the received signal [17].

5.1.2 Algorithms

In this section we will describe the most well known methods used in position determination. A strict classification is difficult due to the fact that the algorithms are developed having in mind different application scenarios, thus making use of different starting hypothesis. A survey of the most used techniques is given in [40].

When talking about localization algorithms, there are several common characteristics one can take into consideration before making a judgment. First of all, the localization algorithms can be *centralized*, *distributed* or *localized*. By centralized algorithms we understand algorithms where all the computations are performed at a single central point. This node has to collect all the data from the network, compute the positions of the other nodes and eventually distribute the position information back to them. The obvious disadvantages are the huge communication costs and the computation power required. Nevertheless, the precision of the position estimates computed by centralized localization algorithms is always superior to other algorithms. This is due to the fact that the central node has a global view of the whole picture and can apply refinement phases to improve the computed positions. Distributed localization algorithms make use of several computing and communication capabilities, with the usual advantages of

not relying on a single point of failure, not requiring a specialized central node and balancing the load by making use of a geographically distributed resource. A localized algorithm is not only distributed, but makes use only of local data, each node having to communicate just to nodes situated in a limited region.

Localization algorithms can be classified also in range-free algorithms (algorithms that do not make use of specialized hardware to perform distance measurements) and range-based algorithms (algorithms making use of measured distances between the nodes to compute positions). Intuitively, the distance-based algorithms should perform better that their distance-free counterparts. This fact is not always true, due to the fact that any measurement contains errors as well. If the precision of measurements is bellow a certain threshold, then not using measurements at all provides better results [39].

Another way to look at localization algorithms is whether they provide local, relative or absolute coordinate systems. An absolute coordinate system has global coherence and is desirable in most of the situations. These algorithms are in general expensive in terms of communication cost. Relative positioning establishes positions in a coordinate system specific to the network. The obtained locations still provide network wide coherence. Local positioning returns relative coordinates to a limited set of nodes (usually the neighbors) and helps communicating parties to position themselves relatively to each other.

Before presenting the most well known methods, let us present a basic localization mechanism, which can be found as the main underlying idea in many localization schemes. The method is called *lateration*. Let us assume the case of a two dimensional space (see Figure 5.1). Assume that that nodes 1, 2 and 3 know their coordinates (throughout this thesis we will refer the nodes 1, 2 and 3 as *anchors*, *beacons* or *seeds*). Assume now also that the exact distances between the node A and the other three nodes are known. This information is enough to determine the coordinates of the A node using simple geometry (as the intersection of the circles centered in 1, 2 and 3 having as radius the known distances).

Unfortunately, in reality, the exact coordinates and the distances are not known exactly. This prevents the three circles to intersect. Various approximation methods have been developed to estimate the intersection point. In the right side of Figure 5.1 we have represented the simplest approximation: the circles are replaced with rectangles. The estimated position is considered the weight center of their intersection. Usually more complex schemes make use of least squares approximation to find the point.

Now, let us take a look at some available methods:

- **Convex optimization** [10] - was one of the first localization schemes available. It treats the localization problem from the point of view of linear

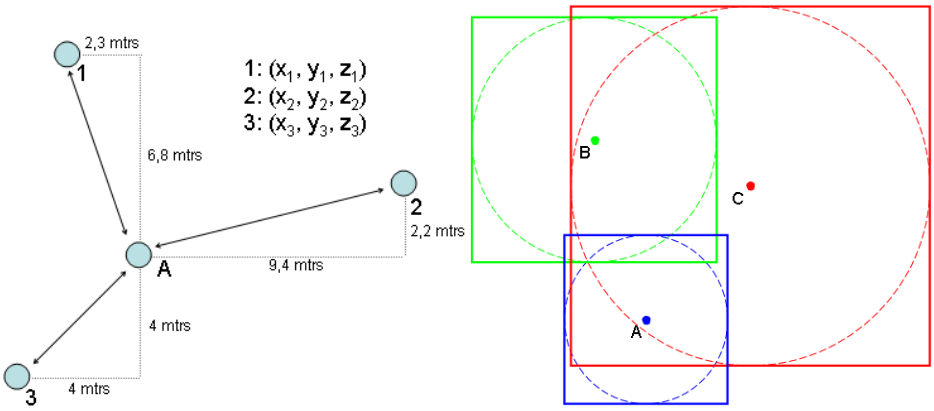

Figure 5.1: Lateration as localization technique

programming and semi-definite programming. The various constraints related to the relative positioning of nodes can be represented as linear matrix inequalities. The advantages of this centralized scheme are that the distance and angle information are simple to model and the solutions provided are optimal. Efficient computational methods have already been developed. The disadvantages are specific to the centralized methods. One important thing is that the bounds of the computation complexity are known (quadratic respective cubic in the number of connections).

- **Multidimensional scaling (MDS)** [52] - is a very robust (centralized in principle) positioning technique leading to results superior to most of all the other existing alternatives. Several versions of the algorithm have been developed: both range free and range based, centralized and localized. They can produce both relative and global coordinates. The method finds an embedding in lower dimensional space for a set of objects characterized by a pair-wise distances between them. Theoretical bounds for the computation complexity exist (cubic with respect to the number of nodes). The localized version of the algorithm [51] can deal with nonuniform topologies and has a reduced computation complexity.

- **Lighthouse system** [47] - the positions of a whole field of sensor nodes can be computed using only a single *lighthouse device* capable of *seeing* all the nodes. This device performs two functions at the same time: collects data from the sensor nodes and helps the nodes to position themselves. Unfortunately, the hardware requirements are quite restrictive: the system requires each node to be equipped with a photo detector and a clock. The

very precise locations provided by the algorithm are shadowed by the strong assumption of line of sight between the central device and each node.

- **Ad hoc localization system (Ahlos)** [49] - defines and combines several types of multilateration. Its main strong point is that it is a completely distributed protocol. Its weakest point is that the number of needed anchors should be very large for a good result. The algorithm is made up of one initial phase when nodes that can communicate to sufficient anchors compute their position using lateration and then behave like anchors (iterative algorithm). Some nodes are not able to compute their position this way, so they organize in collaboration groups. The algorithm may not always converge, so a number of nodes will never be able to compute their positions. Additionally, the algorithm is dependent on the accuracy of the distance measurements.

- **Ad hoc positioning systems (APS)** [40] - are a combination of two major ideas: distance vector (DV) routing (information is forwarded hop by hop from each anchor in the network) and global positioning system (eventually each node will compute its position based on the position of anchors and distance estimates). The localization schemes belonging to this group make use of connectivity, distance measurements, angle of arrival estimates, compass information - six possible combinations make sense and were studied. Although the algorithms span through all the localization algorithms categories, the main disadvantage remains that they target static networks, the DV component requiring a high cost in terms of communication in the case of mobile nodes.

- **Self positioning algorithm (SPA)** [9] - is a relative positioning algorithm. The coordinate system is determined by a location reference group (LRG). Nodes exchange information with neighbors to determine second order neighborhood information (connectivity and distance estimates). The next step is to construct local maps based on this information. LRG helps orienting all the maps by aligning all the coordinate systems.

- **Local positioning system** [38] - extends the ideas used by SPA. It develops a method for nodes to use some capabilities (ranging, angle of arrival, compasses) to establish local coordinate systems in which all immediate nodes are placed. It is then possible to register all these coordinate systems with the source of a packet (positioning is achieved only for the nodes participating in forwarding of the packet).

- **Sequential Monte Carlo approach** [26] - is an adaptation of the Monte Carlo positioning techniques used in robotics (where it was already used in

target tracking, robot localization and computer vision). It is a scheme targeted especially at scenarios including mobile nodes. A discrete model of time is used. During each step of time, two phases take place: a prediction phase (new position estimates are computed) and a filtering phase (the position estimates are filtered and the solution space might be re-sampled). The results given by this localization schemes are completely counter intuitive! It basically shows that mobility improves localization results at a reduced communication and computation cost. More amazing, the original scheme is range-free and the initial work shows results comparable to the distributed localization schemes for static scenarios.

Other approaches exist as well. For example, recently, localization was approached also from the point of view of graph theory. One remarkable result shows that from complexity point of view, graph embedding is a NP complete problem, so the optimal exact solutions cannot be computed in polynomial time [34]. Apart from this result, a localization scheme was proposed for the one dimensional case [7]. For the two dimensional deployment case, a series of properties have been identified but no notable result is available at the moment of writing this thesis.

5.2 A precision-based localization algorithm

In this section we introduce a new distributed algorithm for location discovery. It can be used in wireless ad-hoc sensor networks that are equipped with means of measuring the distances between the nodes (like the intensity of the received signal strength). The algorithm takes the reliability of measurements into account during calculation of the positions of the nodes. The simulation results of our approach yield 2 to 4 times better results in position accuracy than other systems previously described. This level of performance can be reached using only few broadcast messages with small and constant size, for each node in the network.

5.2.1 Assumptions

The specific platform that our proposed system is intended for is that of a sensor network that can be deployed at random, consisting of a (possibly very large) number of similar sensor nodes, and only a very small amount of base stations. The sensor nodes will be small, cheap and battery operated, with short range radio frequency (RF) communication hardware, simple (slow) microprocessors, and additional sensing hardware.

Due to the nature of these nodes, some important design goals for the algorithms used to provide the localization data have been identified, much like the algorithms described and compared in Langendoen et al. [30]. Such algorithms should be truly distributed, as well as self-organizing, robust and energy-efficient. This means that the calculations should be made on each individual node, using as little as possible computation and especially communication, without relying on any fixed infrastructure or network topology, and being able to cope with changing conditions, such as node failures.

A small subset of the nodes in the network, called anchor nodes, will initially know its own location, expressed by its coordinates relative to some network-wide coordinate system, either by manual configuration, or by using other location sensing techniques requiring extra hardware, like for example GPS. All other nodes will initially not know their location. Anchor nodes are assumed to have the same hardware capabilities, so factors like communication capabilities and energy consumption considerations will be equal to non-anchor nodes. Ideally, these nodes should be spatially distributed equal across the network, even though in certain application areas this might not be the case, or cannot be relied upon. Certain flexibility towards this property has to be assured by the localization algorithm. To minimize on installation and maintenance effort the fraction of anchor nodes in the network should be really small, and the location algorithm should be able to deal with this small amount of anchor nodes.

The results presented in this section only focus on situations with fixed sensor locations, since this already proves to be enough of a challenge. Other environmental factors, like the positions of objects in between the nodes, might be changing, however, resulting in varying readings of received signal strength indication (RSSI) measurements between pairs of nodes at different times. Systems where the nodes are mobile can use the described algorithms to update the nodes positions continuously if the movement rate is within certain boundaries.

5.2.2 Precision of measurements

As the basis of an algorithm to determine the location of a node, some measurements have to be available. With the typical sensor network hardware and network structure, the RSSI can be used to obtain an indication about the distance between a pair of neighbor nodes in the network. The specific calculations to translate the RSSI readings into the distance towards the sending node will not be addressed in this thesis. It is assumed that such a calculation can be made and is available to the algorithm presented herein.

In general, if the distance to at least 3 neighbor nodes is known, as well as the locations of those nodes (for example because they are anchor nodes), the

position in 2-dimensional space can be computed using triangulation calculations. Unfortunately, especially in indoors environments, the distances obtained from the measured RSSI will be quite imprecise, because of the fading and multi-path effects of the radio signals meeting with the various surfaces in the surroundings of the node. According to [6], this error can be as large as 50% of the measured distance. The error of the measurement does however conform to a Gaussian distributed random variable. As a descriptive value of the error distribution the standard deviation can be used. For maximum errors of 50% this means standard deviation of about 20% of the distance. This value will be referred to further on as η. Different environments with other error characteristics will result in different (possibly smaller) values of η.

By itself, with distance errors of this size, the computation through triangulation will contain a large error as well, especially because of the accumulation of the errors in subsequent calculations. This might render the result of the calculation practically useless. However, by using the connectivity of the network, which is usually more than 3 neighbors per node, the redundancy in the network can be exploited to improve on the results of estimating a nodes location. Other factors, such as the known properties of the error distributions, as well as obtaining multiple measurements between pairs of nodes instead of just one, can be taken into account as well to try to obtain a reasonable precision of the computed location.

5.2.3 Iterative multilateration

For nodes with more than the minimally required 3 neighbors with known position, the ”iterative multilateration” method, described by Savvides et al. [49] can be used to calculate the position with smallest error. Because of the known error distribution of the distance measurements, the error of the obtained location can be calculated as well. This can be modeled as a Gaussian distributed random variable, denoting the probability of the real location of the node to be within a certain range of the computed value, expressed with the standard deviation of the error distribution. In other words, the standard deviation serves as a measure of the precision of the location estimation.

This newly calculated position, combined with its precision, can now be used in subsequent calculations for other still undetermined nodes. The nodes that did not compute a position yet, will have to take into account also the uncertainty of the newly located nodes in their computations. In fact, they will start by combining the uncertainty of the distance measurements and that of the position estimates. When still undetermined node obtained at least three range measurements to already determined neighbor nodes, it can itself calculate its position. This iterative multilateration can be used again to refine a node’s position, when more

neighbor nodes have their position calculated, or after the neighbor nodes have obtained refined position estimates.

This approach of refining a node's position based on the measured ranges to neighbor nodes is used as well in the refinement stage of the system described in [50], but because a precision value of a node's initial position is known as well in the scenario described above, initial position estimates of non-anchor nodes can be used to calculate a position estimate of undetermined nodes as well. When a possibly very imprecise (with large standard deviation value) location estimate is used in subsequent estimation calculations, the large error is accumulated in the results of the new calculation. Though, this result is provided together with an even larger standard deviation of the position error meaning that the lower accuracy is provided with the result of calculation.

The problem that still arises is that at the start of the location discovery algorithm, only the anchor nodes will have a known location (with infinitely high precision). If there is a small fraction of anchor nodes, then there will be many non-anchor nodes with fewer than three anchor nodes as their direct neighbors, making it impossible for them to calculate their positions. It is possible though, to obtain a distance to the anchor nodes from all non-anchor nodes, by using the distances measured at the intermediate hops on the shortest path to each anchor. The multi-hop distance can be obtained by multiplying the sum of all single hop measurements by a precomputed bias factor (the standard deviation associated with it increases linearly with the number of hops).

5.2.4 Iterative weighted least squares estimation

The calculations of a node's position from a set of range measurements between the node with unknown position and a set of nodes with known or estimated positions can be performed in a way similar to that of GPS [33] and the "iterative multilateration" described in [49]. The main difference with those systems and the one proposed here is that the precision estimates of all data values, expressed as a standard deviation of the error distribution, are also taken into account in the whole calculation. This section deals with the description of the *Iterative Weighted Least Squares Estimation* (IWLS) used in the proposed system.

Starting from an initial estimation, an improvement vector is calculated iteratively and added to the previous estimation until the improvement vector is smaller than a certain value. This vector is obtained through a weighted least squares estimation:

$$wA\mathbf{x} = w\mathbf{b} \tag{5.1}$$

where w is a weight factor, A is a matrix, and $\mathbf{x}$ and $\mathbf{b}$ are vectors.

To calculate the improvement vector we take the true position of the node to be calculated as $\mathbf{x} = (x, y)$ for a 2-dimensional system, or $\mathbf{x} = (x, y, z)$ in the 3-dimensional case. The initial estimation of the position is denoted as $\mathbf{x}^{est}$ and the positions of the n neighbor nodes as $\mathbf{x}_i$, for $i = 1..n$. The measured ranges to those nodes can be denoted as: $r_i = \|\mathbf{x}_i - \mathbf{x}\| + \epsilon_i$, $i = 1..n$, with ϵ_i denoting the measurement error, and the node's true position as: $\mathbf{x} = \mathbf{x}^{est} + \delta\mathbf{x}$.

In the same way, the distances to the estimated position are: $r^{est} = \|\mathbf{x}_i - \mathbf{x}^{est}\|$, $i = 1..n$ and

$$\delta r_i = r_i - r_i^{est} = \|\mathbf{x}_i - \mathbf{x}^{est} - \delta\mathbf{x}\| - \|\mathbf{x}_i - \mathbf{x}^{est}\| + \epsilon_i$$

$$\delta r_i \approx -\frac{(\mathbf{x}_i - \mathbf{x}^{est})}{\|\mathbf{x}_i - \mathbf{x}^{est}\|} \cdot \delta\mathbf{x} + \epsilon_i = \mathbf{1}_i \cdot \delta\mathbf{x} + \epsilon_i,\ i = 1..n \tag{5.2}$$

with $\mathbf{1}_i$ the direction vector of length 1 from the nodes estimated position to node i.

For each range measurement r_i a weight $w_i = 1/\sqrt{\sigma_i^{edge2} + \sigma_i^{node2}}$ is calculated, where σ_i^{edge2} is the variance of the range measurement to node i, and σ_i^{node2} is the variance of the position of the node i.

Matrix A consists of n rows, each filled with the direction vector $\mathbf{1}_i$, one for each anchor node involved in the calculation. Vector $\mathbf{b}$ is constructed as a column vector filled with the values δr_i, one for each anchor node. Note that the i'th row of the matrix A needs to correspond with the i'th row of vector $\mathbf{b}$ with respect to the anchor node the values are calculated off. The least squares solution of equation (5.1) will then look like this:

$$\begin{bmatrix} \mathbf{1}_1 \cdot w_1 \\ \vdots \\ \mathbf{1}_n \cdot w_n \end{bmatrix} \cdot [\delta\mathbf{x}] = \begin{bmatrix} \delta r_1 \cdot w_1 \\ \vdots \\ \delta r_n \cdot w_n \end{bmatrix} \tag{5.3}$$

This way the improvement $\delta\mathbf{x}$ is calculated. In general, the least squares solution of $\mathbf{x}$ for $A\mathbf{x} = \mathbf{x}$ is: $\mathbf{x} = A \cdot C \cdot \mathbf{b}^T$ where $C = (A^T \cdot A)^{-1}$.

Using covariance matrix C, a square matrix with the number of rows and columns equal to the dimensionality of the system, the node's standard deviation is calculated as:

$$\sigma^{node} = \sqrt{\frac{1}{2}\sum_i \sum_j C_{i,j}} \tag{5.4}$$

When the estimated position of the nodes is calculated like this, the optimal location will be calculated, based on the given ranges to and positions of the neighbor nodes.

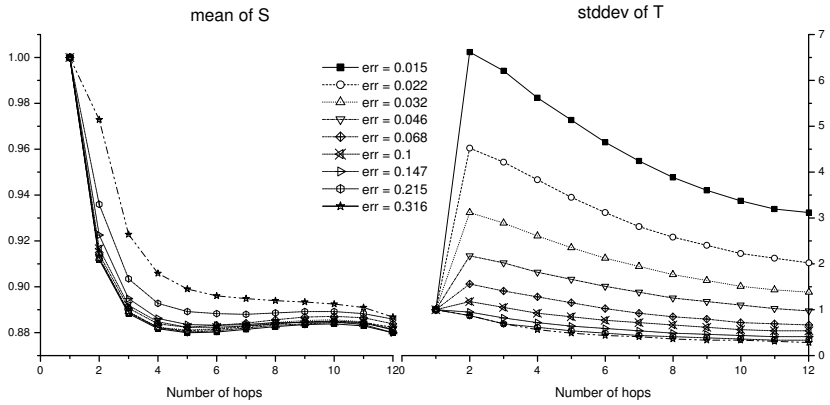

Figure 5.2: Bias table values for μ_S and σ_T

5.2.5 Multi-hop distances

Calculation of the distance between two nodes $\mathbf{A}$ and $\mathbf{B}$ that are connected only by a path of more than one hop can be performed indirectly. On each hop along the shortest path between the nodes, the range and corresponding standard deviation are measured. An estimation of the distance and precision between $\mathbf{A}$ and $\mathbf{B}$ can be made, however, by taking the sum of all these ranges. The distance calculated this way, called r_m will usually be larger than the true distance between the two nodes, and the standard deviation will be larger than the summed single-hop standard deviations σ_m, because of the error introduced by the less precise calculated distances. A better estimation of those values can be made, however, by statistically analyzing large sets of these summed measurements.

Every measured summed distance r_m can be seen as a sample of a random variable R_{r_m}. If this variable is of Gaussian distribution, it serves as a good bases to obtain estimation of the actual distance. The expected value of R_{r_m} is the mean of the distribution, $E(R_{r_m}) = \mu_R$. Normalizing R_{r_m} by the measured distance r_m results in a normalized random variable $Q = R_m/r_m$ for all values of r_m measured along paths with equal number of hops, from which the estimated value of the true distance r_t can be calculated as: $r^{est} = \mu_Q \cdot r_m$. The error between the measured distance and the estimated true distance $\epsilon_{r_m} = r_m - r^{est}$ can, in the same way, be seen as a sample of the random variable S_{r_m}. The standard deviation σ_S of S_{r_m} is the precision of the estimated distance r^{est}.

A normalized random variable T is defined as having samples ϵ_T, obtained by

dividing all samples ϵ_{r_m} by the expected error η on the true distance r_t:

$$\epsilon_T = \frac{(r_m - r^{est})}{r_t \cdot \eta} \tag{5.5}$$

The expected error η is an environment and hardware dependent value of the range measurement error (given as a percentage of the range) as mentioned earlier in Section 5.2.2. From the standard deviation σ_T of T the precision of the estimated true range r^{est} can now be calculated from σ_m and σ_T as: $\sigma^{est} = \sigma_m \cdot \sigma_T \cdot \mu_Q$.

The values for σ_T and μ_Q can be calculated offline, and stored in a table, containing one set of (σ_T, μ_Q) values for all possible hop lengths of the shortest hop paths between any pair of nodes. These values for single hop paths (for nodes that are each others direct neighbors) will of course be $(1, 1)$. The value of σ_T does depend on the η, so for different values of η another table will have to be used. This value does however only depend on the hardware and environment used for the network, which will be known before deployment of the network, so only one table will have to be stored or in active use throughout the whole localization calculation. Figure 5.2 shows the values of σ_T and μ_Q for values of η and different hop counts.

5.2.6 Algorithm details

The protocol uses a two-phase approach, and relies on two corresponding kinds of messages being passed between the network nodes; start-up messages and refinement messages. During the start-up phase, a node attempts to calculate an initial position estimate, based on the distances towards the anchors. This initial position will then be improved to get a more accurate estimate during the refinement phase.

Start-up messages contain information about the distance and hop count, as well as the position of another node (with known, or at least estimated position) to the sending node. *Refinement messages* contain the position of the node that sent it. When the localization algorithm starts, usually when the node is switched on, or awakes from a sleep state, a non-anchor node starts by broadcasting an *announcement message*, indicating its presence and requesting for information from nearby nodes. An anchor node starts directly on activation by broadcasting its own position. For its surrounding nodes, this is an indication of its presence as well. Upon receiving an announcement message, a node broadcasts all information currently known to that node. This way, the new node quickly learns the positions of all nearby nodes, and helps the newly connected node to catch up on the current state of the network. For all nodes in start-up state, all received messages with hop count lower than the maximum are stored, until enough distances (at least 3 in the

2-dimensional simulation) are known to be able to calculate its own position estimate. If an estimate can be calculated, the node will proceed to refinement state, and broadcast the position estimate just calculated.

Nodes in refinement state only collect refinement messages from direct neighbors. After receiving a position update from one of its neighbors, the node can re-calculate its own position, based on the new or more precise position just received, together with all other neighbor's positions. If this new position estimate is more precise than the previously known location (smaller σ), this new position estimate is broadcast in a refinement message. This strategy ensures finiteness of the algorithm. After each round of message passing, the improvement in calculated precision will be smaller. If a node cannot calculate a more precise position update, it will not rebroadcast its position. All other nodes will recalculate their positions based on fewer position updates from their neighbors, and the new estimate will be more likely not to be more precise, in which case the node will stop re-broadcasting as well. After a few rounds no node will be able to improve its position estimate, and the algorithm stops.

As the protocol progresses, nodes update their position estimates based on the improved position estimates of their neighbor nodes, and possibly a neighboring anchor node. These improved position estimates of the neighbor nodes in their turn, were calculated from the previous estimate of their neighbors. So, in effect, a node's new position is calculated indirectly from its own previous improved estimate. It is important that all nodes recalculate their positions on a similar rate, so ensure each node keeps a reliable calculated precision of its position estimate, relative to its neighbors. This is ensured by allowing a node to re-broadcast only after a certain, network wide, fixed time after the previous position estimate was broadcast. This also keeps the number of broadcasts low.

Anchor nodes already know their position from the start. Because their position is obtained in a way external to this algorithm, they do not recalculate it. As a result, for the majority of the time, they can be silent, and only react to announcement messages by rebroadcasting their fixed position. The precision of anchor nodes positions is obtained externally as well, and can be much greater than possible to achieve with the calculations of this localization algorithm. In the simulations, the anchor nodes precision is set to be infinitely high.

5.2.7 Simulation setup

To test the performance of described system, a simulation environment was built using the OMNeT++ discrete event simulator [55] (see Section 4.2). The tests performed in the simulation environment are described is detail in Section 5.2.8. The network communication model used in the simulation makes use

only of local broadcasts, where messages are delivered to all nodes within a certain and fixed range from the sending node. Colliding transmissions and message corruption are abstracted from (it is assumed that lower network layers will be in charge of providing a reliable broadcast service). The positions of all nodes are fixed during a simulation run, and are determined at initialization. The simulation environment uses a 2-dimensional coordinate system. Each node's coordinates are selected randomly and uniformly distributed within a square region of a size in units equal to the amount of nodes in the network.

During network initialization a number of nodes is selected as anchor nodes, in a way similar to the one used by Langendoen et al. [30]. The anchors are selected in a grid-like structure, where the nodes closest to the points of the grid will be chosen to be anchors. The grid is of size s, with $s \times s \leq N$, with N the number of anchor nodes, and s the largest number that still satisfies the inequality. The rest of the anchors are selected randomly from the remaining nodes. This way of selecting the anchor nodes is chosen because it provides a more or less equal environment for all nodes, where the closest anchors will always be a certain maximum distance away. The reasons to choose this strategy is mainly because it offers a uniform distribution of anchors. Additionally, it allows easy comparison to the results described in [30].

The distance measurements between nodes are calculated from the RSSI information. This means that every received message also provides a range indication between itself and the sending node. In reality the calculation of this range infers a certain amount of error, as mentioned earlier. In the simulation the range measurement including error is modeled by drawing a random value from a Gaussian distribution with the true range as its mean, and a standard deviation of $\eta \cdot true\,range$. The implementation of the algorithm records all range measurements from each message it receives, and uses the average distance measured to a neighbor node in all further calculations. This way, as more messages are received, the measured distance will converge to an ever more constant value, which leads to better position estimates during refinement.

5.2.8 Simulation results

We have setup the following parameters for the simulator (this setup is commonly used in referenced papers):

- Number of nodes: 225, placed on a square area of 15 units length;
- Radio range: 2.1 units, results in connectivity of about 12;
- Relative range error η: 10%;
- Number of anchors: 5% = 11 anchors;
- Multihop distance hop count: 4 hops.

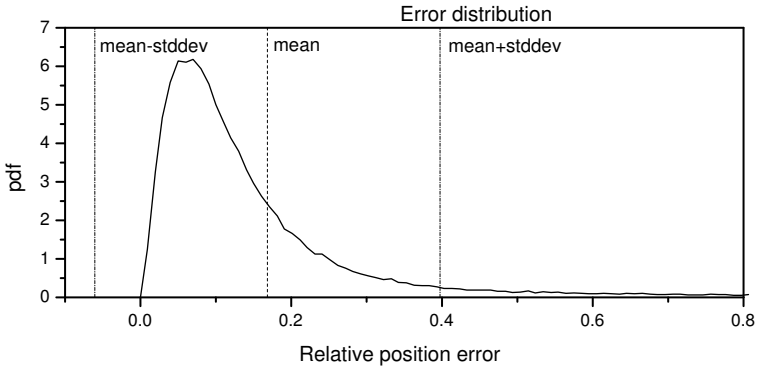

Figure 5.3: Position error distribution for standard scenario

Because of random effects, produced by the use of randomly selected error values, node positions and possibly other factors, consistent results on all of the performed tests can only be obtained by averaging over large quantities of individual test runs. To ensure consistency, all tests performed have been repeated at least 50 times.

As a measure of the performance of the described algorithm, as well as an indication of the usefulness in specific application areas, *the accuracy* (μ) of the algorithm, as well as its *precision* (σ) are important indicators. The accuracy can be described as the average distance between a node's actual position and its position estimate. The precision describes to what extent the real position error is near this average distance error. This is calculated using the standard deviation of the error values. In order to test those values, an initial test has been performed using the parameter values of the standard scenario. Figure 5.3 shows the error distribution of the relative position error. Note that, even though the majority of the error values stay below 10%, the average error value is around 17%. This is due to the fact that a small number of nodes have relatively large errors, of up to and beyond the radio range. This fact is illustrated as well by the large standard deviation of about 0.23.

Even though, as the results show, each node's position cannot always be estimated very accurately, it does perform well on assigning a position estimate to all nodes. On average, over the test runs executed in this first test, a coverage factor of 98.9 % has been measured, meaning almost all of the nodes have obtained a position estimate, with the exception of the few nodes that do not connect to enough neighbors to actually start an estimation.

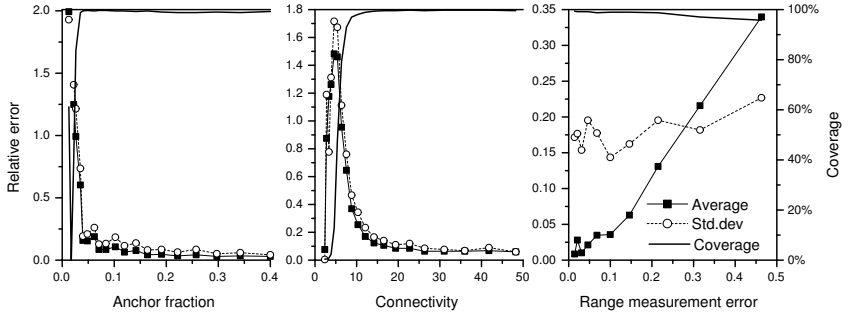

Figure 5.4: Relative position error function of anchor fraction, connectivity and range measurement error: coverage (unmarked, right scale), average (square mark) and standard deviation (circle mark)

In the next series of simulations, the algorithm's sensitivity to variations in fraction of anchors, average connectivity and range measurement error is tested. Figure 5.4 shows the results of these tests. In all three diagrams, both the average distance error and the standard deviation are shown, as well as the coverage factor.

The simulation results show that the system is hardly sensitive towards number of anchors, except for really small numbers. At an anchor fraction value of 1.8%, which is 4 anchors, no nodes are able to obtain a position estimate during the start-up phase. This is because the four nodes are placed at the corners of the area, and there is no node that has 3 anchors near enough to receive multi-hop distance information to them. This can be solved of course by increasing *the time to live* field of the messages, so the distance information from the anchors travels for a greater number of hops. From a value of 4% onwards the system shows little change in both relative error (average and standard deviation) and coverage.

As for the connectivity (controlled by the variation of the transmission range), from a connectivity of about 8 to 10, the result is nearly the same, showing little change in relative range error. In the results for both the anchor fraction and radio range variations, the average and standard deviation of the distance error have proportional values. This indicates that the distribution of the range error is similar to the one shown in Figure 5.3, scaled along the x axis.

With respect to the range error sensitivity, this is somewhat different. While the standard deviation keeps a more or less constant value around 0.27, the average error does change linearly with η. The size of the range error is directly reflected by the accuracy of the system, whereas the precision remains largely unchanged.

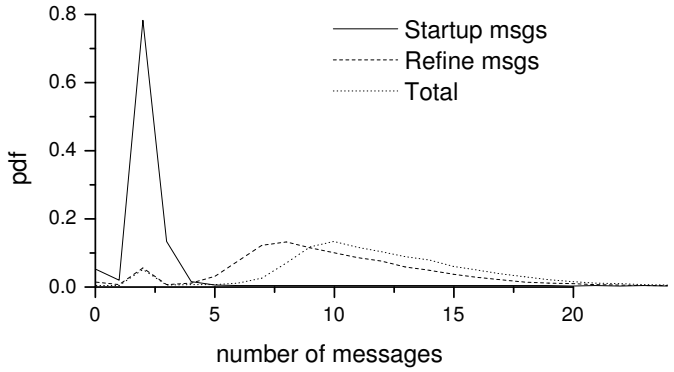

Figure 5.5: Distribution of number of Startup and Refine messages for standard scenario

In a real-world implementation of an ad-hoc sensor network, energy preservation is of great importance. Because network communication is largely responsible for the energy consumption, this metric is of major interest as well. Some additional tests have been performed to quantify the network communication cost of this system.

Both the Startup and Refine messages have a small, fixed size. The network communication cost can thus be quantified by the number of messages broadcast per node. Figure 5.5 shows the probability distribution of both message types, and the total number of messages for the standard scenario. It is clear that the Refine messages take up the majority of the total number. Decreasing the number of Refine messages directly improves the total communication costs.

The algorithm can be adapted so that every node will only send a certain amount of Refine messages. The implications of this in relation to the distance error are shown in Figure 5.6. It clearly shows that after the first 3 or 4 Refine messages more communication hardly makes any improvement in accuracy (mean value of the positioning error) and precision (standard variation of the positioning error), while the average number of messages increases linearly. Limiting the number of Refine messages is an easy way to improve the total performance of the system.

Considering the small size and low cost of the targeted devices, only very limited processing resources will be available. The processing time to complete the localization process is an important aspect of this particular system.

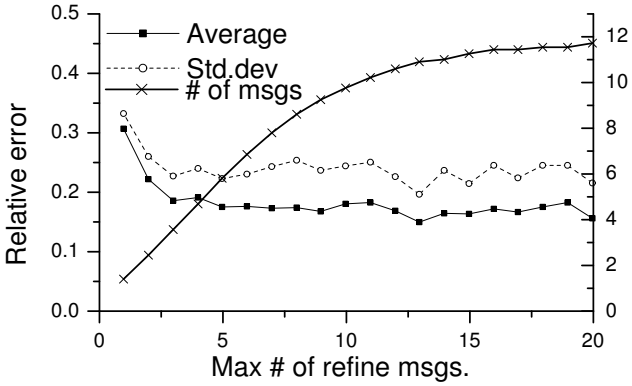

Figure 5.6: Relative distance error and average number of Refine messages per node as a result of limiting the number of Refine messages

As the central part of the system, the Iterative Weight Least Squares Estimation takes up the majority of processor cycles in the total calculation. This part of the calculation is therefore a good indicator of the total processing time used.

A number of factors are responsible for the total time each node spends calculating its (updated) position. At first, the size of the matrices used, and thus the spent calculation time, is proportional to the amount of neighbors involved in the calculation. Besides that, the number of iterations taken for the calculation to complete is a proportional factor to the total calculation time. At last, a third factor is the number of times the IWLS calculation is performed. In general, at least every time before a Refine message is sent, a node's position is re-calculated several times, until a more precise result is found.

The factors involved in calculation time, as mentioned above, have been examined, using the data sets generated for the earlier tests. Figure 5.7 shows the number of times a calculation is performed per node, average number of IWLS iterations per calculation, average number of neighbors involved, and number of calculations per broadcast Refine message. The solid, unmarked line, on the right scale, shows the grand total of the number of calculations, number of iterations and number of neighbors multiplied. This is a number proportional to the total calculation time spent per node in the IWLS calculation for the whole localization algorithm.

For the targeted devices, it is very important to only spend energy and processing time as long as it gives improvement to the positioning accuracy. Selecting the right set of parameters optimal to the deployment area to ensure this is should be

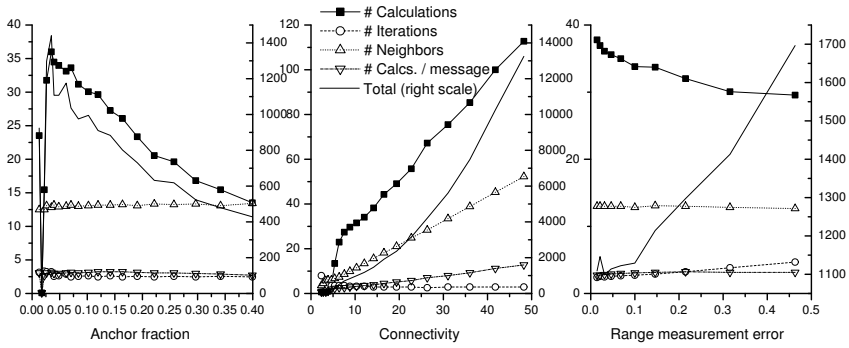

Figure 5.7: Nr. of calculations for different parameter values. The number of calculations, iterations, neighbors and calculations per message are drawn on the left scale; the total calculation time per node is shown by the unmarked solid line, drawn on the right scale.

considered carefully. Reducing the processing cost can be achieved by reducing either of the above-mentioned factors.

Comparison with Figure 5.4 shows that limiting the amount of neighbors in the IWLS calculation greatly reduces the processing cost, while this hardly has an influence on the range error performance of the system. From the graphs it is clear that the number of calculations per broadcast message is an almost constant factor.

5.2.9 Discussions

The previous sections have covered the description of the algorithm used in obtaining the position estimate in a distributed network.

The implementation of the system presented in this section is only a simple, straightforward one. As has been shown in the previous section, performance improvements can be made, by making small changes, based on close inspection of the results obtained by selectively changing certain parameters that are of importance in the whole calculation of the algorithm. As one of such improvements, multi-hop distances towards indirect neighbors could also be used in the refinements phase, instead of just to obtain initial position estimates. This could be especially useful for networks with a low connectivity. It does promise to pay off in terms of performance increase, to spend some time fine-tuning the algorithm,

	Condition	**Pos.err.**	**Coverage**
Traditional alg. + refine	$\eta < 0.05$	20-25%	50-60%
Traditional alg. phase 1+2	$conn. > 12$	35-42%	100%
Precision based algorithm	$\eta < 0.25$	15-25%	95-100%
Precision based algorithm	$\eta < 0.1, conn > 12$	10-15%	98-100%

Table 5.1: Comparison of the presented algorithm with the ones described in [30], on relative position and connectivity

based on a particular application area.

As noted before, similar algorithms have been designed, and can serve as good comparison material. Langendoen ([30]) provides a comparative test for combinations of different algorithms. Even though the general approach is somewhat different, the refinement steps are almost identical to the one used in our system. The data after refinement are comparable to the results presented in this paper.

The multi-hop distances and IWLS calculation are quite similar to the *sumdist* and *lateration* methods described in [30] and others, with the exception that use is made in our system of the precision indications available with the messages.

From the results presented in [30], it is clear that the refinement algorithm only is not a very useful addition to the first two phases (first phase is a distance estimation and phase two is position computation) except in very specific conditions. The use of refinement dramatically reduces the coverage of the whole system to levels below 50% in all cases except when the distance measurement errors are very small ($\eta < 0.05$). Under these conditions, the position error can drop to 25% or even 20%, although coverage of just 60% can be reached. Without the refinement phase, the various combinations of stage 1 and 2 algorithms can only reach a position error of 42%, and less for higher connectivity, but with 100% coverage. Having full coverage is very important, of course, since acquiring a position estimate is the intended goal of these algorithms.

Our results, on the other hand, show better position estimates, while practically full coverage can be reached, in almost all situations. In all but the most extreme environments, a position error of 25% or less can be achieved, while keeping 95% coverage. Large distant measurement errors or low connectivity do make the numbers less feasible, showing the application limits for this algorithm. But in more optimal conditions, position errors as low as 10% can be reached, with near 100% coverage. Table 5.1 summarizes the results for the algorithms mentioned above.

The improvement in position error with our proposed solution is most likely caused by the availability of more measurement data, namely the precision indication, even though not a lot can be said about this in general, since the obtained

location precision is a result of many interrelated factors.

In terms of communication overhead, our solution proves to be competitive as well. In [30] it is stated, that the initial flooding (including calibration) of the network costs about 3.5 to 4.8 messages per node, depending on the algorithm used. This is comparable to the average number of startup messages per node, which is 2.1 in our solution. The amount of messages needed during refinement is largely dependent on the limit imposed on the system, and in both algorithms, the resulting position error is constant from about 5 messages onwards. All systems use messages of a small, constant size.

It is hard to compare the calculation requirements of all algorithms, especially because only simulations are available up to this point in time of both the system described in this paper and the ones in [30]. Besides that, Langendoen et al. do not give any metrics about calculation time or complexity in their paper. At this point, no conclusions can be made about how the different algorithms compare in terms of calculation time.

It is shown that, using RSSI as a distance measurement system, position errors of as low as 10% of the transmission range can be reached. Even though this could be not sufficiently accurate for use in all kinds of situations, it does provide good enough results to at least be able to know about the networks topology, and coverage area of the individual nodes. Altogether, it can be concluded that the goal of obtaining position information of nodes in a distributed ad-hoc network, making use of only rather inaccurate measurement techniques, is within reach.

5.2.10 Conclusions

In this section we presented a distributed localization algorithm for ad hoc wireless networks, which takes into account the precision of measurements. It is aimed at networks where the nodes are static. The algorithm takes a two-step approach: first an initial location estimation is made, based on distance measurements obtained from RSSI readings. Subsequently, refinements are calculated. Both steps, including the calculations involved, are described in detail, after which the performance is discussed. In a 225-node network, of which 5% are anchors, with 10% range error, a relative distance error of about 16% can be achieved, with a nearly 100% coverage. In general, the results of this approach yield 2 to 4 times better results in position accuracy than other systems described previously. This level of performance can be reached with just 10 or fewer messages broadcast per node in the network, which are of small, constant size. Details about the calculation cost are discussed as well, and some suggestions are given on how to optimize the performance of the algorithm for real world implementations.

5.3 One-dimensional random distribution statistics

A lot of effort has been put into characterizing the ad-hoc networks from a statistical point of view. The authors of [28] derive an approximate formula for the expected progress in each hop for a two dimensional case network deployed within a circular region:

$$\overline{z} = \left(\frac{N}{\lambda\pi}\right)^{\frac{1}{2}} \left[1 + e^{-N} - \int_{-1}^{1} e^{-\frac{N}{\pi}} \left[cos^{-1}(t) - t\sqrt{1-t^2}\right] dt\right] \tag{5.6}$$

For the same scenario, an average distance between any two nodes deployed in a circular area is computed ($d = \frac{128}{45\pi}R$) and the average hop count is determined by dividing these two values ($\overline{h} = \frac{d}{\overline{z}}$).

The authors of [4] derive closed form formulas for the probability that two nodes can communicate within one or two hops. The nodes are deployed in a rectangular area. The formulas are based exclusively on the distribution of the distance between two random nodes in such a scenario, derived in [23]. For hop-counts larger than 2, simulation results are presented and analyzed. Lower and upper bounds for the hopcount distribution between two random nodes are derived based on the properties of the network for the density of the nodes (λ) growing to infinity.

In [32] the author investigates the link distance between two randomly positioned nodes, where the network scenario is represented by nodes distributed inside a rectangular surface. Two cases are analyzed: inthe first one the nodes are uniformly distributed and in the second one the position of nodes follows a Gaussian distribution. It is shown that the shapes of the link distributions for these scenarios are very similar when the width of the rectangular area in the first scenario is taken to be about three times the standard deviation of the location distribution in the second scenario. The author of [35] continue the analysis, exploring the underlying spatial distributions in more depth. He also gives two improved versions of the original estimator. Mobility of the nodes is also taken into account, as an extension of the work done in [5].

Average hop distance is investigated for other network topologies as well. For example, in [48] the average hop distance is investigated for ring and street network topologies.

In the following, we take a look at the one dimensional random deployment scenario and try to find the expression for the hop count underlying statistics. These formulas will give a way to estimate distances between nodes only by knowing the average density and the hop count, no distance measuring hardware being necessary. The exact extension to the two dimensional case or to higher

dimensions is very computation intensive (impossible maybe?) as no closed-form formulas can be determined. The approximations cited above try to approach the real case as close as possible.

We model the one–dimensional network scenario through a set of nodes placed on the positive side of the real x axis, where positions are picked according to a one–dimensional Poisson process with known parameter λ [54]. Among these nodes, we consider a special device referred here to as source or *sink node* (SN), that is the originator of the hop count distribution procedure and that is placed at the origin of the real line. The probability of having exactly k nodes within a given interval of length l is a random variable K with the property: Prob$(K = k) = e^{-\lambda l}((\lambda l)^k/k!)$. The distance between the origin of the axis and the first node to its right is a random variable X characterized by an exponential distribution:

$$F_X(x) = 1 - e^{-\lambda x}, \quad f_X(x) = \lambda e^{-\lambda x}. \tag{5.7}$$

The distance between any two consecutive nodes has the same distribution X [43]. Finally, we assume that each node can communicate to all its neighbors whose distances are smaller than R, where R is the node transmission range.

5.3.1 Limits of the hop count intervals

We say that a node has a hop count (HC) equal to n, $n \in \mathbb{N}$, if it is situated n hops away from the source node. Moreover, we say that a given node with HC $n \geq 0$ and in position $\overline{x} \geq 0$ is the *last node* in its hop count if all nodes in $x > \overline{x}$ have a larger hop count value. Let $\Omega = (\tau_0, \tau_1, \tau_2, ..., \tau_n, ...)$ denote the positions of the nodes, $\tau_i \in \mathbb{R}^+$. Moreover, let $\Omega' = (\xi_0, \xi_1, \xi_2, ..., \xi_m, ...)$ be the set containing the positions of the last nodes in each hop count, where ξ_i is the position of the last node in HC i. Observe that, $\Omega' \subset \Omega$ and that SN, placed in $\tau_0 = \xi_0 = 0$, is considered to be the last node in HC 0. Moreover, it can be easily verified that a node belongs to HC n iff is within transmission range with the last node in HC $n-1$ and is not in transmission range with the last node in HC $n-2$. Exceptions to this rule are HC 0 (that contains the source node only) and HC 1 (that contains all the nodes in transmission range with the source). Based on these observations, we can write the following inequalities:

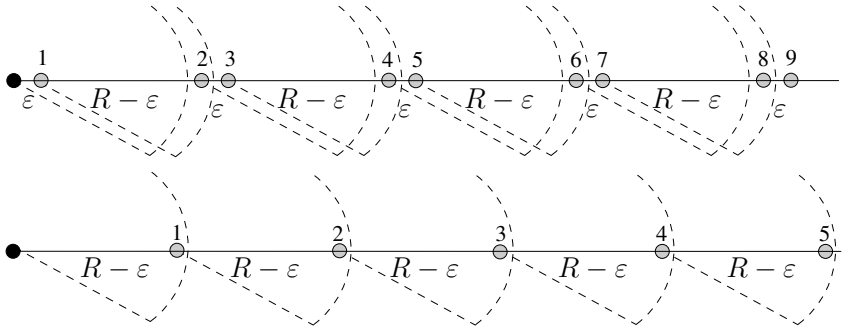

Figure 5.8: Inferior and superior bounds for the hop intervals ($\epsilon \in \mathbb{R}^+,\ \epsilon \to 0$).

$$\begin{aligned} \xi_0 &= 0 \\ \xi_0 < \xi_1 &< \xi_0 + R \\ \xi_0 + R < \xi_2 &< \xi_1 + R \\ &\cdots \\ \xi_{n-2} + R < \xi_n &< \xi_{n-1} + R \\ &\cdots \end{aligned} \tag{5.8}$$

Now, we are interested in finding the boundaries of the interval containing all the feasible positions for those nodes in HC n. The inferior ($\mathcal{D}_{\text{min}}$) and superior ($\mathcal{D}_{\text{max}}$) limits of the interval containing all the nodes with HC n are found by solving (5.8):

$$\mathcal{D}_{\text{min}}[n] = \left\lfloor \frac{n}{2} \right\rfloor R, \quad \mathcal{D}_{\text{max}}[n] = nR \tag{5.9}$$

This result can be also justified by simple geometrical observations (Fig. 5.8). The fact that the distance between the two consecutive points ξ_i and ξ_{i+2} needs to be larger than or equal to $R + \varepsilon$, where $\varepsilon \to 0^+$, leads to the inferior limit. The superior limit comes from the fact that ξ_i and ξ_{i+1}, in the limiting case, can be spaced by at most $R - \varepsilon$, where $\varepsilon \to 0^+$.

To proceed with our analysis, we introduce the following notation. Let $\mathbf{\Gamma}_{l_1}^{l_2}(x)$, $l_1, l_2 \in \mathbb{N}$, $l_1 < l_2$, be defined as:

$$\mathbf{\Gamma}_{l_1}^{l_2}(x) = \begin{cases} 1 \text{ for } l_1 < x < l_2 \\ 0 \text{ otherwise} \end{cases}$$

In other words, $\overset{\mathcal{D}_{\max}[n]}{\underset{\mathcal{D}_{\min}[n]}{\mathbf{\Gamma}}}(\xi_n)$ is 1 in the feasible region for ξ_n and 0 otherwise.

5.3.2 Punctual distribution function of ξ_n

For each hop count $i \in \mathbb{N}$, we define a random variable E_i denoting the position of the last point in that hop. In the following, we compute the probability distribution functions for these random variables, $f_{E_i}(\xi_i) = f(x_i)$, where the joint distribution of $E_1, E_2, \ldots, E_n$ is indicated as $f(\xi_1, \xi_2, ..., \xi_n)$. Citing the total probability formula, we can write the following relations:

$$\begin{aligned} f(\xi_0) &= \delta(\xi_0) \\ f(\xi_1, \xi_0) &= f(\xi_1|\xi_0) f(\xi_0) \\ &\cdots \\ f(\xi_n, \xi_{n-1}, ..., \xi_0) &= f(\xi_n|\xi_{n-1}, ..., \xi_0) \cdots f(\xi_1|\xi_0) f(\xi_0) \end{aligned} \tag{5.10}$$

where $\delta(x)$ is the Dirac distribution function and the above distributions make sense in the intervals where the nodes can actually be placed, i.e., according with the constraints in Eq. (5.8), otherwise the functions equal 0. Moreover, due to the structure of the problem, $\forall\ n \in \mathbb{N},\ n \geq 2$, ξ_n only depends on the positions of the last nodes in the previous two hops (ξ_{n-1} and ξ_{n-2}). This can be written also including the interval of definition as:

$$f(\xi_n, \xi_{n-1}, \xi_{n-2}) = f(\xi_n|\xi_{n-1}, \xi_{n-2}) f(\xi_{n-1}, \xi_{n-2}) \overset{\xi_{n-1}+R}{\underset{\xi_{n-2}+R}{\mathbf{\Gamma}}}(\xi_n) \tag{5.11}$$

We can now rewrite the set of equations (5.10):

$$\begin{aligned} f(\xi_0) &= \delta(\xi_0) \\ f(\xi_1, \xi_0) &= f(\xi_1|\xi_0) f(\xi_0) \overset{\xi_0+R}{\underset{R}{\mathbf{\Gamma}}}(\xi_1) \\ &\cdots \\ f(\xi_n, \xi_{n-1}, \xi_{n-2}) &= f(\xi_n|\xi_{n-1}, \xi_{n-2}) f(\xi_{n-1}|\xi_{n-2}) \overset{\xi_{n-1}+R}{\underset{\xi_{n-2}+R}{\mathbf{\Gamma}}}(\xi_n) \end{aligned} \tag{5.12}$$

The marginal probability function $f(\xi_n)$ can therefore be computed by double integration of $f(\xi_n, \xi_{n-1}, \xi_{n-2})$ over ξ_{n-2} and ξ_{n-1}. The conditional probability

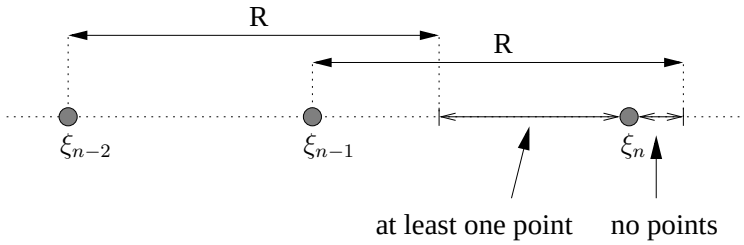

Figure 5.9: Feasible region for ξ_n once ξ_{n-2} and ξ_{n-2} are given.

$f(\xi_n|\xi_{n-1},\xi_{n-2})$ can be computed based on the fact that, in order for ξ_n to be the position associated with the last node in hop count n, the following two events *have to be jointly verified* (see Fig. 5.9):

A: "The interval $(\xi_{n-2}+R,\xi_n]$ is not empty".

B: "There are no points in the interval $(\xi_n,\ \xi_{n-1}+R]$".

The events A and B are independent because the original set of points (nodes) are the realization of a Poisson process. We can therefore write that:

$$\begin{aligned}
\text{Prob}(A) &= \text{Prob}\{K>0 \text{ in } (\xi_{n-2}+R,\xi_n]\} \\
&= 1-\exp\{-\lambda(\xi_n-\xi_{n-2}-R)\} \\
\text{Prob}(B) &= \text{Prob}\{K=0 \text{ in } (\xi_n,\xi_{n-1}+R]\} \\
&= \exp\{-\lambda(\xi_{n-1}+R-\xi_n)\}
\end{aligned} \tag{5.13}$$

We are now in the position of deriving the $F(\xi_n|\xi_{n-1},\xi_{n-2})$ cdf and its derivative $f(\xi_n|\xi_{n-1},\xi_{n-2})$:

$$\begin{aligned}
F(\xi_n|\xi_{n-1},\xi_{n-2}) &= P(A)P(B)\int_{\xi_{n-2}+R}^{\xi_{n-1}+R}\mathbf{\Gamma}(\xi_n) \\
f(\xi_n|\xi_{n-1},\xi_{n-2}) &= \lambda\, e^{-\lambda(\xi_{n-1}+R-\xi_n)}\int_{\xi_{n-2}+R}^{\xi_{n-1}+R}\mathbf{\Gamma}(\xi_n)
\end{aligned} \tag{5.14}$$

From Eqs. (5.14) and (5.12), the distribution function $f(\xi_n)$ can be obtained by induction as reported in Eq. (5.15):

$$f(\xi_n) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\xi_n, \xi_{n-1}, \xi_{n-2})\, \mathrm{d}\xi_{n-2}\, \mathrm{d}\xi_{n-1} =$$
$$= \lambda^n e^{-\lambda(nR-\xi_n)}. \tag{5.15}$$
$$\cdot \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathbf{\Gamma}_{\xi_{n-2}+R}^{\xi_{n-1}+R}(\xi_n) \cdots \int_{-\infty}^{\infty} \mathbf{\Gamma}_{\xi_2+R}^{\xi_3+R}(\xi_4) \int_{-\infty}^{\infty} \mathbf{\Gamma}_{\xi_1+R}^{\xi_2+R}(\xi_3)\, \mathbf{\Gamma}_{R}^{\xi_1+R}(\xi_2) \mathbf{\Gamma}_{0}^{R}(\xi_1)\, \mathrm{d}\xi_1\, \mathrm{d}\xi_2 \cdots \mathrm{d}\xi_{n-2}\, \mathrm{d}\xi_{n-1}$$

$$P_c(x) = \begin{cases} 1 x < R \\ \sum_{i=0}^{\lfloor \frac{x}{R} \rfloor} \left\{ \frac{[-\lambda e^{-\lambda R}(x-iR)]^i}{i!} \right\} - e^{-\lambda R} \sum_{i=0}^{\lfloor \frac{x}{R} \rfloor - 1} \left\{ \frac{[-\lambda e^{-\lambda R}(x-(i+1)R)]^i}{i!} \right\} x \geq R \end{cases} \tag{5.16}$$

In other words, we have decomposed $f(\xi_n)$ as the product of two terms, where the first one accounts for the network connectivity (λ), whereas the second term is independent of λ and, for a given ξ_n, accounts for the feasible region of the remaining $n-1$ points $\xi_0, \xi_1, \ldots, \xi_{n-1}$, according to the constraints in Eq. (5.8). More compactly, $f(\xi_n)$ can be written as:

$$f(\xi_n) = \lambda^n e^{-\lambda(nR-\xi_n)} \times G_n(\xi_n) \tag{5.17}$$

where $G_n(x)$ replaces the succession of integrals in Eq. (5.15). Moreover, $G_n(x)$ can be expressed as (see Section 5.3.3):

$$G_n(x) = \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} \left\{ g_n^i(x)\, \mathbf{\Gamma}_{iR}^{(i+1)R}(x) \right\} \tag{5.18}$$

where the $g_n^i(x)$ are polynomials with the following properties (easy verified by inspecting the number of integrals, the terms that are being integrated and the continuity properties of the punctual distribution functions):

1. The coefficients of $g_n^i(x)$ do not depend on λ.
2. The order of $g_n^i(x)$ is equal to $n-1$.
3. $g_n^i(i+1) = g_n^{i+1}(i+1)$.

Computing the distribution of the position of the last node in hop n makes sense only in the cases where the structure is connected up to that hop (i.e., there

are no *connectivity holes* larger than R up to the end of hop n). The problem of having a connected structure in the one–dimensional case has been studied in [11], where the authors derived a close formula (Eq. (5.16)) to obtain the probability $P_c(x)$ of having a connected structure up to any position $x \in \mathbb{R}^+$. The probability of having a connected structure up to the end of hop n is therefore obtained as $P_c[n] = P_c(x = nR)$. Observe that, for a given hop number n, the probability of having a connected structure is a constant quantity that is used to scale the joint pdf $f(\xi_n)$ in order to obtain the conditional distribution $f_c(\xi_n) = f(\xi_n | \{\text{Structure connected up to hop } n\})$. $f_c(\xi_n)$ is therefore given by:

$$f_c(\xi_n) = \frac{\lambda^n e^{-\lambda(nR-\xi_n)} \times G_n(\xi_n)}{P_c[n]} \tag{5.19}$$

where:

$$P_c[n] = \sum_{i=0}^{n} \frac{(-\lambda e^{-\lambda R}(x - iR))^i}{i!} - e^{-\lambda R} \sum_{i=0}^{n-1} \frac{(-\lambda e^{-\lambda R}(x - (i+1)R))^i}{i!} \tag{5.20}$$

Equation (5.19) gives the distribution of the last node in hop n. It is made up of two parts: an exponential function depending on the parameter λ and a set of polynomials whose coefficients are *independent* of λ. This formula is the central point for the analytical characterization of the hop count statistics for the 1D case.

5.3.3 Determination of the $G_n(x)$ coefficients

In this section we address the computation of the coefficients of the $g_n^i(x)$ polynomials that compose $G_n(x)$. As one can notice by inspecting Eq. (5.15), the coefficients of $g_n^i(x)$ are functions of the transmission range R. This means that, for every particular scenario, they should be computed separately, as the transmission range R is unlikely to be the same in all cases.

Fortunately, there is a way to avoid this. In fact, in [31], it has been proved that for a network whose nodes are distributed according to a Poisson process with parameter λ and R is the value of transmission range, one can vary R and modify λ, such that the properties of the underlying graph remain unchanged (the reverse situation where varying λ while adapting R is also true). In more details, if the distances are scaled by a ratio r/r' in the d-dimensional space, λ parameter must be modified according to:

$$\lambda' = (r/r')^d \lambda \tag{5.21}$$

In what follows, we consider the transmission range to be $R = 1$. Any particular application can then scale its distances such that the transmission range becomes 1 and vary λ accordingly. This means that any application could then use the coefficients of the polynomials as computed bellow at no additional expense. In other words, this makes the coefficients of $g_n^i(x)$ constant and independent of the particular scenario (R and λ). To keep the notation simple, in the following we consider that we already modified R to 1 such that λ is the new value obtained after the normalization step. This will not have any influence on the final results.

Moreover, we observe that the $\mathbf{\Gamma}$ functions only take values in $\{0, 1\}$, and this mean that their only influence is on the intervals over which the integration in Eq. (5.15) takes place. In the form in which it is presented, Eq. (5.15) is impossible to be integrated, as the integration variables are all linked together.

To make this integration possible in numerical form, we adopt the following three–steps procedure:

Step 1: From the definition of the $\mathbf{\Gamma}$ functions and keeping in mind that the variables we use conform to the inequalities described in (5.8), we can write the following identity:

$$\overset{\xi_{i-1}+1}{\underset{\xi_{i-2}+1}{\mathbf{\Gamma}(\xi_i)}} = \begin{bmatrix} \xi_{i+1}-1 & i \\ \xi_{i+2}-1 & \lfloor \frac{i}{2} \rfloor \end{bmatrix} \cdot \begin{bmatrix} \xi_{i+1}-1 & i-1 \\ \xi_{i}-1 & \lfloor \frac{i}{2}-1 \rfloor \end{bmatrix} \cdot \begin{bmatrix} \xi_{i}-1 & i-2 \\ \xi_{i-1}-1 & \lfloor \frac{i}{2}-2 \rfloor \end{bmatrix} \tag{5.22}$$

where $i \in \mathbb{N},\ i = \overline{1..n}$. Therefore, we can apply the transformation in (5.22) to each $\mathbf{\Gamma}$ function in $G_n(x)$. Observe that, in Eq. (5.15), the terms ξ_{n+2} and ξ_{n+1} do not exist, their place being taken by the terms $n + 2$ and $n + 1$, respectively.

Step 2: One can observe that each integration has as limits integer numbers. More than this, the integration is possible only if the integral over the generic variable ξ_i is split into $(i - \lfloor \frac{i}{2} \rfloor)$ disjoint integrals where the $\mathbf{\Gamma}$ functions to be integrated are evaluated; each integral corresponds to one interval of size 1 - the union of these disjoint intervals covers the feasible region for ξ_i, $(\lfloor \frac{i}{2} \rfloor, i)$.

Step 3: For what concerns the combination of any three subsequent variables of the kind $(\xi_{i+2}, \xi_{i+1}, \xi_i)$, only one out of the three situations presented in Fig. 5.10 is possible. That is, if a is an integer such that: $\lfloor \frac{n+2}{2} \rfloor \leq a < n + 2$, then:

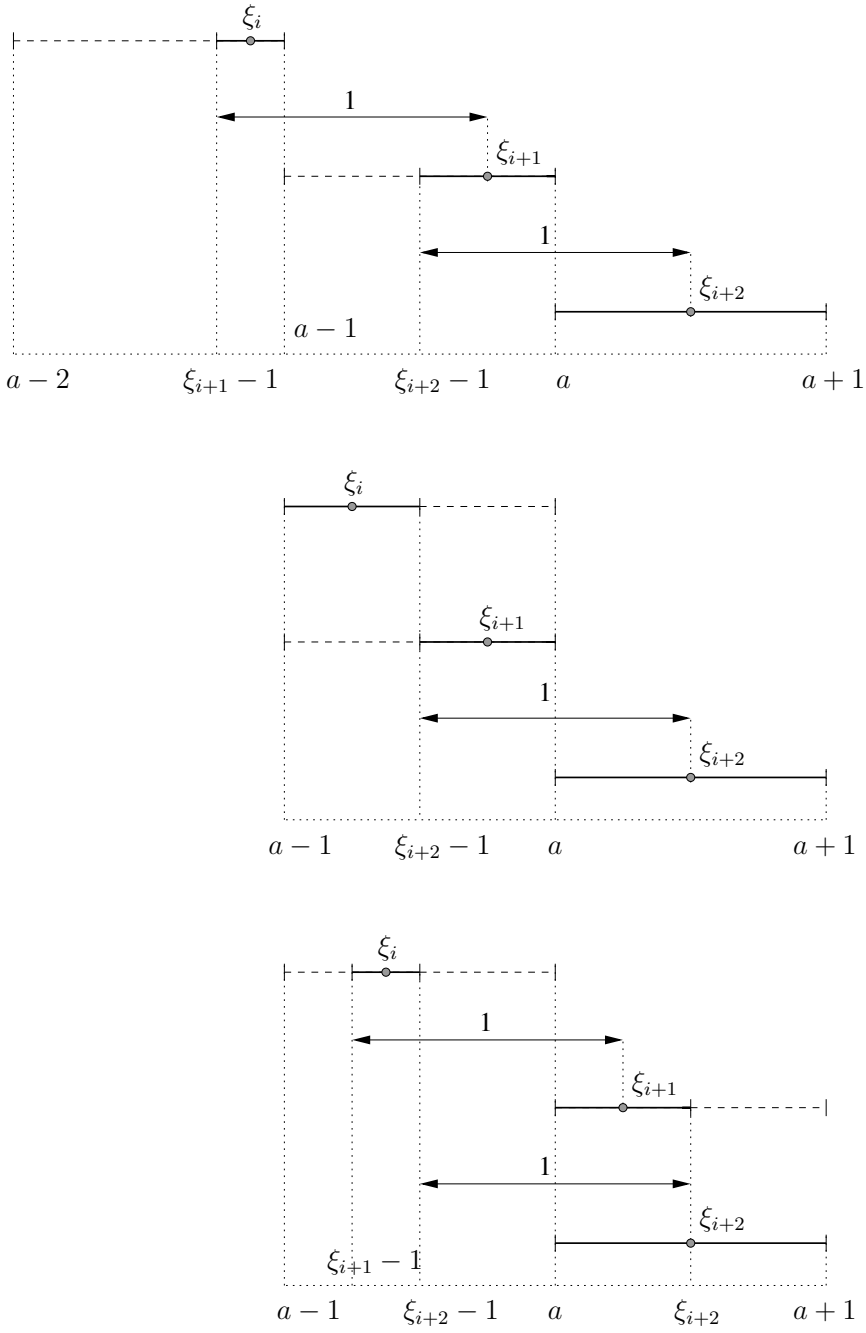

Figure 5.10: Possible configurations of the last nodes in each hop, $a \in \mathbb{N}$ such that $\lfloor \frac{n+2}{2} \rfloor \leq a < n+2$.

$\mathbf{g_n^i(x)}$	$\mathbf{x^5}$	$\mathbf{x^4}$	$\mathbf{x^3}$	$\mathbf{x^2}$	$\mathbf{x^1}$	$\mathbf{x^0}$
g_1^1	–	–	–	–	–	1.0000
g_1^2	–	–	–	–	−1.0000	2.0000
g_1^3	–	–	–	0.5000	−1.0000	0.5000
g_2^3	–	–	–	0.5000	−3.0000	4.5000
g_2^4	–	–	−0.3333	2.0000	−3.5000	1.6667
g_3^4	–	–	−0.1667	2.0000	−8.0000	10.6667
g_2^5	–	0.0833	−0.6667	2.0000	−2.6667	1.3333
g_3^5	–	0.1250	−1.5000	6.2500	10.1667	4.7083
g_4^5	–	0.0417	−0.8333	6.2500	−20.8333	26.0417
g_3^6	−0.0417	0.5833	−3.1667	8.3333	−10.6250	5.2500
g_4^6	−0.0333	0.6667	−5.0833	18.0000	−27.9583	12.7167
g_5^6	−0.0083	0.2500	−3.0000	18.0000	−54.0000	64.8000

Table 5.2: Coefficients of the $g_n^i(x)$ polynomials $(i = \overline{2..6})$.

$$
\begin{aligned}
&1.\ \xi_{i+2} \in (a, a+1) \quad \xi_{i+1} \in (a-1, a) \quad \xi_i \in (a-2, a-1) \\
&2.\ \xi_{i+2} \in (a, a+1) \quad \xi_{i+1} \in (a-1, a) \quad \xi_i \in (a-1, a) \\
&3.\ \xi_{i+2} \in (a, a+1) \quad \xi_{i+1} \in (a, a+1) \quad \xi_i \in (a-1, a)
\end{aligned}
$$

The integration limits (feasible regions) in the previous three cases are accounted for by the following expressions:

$$
\begin{aligned}
&1.\ \int_{a}^{a+1} \mathbf{\Gamma}(\xi_{i+2}) \int_{\xi_{i+2}-1}^{a} \mathbf{\Gamma}(\xi_{i+1}) \int_{\xi_{i+1}-1}^{\xi_{i+2}-1} \mathbf{\Gamma}(\xi_i) \\
&2.\ \int_{a}^{a+1} \mathbf{\Gamma}(\xi_{i+2}) \int_{\xi_{i+2}-1}^{a} \mathbf{\Gamma}(\xi_{i+1}) \int_{a-1}^{\xi_{i+2}-1} \mathbf{\Gamma}(\xi_i) \\
&3.\ \int_{a}^{a+1} \mathbf{\Gamma}(\xi_{i+2}) \int_{a}^{\xi_{i+2}} \mathbf{\Gamma}(\xi_{i+1}) \int_{\xi_{i+1}-1}^{\xi_{i+2}-1} \mathbf{\Gamma}(\xi_i)
\end{aligned}
\tag{5.23}
$$

By applying the two steps above and using the integration limits described in Eq. (5.23), one can perform the integration in a recursive manner. As an example, we present in Table 5.3.3 the $g_n^i(x)$ parameters for $n = \overline{2..6}$. Fig. 5.11 shows the punctual distribution function for the last node in each hop, for the first 10 hops and for $\lambda = 6.7$. The theoretical distributions are plotted against simulation points.

5.3.4 Hop count number and distance statistics

In this subsection we derive the relationships between the hop count number and the actual position of any given node. The basic assumption is that we are

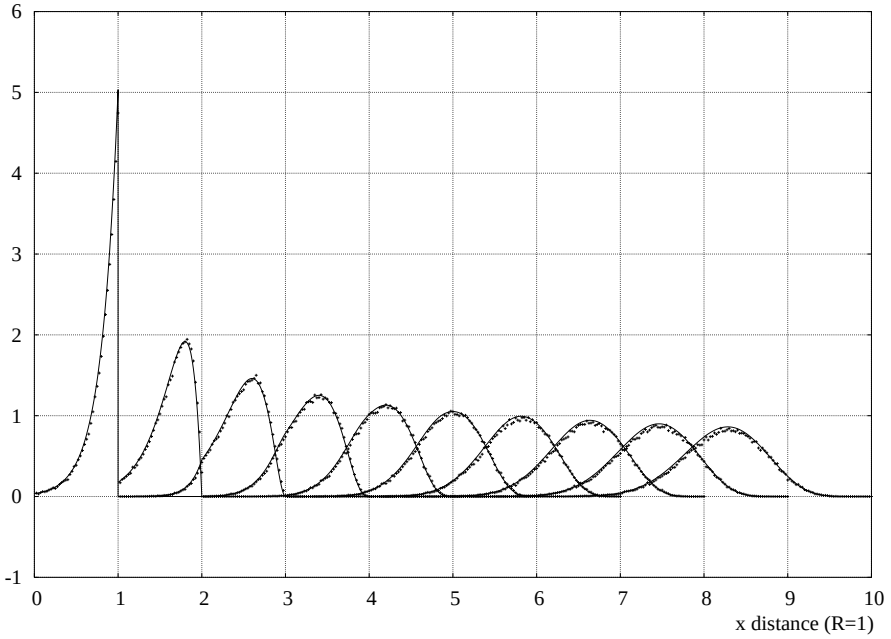

Figure 5.11: Probability distribution functions of the last node in each hop ($n = \overline{1..10}$), $\lambda = 6.7$.

dealing with a connected topology up to and including the node in question. Assume that a node finds out the number of hops it is separated from the source node SN. This information is enough for it to get an estimation of the distance between the two points. From the distribution of the distance, the node can compute the average distance and its associated standard deviation and use it, e.g., in localization algorithms, routing protocols, etc.

The distribution function of the distance conditioned on the number of hops has particular expressions for the case when the node lies in the first and second hop. For the third hop on, we provide a recursive formula for the computation of the distribution. By ξ_i we understand the position of the last node in hop i; and by x_i we understand the position of any node in hop i. The punctual distribution function for the case where the node lies in the first hop (f_{d1}) is given by:

$$f_{d1}(x_1) = \frac{1}{R}\mathbf{\Gamma}\binom{R}{0}(x_1) \tag{5.24}$$

This result is quite obvious: a node is in the first hop if it is in transmission range with the source node. Within this interval, due to the fact that we are dealing

with a Poisson process, it has an equal probability of being at any given position, thus a uniform distribution. Assume now that a node belongs to the second hop. We can write the following expressions:

$$\begin{aligned} f(x_2|\xi_1) &= \frac{1}{\xi_1}\mathbf{\Gamma}_{R}^{\xi_1+R}(x_2) \\ f(\xi_1, x_2) &= f(x_2|\xi_1)f(\xi_1) \\ f_{d2}(x_2) &= \int_{-\infty}^{\infty} f(\xi_1, x_2)\ \mathrm{d}\xi_1 \end{aligned} \tag{5.25}$$

Node x_2 belongs to the second hop. This means that it is in transmission range of the last node in the first hop and out of the transmission range of the source node. Moreover, given that the position of the last node in the first hop is ξ_1, the node in second hop can belong only to the interval $(R, \xi_1 + R]$. Its distribution is again a uniform distribution due to the uniform deployment of the nodes. We can now compute $f(\xi_1, x_2)$ and integrate it over ξ_1 in order to get the marginal distribution $f_{d2}(x_2)$ (the expression for $f(\xi_1)$ has already been determined in Section 5.3.2). Performing these computations leads us to the following result:

$$f_{d2}(x_2) = \lambda e^{-\lambda R}\left(\mathcal{E}i(-\lambda x_2 + \lambda R) - \mathcal{E}i(-\lambda R)\right) \tag{5.26}$$

where $\mathcal{E}i(x)$ is the exponential integral function, defined as:

$$\mathcal{E}i(x) = -\int_{-\infty}^{\infty} \frac{e^{-x}}{x}\mathrm{d}x \tag{5.27}$$

For the distributions of distances in hops further away than the second hop, we use the following reasoning (see Figure 5.9): a node belongs to hop n if it is in the transmission range of the last node in hop $n-1$ and is out of the transmission range of the last node in the $n-2$ hop. Therefore, within the interval $(\xi_{n-2}+R, \xi_{n-1}+R)$ the position of the node is uniformly distributed. The marginal distribution $f(\xi_n)$ is obtained as:

$$
\begin{aligned}
f(x_n|\xi_{n-1},\xi_{n-2}) &= \frac{1}{\xi_2-\xi_1}\overset{\xi_{n-1}+R}{\underset{\xi_{n-2}+R}{\mathbf{\Gamma}}}(x_n) \\
f(x_n,\xi_{n-1},\xi_{n-2}) &= f(x_n|\xi_{n-1},\xi_{n-2})f(\xi_{n-1},\xi_{n-2}) \\
f_{dn}(x_n) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x_n,\xi_{n-1},\xi_{n-2})\,\mathrm{d}\xi_{n-2}\,\mathrm{d}\xi_{n-1}
\end{aligned}
\tag{5.28}
$$

The expression of $f(\xi_{n-1},\xi_{n-2})$ has already been determined in Section 5.3.2. Numerical integration of these expression should follow the method described in Section 5.3.3 of rearranging the integration limits.

5.3.5 Conclusions

In this section we have focused on the case of the sensor networks having the nodes deployed in a random manner, along a line. Knowing the average density of nodes and the hopcount number for a node with respect to a sink, we are able to compute a distance estimate based on the underlying statistics. These method provides a mean of estimating distances without the need of additional distance measurement hardware.

The exact formulas have been deducted for this one dimensional scenario. Unfortunately, for higher order scenarios, the formulas are proved not to have a closed form, thus making computation extremely difficult and making approximate approaches the only solution.

Lately, the one dimensional scenario has been approached also from the graph theory point of view. An optimal algorithm has been given [7] and it is in our intention, as future work, to combine it with the statistical approach to yield to results usable in real applications.

5.4 Statistically enhanced localization algorithms

In this section we focus on two existing localization schemes and study them in the case of semi-static, randomly deployed, wireless sensor networks. In the majority of the application scenarios it is assumed that the nodes of the network are deployed in a random manner such that they cover uniformly a given surface [1, 2, 13]. However, the properties that this hypothesis provides are in general not used in the design of sensor networks protocols. The implications of this assumptions have been studied since the early days of ad-hoc networks [28] yet

are usually not taken into consideration. As we will show in this paper, the usage of the properties of the *randomly deployed* sensor network will lead to major improvements in performance for the chosen localization algorithms, requiring almost no additional resources.

The basic idea that we will be exploiting is that knowing the average density of the nodes and having an expression that can characterize the transmission range of the nodes, one can get a distance estimate between any two nodes if the number of hops separating the two of them is known (see Section 5.3). The information about the distance comes as a pair: mean value and standard deviation. The quality of the estimation depends on at least two factors: the distance separating the two nodes (the closer the nodes, the more exact information available) and the average density of nodes (more nodes, better distance estimates).

The new schemes presented here come with little additional costs: only the average number of nodes needs to be known (or determined with a distributed protocol). This value has to be distributed among the sensor nodes once. This little extra overhead is alleviated since the protocols we propose have at least one phase less than the existing ones. More than this, the node density can be determined locally with almost no effort (for example based on information already available from the media access control protocol).

An additional overhead is the amount of computation needed. The protocols we propose will make use of the weighted least squares method for determining the position of the nodes. While this method is resource and time consuming it might still be considered acceptable for the majority of the cases, as no additional distance measurement hardware is needed and these computation are made only once in the lifetime of the network for static networks.

A difficult problem to be addressed is the model to be chosen for the transmission range of the nodes. There are several models already available to characterize the communication channels, yet none of them is able to characterize all the environments envisioned for sensor networks (ranging from outdoor environments with almost no interference to office spaces with lots of communication disrupting factors [46]). No matter which transmission model is considered, a certain trashhold for the transmission range can be computed, such that, if two nodes are placed at a distance larger than this value, the probability that they can communicate to each other can be approximated with 0. In the following we will assume that such a value R has been determined.

5.4.1 Basic ideas

The existing localization protocols can be classified in two large categories depending on whether nodes make use or not of devices that give range estimates.

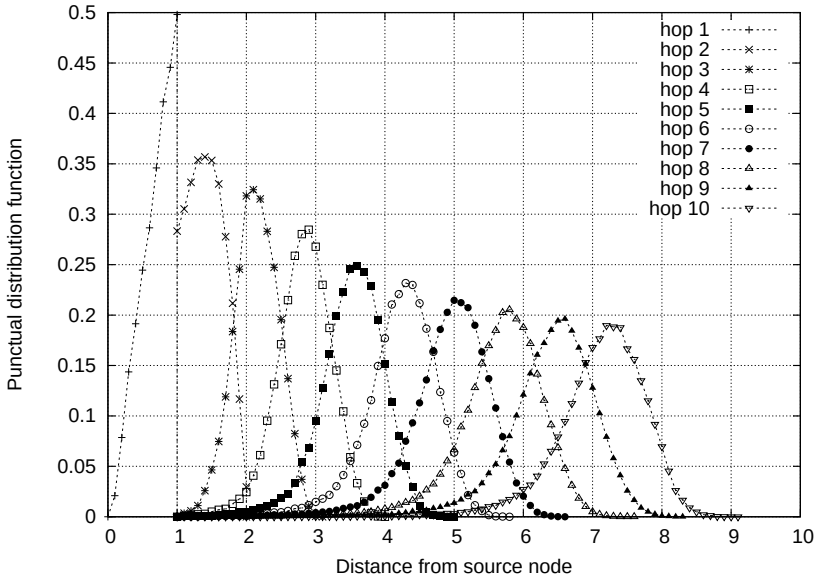

Figure 5.12: Node distribution among the hops

Usually, methods that use distance estimates (based on RSSI, time of flight, angle of arrival, etc.) provide better results than the ones based only on connectivity information. Unfortunately, correct distance estimation is expensive because of the additional hardware that needs to be added to each node and the energy it consumes.

Using the power level at the receiver is a method that gives an estimate of the distance with no additional cost (for the transceiver devices that can supply this information). In the case of randomly deployed sensor networks, a second estimation, based on the density of the nodes (λ) and the model of the transmission range, is available for free.

By random deployment we understand topologies that spread the nodes uniformly over a certain area. In the ideal case, the area is supposed to be infinitely large, but in practice is limited to finite rectangular surfaces. Other random deployment models have been studied, e.g. a Gaussian model where nodes were scattered following a normal distribution around a central base station [32]. For specific situations, the best suited model should be developed and used (e.g. nodes uniformly deployed in an area containing obstacles, or along rectangles mapping over corridors and rooms on a two-dimensional map representation, etc.).

In the following we will consider the case of networks with nodes uniformly deployed within a rectangular area. The transmission model used will be the one

that considers a fixed transmission range for all the nodes. While this model may not well characterize the indoor environments, we will be using it as an exercise tool. More suited probabilistic models can be used afterward for particular environments.

Different application scenarios will have different characteristic values for λ and the transmission range R. It has been proved before (see [31]) that for a network whose nodes are distributed according to a Poisson process with parameter λ and whose nodes have a transmission range R, one can vary R and modify λ accordingly, such that the properties of the underlying graph remain unchanged (the reverse situation where varying λ while adapting R is also true). If the distances are scaled by a ratio r/r' in the d-dimensional space, the λ parameter must be changed to $\lambda' = (r/r')^d\lambda$. For the ease of use, and to make results comparable, one can consider the transmission range $R = 1$ and vary λ accordingly (as will be seen also in the following graphs).

Let us pick one node in the network and call it the source node. We say that a node belongs to the first order neighborhood of the source node (or is in the first hop) if it is in transmission range of the source node. A node belongs to the second order neighborhood (the second hop) if it is in transmission range of any node in the first hop but it is not in the transmission range of the source node. More generally, a node belongs to hop n, $(n \in \mathbb{N},\ n > 2)$ if it is in the transmission range of any node from hop $n-1$ and is outside the transmission range of all the nodes in hops $n-2$ or lower.

The position of the nodes in the given network can be modeled using a two dimensional Poisson process. One of the basic results for this sort of networks is the relationship between the density of the network and the average number of neighbors a node can have in its vicinity. If we assume that the transmission range of any node is fixed and equal to R, then the number of first-order neighbors a node might have is a random variable $\mathbf{n}$, with $P(\mathbf{n} = n) = e^{-\lambda R}\frac{(\lambda R)^n}{n!}$.

The mean distance (and also standard deviation) to a node in the first-order or second-order neighborhood can be easily computed from the expressions of the distribution of the nodes in these hop counts. For higher order neighborhoods, there are no exact analytical expression available as the expression to be computed become too complicated (the major problem is integrating combinations of exponential and linear functions over domains resulting from intersections of circles). But, there have been developed approximate results for the problem [32, 35]. More than this, theoretical bounds were determined and the approximation results stay within these bounds. The problem was studied even further as the influence of the deployment area was taken into consideration (see for an example on how to address these problems [4]).

Let us choose a test environment similar to the ones given as examples in

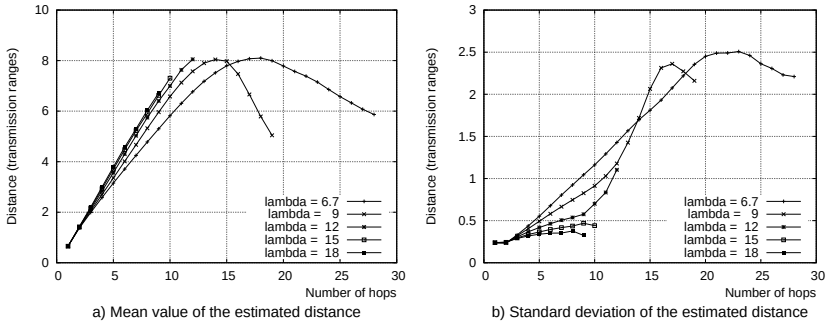

Figure 5.13: Estimated distance function of the number of hops

several papers dealing with localization issues: 200 nodes are randomly deployed in a rectangular area of size 1x1 units^2. As exact analytical expressions are not (yet) available, we will be using information about the distribution of the nodes in the network estimated out of a large number of simulations. The obtained results confirm the existing distributions for the first two hops and are within the determined bounds for larger hopcounts. The values for λ and R were chosen such that the obtained structure is connected with high probability (assuming $R = 1$, the corresponding $\lambda \geq 6.7$).

We recorded the distribution of nodes in each hop both including and ignoring the border effect (see Figure 5.12 for the case of $R = 0.12$ units). As one can notice the curves tend to have a Gaussian form as the number of hops is increasing. There is certainly a relationship between the mean values of these distributions even if the underlying structure is random.

Based on the values in Figure 5.12 we computed the mean and standard deviations of the average distance to a node in a certain hop (shown in Figure 5.13). For the first part of the curves (up to hops 10-15), the border effects can be neglected and both the mean distance and their associated standard deviations can be placed on a line. The right part of the curves shows irregularities due to border effects, as hops with very high numbers appear only for source nodes placed in the corners of the rectangles, etc. This is why, for example, for $\lambda = 6.7$, the mean distance decreases after a hop count larger than 18 and the standard deviations no longer line up. The interpretation of the decreasing standard deviation is that for the case when this occurs, the source and destination nodes are forced to be situated somewhere near one of the diagonals of the square area rather than deployed all over its surface.

It is reasonable to assume that nodes further away from the source node should have a diminished influence on the source position than closer nodes (based on

the fact that precision is affected by distance). This is confirmed by the previous results as well, measuring the increasing standard deviations of the estimated distances with the number of hops. Based on this observation and the previous figure, we are able to quantify the influence of nodes having different hop counts by associating to each node a specific weight. If the distance to node i has a standard deviation of σ_i, then its associated weight is $w_i = \frac{1}{\sigma_i^2}$ (this association should lead to the best results according to the estimation theory).

The localization algorithms we will use for exemplification use a least squares method to determine the position of the nodes. In order to make use of the previously described coefficients we will make use of a weighted least squares method and achieve better results than the original algorithms.

For the range based localization algorithms, usually the distance accumulated over the shortest path is propagated. Figure 5.14 presents a comparison between the mean value of the accumulated distance on the shortest path and the "real" estimated distance to a node n hops away from the source node (and their associated standard deviations). The propagated distance increases linearly with the number of nodes and maintains a relatively small (increasing) standard deviation. The real distance to the nodes can be seen to follow a curved shape due to the border effects of the selected area and has a standard deviation several times larger than the equivalent propagated distance. This shows that measurements coming from further away nodes should have a diminished influence on the position of a source node than closer nodes.

By inspecting Figure 5.14, coefficients can be computed (based on the shapes of the two lines) to correct the distance estimates (a similar coefficient was computed in a different manner in [22] or see Section 5.2). For example, Table 5.3 presents the coefficients and the weights computed for the simulation setup described in Section 5.4.6. It can be noticed that the weights decrease towards zero in fast, meaning that the influence of further away nodes becomes negligible after a relative small number of hops. This information can be used as the upper bound on the *time to live* field associated with the messages used for localization purposes (the time to live field shows how many hops a message is allowed to travel before it expires). The correction coefficients for the distances have been computed based on the mean of distances (both measured and propagated via the shortest path) for each specific hop count. As expected, the coefficients decrease with the number of hops, but not so abruptly as the associated weights.

We chose two localization schemes (DVHop and DVDistance) in order to show that the basic observations described in this section can lead to major improvements in terms of localization errors. The first protocol belongs to the group of range free protocols while the second one is a distance based one. In this section we will briefly describe the original protocols and the statistically enhanced

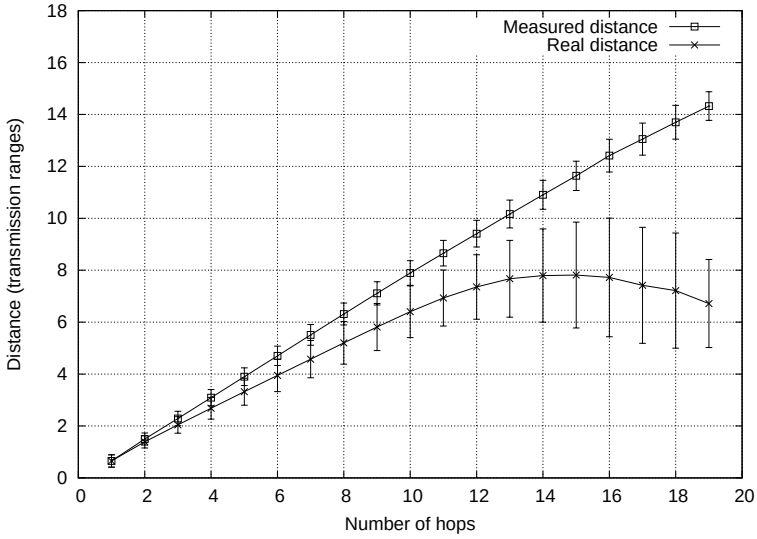

Figure 5.14: Relationship propagated distance - real distance

derived protocols.

5.4.2 DVHop protocol

In the following we will address as *anchors* the nodes that know their positions (for example given by additional hardware as GPS or because the user specified their position). DVHop is made of the following phases:

1. nodes get distances, in number of hops, to the anchors via a distance vector exchange algorithm (the anchors also get distance information to the other existing anchors in the system).
2. each anchor node computes an average hop distance, based on the information received from the other anchors. In the protocol, the sum of all distances to the other anchors is divided by the sum of the hopcounts to these anchors. Each anchor distributes the computed value to the neighboring nodes via controlled flooding.
3. each node computes its location based on the estimated distances to the anchors (based on the number of hops and the average hop distance). Because of the errors, the final position is chosen as the one that minimizes the errors in least squares sense with respect to the known distance estimates (in the literature the method is referred to as *lateration* - see Section 5.1.2).

The protocol produces positioning errors comparable to the transmission range of each node. It produces acceptable results in uniform networks and can be used on networks with nodes having no means of measuring distances. The protocol is sensitive mainly to the number of anchor nodes.

5.4.3 DVHopSE protocol

In order to make use of the information in Section 5.4.1 we need to distribute the value of λ to all nodes. This can be done with an initial flooding phase if the user knows it beforehand or can be performed locally at each node. There can be derived a full variety of algorithms, the simplest ones being: each node counts the nodes in its neighborhood and based on this it can estimate the λ parameter. The larger the neighborhood considered, the better the result (the basic assumption is that the network is *uniformly distributed*). This phase can be easily embedded into the distance vector exchange phase.

The phases of the DVHop modified algorithm (DVHopSE) are then:

1. distance vector exchange such that each node gets distance information to the beacons. This phase also includes the initial learning of the λ parameter in one of the previously described ways. The anchor nodes need not to compute or distribute hop distance estimates.
2. computation of the location - each node has hopcount distances to the anchors and knows the density of the nodes, thus they can determine distance estimates to the anchors and then compute their position.

The position is computed using a weighted least squares method. The weights to be used are computed based on the λ coefficient and the hop count to each of the anchors. These coefficients can be given as look up tables or even distributed with the λ value in the initial flood phase if needed.

5.4.4 DVDistance protocol

DVDistance is a distance based localization protocol meaning that the nodes have to be equipped with distance measurement hardware (which for example could be based on RSSI). Accumulated distances over the shortest paths are propagated in the network and nodes estimate their position based on these results.

The phases of the algorithm are:

1. anchor nodes start flooding the network with their position. Each node receiving the first message from an anchor adds to it also the distance estimate from the node it received the message. Subsequent messages from the same anchor are discarded. This way, each node learns accumulated shortest path

Hop number	1	2	3	4	5	6	7	8	9
Weights (%)	26.48	27.05	14.84	8.59	5.53	3.86	2.91	2.23	1.82
Dist. coeff. (%)	100	92.76	89.17	86.77	85.15	83.93	83.06	82.40	81.77

Table 5.3: Weights and distance correcting coefficients (R=0.119 units, λ=9)

distance to the surrounding anchors.

2. nodes compute their position based on the information received from the anchors and the accumulated shortest path distances. They use a least squares method to determine the position leading to the smallest errors.

This method is sensitive to the errors introduced by the distance measuring hardware and also to the *time to live* field of the messages (the number of hops a message is allowed to travel before it expires) and to the number of anchors.

5.4.5 DVDistanceSE protocol

DVDistance produces better results than DVHop, but is sensitive to the distance measuring hardware errors and the time to live field of the messages. In the same time, the distance obtained is different from the real distance (see Figure 5.14) due to the fact that the shortest path is longer than the actual real distance between the nodes. This last problem can be addressed with a correcting coefficients computed for each particular deployment case.

The time to live associated with each message is a very important parameter. In order to accentuate this idea, let us suppose that all the messages are allowed to travel throughout the whole network. This will lead to big errors in the global positioning of the nodes: each node will have the majority of the anchors positioned on one side, exceptions being the nodes in the center of the network. The position of any node will be influenced by the majority of the anchors, so its computed position will be shifted against this side with respect to the center of the area (the final topology would be like the original one *exploded*).

Figure 5.15 captures this phenomena. The dots indicate the actual position of the nodes while the crosses represent the computed position. The left part of Figure 5.15 illustrates DVDistance protocol, where the messages had no expiring time set. All the estimated positions are moved towards the exterior of the figure. To better illustrate the concept, we divided by 1.5 all the received distance estimates that reached the nodes. The results are presented in the right part of the same figure. As expected, each node got underestimated distances to the anchors, leading to the reversed effect: each node computed position is moved towards the majority of the anchors, so it is attracted to the center of the figure. An equilibrium point between the two situations exist, and a parameter specific to

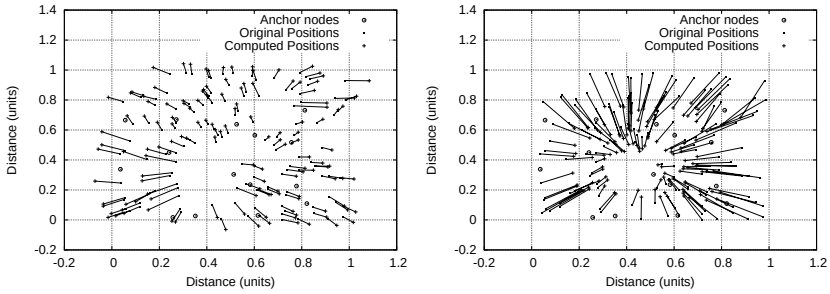

Figure 5.15: Influence of the time to live parameter (left side DVDistance with TTL=∞, right side DVDistance with TTL=∞ and distances reduced with 150%)

a particular implementation can easily be found.

We choose the time to live parameter based on the following two bounds: the inferior bound is based on the fact that each node should have information from at least 3 anchors, and the superior bound is given by the fact that their influence should be major. Based on these two factors we decided for the particular configuration to choose a time to live equal to 6 (98.23% nodes have access to more than 3 anchors).

The phases of the DVDistanceSE protocol are similar to the original one:

1. anchor nodes start flooding the network with their position in the same manner as in DVDistance algorithm. Additionally the messages contain also the number of hops they traveled. In the same time information about the λ parameter and the area of the network is spread into the network.
2. nodes compute their positions based on the received information from the anchors, the computed coefficient and the weights. They use a weighted least squares method to determine the position leading to the smallest errors.

The computation could stop here (see Section 5.4.6 for a description of the obtained results). But, when computing its position, each node can also get an estimate of how precise this position is (this is possible because the standard deviations for the measurements that led to a certain position are known). Some additional minor improvement can be obtained if a third phase is added in which the nodes broadcast also their location together the associated precision. A node will correct its position based on the neighbors information if their precision is higher than its current one. This phase is usually costly in terms of number of messages exchanged and duration in time [22].

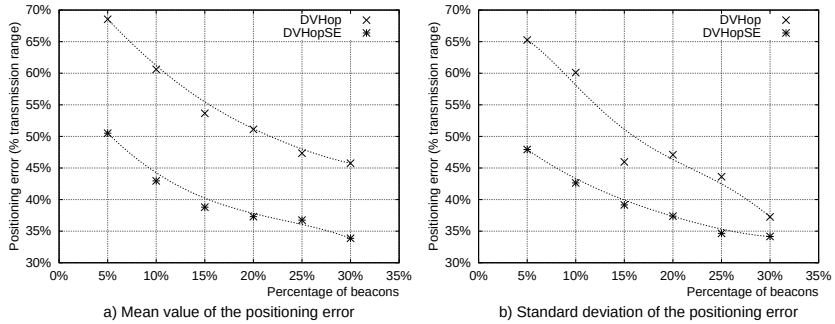

Figure 5.16: Comparison DVHop - DVHopSE

5.4.6 Simulation results

We have simulated the previously described four protocols using MATLAB. In this section we will discuss the precision of the obtained results in terms of overall positioning error. The communication issues as collisions, number of messages and delays were not taken into considerations due to the fact that the differences between the presented protocols are minor from this point of view and we are interested in characterizing the precision of the localization method rather than the efficiency of the particular underlying network protocol stack.

The simulation scenario consisted of 200 nodes distributed onto a square area (1x1 units^2). From the set of random topologies created we have discarded the partitioned network topologies. The simulations were repeated several hundred of times to obtain results characterized by high confidence.

Figure 5.16 presents a comparison between DVHop and DVHopSE protocols. The transmission range was fixed to 0.119 units and a number of randomly chosen nodes were designated as anchors (varying from 5% to 30% from the total number of nodes). The graphs show the mean and standard deviation values of the distribution of the overall positioning error. It can be seen that an improvement in mean error between 22% and 29% was achieved, while the improvement in the standard deviation lie in the range 8% and 29%.

The explanation for this improvement lies in the fact that the closer anchors had a bigger influence in the position of the nodes than anchors being further away. DVHop computes hop estimates that are best fit the local topology: if nodes are denser in a certain part of the network, they will get a smaller value for average hop distance than nodes in a less denser part of the network. DVHopSE averages these values to only one. Simulation results show that the weights associated to how far the anchors are situated play a bigger role than the local densities.

The second simulation scenario involves DVDistance and DVDistanceSE pro-

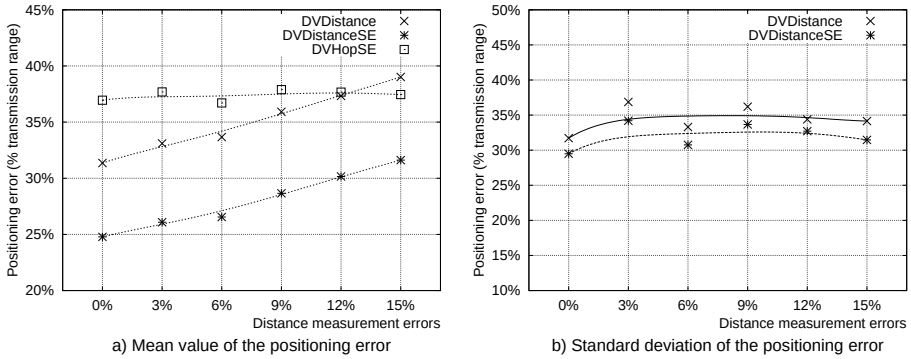

Figure 5.17: Comparison DVDistance - DVDistanceSE

tocols. We have chosen for the average connectivity the value 9 and varied the precision of the distance measuring devices: they introduced a Gaussian error with mean 0 and standard deviation ranging from 0% and 15% of the measured distance. A comparison between these two protocols is illustrated in Figure 5.17. DVDistanceSE protocol brings an improvement of 19% to 21% improvement in the mean value of the positioning error. The standard deviation of the positioning error also is improved, but with a smaller amount (the improvement is situated between 5% and 8%). The improvement achieved is due to the two new factors introduced: the shortest path distances correction coefficients and the weights associated to each hop.

In Figure 5.17 we also plotted for comparison the performance of the DVHopSE protocol (where 10% of the nodes were considered anchors). This has a constant performance because it is not influenced by the distance measuring hardware errors (it is a range free protocol). For the case when perfect distance measures are available, the improvement that DVDistanceSE brings is of 33%. This value decreases with the amount of errors added up to 16% improvement for the worst case considered in the simulation.

It is interesting to notice that for errors larger than 12% the distance based algorithm produces worst results than the range free algorithm. In such situations, the later is prefered as not only the results are better but also th amount of needed resources is considerable smaller.

Both statistically enhanced protocols perform better in terms of mean value of the positioning error and standard deviation.

5.4.7 Conclusions

In Section 5.4.1 we have focused on the particular case of sensor networks that respect the property that their nodes are uniformly distributed in a random manner on a certain surface. Based on this hypothesis, we have extracted distance estimates based on the hopcount of the network and used them to show that improvements are possible in the currently available localization protocols.

The main idea of this paper is that *random deployment* assumption that is present in the vast majority of scenarios is not exploited. The underlying random topology has certain properties that can be actually used in protocols, instead of being rediscovered with the cost of additional resources and time.

We have chosen as examples two localization protocols. We have obtained improvements (some of them already predicted by theory) that otherwise would cost additional software and hardware resources. In our case, we have managed to obtain with range free protocols the results similar to those obtained using their range-based equivalents.

Future work is aimed at better understanding the influence of underlying statistics in the localization protocols. Other deployment schemes apart from the uniform deployment (Poisson points) in both two dimensional and three dimensional cases will be examined.

5.5 Conclusions and future work

In this chapter we have addressed some of the aspects of positioning problem in wireless sensor networks. A large number of prototypes have already been built and give the user the possibility of determining his position with a certain degree of precision. At the same time, the problem received attention from theoretical perspective as well, results from various research fields and various approaches being aggregated together.

Still, we can say that the localization problem is open for research. There is still no algorithm to be adopted as a standard one and which could work in the majority of situations. The gap between the theoretical results and practical implementations is large as the practical implementations make use of additional hardware and the theoretical studies are based on idealized hypothesis too far away from reality.

There are various mechanisms that can be used when designing a localization algorithm. The possible combinations produce results more or less fitted for various application scenarios. For example, we have shown how using the precision of the computed position can be used to obtain improved results in certain cases.

The study of the randomness implications in designing the algorithms was also an example.

Future work should focus on studying the available mechanisms and finding the combinations between them that will produce the best results. At the same time, future investigations should analyze the combination of results from graph theory (giving clear bounds of what can be achievable) and statistical techniques specific to estimation theory (giving results characterizing the average cases rather than the worst ones).

Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks (Elsevier) Journal*, 38(4):393–422, March 2002.

[3] P. Bahl and V.N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *INFOCOM vol.2*, pages 775–784, 2000.

[4] C. Bettstetter and J. Eberspacher. Hop distances in homogenous ad hoc networks. In *Proceedings of IEEE Vehicular technologies conference(VTC), Korea*, 2003.

[5] C. Bettstetter and C. Wagner. The spatial node distribution of the random waypoint mobility model. In *1st german workshop on mobile ad-hoc networks (WMAN02), Ulm, Germany*, 2002.

[6] J. Beutel. Geolocation in a picoradio environment. In *MS Thesis, ETH Zurich, Electronics Lab*, 1999.

[7] R. Bischoff and R. Wattenhofer. Analyzing connectivity-based, multi-hop ad-hoc positioning. In *2nd IEEE International Conference on Pervasive Computing and Communications (PerCom), Orlando, Florida, USA*, 2004.

[8] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor localization for very small devices. In *IEEE personal communications*, pages 28–34, 2000.

[9] S. Capkun, M. Hamdi, and J.P. Hubaux. GPS-free positioning in mobile ad-hoc networks. In *Proceedings of the Hawaii Int'l Conf. Sys. Scis.*, 2001.

[10] L. Doherty, K. Pister, and L.El Ghaoui. Convex position estimation in wireless sensor networks. In *IEEE INFOCOM, Anchorage, AK*, 2001.

[11] O. Dousse, P. Thiran, and M. Hasler. Connectivity in Ad Hoc and Hybrid Networks. In *IEEE Infocom*, volume 2, pages 1079–1088, 2002.

[12] Ekahau positioning system. http://www.ekahau.com.

[13] D. Estrin, R. Govindan, J.S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.

[14] Amorai-Moriya et al. United states invention 6,316,934, november 13, 2001.

[15] Hamdi et al. Method and wireless terminal for generating and maintaining a relative positioning system. World Intellectual Property Organization WO 02/03091 A2, January 2002.

[16] Maloney et al. United states invention 6,119,013, september 12, 2000.

[17] Maloney et al. United states invention 6,127,975, october 3, 2000.

[18] Nysen et al. United states invention 6,259,991, july 10, 2001.

[19] Richley et al. United states invention 6,542,083, april 1, 2003.

[20] Stilp et al. United states invention 6,603,428, august 5, 2003.

[21] Want et al. United states invention 6,292,744, september 18, 2001.

[22] L. Evers, S. Dulman, and P. Havinga. A distributed precision based localization algorithm for ad-hoc networks. In *Proceedings of Pervasive Computing*, 2004.

[23] B. Ghosh. Random distances within a rectangle and between two rectangles. In *Bull. Calcutta Math. Soc., vol.43*, pages 17–24, 1951.

[24] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network*, 8(1):62–70, February 1994.

[25] J. Hightower, R. Want, and G. Borriello. SpotON: An indoor 3D location sensing technology based on RF signal strength. Technical Report UW-CSE 00-02-02, University of Washington, Seattle, February 2000.

[26] L. Hu and D. Evans. Localization for mobile sensor networks. In *Tenth Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2004.

[27] A.R. Jiminez, F. Seco, R. Cere, and L. Calderon. Absolute locatilization using active beacons :a survey and IAI-CSIC contribution.

[28] Leonard Kleinrock and John Silvester. Optimum Transmission Radii for Packet Radio Newtorks or why Six is a Magic Number. In *Proceedings of IEEE National Telecommunications Conference*, pages 4.3.1–4.3.5, Birminghan, Alabama, 1978.

[29] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *Proceedings of the Third IEEE International Workshop on Visual Surveillance (VS'2000)*, pages 3–10. IEEE Computer Society, 2000.

[30] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: A quantitative comparison. In *Computer Networks (Elsevier), special issue on Wireless Sensor Networks*, 2003.

[31] R. Meester and R. Roy. *Continuum Percolation*. Cambridge University Press, 1996.

[32] L. Miller. Distribution of link distances in a mobile network. In *Journal of Research of the National Institute of Standards and Technology*, 2001.

[33] P. Misra, B.P. Burke, and M.M. Pratt. Gps performance in navigation. In *Proceedings of IEEE, vol.87, nr.1, pp.65-85*, 1999.

[34] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *ACM Joint Workshop on Foundations of Mobile Computing (DIALM-POMC), Philadelphia, Pennsylvania, USA*, 2004.

[35] J. Mullen. Robust approximations of the distribution of link distances in a mobile network. In *Mobile computing and communications review, vol. 7(2)*, 2003.

[36] K. Muthukrishnan, M. Lijding, and P. Havinga. Towards Smart Surroundings: enabling techniques and technologies for localization. In *Proceedings of International Workshop on Location and Context Awareness (LoCA2005)*, 2005.

[37] L.M. Ni, Y. Liu, Y.C. Lau, and A.P. Patil. LANDMARC: Indoor location sensing using active RFID. *Wireless Networks. Special Issue on Pervasive Computing and Communications*, 10(6):701–710, 2004.

[38] D. Niculescu and B. Nath. Localized positioning in ad hoc networks. *Elsevier journal of Ad Hoc Networks*, 1(2-3):247–259, 2003.

[39] D. Niculescu and B. Nath. Error characteristics of ad hoc positioning systems. In *ACM MOBIHOC 2004, Tokyo*, 2004.

[40] D. Niculescu and B. Nath. Position and orientation in ad hoc networks. *Elsevier journal of Ad Hoc Networks*, 2(2):133–151, 2004.

[41] R.J. Orr and G.D. Abowd. The smart floor: a mechanism for natural user identification and tracking. In *CHI '00 extended abstracts on Human factors in computing systems*, pages 275–276. ACM Press, 2000.

[42] S. Pace, G. Frost, I. Lachow, D. Frelinger, D. Fossum, D.K. Wassem, and M. Pinto. *The Global Positioning System*, chapter GPS history, chronology and budgets, pages 237–270. RAND Coorporation, 1995.

[43] A. Papoulis and S.U. Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw–Hill Higher Education, fourth edition, 2002.

[44] Pinpoint 3D positioning system. http://www.pinpointco.com.

[45] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom'00)*, pages 32–43. ACM Press, 2000.

[46] N. Reijers, G. Halkes, and K. Langendoen. Link layer measurements in sensor networks. In *First IEEE conference on mobile ad-hoc sensor systems (MASS2004), SUA*, 2004.

[47] K. Romer. The lighthouse location system for smart dust. In *In Proceedings of MobiSys 2003 (ACM/USENIX Conference on Mobile Systems, Applications, and Services), pages 15–30, San Francisco, CA, USA*, 2003.

[48] C. Rose. Mean internodal distance in regular and random multihop networks. In *IEEE Transactions on Communications, vol.40*, Aug. 1992.

[49] A. Savvides, C.C. Han, and M. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *7th ACM Intl. Conf on Mobile Computing and Networking (MOBICOM), p.166-179*, 2001.

[50] A. Savvides, H. Park, and M.B. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *WSNA, Atlanta, GA, pp.112-121*, 2002.

[51] Y. Shang and W. Ruml. Improved mds-based localization. In *Proceedings of the 23rd Conference of the IEEE Communicatons Society (Infocom 2004)*, 2004.

[52] Y. Shang, W. Ruml, Y. Zhang, and M.P.J. Fromherz. Localization from mere connectivity. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc'03)*, pages 201–212. ACM Press, 2003.

[53] P. Steggles and J. Cadman. A comparison of RF tag location products for real world applications. Technical report, Ubisense limited, March 2004.

[54] D. Stoyan, W.S. Kendall, and J. Mecke. *Stochastic Geometry and its Applications*. John Wiley & Sons, 2 edition, 1995.

[55] A. Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, 2001.

[56] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. In *ACM Transactions on Information Systems*, pages 91–102, 1992.

[57] M. Zorzi and R.R. Rao. Geographic random forwarding GeRaF for ad hoc and sensor networks: energy and latency performance. In *IEEE Transactions on Mobile Computing*, Oct.-Dec. 2003.

Chapter 6

Conclusions

Experimental sensor networks prototypes open up new vistas for scientists and engineers to observe physical phenomena and react to it. Monitoring either the environmental factors affecting the shy sea birds creatures comings and goings from their burrows on a uninhabited island coast of Maine, or the temperature and condition of the 165,000 tonnes of malting barley at the Holland Malt in the Netherlands, sensor networks start becoming useful and overcoming the experimental status.

The field of wireless networked sensors is still a young, new field. There are a lot of questions to be answered at all the levels. Theoretical research, fancy application domains will answer questions such as which is the best routing protocol to be used, what trade-offs need to be made to acquire a smaller latency, how to combine and present the pieces of data such that the big picture makes sense, etc.

6.1 High-level overview of the thesis

This thesis dealt with the first basic question that arises when starting building a sensor network: how should the system look like? The traditional answer to this question is: use a fixed protocol stack. There are several advantages given by this approach: standardization, ease of understanding of the system, etc. and the certainty given by the myriad of already built systems. But still there is a major disadvantage: the resulting system is not optimized for the task at hand, leading to a short life of the limited energy resources and more often will not fit the targeted hardware. Add to this the big latencies achieved and the annoyances of cross-layer design and optimization and it will soon be clear that the fixed protocol stack is not an option.

Various systems solved this problem by using a component graph approach:

the clear layered separation disappeared but the rigidity of the fixed stack still persisted. The nodes are deployed in dynamic environments where the monitored characteristics vary with time, requiring dynamic topologies and various versions of applications to be run. The fixed architecture solutions can answer this issue by erasing the code in the sensor nodes and downloading whole new versions in them. The cost is tremendous in terms of energy and latency and thus this is not a good approach.

We have proposed a data centric architecture. We placed the notion of data in the center of the whole design and built a system around it. The system is made up of software components that connect together to a central entity. The required/produced data by each one of them is managed by this central entity as well as the whole set of running protocols. Dynamic reconfiguration is as simple as connecting/disconnecting modules to the central entity. The advantages are that the operating system inside the sensor nodes came one step closer to what are we used to in the personal computers: management of a local storage containing programs, executing the needed program at the needed time without the having to recompile the whole system.

We have taken this simple ideas and implemented a real time operating system and a simulator. The simplicity of the approach allowed us to pack into the operating system a number of features, each though not to be possible in the limited target environment. The two systems are functional and are already being used in a large number of research projects.

Then, we have focused on a particular building block of wireless sensor networks: the positioning protocols. We have developed several new algorithms and implemented them. These algorithms integrate perfectly into the data centric architecture as they need data from almost all the other building blocks to function. The data sharing mechanism shows its power here, the designer does not have to worry about the internal data exchange mechanisms.

The new algorithms are based on two features previously not used in localization protocols. The first one was the fact that a node that already computed its position has also some information regarding the precision of this result. By communicating it to its neighbors, the overall positioning solution lies closer to the real situation. The second feature was the property of the sensor networks of having the nodes randomly deployed. The underlying statistics contain some properties that are usually not used and thus, certain properties have to be “rediscovered” at run time using expensive protocols.

6.2 Strengths and weak points

Let us take a look, also from a high level perspective, at the strengths and weak points of the proposed solutions:

- **Data centric architecture -** The main advantages of using this architecture in a system is the fact that the need for a fixed protocol stack disappears. Various combinations of protocols can be employed when needed, new protocols can be started, in a word, the best configuration can be obtained with minimal effort at real-time. Exchanging data among the entities is also simple, as the whole scheme relies on this mechanism.

  There are some disadvantages as well: the scheme works well for configurations containing a small number of components. As the number of entities increases and the sorts of produced/required data per entity increases, the complete architecture can be hard to understand at a first glance. Logical clustering of sets of entities into functional blocks might help to understand better what is going on. Debugging or surveillance of the data traffic is very simple: just create an entity that requires to receive all the published data. An interesting observation is that this dynamic architecture can emulate any fixed protocol stack architecture in a straight forward manner.

  The overhead paid for employing the data centric architecture is the need for a robust way of naming the data (assuming networks built out of software components developed by various parties). From implementation point of view, a central component doing the data management has to be implemented in an efficient manner as it will run each time new data is produced.

- **Data centric operating system (DCOS) -** The feasibility of an operating system offering the capabilities of DCOS was regarded with skepticism by the community in the early phases of its development. Real time capabilities, memory management and dynamic reconfiguration were thought impossible to achieve even individually on a processor having two kilobytes of data memory. Luckily, the combination between the scheduler and the data manager of the data centric architecture, the employment of the EDFI scheduling mechanism and the usage of linked lists as a basic data type led to a working system.

  Among the disadvantages of the system is the fact that the scheduling mechanism does not cover the latency introduced by the scheduler itself. This prevents the system from offering hard real time capabilities, stopping at soft real time level only. But advances in the theory behind the scheduler will overcome this in future work. Another disadvantage is related to the

estimation of the cost of the tasks (the time needed by a task to run on the processor). As software almost always contains loops with the number of cycles depending on state of the task, there are two approaches to this: consider the worst case scenario or consider the mean value of the associated cost. Both approaches are valid, the choice depending on the application at hand and on the set of tasks needed.

- **Simulation framework** - The simulation tool has proved itself an invaluable tool. We have designed and tested a large number of protocols with its aid and found out problems very hard to trace otherwise. Its future use will improve its functionality and speed, as almost each of its users gave suggestions and traced existing bugs. It is quite easy to learn (as OMNet++ as well), we have been noticing that the average computer programmer with basic knowledge of C++ needed around one day of practice before plunging into the description of his own protocols.

- **Localization schemes** - The localization schemes we described can be arranged into two categories: the first one brings improvements based on an existing "trick" not used (available precision of the computed positions), and the second one invents a new "trick" (the statistics of a random deployment) and uses it.

  The net advantages of these schemes are that they bring improvements without requiring additional hardware. There are costs to be paid, nevertheless: in the first case the additional energy spent on communications, and in the second case the assumption of random deployment. Due to the different sets of hypotheses on which they are based the usability of the methods depend on the applications considered. Further enhancements are possible and indicate the future research directions we have in mind.

  We feel that the statistics lying under the random distribution assumption should receive a higher attention. Although the results are not so simple to obtain (integration over domains resulting from intersections of circles seems to appear everywhere), at least some trends can be identified and used. These statistics have the nice advantage that probably they are the only useful characteristic of the sensor networks (from the point of view of the network designer) that scales with the number of nodes: more nodes deployed, more precise results and decreasing standard deviations.

6.3 In the end...

The author of the thesis has gained a lot of information on a field of research about which he knew nothing three years prior to the writing of this document. There were many "obvious" things that proved themselves wrong during the development of various protocols and systems.

The main lesson learned was: ***Keep things simple!***

The best algorithms, protocols and systems are the simple ones. Their description, implementation, particular cases, number of additional tools, mechanisms, data structures, etc. are always kept at a minimum. The simpler the better. This does not necessarily mean that the problems should be considered in a simplistic manner. On the contrary, the environment on top of which the systems are built may be very complex. Studying the underlying mechanisms can take a tremendous amount of time, effort and frustration. But, by understanding these mechanisms, there is always a way in which they can be used to the advantage of the system rather than counteracted via "smart algorithms". A simple and elegant solution proves the best understanding of the underlying phenomenon and minimizes the so much hated cost functions.

At this moment we can say that the research we performed during the last three years, some of it presented in this document, leads to the conclusion that the basic assumption that sensor networks are feasible is true. In the last year we have witnessed an increasingly number of companies interested in the possibilities of this technology and the number of applications ordered by the industry raises every day.

Sensor networks were listed as one of the future technologies, maybe *the next big thing*. A lot of effort has still to be put in and prototypes will be build until they will reach the level of reliability to be assimilated on a large scale in the everyday life. The author of this thesis personally believes that the technology can be considered to have reached its maturity at the moment when he will dare to board a plane that contains at least one such system as a key point in its design.

The information and solutions presented in this thesis required a lot of time and effort and endless discussion sessions to be "invented", studied and verified. It would be more than rewarding if any part of this thesis would serve as a small "building block" to some theory or system useful for all of us.

Appendix A

List of publications

Chapters in books and papers in various workshops, conferences and journals:

- S. DULMAN, M. ROSSI, P. HAVINGA, M. ZORZI On the Hop Count Statistics for Randomly Deployed Ad-Hoc and Sensor Networks - *under submission* -

- S. DULMAN, S. CHATTERJEA, P. HAVINGA Chapter 31: Introduction to Wireless Sensor Networks *The Embedded Systems Handbook, Editor Richard Zurawski, Taylor & Francis CRC Press,ISBN 0-8493-2824-1*

- S. DULMAN, S. CHATTERJEA, T. HOFMEIJER, P. HAVINGA, J. HURINK Chapter 33: Architectures for Wireless Sensor Networks *The Embedded Systems Handbook, Editor Richard Zurawski, Taylor & Francis CRC Press, ISBN 0-8493-2824-1*

- S. DULMAN, P. HAVINGA Data Centric Wireless Sensor Networks *Proceedings of the 2005 NSTI Nanotechnology Conference (Nanotech2005)* Anaheim, California, USA, May 2005

- S. DULMAN, P. HAVINGA - Statistically enhanced localization schemes for randomly deployed wireless sensor networks, *Proceedings of DEST International Workshop on Signal Processing for Sensor Networks,* Melbourne, Australia, December 2004

- T.J. HOFMEIJER, S.O. DULMAN, P.G. JANSEN, P.J.M. HAVINGA AmbientRT - Real Time System Software Support for Data Centric Sensor Networks *Proceedings of DEST International Workshop on Signal Processing for Sensor Networks (ISSNIP2004),* Melbourne, Australia, December 2004

- S. DULMAN, T. HOFMEIJER, P. HAVINGA AmbientRT - Real Time, Data Centric System Software for Wireless Sensor Networks *Proceedings of the 21st Sensor Symposium,* Kyoto, Japan, October 2004

- S. DULMAN, T. HOFMEIJER, P. HAVINGA Wireless sensor networks dynamic runtime configuration *Proceedings of ProRisc 2004* the Netherlands, November 2004

- T.J. HOFMEIJER, S.O. DULMAN, P.G. JANSEN, P.J. HAVINGA DCOS, a Real-Time Light-Weight Data Centric Operating System *Proceedings of the IASTED International Conference on Advances in Computer Science and Technology (ACST 2004)* Virgin Islands, USA, November 2004

- J. WU, S. DULMAN, P. HAVINGA Reliable Splitted Multipath Routing for Wireless Sensor Networks *Proceedings of Building Intelligent Sensor Networks (BISON'04)* Wuhan, China, October 2004

- A.G. RUZZELLI, L. EVERS, S.O. DULMAN, L.F.W. VAN HOESEL, P.J.M. HAVINGA On the design of an energy-efficient low-latency integrated protocol for distributed mobile sensor networks *Student paper, Proceedings of the International Workshop on Wireless Ad hoc Networks,* Oulu, Finland, June 2004

- L. EVERS, S. DULMAN, P.J.M. HAVINGA A Distributed Precision Based Localization Algorithm for Ad-Hoc Networks *Proceedings of Pervasive Computing (PERVASIVE 2004),* Vienna, Austria, April 2004

- J. WU, P. HAVINGA, S. DULMAN, T. NIEBERG Routing Protocol for Wireless Sensor Networks *Poster Session of the European Workshop on Wireless Sensor Networks* Berlin, Germany, January 2004

- S. DULMAN, L. VAN HOESEL, P. HAVINGA AND P. JANSEN Data Centric Architecture for Wireless Sensor Networks *Proceedings of the ProRISC Workshop,* Veldhoven, the Netherlands, November 2003

- J. WU, P. HAVINGA, S. DULMAN, T. NIEBERG Energy Efficient Routing in Wireless Sensor Networks *Proceedings of the ProRISC Workshop,* Veldhoven, the Netherlands, November 2003

- T. NIEBERG, S. DULMAN, P. HAVINGA, L.VAN HOESEL, J. WU Collaborative Algorithms for Communication in Wireless Sensor Networks *in Ambient Intelligence: Impact on Embedded Systems, edited by T.Basten and M.Geilen and H.de Groot, Kluwer Academic Publishers,* November 2003

- S. DULMAN, L.VAN HOESEL, T. NIEBERG, P. HAVINGA Collaborative Communication Protocols for Wireless Sensor Networks *Proceedings of European Research on Middleware and Architectures for Complex and Embedded Systems Workshop,* Pisa, Italy, April 2003

- S. DULMAN, J. WU, P. HAVINGA An Energy-Efficient Multipath Routing Algorithm for Wireless Sensor Networks *Supplement of the The Sixth International Symposium on Autonomous Decentralized Systems,* Pisa, Italy, April 2003

- S. DULMAN, P. HAVINGA A Simulation Template for Wireless Sensor Networks *Supplement of the The Sixth International Symposium on Autonomous Decentralized Systems,* Pisa, Italy, April 2003

- S. DULMAN, T. NIEBERG, J. WU, P. HAVINGA Trade-Off between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks *Proceedings of WCNC Workshop,* New Orleans, Louisiana, USA, March 2003

- S. DULMAN, P. HAVINGA, J. HURINK Wave Leader Election for Wireless Sensor Networks *Proceedings of MMSA Workshop,* Delft, the Netherlands, December 2002

- S. DULMAN, P. HAVINGA Operating System Fundamentals for the EYES Distributed Sensor Network *Proceedings of PROGRESS Workshop,* Utrecht, the Netherlands, October 2002

CTIT Technical reports:

- O.S. KAYA, O. DURMAZ INCEL, S. DULMAN, R. GEMESI, P. JANSEN, P. HAVINGA *Using TinyOS Components for the Design of an Adaptive Ubiquitous System* TR-CTIT-05-17, May 2005, 10 pp.

- T.J. HOFMEIJER, S.O. DULMAN, P.G. JANSEN, P.J.M. HAVINGA *AmbientRT - real time system software support for data centric sensor networks* TR-CTIT-05-02, January 2005, 10 pp.

- L.F.W. VAN HOESEL, S.O. DULMAN, P.J.M. HAVINGA, H.J. KIP *Design of a low-power testbed for Wireless Sensor Networks and verification* TR-CTIT-03-45, September 2003, 10 pp.

- T. NIEBERG, S. DULMAN, P. HAVINGA, L. VAN HOESEL, J. WU *Collaborative Algortihms for Communication in Wireless Sensor Networks* TR-CTIT-03-44, September 2003, 28 pp.

- S. DULMAN, P. HAVINGA *A Simulation Template for Wireless Sensor Networks* TR-CTIT-03-15, April 2003, 6 pp.
- S. DULMAN, T. NIEBERG, J. WU, P. HAVINGA *Trade-Off between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks* TR-CTIT-03-14, April 2003, 9 pp.
- S. DULMAN, J. WU, P. HAVINGA *An Energy Efficient Multipath Routing Algorithm for Wireless Sensor Networks* TR-CTIT-03-13, April 2003, 6 pp.
- S. DULMAN, L. VAN HOESEL, T. NIEBERG, P. HAVINGA *Collaborative communication protocols for wireless sensor networks* TR-CTIT-03-08, March 2003, 9 pp.
- S. DULMAN, P. HAVINGA, J. HURINK *Leader Election Protocol for Energy Efficient Mobile Sensor Networks (EYES)* TR-CTIT-02-21, July 2002, 16 pp
- S. DULMAN, T. NIEBERG, P. HAVINGA, P. HARTEL *Multipath Routing for Data Dissemination in Energy Efficient Sensor Networks* TR-CTIT-02-20, July 2002, 6 pp
- Y.W. LAW, S. DULMAN, S. ETALLE, P. HAVINGA *Assessing Security-Critical Energy-Efficient Sensor Networks* TR-CTIT-02-18, July 2002, 13 pp