Spectrum-efficient Architecture for Cognitive Wireless Sensor Networks

Thesis presented by
Saül Garcia Huertes

Mémoire de fin d'études

2009
(Submitted May 18, 2009)

To my parents and my sister.

Acknowledgments

In the first place, I would like to thank my promotor, Prof. Ph. De Doncker, for accepting me as a Erasmus student here, at the Université Libre de Bruxelles (ULB), and my co-promotor, Dr. J-M Dricot, since he has been so helpful and, in the meantime, he has made me feel comfortable at the working time along this season.

Firstly, the most deserved acknowledgment to my parents, Matilde and Antonio, and my sister, Selena. It is because of your wholehearted support that I finished my degree, and it has been easier with you by my side, helping me always when I needed.

Secondly, I am grateful to all my new friends, and sometimes housemates, here in Brussels, Noelia, Pablo, Tomás, Carol, Ilaria, Alberto, Adriana, Giovanni, Javi, Paquito, Coral, Alba, Andrea, Belén, Beniamino, Michele and Victor. Together with Albert, Elo, Isa, Joan, Júlia, Laia, Marta, Mónica and many more. I would like to thank you all for your help and for the great moments that we had here. I am sure, we will repeat that who knows where. I know that by mentioning you all it may not be as special as that it could be, however, I could not forget to any of you.

To all my old friends from Madrid and Barcelona, like David, Ander, Anna, Laura, Muntsa and Rosa. Thank you a lot since I have just realized that the time passes by but you are still there. Besides, I am lucky to have friends like you Alex, Ana B., Ana E., Anxo, Luis, Mireia, Nacho and Zaida. After all that we have enjoyed together and everything that we have next, I felt like telling you that.

As well as the people who I have to thank most, my closest friends, my adventure mates in some cases, my classmates in a few occasions, and even my everything once. I am talking about Aleix, Guarch, Adri, Tutu, Pau, Sobri, Ruth, and specially Vivi since I had my best time ever with you, so that my most affectionate gratitude goes to you, and that is the reason why I really want to thank you how happy you made me and, in the meantime, how you help me as well. Without all of you, all these tough years would not have been the same including our trips, our spare time and, above all, the time that we spent together at the university.

All of you know me well and, so that, you know how shy I am while talking about my feelings, then, I hope you appreciate these acknowledgments.

Saul

“*I do not think that the wireless waves I have discovered will have any practical application*”

Heinrich Rudolf Hertz

Abstract

Nowadays there is the envision that in a few years the current Wireless Sensor Networks (WSNs) will be present in many applications in the near future. As long as they keep performing on the ISM 2,4GHz unlicensed band, they will have to coexist with other succesful technologies such as Wi-Fi or Bluetooth. Therefore, it is obvious that specially the mentioned band is becoming overcrowded. However and thanks to the emerging Cognitive Radio (CR) techniques, they will allow to apply an effective Dynamic Spectrum Access (DSA) that will address the problem of the scarce available spectrum by a rational allocation of the current wireless communications within the available spectrum on a given time and place. This actuation will permit accessing to less crowded frequencies in order to perform with less interferences and even with less propagation losses.

Along this work, an Spectrum-efficient Architecture for Cognitive Wireless Sensor Networks is introduced. The scheme performs a data collection protocol, with a tree-based topology, completely scalable and with a generic purpose. Through real testing, we can state that our scheme, without altering the normal data collection cycle, is able to feel the presence of other WSN and, consequently, it will migrate the entire network to a new frequency while all these operations are transparent to the final user. It is energy-efficient as well, since no redundant scanning is carried out. As a conclusion, our proposal assures a better performance in case of an external WSN interference without adding neither complex operations nor more traffic to the network.

Hoy en día existe la creencia de que en unos pocos años las actuales Redes Inalámbricas de Sensores estarán presentes en muchas aplicaciones. Mientras estas sigan actuando en la banda sin licencia de ISM 2,4GHz, tendrán que coexistir con otras exitosas tecnologías como Wi-Fi o Bluetooth. En consecuencia, resulta obvio asegurar que la banda en cuestión estará superpoblada en un futuro próximo. Sin embargo y gracias a las nuevas técnicas de Radio Cognitiva, que permitirán la aplicación de un eficiente Acceso al Espectro Dinámico, se conseguirá una distribución racional, dentro del espectro disponible en ese momento y lugar, de las comunicaciones inalámbricas que se estén llevando a cabo. Esta actuación permitirá acceder a frecuencias menos pobladas para poder transmitir con menos interferencias e incluso con menos pérdidas de propagación.

A lo largo de este trabajo se va a presentar una arquitectura eficiente, espectralmente hablando, para Redes Inalámbricas de Sensores y Cognitivas. Este esquema desarrolla un protocolo de recolección de datos, para una red con topología de árbol, totalmente escalable y con finalidades genéricas. A través de las pruebas realizadas, podemos afirmar que nuestro esquema, sin alterar el ciclo normal de recolección de datos, puede detectar la presencia de otras Redes Inalámbricas de Sensores y, consecuentemente, migrar la red a nueva frecuencia mientras que todas estas operaciones están ocultas al usuario final. También es eficiente a nivel de energía, ya que no se realizan comprobaciones redundantes de la presencia de otras redes. De esta manera, nuestra propuesta asegura un mejor comportamiento en caso de la existencia de una Red Inalámbrica de Sensores externa, sin realizar operaciones complicadas ni añadiendo más tráfico a la red.

Contents

Acknowledgments **iii**

Abstract **v**

1 Introduction **1**

2 Cognitive Radio **3**

2.1 Background 4

2.2 Cognitive Radio Networks 6

2.2.1 Spectrum sensing 7

2.2.1.1 Transmitter detection 7

2.2.1.2 Cooperative detection 7

2.2.1.3 Interference-based detection 8

2.2.2 Spectrum management 8

2.2.2.1 Spectrum analysis 8

2.2.2.2 Spectrum decision 9

2.2.3 Spectrum mobility 9

2.2.4 Spectrum sharing 9

2.3 Future challenges 9

3 Wireless Sensor Networks **11**

3.1 Background 11

3.2 Applications 12

3.3 Sensor networks features 13

3.4 Standards 15

3.4.1 IEEE 802.15.4 15

3.4.1.1 Topology 16

3.4.1.2 PHY layer 17

3.4.1.3 MAC layer 18

3.5 Comercial products 18

3.5.1 IRIS 19

3.6 Operating systems 19
3.6.1 TinyOS 20
3.7 Programming languages 21
3.7.1 nesC 21

4 Design approach **23**
4.1 Related work 24
4.2 Channel interference 25
4.3 Suggested scenario 26
4.4 Application description 28
4.4.1 CTP on TinyOS 28
4.4.1.1 Collection 30
4.4.1.2 Dissemination of Small Values 31
4.4.2 Role description 31
4.4.2.1 Root node 32
4.4.2.2 Client node 33

5 Implementation **34**
5.1 Performance 34
5.2 Events execution 37
5.3 Baring the code 37
5.3.1 Variables, structs and constant definitions 39
5.3.2 Functions and event handlers 40
5.4 Global application issues 42
5.4.1 Considerations 42

6 Evaluation **44**
6.1 Results 45

7 Conclusions and future work **47**
7.1 Future work 47

Bibliography **49**

A Application code **52**

List of Figures

2.1 Measurement of 0-6GHz spectrum utilization 4
2.2 Cognitive cycle 5
2.3 Cognitive radio flowchart 7

3.1 Current environmental WSN applications 13
3.2 The overall Zigbee protocol stack 15
3.3 A few examples of 802.15.4 existing topologies 16
3.4 The IEEE 802.15.4 and IEEE 802.11 PHY channels disposition 17
3.5 Sensor node architecture 18
3.6 The TinyOS 802.15.4 frame format 20
3.7 Structure files for the TinyOS application definition 21

4.1 Interference due to domestic devices while a MicaZ is transmitting 24
4.2 Hospital floorplant diagram distribution for a possible WSN application . . . 27
4.3 CTP either data and routing frames 30
4.4 Channel distribution before and after the cognitive performance 32
4.5 Root node flowchart 33

5.1 Application description example 36
5.2 Events execution for both existing roles on the network 38

6.1 Network distribution during the test 44
6.2 Root node event log 46

A.1 RssiDetection.h 53
A.2 RssiDetectionC.h 53
A.3 RssiDetectionC.nc 54
A.4 RssiDetectionC.nc 55
A.5 RssiDetectionC.nc 56
A.6 RssiDetectionC.nc 57
A.7 RssiDetectionC.nc 58
A.8 RssiDetectionAppC.nc 59

List of Tables

3.1 Frequency bands and data rates for the IEEE 802.15.4 standard 17
3.2 Hardware platform evolution 19

4.1 Output power comparison on the 2,4GHz ISM band 25

5.1 Preset values for the global application 42
5.2 Memory requirements for the global application 42

Nomenclature

ACK	Acknowledgment packet
AM	Active Message
CCA	Clear Channel Assesment
CR	Cognitive Radio
CSMA-CA	Carrier Sense Medium Access - Collision Avoidance
CTP	Collection Tree Protocol
DSA	Dynamic Spectrum Access
DSSS	Direct Sequence Spread Spectrum
ED	Energy Detection
ETSI	European Telecommunications Standards Institute
ETX	Expected Transmissions
FCC	Federal Communications Comission
FFD	Full Function Device
FIFO	First In First Out
GTS	Granted Time Slot
IEEE	Institute of Electrical and Electronic Engineers
ISM	Industrial, Science and Medical
LQI	Link Quality
LR-WPAN	Low Rate - Wireless Personal Area Network
MAC	Medium Access Control

NTIA	National Telecommunications and Information Administration
OS	Operating System
PHY	Physical layer
QoS	Quality of Service
RFD	Reduced Function Device
RKRL	Radio Knowledge Representation Language
RSSI	Radio Signal Strength Indicator
THL	Time Has Lived
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

Chapter 1

Introduction

The relatively new Wireless Sensor Networks (WSNs) are currently subjected to a deep development in order to exploit the benefits of these simple, cheap, and robust networks. The starting point was over a decade ago when, according to their characteristics, a wide range of application possibilities, beyond the military domain, appeared. Mainly, commercial domains such industrial controls, automotive security, home welfare, surveillance controls or even medical monitorization are the current research topics within the WSNs. All these applications have appeared under the commercial name of Zigbee, based on the IEEE 802.15.4 standard [1]. The current standard, agrees in terms of the scalability and low-cost requirements needed for the developing of the previously mentioned applications.

A remarkable feature on the current WSNs is that they share the same 2,4GHz ISM[1] band with previous standards such as the IEEE 802.11 and the IEEE 802.15.1, named commercially as Wi-Fi and Bluetooth respectively. In fact, the Wi-Fi has become a very successful technology and, consequently, has occupied almost globally the mentioned band. What is more, any emerging wireless communication to be settled on the 2,4GHz unlicensed band will have to struggle with domestic devices such as microwaves as well, since they provoke the apparition of high-powered interferences within the mentioned frequency range. Hence, this fact increases the evidence that the unlicensed spectrum is becoming overcrowded. Furthermore and following the philosophy of the current WSNs, the 802.15.4 because of the low-power communication feature will be always affected by any other communication or interference done by some of the previous devices. This topic has been subjected to an extensive research since the efficient use of the scarce available spectrum is one of the main concerns of the community, and several experimentation has been done to show the noticeable degradation on the performance in the 802.15.4 networks when there is a 802.11 network operating in an overlapping frequency bands [2, 3, 4].

By contrast, some of the published works regarding the utilization of the available spectrum under 3GHz, has shown the inefficient use of the current frequencies distribution,

[1]Industrial, Scientific and Medical (ISM) radio band, reserved globally for application purposes on the mentioned fields without license requirements.

specially on the unlicensed band where even being overcrowded the spectrum distribution is not homogeneusly allocated on the given band, Figure 2.1 on page 4 [5]. Thus, an optimal distribution on the unlicensed band will address the problem of the limited spectrum currently available. Furthermore, considering the time and spatial spectrum variations, the optimal distribution has to be dynamic and to achieve it, is when enter on scene the Cognitive Radio (CR) techniques [6]. By using CR devices, we will endow the network with intelligence in order to choose at a given time and place the best available spectrum range to perform. Applying these techniques to WSNs, it will permit to cover larger areas with less devices since the lower is the frequency, the higher powered is the signal at the receiver.

Nevertheless, the great challenge that implies endowing the current networks with CR devices, goes first by proposing a valid Dynamic Spectrum Access (DSA) and it is where the research is headed. Current CR algorithms and protocols carry out very specific DSA accordingly to a final application purpose. In fact, the current CR algorithms differ among them since they try to avoid different interference from different devices and, consequently, they select the best available frequency range following different criteria. Besides, these algorithms are usually limited by the hardware possibilities, specially on WSNs where devices are typically single-radio, low-powered and battery-feeded.

In this work, an Spectrum-efficient Architecture for Cognitive Wireless Sensor Networks is introduced. The proposed scheme is characterized by a typical WSN with a tree topology and completely scalable, where the root collects any data sent by the network's leafs for a generic purpose using a collection protocol. Our scheme will detect the presence of any other WSN in the coverage area and, subsequently, it will migrate the network to an interference-free area. Through real testing, the advantatges of CR-based networks are demonstrated just by using an ease to implement scheme that exploits the opportunities presented by the given data collection protocol. In other words, our scheme without altering the normal data collection cycle is able to feel the presence of other WSN and, consequently, it will migrate the entire network to a new frequency while all these operations are blind to the final user. It is energy-efficient as well, since no redundant scanning is carried out. As a conclusion, our proposal assures a better performance in case of an external WSN interference without adding neither complex operations nor long overheads.

The rest of the thesis is organized as follows: In Chapter 2, the CR techniques are illustrated and in Chapter 3, some features of the current WSNs are introduced. In Chapter 4, our design approach together with the related work and a possible application scenario are described. In Chapter 5, there is a detailed explanation of the final application code. In Chapter 6, the evaluation results are presented so as to, finally, in Chapter 7 the conclusions together with a few future work guidelines are discussed.

Chapter 2

Cognitive Radio

Currently, wireless communications are still assigned by a fixed spectrum policy, regulated either by local governmental agencies or international commissions, like the Federal Communications Commision (FCC) in the United States or the European Telecommunications Standards Institute (ETSI) in Europe. This fixed spectrum assignment is due to the impossibility of performing a dynamic assignment with the existing devices when the firsts wireless standards appeared. This dynamic performance requires an important computional capability that radio devices have been acquiring over time. Thus, actual devices are more able to act autonomously and a new spectrum assignment policy is likely to come. Through the National Telecommunications and Information Administration (NTIA) allocation chart [7], we can realise how exploited is the available spectrum for wireless communications. The main advantage of this spectrum allocation is the total control over the communications in the licensed bands, what in the meantime offers the Quality of Service (QoS) required for each communication. However, any allocation policy established a few years ago by those agencies or comissions could be considered outdated, since the current wireless scenario is totally different.

The lack of free *spectrum holes*[1] is not alarming by now, but in the overcrowded 2,4GHz ISM band. Hence, is on that unlicensed band where there is the major concurrency of wireless communications, such as Wi-Fi, Bluetooth, Zigbee and even the cordless phones and domestic microwaves saturate the mentioned band. Nevertheless, despite the already known wireless *pollution* in big city areas wireless communications only uses from 15% up to 85% of the available spectrum according to FFC [9]. In addition, more than 95% of this usage is below the 3GHz. Furthermore, most of the time large portions of the spectrum remain unused or they are only use sporadically due to default presets of the commercial wireless devices, as long as they all perform at the same band. On the other hand, this wireless activity is heavier in certain moments during the day depending on the finality of these communications. Then, it is obvious that the available spectrum varies in space

[1]Spectrum hole definition[8]: *"band of frequencies assigned to a primary user, but, at a particular time and specific geographic location, the band is not being utilized by that user"*

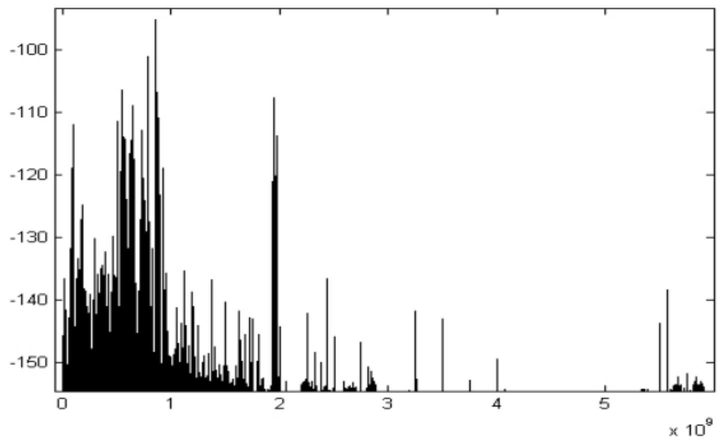


Figure 2.1: Measurement of 0-6GHz spectrum utilization

as well as in time. A measurement done in Berkeley [5] reinforces this statement, besides showing up that the 2,4GHz presents occupation but without being totally saturated on the whole range, Figure 2.1 on page 4. Thus, new techniques have appeared in order to make an effective utilization of the allocated, still unused, wireless spectrum. Those new techniques are all based on nodes[2] equipped with *cognitive radios* that provide flexibility to the network. As a general idea, this cognitive way of act identifies the *spectrum holes* on the electromagnetic spectrum depending on the space and time, and rebuilds the network on an interference-free area, taking into account the priority of the licensed users over the unlicensed. As a result, high-troughput networks can be achieved without major improvement on the existing protocols.

2.1 Background

The CR was formerly defined as *"the point in which wireless personal digital assistants (PDAs) and the related networks are sufficiently computationally intelligent about the radio resources and related computer-to-computer communications to: (a) detect user communications needs as a function of use context, and (b) to provide radio resources and wireless services most appropiate to those needs"* by Mitola [10]. Before, Mitola defined a new radio language called *Radio Knowledge Representation Language (RKRL)* in order to make cognitive radios achieve the flexibility of personal wireless services [11].

After this revolutionary concept the FCC appart from drawing up several guidelines in order to support the development of CR, they also provided their own definition: *"A Cognitive Radio is a radio that can change its transmitter parameters based on interaction with the environment in which it operates"* [9]. It can be sum up in two main ideas: cognitive

[2]From now on, device or node are used indistinctively.

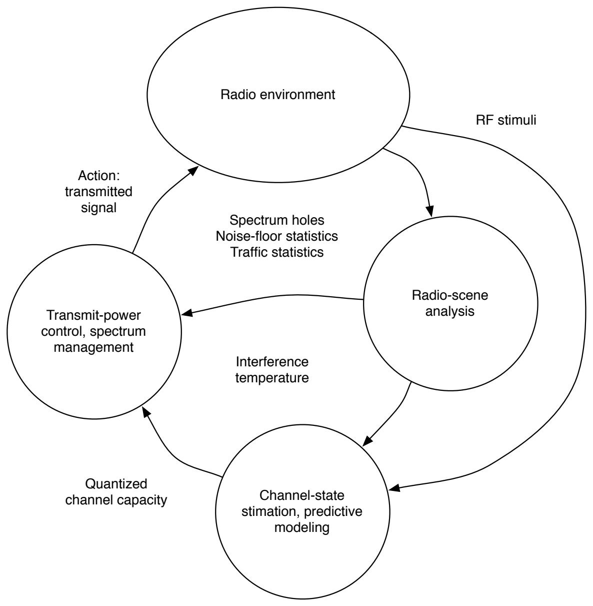

Radio environment
RF stimuli
Action: transmitted signal
Spectrum holes
Noise-floor statistics
Traffic statistics
Radio-scene analysis
Transmit-power control, spectrum management
Interference temperature
Quantized channel capacity
Channel-state stimation, predictive modeling

Figure 2.2: Cognitive cycle

capability and reconfigurability. The first one is related to the possibility of providing a good performance autonomously, without any human supervision. This capabability includes not only the monitoring tasks but the power of decision as well. The main tasks to perform this environment interaction are well depicted by the *cognitive cycle,* Figure 2.2 on page 5, with the following steps:

1. *Spectrum sensing:* scans all the available bands in order to get the complete picture of the current spectrum.
2. *Spectrum analysis:* lists all the *spectrum holes* with their features.
3. *Spectrum decision:* according with the decision policy predefined and the user needs, chooses the best channel[3].

Reconfigurability appears when a decision is taken, CRs must coordinate with the entire network where they are involved to scroll all the current communications to the new channel. This action should be hidden to the current users of the network and should involve operating parameters such as the operating frequency, the modulation, the transmission power and the communication technology. The possibility of variating those parameters at run-time is required regarding to the likely variation of the environment.

[3]From now on, frequency or channel are used indistinctively.

After the great interest provoked by the guidelines' publication, some developments have been headed to separate ways. Thus, several literature about cognitive radios can be found. As an example, a latter definition by Haykin [12]: *"Cognitive radio is an intelligent wireless communication system that is aware of its surrounding environment (i.e. outside world), and uses the methodology of understanding-by-building to learn from the environment and adapts its internal states to statistical variations in the incoming RF stimuli by making corresponding changes in certain operating parameters in real-time, with two primary objectives in mind: (1) highly realiable communications whenever and wherever needed; (2) efficient utilization of the radio spectrum".* However, they all finally converge in the idea of taking advantage of the unused spectrum when possible and without interfering any licensed user communication.

2.2 Cognitive Radio Networks

Also known as Dynamic Spectrum Access Networks (DSANs), are a DARPAs[4] approach that aims to implement this new DSA. They will provide high bandwith to any wireless communications through one of the available network architectures at that very spot and at that very moment. Through opportunistic access within either licensed or unlicensed bands, respecting the privileges of the licensed networks users and without interfering the already existing communications. However, the key fact to reach this level of efficiency is to endow all the wireless devices with CRs.

CR techniques will permit to operate in the best available channel according to an environmental survey done by the device. Concretely, these techniques will enable the devices to detect the *spectrum holes* surrounding them, taking into account the presence of licensed users if they operate on a licensed band. Subsequently, they will select the best available channel and they will coordinate with the rest of the users. Eventually, they would also vacate the current channel in case of a licensed user presence. In [6], a formal division of these tasks is presented as: collecting information of the current state of the available spectrum *(Spectum sensing)*, synthesizing the environmental information to select the best frequency available *(Spectrum management),* coordinating the network to hold the existing communications while migrating to the new channel *(Spectrum mobility)* and respecting the network privileges to determinated users *(Spectrum sharing).* In addition, if we consider the data transmission, the flowchart of a generic cognitive radio device is presented in Figure 2.3 on page 7.

[4]The Defense Advanced Research Projects Agency (DARPA) is an agency of the US Defense Dpt. responsible for the the research and development of new technlogies with military applications.

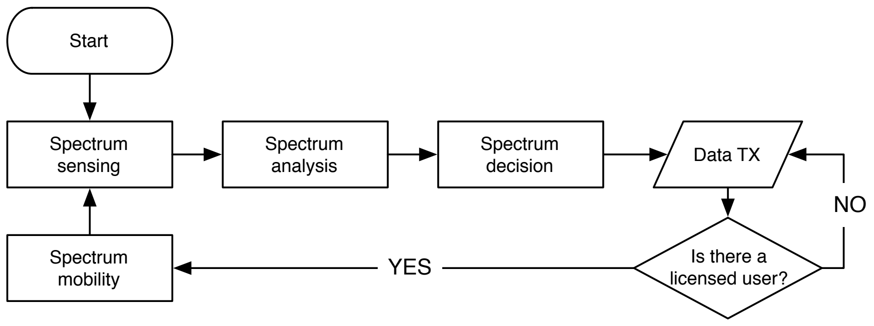


Figure 2.3: Cognitive radio flowchart

2.2.1 Spectrum sensing

The final aim is to seek *spectrum holes* and the most efficient way to detect them is to identify primary users that are receiving data from secondary users within our communication range. This detection takes place in the lower layer of the network protocol and it will be based on the signal level that the user receives while performing on a given channel. Eventually, *spectrum sensing* techniques can be classified as transmitter detection, cooperative detection and interference-based detection. Both transmitter and cooperative detection are based on detecting if the present channel is available locally or in the whole network respectively, whereas interference-based plays with the signal level opportunities.

2.2.1.1 Transmitter detection

Based on detecting weak signals locally from primary users and following the hypothesis below to establish if there is either vacancy or occupancy of the current channel:

$$x(t) = \begin{cases} n(t) & H_0, \\ As(t) + n(t) & H_1 \end{cases}$$

Where the $n(t)$ is the noise floor, A the amplitude and $s(t)$ the signal received at the secondary user defined as $x(t)$. Then the secondary user, will follow this simple rule to state if there is vacancy H_0 or occupancy H_1, and act consequently. Hence, there is no more traffic added to the network as long as the hypothesis is considered only in the user.

2.2.1.2 Cooperative detection

It can be considered an extension of the transmitter detection, as long as it is based on the same channel occupancy hypothesis. However, what differs is that in cooperative detection there is the figure of a main user that gathers all the secondary users' local information. Hence, the main user has a global vision of the available spectrum and it can discard those channels that are occupied on each different secondary users' environment. Consequently, it may be discarded a channel that interferes uniquely a certain node. Despite being more accurate, cooperative detection adds more traffic operations to the network that in certain cases would be counterproductive regarding to the final aim of cognitive networks.

2.2.1.3 Interference-based detection

The FCC [9] makes reference to a new procedure for interference detection, based on the *interference temperature,* and this detection takes place in the radio station. Primary users will not have any restriction regarding to the signal level while communicating, however, secondary users will be allowed to operate on a given *spectrum hole,* but just when their signal level is below a certain threshold. This admission level will be established after taking into account all the cumulative RF existing on the current channel from multiple transmissions, so that the maximum level will be an acceptable level for data transmissions of secondary users.

2.2.2 Spectrum management

While *spectrum sensing,* has been already explained in the last section, now we are going to take into account the remaining two states of the *cognitive cycle*, Figure 2.2 on page 5: the *spectrum analysis* and the *spectrum decision.* Both tasks are close among them, since they operate in an upper layer unlike the *spectrum sensing* more related to the PHY layer. Therefore, the spectrum management will collect all the information, it will process all the information, and finally it will compare all the information so that it will be able to choose the best channel to perform the communication.

2.2.2.1 Spectrum analysis

Considering how heterogeneus could be the *spectrum holes* detected, several parameters that define the network are required to unify the decision criteria. Some of them would be: interference level, path loss, wireless link errors, link layer delay or holding time. All gathered we could extract the channel capacity as one of the most important parameters at the time of selecting the best available channel. Formerly, SNR[5] was used to estimate the capacity, however, that estimation was local, and that follows the same idea of the non-cooperative detection, that a local vacancy of the current channel is meaningless at the time to assure there is an *spectrum hole.* Hence, many primary users could be detrimented. Therefore, channel vacancy makes a given channel likely to be selected, though, it is interesting to take into account the previous parameters as well, in order to select

[5]Signal to Noise Ratio (SNR), mesure to compare the desired signal to the level of background noise.

actually the best available channel.

2.2.2.2 Spectrum decision

Subsequently, once all *spectrum holes* have been characterized it is time to take the decision. Despite making clear which channel is the best available, user requirements must be considered as well. That means, once we know the QoS needed, it is on that point that the *spectrum decision* will weight up the cost of the channel switching in order to achieve better channel conditions, instead of working on the current channel with worse conditions.

2.2.3 Spectrum mobility

The process when a secondary user switches the best frequency of operation should be smooth and rapid. Nevertheless, we have to keep in mind that each time that an user changes its frequency of operation, subsequently there are other parameters to be reset to the new situation, and that changes are likely to degrade the performance. In addition, the network coordination will take some time as well, as long as the entire network has to converge into the new situation. Thus, the spectrum management would have to consider the degradation involved in a channel switching in exchange of operating on a better channel, and it will usually be related to the amount of information to be sent.

2.2.4 Spectrum sharing

Following the general idea of sharing the available spectrum while unused, after a data transmission the secondary user should abandon the used *spectrum hole,* just in case any other incoming communication by other user requires the same frequency. Besides, as pointed out earlier, if a primary user requires its licensed band to perform, that one should be cleared of any current communication by secondary users.

2.3 Future challenges

The performance of CR networks directly depend on the properties of the spectrum band in use. This direct relationship necessitates a cross-layer design in the entire CR networks protocol stack. Therefore, several open challenges for the CR networks, regarding to different features of these kind of networks, have entailed the apparition of several literature trying to solve these challenges one by one. However, they are not often connected as a consequence of neither having standardization guidelines nor a main work to follow yet. The main issues to address regarding the CR are explained next [13].

Despite having a number of cross-layer design proposals, there is not clear which one presents the most robust performance. This is a consequence of the different scenarios where the development takes place and the different chosen metrics to evaluate the proposal. Then, it is clear that a harmonization of these scenarios and these metrics is required to head towards a main proposal. As long as new proposals enter on scene, it is important to endow these proposals with the capability of coexistence among them since they will have to coexist without detriment the rest of schemes until the standardization arrives. Furthermore, it is necessary to think about the interaction between the already existing schemes due to the fact that is mandatory to carry out the cognitive cycle and, subsequently, it may provide the user the best available frequency of operation in the given area. Therefore, some standard signaling between the different technologies are needed.

Another key challenge to address concerns the provision of an optimal operating to the final user. To achieve it, the idea is to make the protocol stack responsible to the variations in the underlying network conditions. Nevertheless, this solution needs first the establishment of the network conditions under which the proposed design would improve the performance. Secondly, the implementation of efficient mechanisms on the stack to make a timely and accurate assessment of the state of the network and, in the meantime, the corresponding overheads need also to be taken into account. Moreover, the interfaces need to be standardized to the current protocol designs since several options are available to the developers. This process usually takes place during the process of drafting technical standards, normally by the IEEE.

Chapter 3

Wireless Sensor Networks

WSNs are defined as wireless networks of sensor-powered devices interconnected among them that cooperatively describe their environment through their sensor measures. The basic device can be considered as a small computer and it is formed by a microcontroller, several sensors, a radio transceptor and batteries Figure 3.5 on page 18. Within these network features, the most remarkable ones could be: the reconfigurability and the energy-efficient in both hardware and communication protocols. The reconfigurability, already seen as a requirement for the CR devices, regards to the versatility and the facility to set up the network. On the other hand, low consumption is mandatory as long as these devices must work autonomously, for long periods without any maintenance.

In the present days, the current development is headed towards the *energy harvesting,* where the energy derived from environmental resources such as solar power, kinetic energy or wind energy can be captured and stored. Furthermore, low consumption of the devices is also being improved. Therefore, all gathered we will have at our disposal sensor devices with upper autonomies. Consequently, such autonomy, which is one of the main cons of the WSNs, is becoming less problematic [14].

3.1 Background

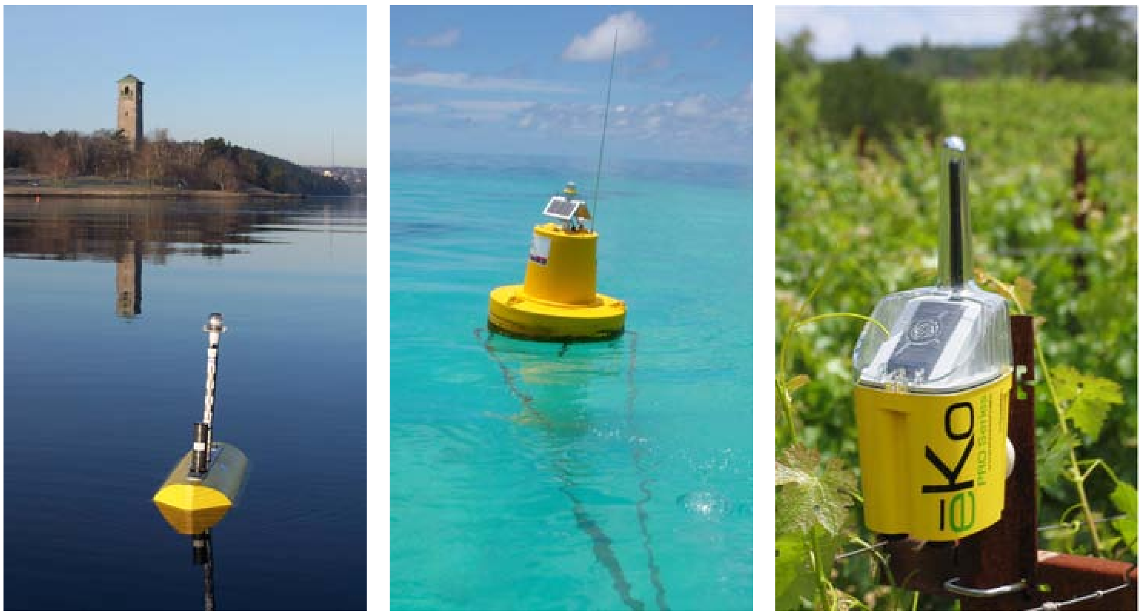
It was in the late nineties when the firsts WSNs were launched with civil applications finalities. Nowadays, companies as Crossbow[1] have developed reduced motes with great autonomy just being supplied by batteries and, as a result, current WSNs are mostly constituted by Crossbow products. Eventually, according with the recent achievements on integration and miniaturization we can assume that in a few years we will be able to sensorize and, consequently, control several unthinkable situations nowdays. In fact, Dust Inc. together with the members of the Smart Dust project [15], have developed pea-sized sensor devices that can be smartly situated almost everywhere.

[1]Crossbow Technology, Inc, commercializes the IRIS platform devices among others. This company, with headquarters in California, US, has taken investment from Cisco and Intel as main investors.

3.2 Applications

Considering all the existing Ad-Hoc wireless networks and taking into account the current development of either new or existing WSNs applications, they have been succesfully accepted and, consequently, WSNs are fully integrated within the following fields [16]:

- *High-security areas:* restringed areas such as government buildings, nuclear plants or airports, where typical monitorization can not be placed, whereas WSNs can be discreetly placed almost everywhere. It was one of the firsts application fields of the current WSNs.
- *Environmental sensors:* monitoring vast areas such as great forests or oceanic areas could be impossible without WSNs. Moreover, the numerous sensors provide information of many variables that can help to prevent certain natural disasters. All of that while minimizing the environmental impact of the human activities, Figure 3.1 on page 13.
- *Industrial sensors:* some of the quality controls within the manufacturing industry are being done by WSNs since they can be placed easily and they accomplish their tasks more efficiently than the traditional quality controls.
- *Automotive industry:* as a complement for the control cameras, WSNs can add information of the situation of the car either locally by built-in sensors around the car, or globally by getting live information of the state of our route, warning us from traffic incidents.
- *Medicine:* due to the easy integration in small devices, WSNs will improve the control of patients' vital signs and act consequently if there is any variation on them. It can be considered one of the most promising fields of actuation as long as the devices become smaller.
- *Domotics:* affordable technology for the next comming homes, WSNs will automatically act according to some user presets and taking into account the environmental issues as time, temperature, humidity or even the energy consumption. In addition, this technology is easy to deploy in the already existing homes, without requiring a great construction work because of the its wireless feature. Furthermore, this technology is likely to become cheaper than the wired one.



(a) Wireless sensor monitoring river environmental parameters

(b) Wireless sensor monitoring open-sea environmental parameters

(c) Wireless sensor monitoring weather conditions within a forest environment

Figure 3.1: Current environmental WSN applications

3.3 Sensor networks features

Regarding to the applications and the way they were concieved, WSNs have a few unique characteristics that can be sum up as follows [14]:

Regarding to the energy consumption:

- *Power limitation:* both hardware and software must be restrictive with the consumption of the processing and limitate the processes if they are not required. Hence, we are facing one of the most remarkables features on WSNs: the autonomy. Subsequently, they are able to operate in isolated areas far from energy sources appart from the natural ones that they could harve.
- *Low range coverage:* as a consequence of the power limitation, their transceivers are not able to reach far distances.
- *Multihop protocols:* following the idea of the low consumption and the low range transmissions, WSNs require multihop protocols to cover long distances.

Regarding to the isolation:

- *Ability to withstand with adverse weather conditions:* Despite being fully protected, the devices should be as sturdy as possible since they may face sometimes hard situations that can compromise their performance.

- *Malfunction management:* in case of system errors, WSNs devices should solve them by themselves since sometimes they may be placed in isolated areas far from any human supervision.

Regarding to the environment:

- *Devices mobility:* according to the application some devices may not be fixed to a certain position, therefore, the operability of the network can be interrupted. As a result, WSNs present a dynamic network topology, being able to change from one topology to another accordingly to the node mobility.

- *Channel variations:* as stated before, the radio channel is likely to suffer variations depending on the space and time, so that some devices may become incommunicate along a certain period. Then, and taking into account the possibility of a device malfunction as well, WSNs are able to reconfigure the routes in order to keep the network operative.

Regarding to the product acquisition:

- *Cost of the device:* as long as several devices are needed to dispose a common WSN, the comercialization price of each device has to be affordable taking into account the final purpose as well. In addition, we have to consider the possibility of either loses or damages of our devices. Moreover, as other electronic devices the unitary price while producing large quantity of them is decreased.

Regarding to the network infrastructure:

- *Absence of predefined structure:* as a default there is not any main entity that centralizes all the data, each node can act either as a routing node or as a terminal node. Nevertheless, there is usually the figure of the *root node* while talking about the current protocols for the WSNs. Those protocols habitually enroute the packets *economically* since they generate routes on-demand or creating *sink networks* where all packets within the network reach their final destination through a main device. However, broadcasting techniques are also available, although they are not very common.

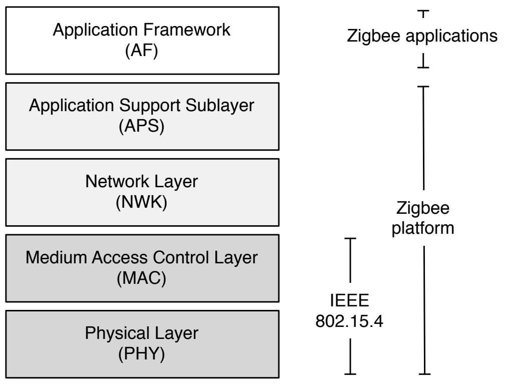


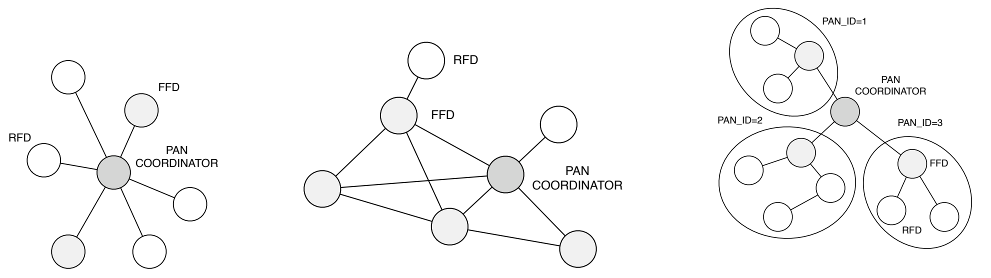
Figure 3.2: The overall Zigbee protocol stack

3.4 Standards

Despite the fact that several protocols have been developed, only a few have been succesful and have actually become the standards for the WSNs. One of these standards is the Zigbee protocol that have specifically become the main standard for the WSN communications. It was born to address the lack of an standard able to operate along with the low consumption and the low data transmissions, by a conglomeration of hundreds of enterprises that established the *Zigbee Alliance* to define the current standard. Their specification was conceived to complement the already existing IEEE 802.15.4 but in the upper layers, Figure 3.2 on page 15. Hence, these networks were renamed as Low Rate - Wireless Personal Area Network (LR-WPAN), that provide low range coverage, low data transmissions and low consumption. WirelessHART is the other famous standard that was specifically designed for industrial applications and it is all englobed in the HART protocol suite approved by the HART Communication Foundation. Later appeared the 6LoWPAN aimed at making an adaption of the IPv6 communications over Low power WPANs, and the ISA100 that makes completion to the 6LoWPAN for industrial applications but it is not yet finished. Although all of them were designed for different applications, they all share they same underlying radio standard the: 802.15.4 -2006 [1, 14].

3.4.1 IEEE 802.15.4

The main objectives of a LR-WPAN are ease of installation, reliable data transfer, short-range operation, extremely low cost, and a reasonable battery life. Moreover, a LR-WPAN device comprises a PHY, which contains the radio frequency (RF) transceiver along with its low-level control mechanism, and a MAC sublayer that provides access to the physical channel for all types of transfer. Both layers together with the available topologies on the IEEE 802.15.4 are presented in the following sections.



(a) Star topology (b) Peer-to-peer topology (c) Cluster-tree topology

Figure 3.3: A few examples of 802.15.4 existing topologies

3.4.1.1 Topology

The communication among two or more nodes requires that at least one of them acquires a coordinator role to control the network, whilst the rest can be considered only as regular nodes with the commitment of sending the information required for the correct functioning of the network. Thus, the IEEE 802.15.4 [1] defines two different types of nodes:

- *Full Function Device (FFD):* is able to receive and process the IEEE 802.15.4 messages. In addition it can act either as a router or as a coordinator as well as provide the application level required interaction for an user interface.
- *Reduced Function Device (RFD):* has reduced capability and functionality since it has to operate as a sensor, just sending the specific information and without processing any of them. As a consequence, RFDs need less energy than FFDs and, consequently, they have more autonomy.

Then, according to their characteristics we have to take into account that a RFD can only communicate with a FFD. Subsequently and considering the final purpose of the network, there are different topologies, Figure 3.3 on page 16 from [1]:

- *Star topology:* all the nodes within the same WPAN are coordinated by the same FFD, coordinator's main duty is to manage the medium access.
- Peer-to-peer topology: mostly all nodes are FFDs, however, a coordinator node and some RFDs can be placed without modifying the topology. The main difference with the star resides on the added redundancy on the network. Besides, more complex topologies can be achived from peer-to-peer such as mesh networks or cluster-tree networks.

PHY (MHz)	**Frequency (MHz)**	**Chip rate (Kchip/s)**	**Spreading Modulation**	**Bit Rate (Kb/s)**	**Symbol Rate (Ksymbol/s)**	**Data Modulation**	**Field of influence**
868-915	868-868,6	300	BPSK	20	20	BPSK	Europe
868-915	902-928	600	BPSK	40	40	BPSK	USA
2450	2400-2483,5	2000	O-QPSK	250	62,5	16-ary orto	Worldwide

Table 3.1: Frequency bands and data rates for the IEEE 802.15.4 standard

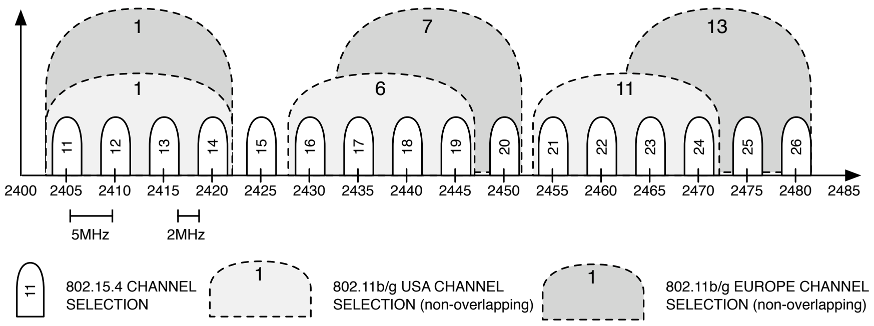

1
7
13
1
6
11
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
5MHz
2MHz
802.15.4 CHANNEL SELECTION
802.11b/g USA CHANNEL SELECTION (non-overlapping)
802.11b/g EUROPE CHANNEL SELECTION (non-overlapping)

Figure 3.4: The IEEE 802.15.4 and IEEE 802.11 PHY channels disposition

3.4.1.2 PHY layer

The lowest layer of the stack is already divided in two sublayers: PHY data and PHY management. Globally it is based on the Direct Sequence Spread Spectrum (DSSS) that gives the possibility of coexistence with the other wireless standards on unlicensed bands. There are two separate bands, one of them with a spectrum range from 868MHz to 915MHz which posses a great sensitivity and coverage but, conversely, has a rather slow bit rate. The other band, totally opposite to the former one, is on the ISM 2,4GHz band as we can see in Table 3.1 on page 17, with higher rate data transmission and less coverage. They are classified taking into account their field of influence. In the 2,4GHz band, the spectrum usage is more efficient, it is separate in 16 channels of 2MHz and it includes a channel separation of 5MHz to avoid co-channel interference [1]. In Figure 3.4 on page 17, the channel disposition is presented comparing the IEEE 802.15.4 PHY channels disposition with the three non-overlapping 802.11 channels on each continent and, as we can see there is overlapping among the two IEEE standards either ways.

In addition, the PHY layer is responsible for the following tasks: in first place the activation or deactivation of the radio transceiver, secondly the Energy Detection (ED) within the current channel, as well as the Link Quality (LQI) for received packets, the Clear Channel Assesment (CCA), the channel selection, and finally the data transmission and reception.

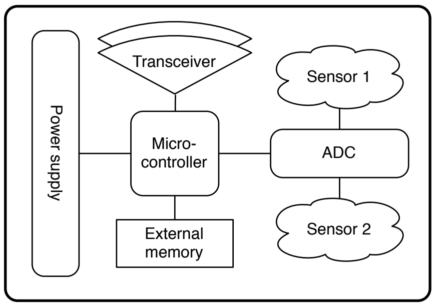


Figure 3.5: Sensor node architecture

3.4.1.3 MAC layer

The Medium Access Control (MAC) sublayer manages all access to the physical radio channel and is responsible for the following tasks [1]:

- Generating network beacons if the device is a coordinator
- Synchronizing to network beacons
- Supporting device security
- Employing the Carrier Sense Medium Access - Collision Avoidance (CSMA-CA) mechanism for the channel access
- Granting QoS by means of Granted Time Slot (GTS)
- Providing a reliable link between peer MAC entities

3.5 Comercial products

A sensor node, as introduced before, is the key part on the WSNs. They are capable of sensing, processing with their own microcontroller and either storage a small amount of information on their own external memory, or interchange this information with other sensor nodes through the radio transceiver, Figure 3.5 on page 18.

The starting point was the Smartdust project [15] in 1998. Afterwards, that first iniciative was bifurcated into several projects including the major research centers as Berkeley Wireless EmBedded Systems, the birthplace of several of the most used motes[2]. All the Berkeley mote's evolution, gathered with other research groups, is presented in Table 3.2 on page 19.

[2]A sinonym for a sensor node, commonly used in the United States.

Mote	WeC	rené	dot	mica	mica2	mica2dot	iris
Released	1999	2000	2001	2002	2003	2003	2007
Processor	4 MHz				7 MHz	4 MHz	16MHz
Flash (code, kB)	8		16	128			
RAM (kB)	0.5		1	4			8
Data rate (kbps)	10			40	38.4		250
Radio type	RFM				ChipCon		Atmel
Minimum operation (V)	2.7						
Expandable	no	yes	no	yes	yes	yes	yes

Table 3.2: Hardware platform evolution

3.5.1 IRIS

The last column of the Table 3.2 on page 19 refers to the IRIS platform from Crossbow and the evolution is noticeable. These motes are the ones used on this work, and regarding to the fabricant specifications [17], we can highlight some of several improvements done on this recent platform:

- Up to three times improved radio range and twice the program memory over previous MICA Motes
- Outdoor line-of-sight tests have yeilded ranges as far as 500 meters between nodes without amplification
- 250 kbps data rate

Therefore, it is obvious that a huge work has been done on this platform as long as the technical features have increased in terms of range and data rate. Finally, in order to load the applications programmed for this work, a MIB520CA mote board has been used. This board provides a serial/USB interface for both programming and data communications.

3.6 Operating systems

Presumably WSNs operating systems (OS) are less complex than any other one developed for any other purpose, because of the special features that WSNs have. Furthermore, the hardware limitations reinforce this theory. Other characteristics that usually are related to WSNs are the isolation and the autonomous operation, that make unnecessary having user interfaces. Consequently, we are facing extremaly basic OS. Several platforms have appeared in last years [18], some of them such as SOS, MANTIS or Contiki have

```
+-------------------+---------+------------------------------+--------------+
| 802.15.4 Header   | AM type |            data              | 802.15.4 CRC |
+-------------------+---------+------------------------------+--------------+
```

Figure 3.6: The TinyOS 802.15.4 frame format

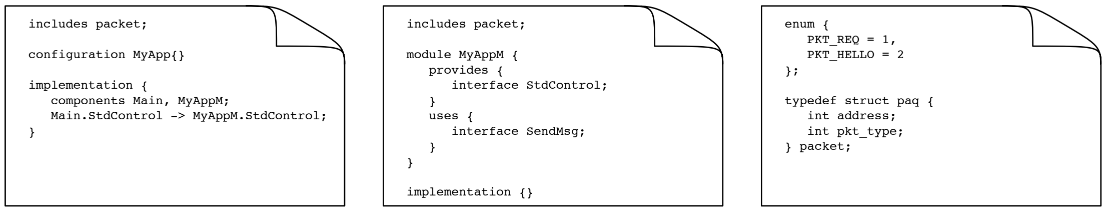
been described under C language. On the other hand we find TinyOS, an open-source OS, described in network embedded systems C (nesC) and especially designed for the WSNs. Unlike other OS, TinyOS is event-driven programming model instead of multithreading. As is the case of this work, the fact of being open-source is what brings the major application development to this platform.

3.6.1 TinyOS

Nodes on WSNs need to be robust and execute several operations at once, but have limited storage. In order to abstract upper layers from the PHY and MAC layers provided by 802.15.4 standards, TinyOS creates an intermediate layer called Active Message (AM), the resulting frame in Figure 3.6 on page 20 [19]. Besides, TinyOS has an architecture based on components that reduces to the minimum de code size. TinyOS applications are built out of software components, some of them present hardware abstractions. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage. In order to facilitate modularity each component declares the commands and events that signals. Commands are non-blocking requests made to lower level components and that is the reason why event-driven concurrency model suits perfectly. Furthermore, it is optimized in terms of energy-saving and memory utilization [20].

Appart from the event-driven programming, TinyOS provides an efficient framework for modularity. Regarding to the program execution, it is divided into hardware events and tasks. The former are interrupts caused by a timer, sensor or communication device and they can interrupt tasks that are currently executing. On the other hand, tasks are a form of deferred procedure call that allow a hardware event or task to postpone processing. 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 though they can be preempted by events. Tasks can call lower level commands, signal higher level events, and schedule other tasks within a component. Tasks are non-preemptive and run in FIFO[3] order. When the task queue is empty, the system stands by until the next interrupt, that contributes to the energy-saving. Tasks are atomic with respect to each other [21].

[3]First In First Out (FIFO), that describes how to manage a processing queue. Thus, the oldest process in the system will be the first to be handled.



```
includes packet;

configuration MyApp{}

implementation {
   components Main, MyAppM;
   Main.StdControl -> MyAppM.StdControl;
}
```

(a) MyApp.nc

```
includes packet;

module MyAppM {
   provides {
      interface StdControl;
   }
   uses {
      interface SendMsg;
   }
}

implementation {}
```

(b) MyAppM.nc

```
enum {
   PKT_REQ = 1,
   PKT_HELLO = 2
};

typedef struct paq {
   int address;
   int pkt_type;
} packet;
```

(c) packet.h

Figure 3.7: Structure files for the TinyOS application definition

Regarding to the motes, there is no preinstalled operating system on them. Therefore, while compiling an application the TinyOS is added to it and, subsequently, every time that the mote is flashed the application is loaded all together with the operating system. Nevertheless, the overall size of the final file to be upload is far to be heavy.

3.7 Programming languages

Along with the operating systems' development new programming languages have appeared. Mostly, they all are dialects of C language but there is C-based OS as well like SOS, MANTIS and Contiki, that were introduced previously. However, as TinyOS has been the most spread OS for the WSNs, its programming language has become the most used language for the current development on the WSNs, the nesC language.

3.7.1 nesC

nesC is an extension of the C language that, in order to accomplish the requirements of simplicity and optimization, is a component-based language. To define these components nesC uses modules and configurations. Inside each module there is the code for a single component, whereas the configuration is needed for the *wiring*. A final application can be formed by several configurations if and only if there is a defined top-level configuration that *wires* the whole application modules and configurations. A module implements one or more interfaces. In the meantime, interfaces describe the commands and events provided by a component and they are bidirectional. Therefore, configurations will wire modules using a given interface to a component providing an implementation of this interface [22].

It is assumed an execution model of asynchronous handlers and tasks that have to be executed totally. The scheduler for those tasks can execute them in any order, normally FIFO in TinyOS, but always until completion. Since tasks are not preempted but executed until completion, they are atomic between each one of the others tasks. However, they are not atomic with respect to interrupt handlers. Thus, nesC programs are likely to race conditions and they, if existing, are reported at compile-time. They can be avoided either by using atomic sections to turn off hardware interrupts on certain block of code, or by converting the conflicting code into atomic tasks [22].

A TinyOS application implemented in nesC needs a few diferent files to compile, using gcc, the global C-file to flash the mote. Typically these files are formed by a module file, a configuration file, a header file and a Makefile with the compiling directions, Figure 3.7 on page 21. Through the configuration file, it will be included any module of TinyOS if and only if the module is actually used on the final application. In addition, having a single file makes possible a whole-program analysis and, consequently, an efficient optimization.

Chapter 4

Design approach

The overcrowded ISM 2,4GHz band, is shared by the IEEE 802.11 networks, the IEEE 802.15.4 networks, Bluetooth devices and even domestic microwaves and cordless phones, are likely to make interferences to them like in Figure 4.1 on page 24 [23]. All together makes this band undoubtedly one of the most highly occupied bands on the scarce available spectrum. Focusing on the subject of this work, is on the IEEE 802.15.4 networks where is taking place the major development currently since they are a brand new technology. Furthermore, this notorious progress on this technology, helps on the envision that in a few years these devices will multiply their current population. However, these small networks are on a bad position in comparison with other communication networks as long as they are low power emissions. Thus, they are more able to be detrimented by the presence of high-powered IEEE 802.11 signals, since the power transmissions are 0 dBm for the WPAN devices and 16 dBm for the Wireless Local Area Network (WLAN) devices, whereas Bluetooth varies from 0 dBm to 20 dBm. As long as the IEEE 802.11 has been settled down as a one of the most spreaded wireless communication technologies, seems like it is on the WSNs where the functional changes regarding to dynamic spectrum assignment should be taken. Then, the global idea of CRs applied to the IEEE 802.15.4 communications, looks like a great improvement on the current WSNs.

In practice, along with the integration progress on the sensor nodes, a software-level improvement regarding to the adaptive spectrum assignment seems to be necessary. Regarding to the current work on the subject, the several methods proposed in order to upgrade WSNs into cognitive WSNs have been bifurcated considering the nature of the interference, the hardware possibilities and the network features such as topology and routing protocol. On this work, an Spectrum-efficient Architecture for Cognitive WSNs is introduced in the following sections, all along with the state-of-art on cognitive WSNs protocols, the channel interference research done in previous works, a suggested application scenario and a low level description of the scheme.

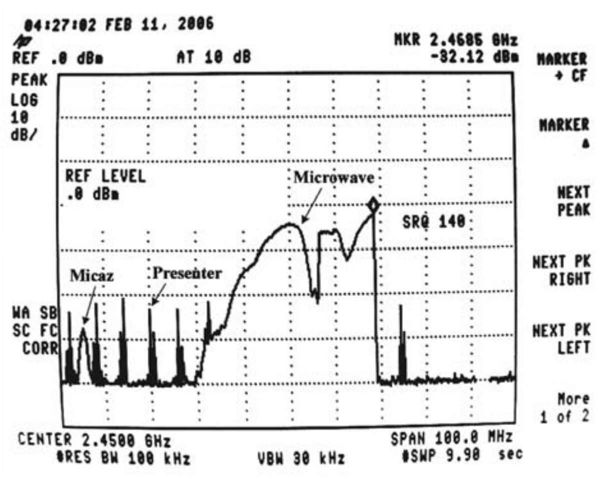


Figure 4.1: Interference due to domestic devices while a MicaZ is transmitting

4.1 Related work

There are several related proposals on cognitive networks and they all can be classified depending on the interference to be avoided or according to the hardware possibilities. In the following protocols, the authors use either multi-radio devices or use control messages to set the channel for the network performance. A multi-radio scheme is presented in [24], where the authors use nodes equipped with more than a single transceiver to separate the control packets from the data packets. Besides, in [25] and following the idea of tree networks, the authors provide a solution to complement a generic data collection application in WSNs. The main idea is to divide the network into subtrees all rooted to the base station, where each subtree will be allocated on a different channel and, consequently, it does not need time for synchronization. This feature is assuming that the base station is equipped with as much transceivers as subtrees rooted to it, however, usually WSN nodes are equipped with a single-radio. On the other hand, in [26, 27, 28] the authors provide different solutions based on a previous channel negotiation. Nevertheless, again these schemes are not efficient in terms of throughput since usually in WSNs the data packets are limited due to the small bandwidth available and the fact of adding more packets for the negotiation is not probably the best solution.

Focusing on WSNs, in [29, 30, 31, 32] three multi-channel MAC protocols conceived for WSNs are introduced. After their tests, all of them coincide to present a noticeable improvement while performing. They all basically try to assign dynamically the channel in order to avoid potential interferences of 802.11 networks. They address the interference avoidance by scrolling the affected nodes into a new channel, constituting a two or more levels network according with the affected areas. The border nodes of the affected regions need to keep updated the neighbor's table with the actual frequencies of each one and they

	802.11a/b/g	**Bluetooth Class 1**	**Bluetooth Class 2**	**Bluetooth Class 3**	**Microwaves**	**802.15.4**
Output	0-20 dBm	20 dBm	4 dBm	0 dBm	16-33 dBm	0 dBm

Table 4.1: Output power comparison on the 2,4GHz ISM band

will have forward messages by switching their frequency as well. Therefore, regarding to multi-hop networks, a very frequent channel switching can cause considerable packet losses and, to address this potential problem, they use negotiation and scheduling schemes to coordinate the channel migration. They face some of the typical problems on the WSN as performing in areas with high number of networks. As a result, a high number of orthogonal channels are required, an accurate time synchronization among nodes has to be set, the channel switching delay has to be taken into account, and more resources at the nodes, due to the high number of functionalities added, are required.

Finally, in [33] the authors simulated a WSN with a sink topology affected on a given area by 802.11 interference. This interference is assumed to vary over time, space and frequency. Their algorithm works by using a *reward* function which indicates if a channel switching is advisable, and thus, the protocol will migrate the whole network to the best channel available given by the mentioned function. Therefore, this scheme is presumably more energy-efficient since the migration takes place only if the new conditions are better than the current ones even with the presence of an interfering device.

4.2 Channel interference

The motivation of this work and the related came up, among other reasons, from several researches done previously about channel interference on the 2,4GHz ISM band. As an introductory fact a comparison is presented in Table 4.1 on page 25, where the typical output powers of the different devices that coexist on the ISM band are listed. Previously mentioned, the WSNs are in clear disadvantage in comparison with the rest of the wireless technologies, [17, 34, 35]. Even Crossbow, as one of the most important manufacturers, has published an application note regarding to the RF interference between WLAN networks and Zigbee networks [2]. They both are affected by the existing problem of not taking a proper care of the software configuration. Thus, with their network configuration and utilizing high-powered WLAN cards, they reach up to 20% of packet loss in the WSN performing in the same spot. However, in [3] the authors state that even taking into account the DSSS to reduce interference, a WLAN device can provoke up to 58% of packet loss in a WSN. Eventually, in presence of an IEEE 802.11b interference, the worst one appart from domestic devices, the performance degradation is noticeable in [4], where almost 92% of the IEEE 802.15.4 frames were lost.

For the already existing IEEE 802.11g/n drafts, which may be assumed as the brand new standards for WLAN in the next future, the authors of [36] conclude through their experimentation, that is hard to guarantee the quality of the present WSNs while a IEEE 802.11n is performing with a middle or high traffic load next to it. Therefore and again, if the channel assignment is not done in a properly manner, the WSNs will be detrimented to their packet received ratio. Furthermore, they also appeal to a dynamic threshold setting for the IEEE 802.15.4 devices since high-powered transmissions can increase considerably the current environmental interference level. Thus, the highly sensitive CCA would reach such an adaptive sensitivity that consequently would improve the performance of the both overlapping and non-overlapping IEEE 802.15.4 channels.

Concerning to domestic devices such as microwaves in [23], as shown in Figure 4.1 on page 24, the authors obtained a packet loss ratio from 19% to 54% on the Crossbow's MICAz receiver operating in the 2.45GHz channel when the microwave was on, whereas the packet loss was fixed to 0% when the microwave was off. Their experimental observation demonstrates that the already implemented DSSS is not enough to solve the crowded spectrum issue.

Regarding to the interference among WSNs, there are several studies applied to depict this interference. Typically, if multiple transmissions are taking place on a given spot, they are likely to be interfered with them but when they use orhogonal channels. Therefore, it is commonly known that adjacent channels to the present transmission channel provoke interference due to the existing overlapping in the IEEE 802.15.4 standard. Thus, in [35] the interference due to other WSNs working in adjacent channels is studied and, after experimentation, it can be observed that when several transmissions perform on the adjacent channel their co-channel interference appears.

Finally, again in [35] a relation between the interference at the receiver and the distance with the interfering jammer is shown. This interference level is aproximated as the loss rate at the receiver. Eventually, the authors establish that there is a high correlation between channel spacing and spatial spacing. Hence, the researching line headed to cognitive radios able to recognise at a given time and on a given spot their environment and, subsequently, perform on the less harmful frequency available, is more than justified.

4.3 Suggested scenario

Focusing on the main matter of this work, the final aim is to provide an applicable scheme for the current WSNs. That scheme will have to act cognitively while performing since they are likely to be deployed on unsettled environments. The generic network is formed by a root node, that coordinates the rest of the network, and several client nodes under the directions of the root node. The network topology will be a sink network or tree-based network, where the client nodes or leafs capture the information from their sensors and they send it, periodically or everytime there is a variation, to the sink or the

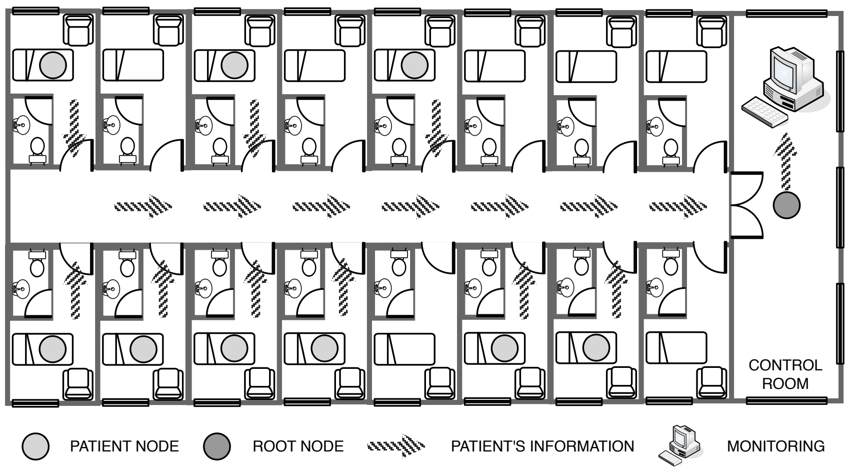


Figure 4.2: Hospital floorplant diagram distribution for a possible WSN application

root node. With preciseness, the protocol utilized is the Collection Tree Protocol (CTP) on its implementation in TinyOS that will be described later.

For a better comprehension a possible application field is provided. As seen before, WSNs are likely to be omnipresent almost everywhere, and one of these possible places are hospitals. In medicine, WSNs can monitor vital signs and act consequently according to any patient health variation. Then, we can deploy several WSNs on a hospital where each one will control the existing patients per floor. Thus, a control room with the personal on duty are able to follow the behaviour of the patients and anticipate possible actions involving surgeons or any other high-qualified personal that usually is scarce, even more at night's shift. The deployed nodes are constantly sending the patient's information to the main node in charge of the floor and, in the meantime, the root node is connected to a computer that processes all the information and shows the patient's evolution, and that floorplant diagram could be repeated according to the number of floors in Figure 4.2 on page 27.

Moreover, we have to take into account that nowadays on most working environments there is a working wireless network, normally a regular WLAN, to carry out the different administrative tasks involving the hospital burocracy. In addition, microwaves or similar electronic devices are likely to be present on the day by day in a hospital. Therefore, this typical environment has been studied in [37], where the authors tried to characterize it. As a conclusion, they found high activity and the interference level measured was time varying and dynamic, since there were high variations depending time of the day and the floor that they took measures. Furthermore, it is necessary consider the patients' mobility within the hospital whether they have to be tested or simply they have to be transfered into another room.

All gathered, we are facing a high activity environment regarding to the 2,4 GHz band and it is also time varying and dynamic. Thus, an adaptive scheme for the current conditions at each moment and in any place is required, as long as the information of all the monitored patients must reach the root node regardless of the existance of interferences or not. In addition, the deployed network should be flexible in number of patients to monitor since the number of patients could vary in time. This problem is what is tried to be addressed along this work providing an standard solution for WSNs to be deployed on crowded spectrum areas.

4.4 Application description

Previously exposed while talking about the *cognitive cycle*, Figure 2.2 on page 5, different techniques for the *spectrum sensing* were introduced. This former step will characterize the global application developed that is presented in the following sections. The proposed technique is related to detecting presence through packet snooping, thus, the channel occupancy by a nearby subnetworks will suppose the migration into the next available channel. From now on we can refer to this technique as jammer[1] detection. This detection will be cooperative since each node will report to the root node the presence of a jammer and subsequently, the root will dispose an overall situation of the network considering not only its environment but also each node's environment.

The application has been developed using the CTP already implemented on the TinyOS. Therefore, the provided code will act on the application layer as well as the CTP layer and they will interact between them at the runtime. In other words, the system will start and the CTP will form the network with the existing devices and, once the protocol provides routes to the whole network from the root, the developed application will perform the *cognitive cycle*. Afterwards, if a channel migration is required the application through the root node will interact with the hanging network to scroll it to a given channel by using the CTP. Finally, the application will restart the subnetwork and, consequently, the CTP layer will form again the network on the new channel.

4.4.1 CTP on TinyOS

The Collection Tree Protocol sets a tree-based topology network in order to collect all the information from the hanging nodes or leafs. One or a few nodes can be marked as the tree roots and the rest of the network establish routes to these roots, since a leaf node can not send a packet to another leaf. The resulting routes are established by using a routing gradient. As a result CTP can be considered as an address-free protocol. The

[1]The radio transmission that interfere into our desired signal from an external device by decreasing the SNR at the receiver.

CTP considers that the data link layer provides:

- an efficient local broadcast address
- synchronous acknowledgments for unicast packets
- a protocol dispatch field to support multiple higher-level protocols
- a single-hop source and destination fields

Besides, the CTP assumes that the link quality provides the value with the nearby neighbors. Hence, the number of retransmissions to send an unicast packet whose acknowledgment is received can be achieved [38].

As mentioned before, CTP uses a routing gradient to calculate the best route to the tree root. Specifically this gradient is the ETX[2], therefore, a root node will have typically ETX of 0, for the rest of the nodes the ETX will be the ETX of its parent plus the ETX of its link to its parent. Once all the possible routes are listed, CTP selects the ones with the lowest ETX values.

Due to the changing environment that usually surrounds the network, routing loops are likely to appear. However, CTP has two mechanisms to address this situation, in fact they are applied sequentially just in case the former fails. Firstly, if CTP detects a packet with a lower ETX than the one it has, it will report this anomaly by broadcasting a beacon frame, expecting that the causing node receives the frame and updates its routes accordingly. Nevertheless, if the causing node does not receive the frame the second mechanism to avoid the routing loops is started. Basically, it consists on checking the ETX value is below a reasonable constant, and as a result all the routes with a higher ETX will be discarded. Another important issue that can entail some problems is the packet duplication. This common problem while treating with Acknowledgment packets (ACK), occurs when an ACK is not received by the sender and the packet is retransmitted again. This can provoke congestion within the network since the packet duplication will be exponential according to the number of hops to reach the tree root. The solution is the Time Has Lived (THL) field that is incremented on each hop. Hence, a looped packet THL will differ from the THL of a link-level retransmission and, consequently, it will be dropped. Either ETX and THL field disposition on the CTP frames are presented in Figure 4.3 on page 30 [38].

On the other hand, and as its name reveals, CTP uses the Collection services also implemented on TinyOS. In fact, is because of the Collection that a node can advertise itself as a root and, consequently, collect all the information. Another important remark

[2] Expected Transmissions (ETX), measures the quality of the path between two nodes.

```
                    1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|P|C| reserved  |      THL      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              ETX              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            origin             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     seqno     |  collect_id   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    data ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(a) Data frame

```
                    1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|P|C| reserved  |     parent    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     parent    |      ETX      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      ETX      |
+-+-+-+-+-+-+-+-+
```

(b) Routing frame

Figure 4.3: CTP either data and routing frames

is that the CTP is not a bidirectional protocol, following the idea of sink networks by which all the information flows like a stream towards the sink and there is no possibility to go backwards. However, on TinyOS there is the possibility to spread short packets from the root to the entire network by using the Dissemination of Small Values service. This service, makes possible to disseminate a small value, typically an integer, within the network without using any collection service. Both Collection and Dissemination will be explained in detail on the following sections.

4.4.1.1 Collection

Mostly all WSNs require at least the minimum expression of a Collection protocol. The network topology is still tree-based subnetworks gathered among them by a base station and the data is managed to reach the tree root as well. While having several tree subnetworks within the same network, the protocol provides a best-effort packet delivery to one of the roots but there is no total guarantee of delivery and no ordering guarantees either. On the contrary, duplicate packets can be delivered to one or more roots. Collection and CTP share the same problematic situations and in the same way, the share the same solutions as loop detection, duplicate suppression, link estimation and self-interference. A node can have four different roles in Collection protocol: producer, snooper, in-network processor and consumer. According to each one, the node will choose the needed interfaces to perform its role. The nodes that capture the information to be sent are the producers. Secondly, the snoopers overhear the packet and if they are not a root, they will forward the packet up to reach the nearest root. If required, a node can update a packet by adding some information of its environment and subsequently forward it as well, becoming an in-network processor. Finally, if the node is a root that is supposed to be the final destination of the data packets, the node act as a consumer [39].

4.4.1.2 Dissemination of Small Values

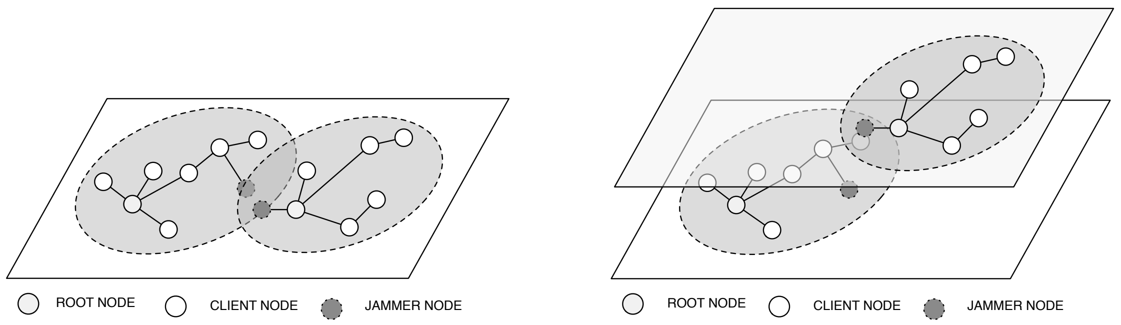
In order to fulfill the point regarding to the bidirectional communications, the dissemination services are required as long as they are able to update homogeneusly the value of a shared variable within the network, and that is the reason why is denoted as Dissemination of Small Values. The service warns nodes when the value of the shared variable has been updated by interchanging packets in order to reach eventual consistency on the whole network. It may occur, however, that a few nodes disagree momentarily on the value but after a while the disagreements will shrink and the network will converge on a single value. This eventual consistency is robust to temporary disconnections and high packet loss rates, the dissemination assures this consensus as long as the node is not disconnected permanently [40]. This fact is going to be important at the programming time since a channel switching by node can be considered a disconnection from the network.

Furthermore, this consistency across the network considers the newest value to establish the consensus. When a node receives a supposed new dissemination value that it is older than the one already existing in the node, it will be able to prompt the network and it will warn the rest of the network that there is a new value. Once this new value is updated by several nodes, the mechanism will make the rest converge into the real new value. Finally, the consistency assures that the network will share the same value of a given variable after a while, however, it is probably that some nodes on the network do not see the different values that the variable has along a given period, and they just get the last new value. This situation could be due to a temporary disconnection or because of several updates done in a short period [40]. Therefore, it is not possible to guarantee that a node's variable will be updated with all the new values and, consequently, a proper programming is required if all of these values are needed along the performance.

4.4.2 Role description

As CTP-based protocol, the jammer detection scheme has two main roles on the network topology: the root node as the main control entity and the client nodes as sensing captors. As a low level description of the cognitive WSN the Figure 4.4 on page 32 represents an understandable diagram that shows the final aim of the application. Firstly, the network is formed on the default channel through the CTP procedure. However, afterwards appears an external network operating in the same frequency plane within the coverage range. Later, the affected node reports this situation to the root node and proceed to the channel migration in order to avoid this external interference. Eventually, both networks will still be operative but in different frequencies and the overall operation has been transparent to the final user.

The application has been implemented without any specific purpose, thus, the data information to be sent by the client nodes is easily adaptable in order to send certain



(a) Both subnetworks performing on the same frequency plane

(b) The application scrolls its subnetwork into another frequency plane after a jammer detection

Figure 4.4: Channel distribution before and after the cognitive performance

information captured by the client nodes. Moreover, the cognitive process that takes place on the application is hidden to the user interface since neither the jammer reporting nor the channel migration are lasting operations. It is worth pointing out another feature of the scheme, whereas the network is starting the root node receives data messages from the client nodes already operative, the root node will consider, after a preset time, that there is no more clients on its network and will report the network size, that has been checking all along this period, to the client nodes. Hence, code modifications are not required in order to set the network size and, consequently, all the network will be able to determine if a received packet comes from a known user of their network or if it can be considered as a jammer.

4.4.2.1 Root node

As the main entity, the root node coordinates the whole network. This responsibility implies, in addition to the default tasks of a root node, most of the cognitive tasks. Although the *spectrum sensing* is cooperative within the entire network, the remaining tasks as *spectrum analysis* and *spectrum decision* from the cognitive cycle, Figure 2.2 on page 5, are all performed in the root node. According to this point of view, it is fair to say that the root node, as the intelligent device of the network, is what provides the cognitive adjective. In Figure 4.5 on page 33, the flowchart for the root node is presented and, as simple as it is, the cognitive operations are summed up in detecting a jammer and moving to a free channel if required, besides the data reception period that can be considered, in a certain way, part of the cognitive operations, since if there is a jammer on the coverage area the client nodes will report that situation through their data packets. Finally, a serial communication has been implemented on the root node code since, as seen in Figure 4.2 on page 27, usually the root node will be connected to a computer in order to process and monitorize the information received.

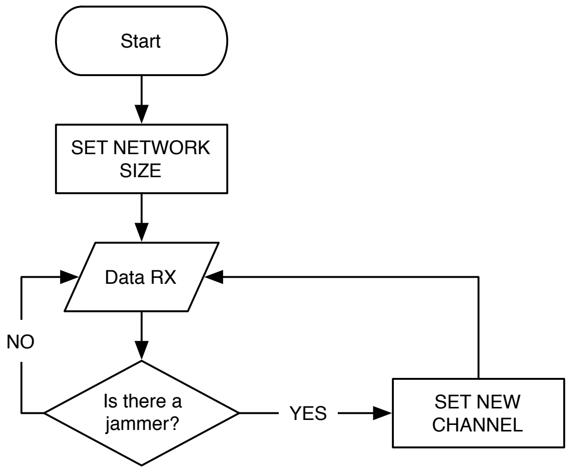


Figure 4.5: Root node flowchart

4.4.2.2 Client node

The client node will only need a few lines of code for its performance, in as much as sending information is its unique commitment. However, there are small code modifications regarding to a standard sending node for a regular CTP networks. Those modifications are basically implemented for the jammer detection to make them to be able to determine the procedence of a received packet. On the other hand, they will send packets randomly to the root with the desired information captured by their sensors.

Chapter 5

Implementation

After an overall explanation of the application behaviour in the previous sections, now it is time to a detailed explanation taking into account the actual code used for this work. On the current section firstly, a detailed performing description is presented by using an application example showing the possible packet interchange concerning three nodes on the network, during a certain period of time. However, the implicit packet transmissions due to the standard CTP network formations and data disseminations will be ignored since they are not part of this work and they are detailed in [38, 39, 40]. Secondly, the resulting event execution will be explained briefly in order to remark how this event-driven programming influence in the resulting execution threads on the node. Later, some of the important variables, structs and constant definitions that take part on the performance are listed together with the main functions and event handlers defined on the application code. Finally, some considerations regarding the possibilities and limitations of the scheme are presented.

5.1 Performance

Following the network performance example in Figure 4.4 on page 32, a packet transmissions diagram is presented in Figure 5.1 on page 36 in order to complete totally the description of the application developed on this work. The example represents the communication between three nodes: the root node and two client nodes. The root node, named as NODE 0, is only at one hop distance of one of the clients named NODE 1. The last node, NODE 2, is at two hops distance of the NODE 0 and needs the NODE 1 to forward its either sending or receiving packets.

Once the network is started, there is a dedicated period for constituting the network routes using the CTP and inmediately the nodes will start sending data messages to the NODE 0. At this time, there is no node able to detect any jammer, since they do not know the network size yet. When the `TIMER_1` fires, the NODE 0 disseminates the `netsize` value, and from then on the network is able to perform the jammer detection. After a reasonable time of data collection, specifically `TIMER_2`, the NODE 0 will check if any jammer has been reported. On the example, NODE 1 has reported the presence of other network on the present channel. Consequently, NODE 0 disseminates a new `channel` value and starts its `TIMER_3` and, NODE 1 and NODE 2 start their `TIMER_3` as well when the new value is received. These `TIMER_3` are needed since an inmediate channel change will be understood as a disconnection. Thus, when the first `TIMER_3` of the whole network fires the dissemination service, it has to reach consensus on the new value of `channel` variable. When the last of the `TIMER_3` fires, the CTP is restarted in order to form again the routes but this time at the new frequency. Eventually, to close up the circle, a new data collection is started. However, at this time is noticeable that there is not a period to update the `netsize`, since as mentioned before this variable will remain with the same value until the end of the session. In addition, we can realise how in the example this description is separate into three stages: the former is the *group formation*, then the *data collection* and finally the *channel migration* if needed.

On the other hand, is worth to mention that there is neither specific functions nor longer periods dedicated just for the cognitive actuation. In fact, just `TIMER_3` is totally dedicated to cognitive finalities, however, as it will be explained later, this timer is a small period that can be depreciated while comparing with the data periods. Hence, the whole performing procedure is also energy-efficient, as long as there is no continuous scan while performing the *spectrum sensing* like other similar schemes presented on a previous chapter. Besides, the *spectrum analysis* takes place discretely, and at the same time that the root is acquiring the data from the client nodes. Moreover, the decision procedure, concerning the *spectrum decision* does not either delay or postpone the data collection period. Therefore, the three steps of the cognitive cycle, in Figure 2.2 on page 5, are efficiently carried out by the scheme since they are all done while the main data operations take place and taking advantatge of them. Finally, a generic step-by-step description of the network events is presented as follows:

1. The root node together with the client nodes are switched on and the group formation period is started

2. The client nodes send randomly data messages following the CTP formation routes and the root node collects them

3. After a given time, the root node has enough information to set the network size and, consequently, disseminates the value of the `netsize` variable

4. From this moment the data collection period is started and the nodes keep on sending information to the root

5. However, at this time jammer detections can be reported since the network size has already been established

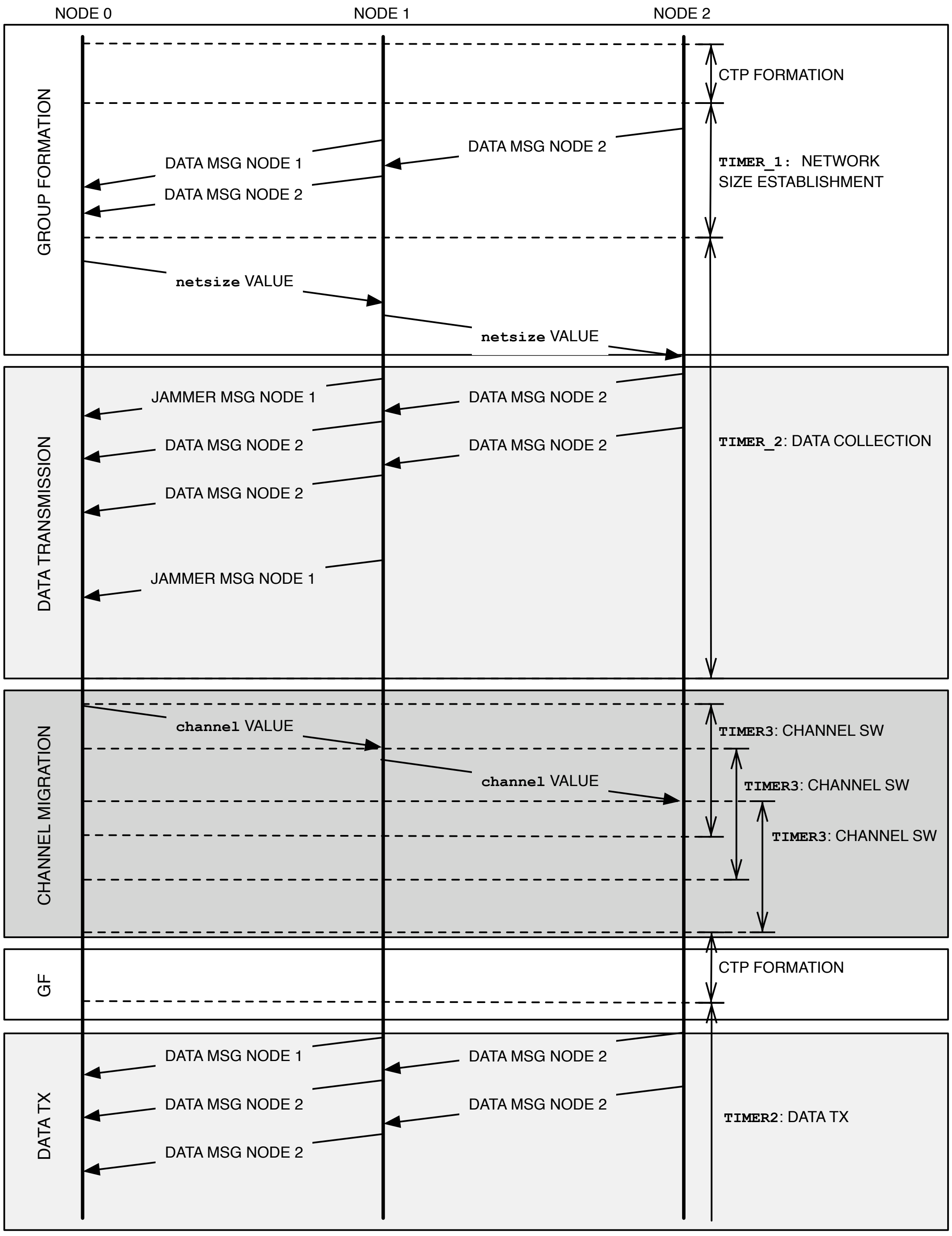


Figure 5.1: Application description example

6. After the data collection period, the root node checks if any jammer has been reported on the current channel by any of the nodes
7. If detected, the root node disseminates the new `channel` value and starts the channel migration period by calling its timer
8. At the time that the dissemination service warns the value has been updated, on the remaining nodes the timer is started as well
9. Before the first timer fires, the network has to reach consensus on the new `channel` value and when the timer fires, each node changes its frequency
10. The root restarts the network by using the CTP to constitute the new routes but this time at the new channel

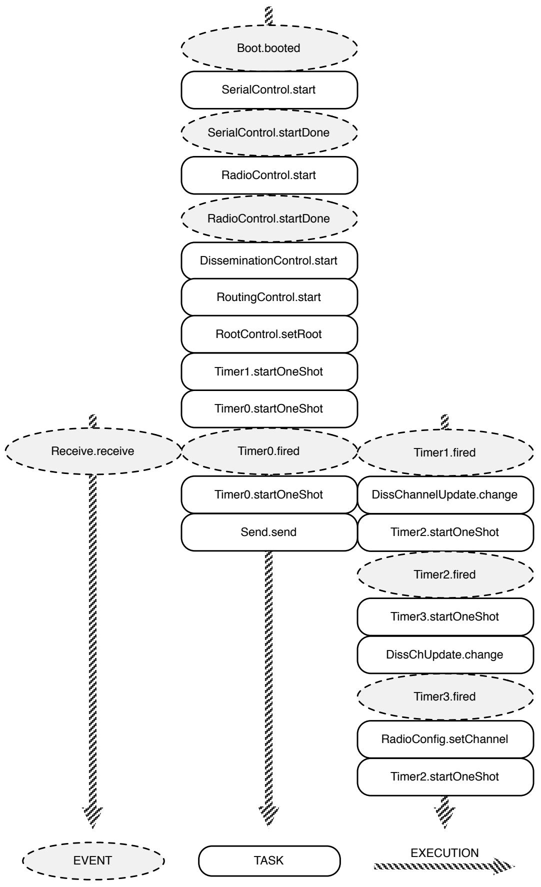
5.2 Events execution

As stated before, TinyOS uses an event-driving execution that makes easier to describe the application. In Figure 5.2 on page 38, both possible event executions depending on the assigned role are presented. The single execution thread, as a consequence of the event-driving programming, can be described as three threads for a better understanding. After all the network formation events have finished, the data collection period is started. During this period, on the root node three different events may happen: either receive or send a message or to have a fired timer. In the same way, on the client nodes three different situations can occur as well: either forward or send a message or having a variable update by the dissemination service. In practise, due to the event execution architecture, it is not possible to consider three execution threads since only one event handler will be executed at the time. Nevertheless, if after the handler has been executed there is a long function call, that does not prevent the execution of the next event to occur. Therefore, the example in Figure 5.2 on page 38 shows the clear difference between the group formation period and the rest of the time since the former is more predictable regarding to the events to be executed.

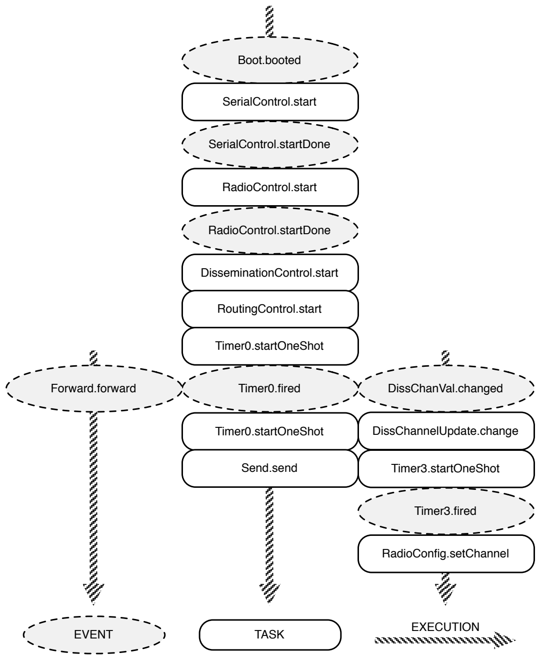
5.3 Baring the code

The IRIS motes incorporate the Atmel RF230 radio transceiver that entails the utilisation of the RF230 stack interfaces on the TinyOS 2.x tree. This stack, as almost the whole TinyOS tree is under development and, that is the reason why at the beginning the `RadioConfig interface` was not yet implemented. However and thanks to Miklos Maroti[1], at the programming time the `RadioConfig interface` was just implemented

[1]Currently, associated professor at the University of Szeged and core developer of the TinyOS tree.



(a) Main events execution for the root node



(b) Main events execution for the client node

Figure 5.2: Events execution for both existing roles on the network

and it has become essential for the final application. This interface is needed in order to change into a new channel at the runtime and is provided by the `RF230ActiveMessageC module` as seen in Figure A.8 on page 59. The application was deployed using the already existing `TestNetwork` application as a starting model since it was designed to implement a simple data collection by the root node. There were no multihop implemented since the `forward event` was not deployed. Obviously, all the functions and events regarding to the cognitive cycle were not either deployed. A code description of the `RssiJammer` application deployed for this work is presented in the next sections.

5.3.1 Variables, structs and constant definitions

On the global application many variables either local or global are defined. However, just a few have a remarkable importance since they will be disseminated, if required, within the network. In Figure 5.1 on page 36 an application example is presented and these variables are shown while being disseminated. As comented before, while disseminating only small values can be sent, and that is the reason why they will be typically integers:

`netsize:` It applies on all nodes. This variable has a shared value in the whole network and is initialized with 0 as a non-valid network size. Thus, all networks expect the root to modify this value in order to detect any jammer presence while performing. The variable is modified only once at the network starting and remains with the same value unless the network is rebooted.

`channel:` It applies on all nodes. This variable has a shared value in the whole network and this is how it has to be as long as the node wants to be connected to the current network. This variable is initialized with the default radio channel and it is supposed to change everytime there is a jammer detection reported to the root.

For the data collection, a few structs are required in order to encapsulate the information while sending it through the network and to storage the valid information in the root node. The following structs will be implemented:

`RssiDetectionMsg:` An standard network struct to encapsulate the data to be sent towards the root node. The data field, will be filled with the RSSI[2] of the received packet at the first hop. It will also include a source field in order to report if there is a jammer on the current channel.

`RssiRoot:` A simple struct to storage the data collected for a later network analysis.

[2]The Radio Signal Strength Indicator (RSSI) measures the power received due to a radio signal.

The are a few constant definitions that influence on the behavior of the network, regarding basically to the time that the application will dedicate to each action. Again, in the Figure 5.1 on page 36, a typical ordering of execution of these constants is presented. These constants are the following:

`TIMER_1`: It only applies on the root node code. This variable sets the time that will take to define the network size since the timer associated to this variable fires, the root node will disseminate the current value of the network size to the rest of the network. Subsequently, the client nodes will consider as a jammer any other node if they have been switched on after the timer fires. Therefore, the value should be long enough to switch on all the devices that will constitute the network.

`TIMER_2`: It only applies on the root node code. This variable sets the period for the data reception from the client nodes at the root node. As this period concerns to the main duty of the network, the data collection, it will typically be the longest period. Nevertheless, it is advisable to cut this time short in time varying and dynamic either networks or environments to assure a free-interference performance during most of the time.

`TIMER_3`: It applies on all nodes. This variable sets the time between the moment when node is warned about a new `channel` value and the channel switching into the channel pointed by the root. This timer value must be set carefully since a small value could lead to a network division. If the time is less than the required for the dissemination service to reach consensus, the non-updated nodes will not change their frequency and, consequently, they will be considered as disconnected for the network since it will already perform in the new channel. In contrast, a long timer value obliges the network to remain in the channel supposedly affected by a jammer.

5.3.2 Functions and event handlers

In the execution code presented in RssiDetectionC.nc there are a few relevant functions and event handlers regarding mainly to the cognitive operations. They are all listed as follows together with a brief description of their duties.

`void setChannel(uint16_t chan)`

By using the `RadioConfig interface` the application is able to set a new given channel at the runtime. In case an error returned by the `RadioConfig interface`, `setChannel` will incide in the channel switching until there is no error returned.

`void Decision(RssiRoot rssi[MAX_NODES][MAX_JAMMERS], bool jammed[], uint16_t mean[])`

The most important function regarding to the cognitive actuation of the application. This function makes the decision of a possible channel migration regarding to the data collected along the previous data collection period. If a channel migration is required the function will call the dissemination services to warn the rest of the network about the new performing channel and it will call the timer whose handler will call the `setChannel` function.

`void sendMessage()`

By calling this function, a generic message is send towards the root node. The message's information is filled with generic fields totally modifiables according to the final purpose.

`event void Timer0.fired()`

Event handler for the `TIMER_0` that only calls the `sendMessage` function. It also starts the next `TIMER_0` but randomly.

`event void Timer1.fired()`

Event handler for the `TIMER_1` that sets the `netsize` variable for a later dissemination of the value. In addition, it starts the `TIMER_2`.

`event void Timer2.fired()`

Event handler for the `TIMER_2` that simply calls the `Decision` function.

`event void Timer3.fired()`

Event handler for the `TIMER_3` that calls the `setChannel` function and starts the new period of data collection by using the `TIMER_2` again.

`event void DissChannelValue.changed()`

Event handler for a DissChannelValue change, it updates the `channel` value and starts the `TIMER_3`.

`event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len)`

Event handler for a received message at the root node. Depending on the data payload, a jammer detection will be reported. It also stores the RSSI data on the `RssiRoot` struct.

RF230.h	
RF230_TX_PWR_DEFAULT	0
RF230_CHANNEL_DEFAULT	11

RssiDetectionC.h	
MAX_CHANNELS	16
MAX_NODES	10
MAX_JAMMERS	50

RssiDetectionC.nc	
TIMER_1	10000
TIMER_2	50000
TIMER_3	10000

Table 5.1: Preset values for the global application

	Memory required	**Memory available on the IRIS motes**	**Percent**
Flash memory	28758bytes (28kB)	128kB	22%
RAM	5384bytes (5kB)	8kB	66%

Table 5.2: Memory requirements for the global application

`event bool Forward.forward(message_t* msg, void* payload, uint8_t len)`

Event handler for a received message at a client node. In case of an empty value of the RSSI the current node will add the information and will return `TRUE`, indicating that this message must be forwarded to its final destination, the root node.

5.4 Global application issues

After compiling, the global application is ready to be flashed on the nodes. There is only a single file to flash and regarding to the ID given to each node the code will assign the corresponding role. The ID's of each device must go from 0 to 9 as a maximum defined for this application, Table 5.1 on page 42. The root node will be the mote flashed with the ID=0, and it is mandatory the existance of a root node. The rest of the nodes can be the other ID from 1 to 9 and they have to be ordered.

Again in Table 5.1 on page 42, the default value for the transmission power will be 0 dBm and the default channel is 11. Regarding to the timers, these values have been assigned in order to assure the correct behaviour at the time of the network formation and the channel migration. Pressumably, `TIMER_3` can present a lower value since the time that takes to reach consensus while disseminating is not noticeable after testing the application. Furthermore, the global application to be flashed presented the following memory needs after compilation, Table 5.2 on page 42. These values can be considered reasonable since there is some information to store in the device.

5.4.1 Considerations

Due to design issues together with WSNs limitations, a few considerations are exposed chronologically about the spectrum-efficient architecture for cognitive WSNs presented on this work as follows:

- The network size is limited in number of nodes
- The devices offer a limited storage of information for a later process
- The timers' value must be adapted to the spatial disposition of the network since longer distances requires more propagation time
- To increase the network, it must be restarted since the new node will start its performing on the default channel and the network is likely to have changed its channel

Chapter 6

Evaluation

Eventually, a formal evaluation of the implemented code is carried out on the current chapter. For the final testing, we have used IRIS motes together with the MIB520 programming board. Regarding to the network topology proposed, we tried to deploy a generic data collection network with a tree topology imposed by the CTP. Besides, we distributed the nodes estrategically in order to test the multi-hop feature added to the application. The final testing took place in a common room, thus, the distances were short. However, during the programming time the motes range were tested indoor, and they perform normally with four floors of separation in a regular building.

Therefore, in Figure 6.1 on page 44, a diagram of the distribution is presented. Because of the disposition of the motes, a few different topologies can occur. Even they are likely to switch from one to another at runtime because of the variability, constantly mentioned along the work, of the spectrum conditions. After the network formation, a jamming node will enter on scene in order to prove the cognitive capabilities of our application. The mentioned jammer will vary slightly its position while performing.

In the following section, by analizing the event's log at the root node, we will be able to compare the final performing conditions of the network with the expected behaviour of the application.

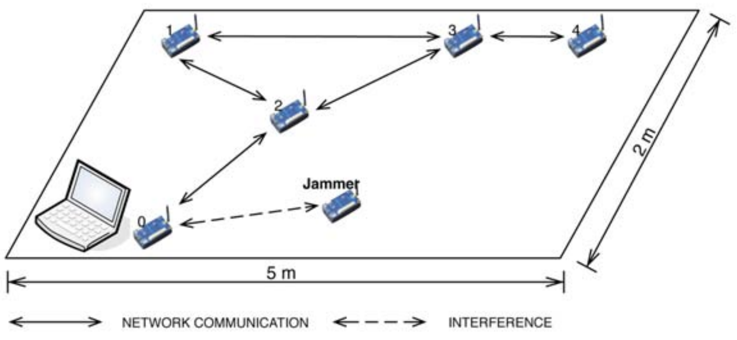


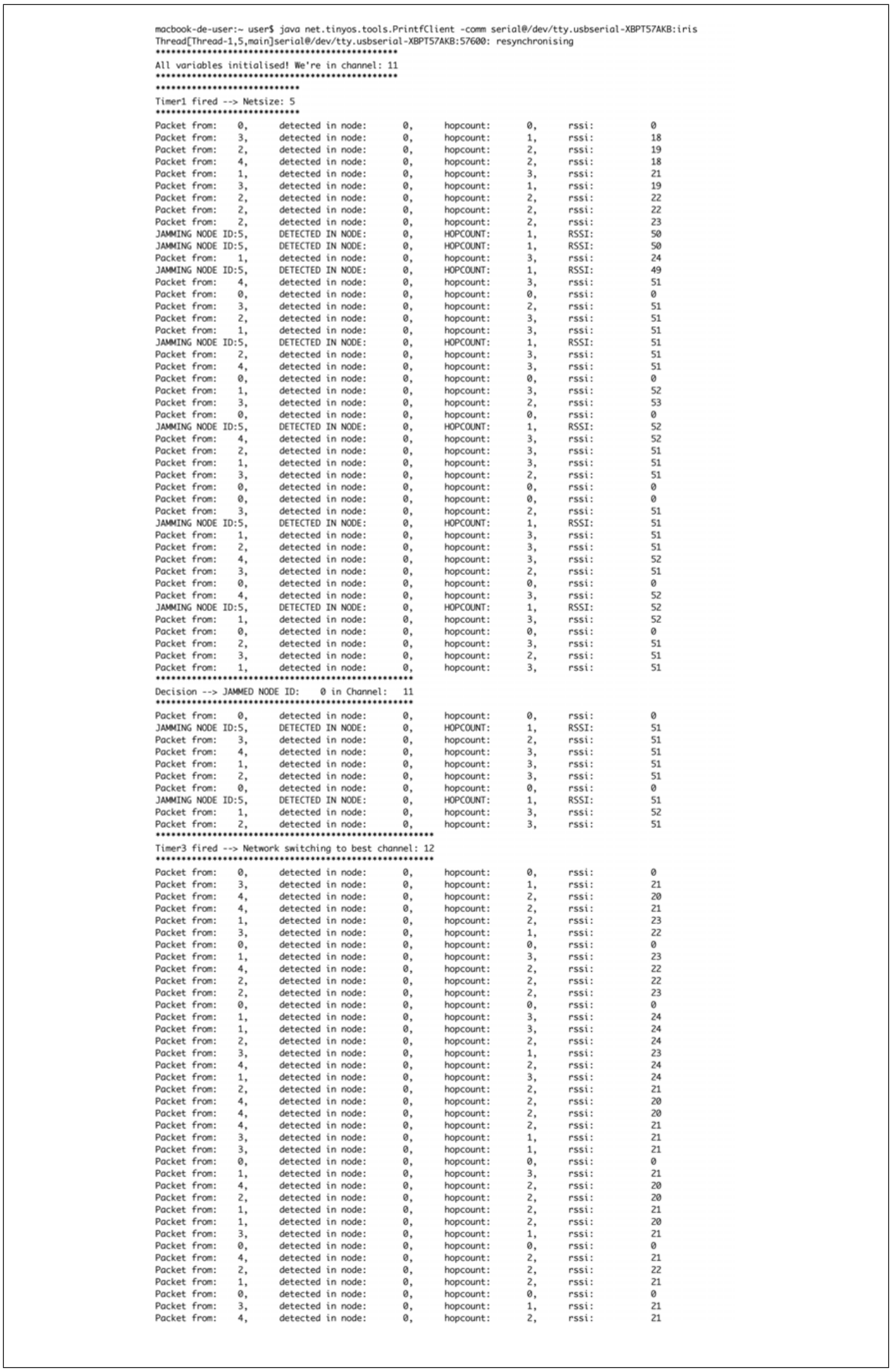
Figure 6.1: Network distribution during the test

6.1 Results

Taking a look at the event's log in Figure 6.2 on page 46, it is easy to state that the protocol has worked. All the devices are initialized on `channel 11`, as well as the jammer mote, and they perform the data collection but, in the meantime, the root node detects a jammer node interference. Therefore, after the data collection period is finished the root node decides to migrate the network to the next available channel, that is the reason why they continue with the data collection at `channel 12`. After the migration, no other jammer is reported, at least the jammer with the `ID:5` is not longer interfering. As not being part of the network, the jammer does not interprete the root's message and, consequently, it remains at the same channel.

Another remarkable issue, would be the multi-hop behaviour of the network. On the event's log there is a field named `hopcount` that shows the number of hops that the packet has done until it has reach the final destination. By inspection, one can see that the number is not constant along the packets with the same source. This fact reinforces the ability of the continuous communication in spite of either the mobility of the network or the spectrum variations. However, these topology variations in our results can be atributted fundamentally to the spatial variation of the jammer node, since its interference can provoke path modifications due to a failed ACK packet from the parent.

Finally, regarding to the last field, the `RSSI` received values are shown. These values suffer a noticeable change when the jammer is switched on, and after the application migrates the network to the new channel, they return to their previous values. Regarding to the `RSSI` values, they are not shown in dBm, and they need obviously to be converted. Nevertheless, we should take into account the noticeable difference that one can observe in Figure 6.2 on page 46 between the non-interfered values and the interfered ones, even without applying the conversion. Hence, our scheme has improved the performance by bringing the signal levels to the non-interfered values once a jamming device has appeared. However, the most imporant fact is that all the operations involving this behaviour, have been done autonomously and without any human supervision, in other words, cognitively.



```
macbook-de-user:~ user$ java net.tinyos.tools.PrintfClient -comm serial@/dev/tty.usbserial-XBPT57AKB:iris
Thread[Thread-1,5,main]serial@/dev/tty.usbserial-XBPT57AKB:57600: resynchronising
***********************************************
All variables initialised! We're in channel: 11
***********************************************
***************************
Timer1 fired --> Netsize: 5
***************************
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          18
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          19
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          18
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          21
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          19
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          22
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          22
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          23
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          50
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          50
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          24
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          49
Packet from:    4,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          51
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          51
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          51
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    4,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          52
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          53
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          52
Packet from:    4,      detected in node:      0,      hopcount:      3,      rssi:          52
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          51
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          51
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          51
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    4,      detected in node:      0,      hopcount:      3,      rssi:          52
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          51
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    4,      detected in node:      0,      hopcount:      3,      rssi:          52
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          52
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          52
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          51
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          51
**************************************************
Decision --> JAMMED NODE ID:    0 in Channel:   11
**************************************************
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          51
Packet from:    3,      detected in node:      0,      hopcount:      2,      rssi:          51
Packet from:    4,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
JAMMING NODE ID:5,      DETECTED IN NODE:      0,      HOPCOUNT:      1,      RSSI:          51
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          52
Packet from:    2,      detected in node:      0,      hopcount:      3,      rssi:          51
******************************************************
Timer3 fired --> Network switching to best channel: 12
******************************************************
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          21
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          20
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          21
Packet from:    1,      detected in node:      0,      hopcount:      2,      rssi:          23
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          22
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          23
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          22
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          22
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          23
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          24
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          24
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          24
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          23
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          24
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          24
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          21
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          20
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          20
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          21
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          21
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          21
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    1,      detected in node:      0,      hopcount:      3,      rssi:          21
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          20
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          20
Packet from:    1,      detected in node:      0,      hopcount:      2,      rssi:          21
Packet from:    1,      detected in node:      0,      hopcount:      2,      rssi:          20
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          21
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          21
Packet from:    2,      detected in node:      0,      hopcount:      2,      rssi:          22
Packet from:    1,      detected in node:      0,      hopcount:      2,      rssi:          21
Packet from:    0,      detected in node:      0,      hopcount:      0,      rssi:          0
Packet from:    3,      detected in node:      0,      hopcount:      1,      rssi:          21
Packet from:    4,      detected in node:      0,      hopcount:      2,      rssi:          21
```

Figure 6.2: Root node event log

Chapter 7

Conclusions and future work

All along this work, we have introduced the emerging CR techniques, and justified their application under WSNs due to their low-powered emissions. Furthermore, there is the envision that in the next years the population of the WSN will increase considerably. Thus, an extensive development on these networks is required to address the distribution of the available spectrum for wireless communications. The aim of the already presented work is to contribute with an adaptive scheme, that is able to coexist with other WSNs. The design of the application has tried to accomplish with the main features of the WSNs that may be affected by the application design, such as autonomy, mobility and robustness. As a result, our proposal does not consume more resources than the expected for a regular collection protocol, and in the meantime, offers a clear advantage than other similar data collection applications when another WSN is present.

Through real testing, at the programming time we have debugged the application in order to obtain the desired behaviour, and we have validate our scheme with favourable results by testing a real application scenario. The application performs smoothly under ideal spectrum conditions, and keeps performing fine in spite of a jamming interference, because of endowing a cognitive behaviour to the existing collection protocol . Hence, the commitment of this work has been achieved, however, further development on cognitive schemes is needed. In the next section, a few future working guidelines are suggested.

7.1 Future work

As a first step, our scheme could be complemented by considering the RSSI level at the receiver as a decision factor. Therefore, our scheme would be sensitive to other interferences appart from the WSNs. However, in the current devices the RSSI measurement is not totally reliable and, consequently, a new procedure to characterize the interference has to be devise. This first actuation, would position the WSNs on a better place since they would not struggle with other powerful devices and, moreover, they would be intelligently placed on the best available frequency.

As a middle step, all technologies that coexist on the ISM band should incorporate a similiar adaptive scheme as our proposal, although, a standard negotiation protocol understandable by all the devices, regardless the standard they use, would be necessary as well. This way, they could arrange the best spectrum disposition for the present communications by negotiating the utilisation of a certain frequency range, at each moment that an incoming communication appears.

Finally, as the last step and according to the future challenges of the CR networks, the mentioned procedure should be extrapolated to all the frequency bands available for wireless communications. Thus, appart from obviously having more spectrum range to share, the QoS of certain standards, such as WSN, would be improved since they could take advantage of the lower losses in the lower frequencies.

Bibliography

[1] IEEE Computer Society. Ieee standard for information technology— telecommunications and information exchange between systems— local and metropolitan area networks— specific requirements. In *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, 2006.

[2] Crossbow. Avoiding rf interference between wifi and zigbee.

[3] Razvan Musaloiu-E. and Andreas Terzis. Minimising the effect of wifi interference in 802.15.4 wireless sensor networks. *Int. J. Sensor Networks, Vol. 3, No. 1*, 2008.

[4] Compatibility of ieee802.15.4 (zigbee) with ieee802.11 (wlan), bluetooth, and microwave ovens in 2.4 ghz ism-band. Technical report, Steibeis-Transfer Centre, 2004.

[5] Robert W. Brodersen Danijela Cabric, Shridhar Mubaraq Mishra. Implementation issues in spectrum sensing for cognitive radios. *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference*, 2004.

[6] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: a survey. *Comput. Netw.*, 2006.

[7] National Telecommunications and Information Administration. U.s. frequency allocation chart as of october 2003.

[8] P. Kolodzy et al. Next generation communications: Kickoff meeting. *DARPA*, 2001.

[9] ET FCC. Notice of proposed rule making and order. *Docket No 03-222*, 2003.

[10] Joseph Mitola III. *An Integrated Agent Architecture for Software Defined Radio*. PhD thesis, Royal Institute of Technology (KTH), 2000.

[11] Joseph Mitola III and JR. Gerald Q. Maguire. Cognitive radio: Making software radios more personal. *IEEE Personal Communications*, 1999.

[12] Simon Haykin. Cognitive radio: Brain-empowered wireless communications. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 2005.

[13] Qusay Mahmoud. *Cognitive Networks: Towards Self-Aware Networks.* Wiley-Interscience, 2007.

[14] Edgar H. Callaway. *Wireless Sensor Networks: Architectures and Protocols.* CRC Press, Inc., Boca Raton, FL, USA, 2003.

[15] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S.J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[16] D. Culler, D. Estrin, and M. Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 2004.

[17] *IRIS Datasheet.*

[18] D. Manjunath. A review of current operating systems for wireless sensor networks. Technical report, Department of ECE, IISc, Bangalore, INDIA.

[19] Philip Levis Jonathan Hui and David Moss. *TEP 125 : TinyOS 802.15.4 Frames.*

[20] Tinyos community forum.

[21] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.

[22] David Culler Eric Brewer David Gay, Philip Levis. *nesC 1.1 Language Reference Manual*, 2003.

[23] John A. Stankovic Gang Zhou and Sang H. Son. Crowded spectrum in wireless sensor networks. *Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets)*, May 2006.

[24] Jungmin So and Nitin H. Vaidya. Multi-channel mac for ad hoc networks: handling multi-channel hidden terminals using a single transceiver. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 222–233, New York, NY, USA, 2004. ACM.

[25] Yafeng Wu, J.A. Stankovic, Tian He, and Shan Lin. Realistic and efficient multi-channel communications in wireless sensor networks. 2008.

[26] Jiandong Li, Z.J. Haas, Min Sheng, and Yanhui Chen. Performance evaluation of modified ieee 802.11 mac for multi-channel multi-hop ad hoc network. 2003.

[27] A. Tzamaloukas and J.J. Garcia-Luna-Aceves. A receiver-initiated collision-avoidance protocol for multi-channel networks. volume 1, pages 189–198 vol.1, 2001.

[28] Paramvir Bahl, Ranveer Chandra, and John Dunagan. Ssch: slotted seeded channel hopping for capacity improvement in ieee 802.11 ad-hoc wireless networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 216–230, New York, NY, USA, 2004. ACM.

[29] G. Zhou, C. Huang, T. Yan, T. He, J. A. Stankovic, and T. F. Abdelzaher. Mmsn: Multi-frequency media access control for wireless sensor networks. 2006.

[30] Jingbin Zhang, Gang Zhou, Chengdu Huang, S.H. Son, and J.A. Stankovic. Tmmac: An energy efficient multi-channel mac protocol for ad hoc networks. 2007.

[31] Xun Chen, Peng Han, Qiu-Sheng He, Shi liang Tu, and Zhang-Long Chen. A multi-channel mac protocol for wireless sensor networks. 2006.

[32] Chulho Won, Jong-Hoon Youn, H. Ali, H. Sharif, and J. Deogun. Adaptive radio channel allocation for supporting coexistence of 802.15.4 and 802.11b. 2005.

[33] Sofie Pollin, Mustafa Ergen, Michael Timmers, Antoine Dejonghe, Liesbet van der Perre, Francky Catthoor, Ingrid Moerman, and Ahmad Bahai. Distributed cognitive coexistence of 802.15.4 with 802.11. 2006.

[34] A. Kamerman and N. Erkocevic. Microwave oven interference on wireless lans operating in the 2.4 ghz ism band. volume 3, pages 1221–1227 vol.3, 1997.

[35] Pierre Jansen Sape Mullender Ozlem Durmaz Incel, Stefan Dulman. Multi-channel interference measurements for wireless sensor networks. *Local Computer Networks, Proceedings 2006 31st IEEE Conference on Volume*, 2006.

[36] M. Petrova, Lili Wu, P. Mahonen, and J. Riihijarvi. Interference measurements on performance degradation between colocated ieee 802.11g/n and ieee 802.15.4 networks. 2007.

[37] S. Krishnamoorthy, J.H. Reed, C.R. Anderson, P. Max Robert, and S. Srikanteswara. Characterization of the 2.4 ghz ism band electromagnetic interference in a hospital environment. volume 4, pages 3245–3248 Vol.4, 2003.

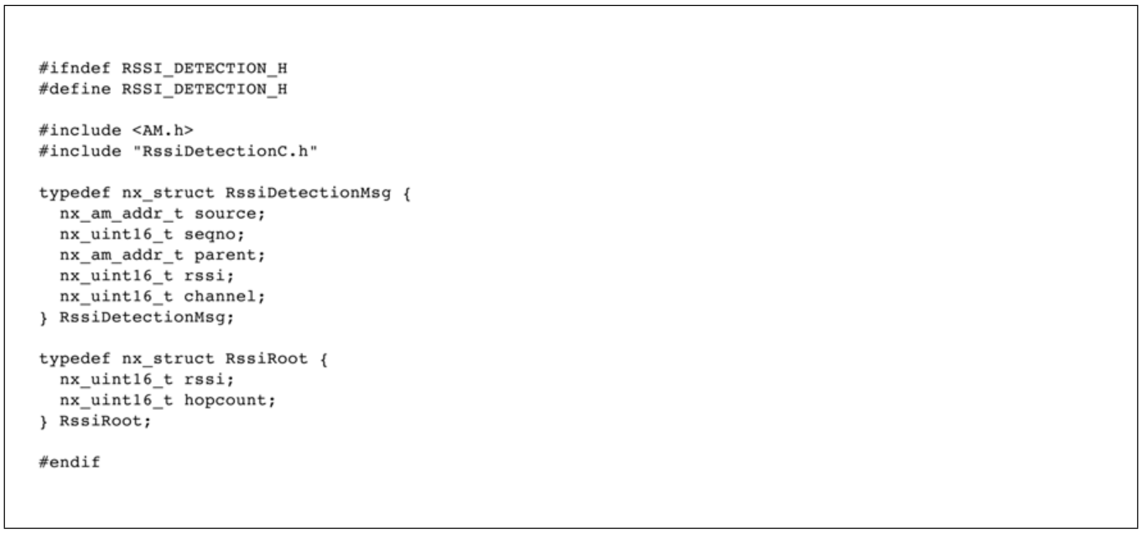
[38] Kyle Jamieson Sukun Kim Philip Levis Rodrigo Fonseca, Omprakash Gnawali and Alec Woo. *TEP 123 : The Collection Tree Protocol (CTP).*

[39] Kyle Jamieson Rodrigo Fonseca, Omprakash Gnawali and Philip Levis. *TEP 119 : Collection.*

[40] Philip Levis and Gilman Tolle. *TEP 118 : Dissemination of Small Values.*

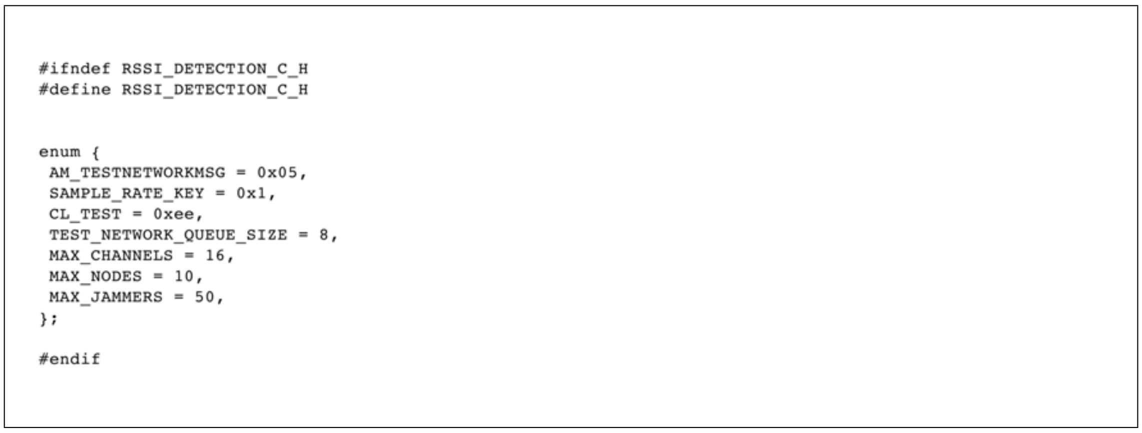
Appendix A

Application code



```
#ifndef RSSI_DETECTION_H
#define RSSI_DETECTION_H

#include <AM.h>
#include "RssiDetectionC.h"

typedef nx_struct RssiDetectionMsg {
  nx_am_addr_t source;
  nx_uint16_t seqno;
  nx_am_addr_t parent;
  nx_uint16_t rssi;
  nx_uint16_t channel;
} RssiDetectionMsg;

typedef nx_struct RssiRoot {
  nx_uint16_t rssi;
  nx_uint16_t hopcount;
} RssiRoot;

#endif
```

Figure A.1: RssiDetection.h

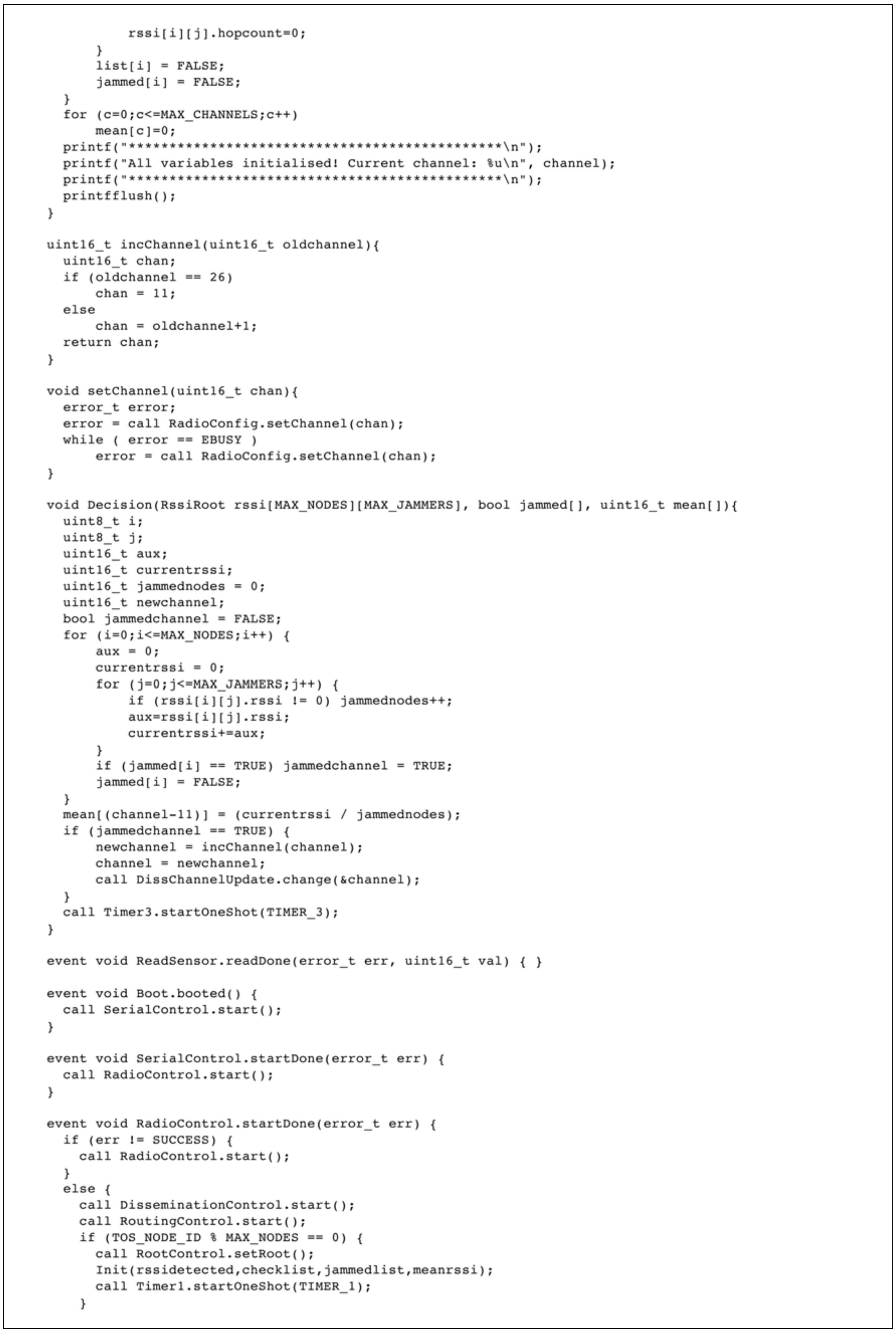


```
#ifndef RSSI_DETECTION_C_H
#define RSSI_DETECTION_C_H


enum {
 AM_TESTNETWORKMSG = 0x05,
 SAMPLE_RATE_KEY = 0x1,
 CL_TEST = 0xee,
 TEST_NETWORK_QUEUE_SIZE = 8,
 MAX_CHANNELS = 16,
 MAX_NODES = 10,
 MAX_JAMMERS = 50,
};

#endif
```

Figure A.2: RssiDetectionC.h

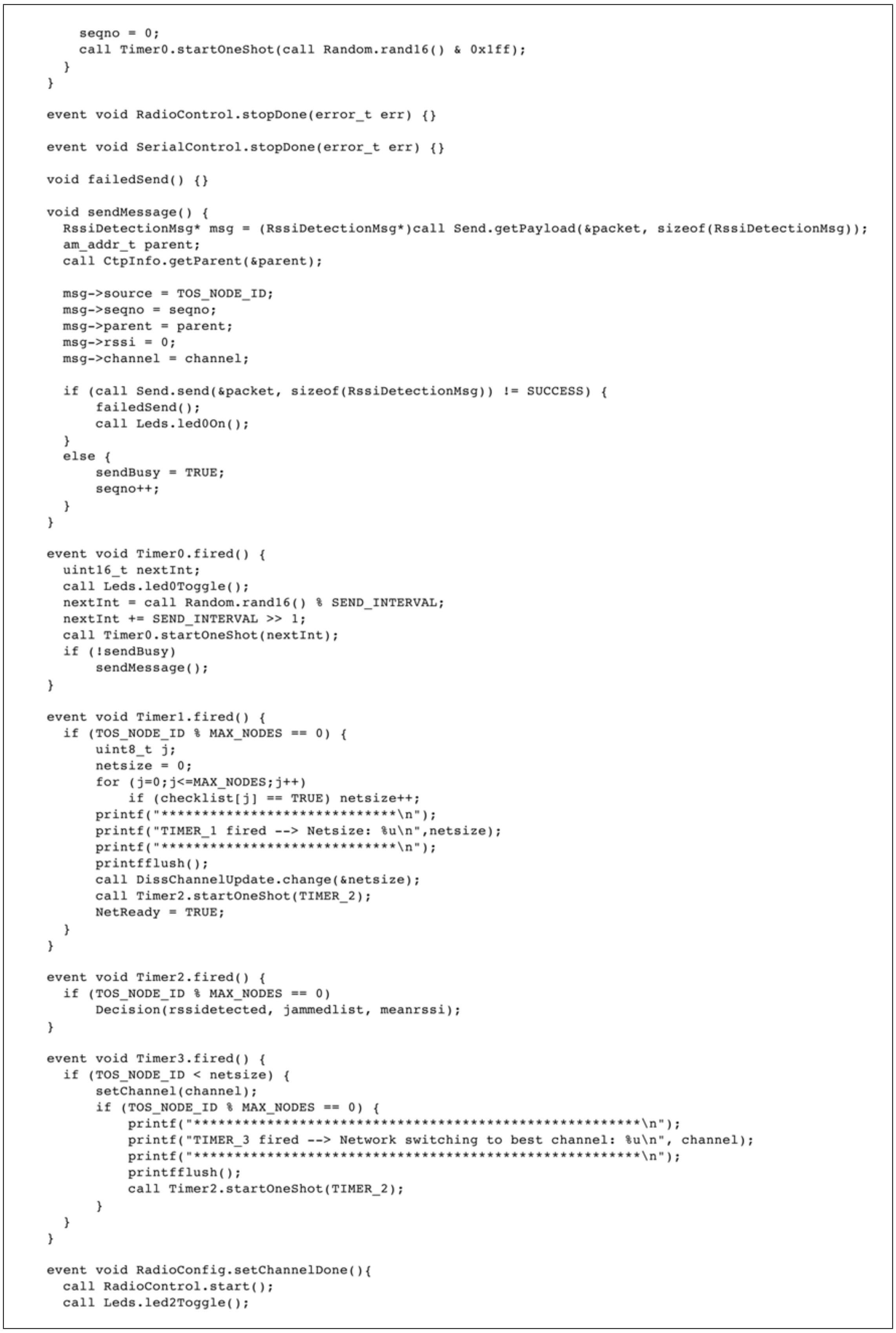
```
#include <Timer.h>
#include <HplRF230.h>
#include "RssiDetection.h"
#include "printf.h"

module RssiDetectionC {
  uses interface Boot;
  uses interface SplitControl as RadioControl;
  uses interface SplitControl as SerialControl;
  uses interface StdControl as RoutingControl;
  uses interface StdControl as DisseminationControl;
  uses interface DisseminationValue<uint16_t> as DisseminationPeriod;
  uses interface DisseminationValue<uint16_t> as DissChannelValue;
  uses interface DisseminationUpdate<uint16_t> as DissChannelUpdate;
  uses interface Send;
  uses interface Leds;
  uses interface Read<uint16_t> as ReadSensor;
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Timer<TMilli> as Timer3;
  uses interface RootControl;
  uses interface Receive;
  uses interface Intercept as Forward;
  uses interface AMSend as UARTSend;
  uses interface CollectionPacket;
  uses interface CtpInfo;
  uses interface CtpCongestion;
  uses interface Random;
  uses interface Queue<message_t*>;
  uses interface Pool<message_t>;
  uses interface AMPacket;
  uses interface Packet as RadioPacket;
  uses interface PacketField<uint8_t> as PacketRSSI;
  uses interface CtpPacket;
  uses interface RF230Config;
  uses interface RadioConfig;
  }

implementation {
  task void uartEchoTask();
  message_t packet;
  message_t uartpacket;
  message_t* recvPtr = &uartpacket;
  uint8_t msglen;
  bool sendBusy = FALSE;
  bool forwardBusy = FALSE;
  bool uartbusy = FALSE;
  bool firstTimer = TRUE;
  uint16_t seqno;
  uint16_t netsize = 0;
  uint16_t channel = RF230_DEF_CHANNEL;
  uint16_t meanrssi[MAX_CHANNELS];
  uint16_t minrssi = 100;
  RssiRoot rssidetected[MAX_NODES][MAX_JAMMERS];
  bool checklist[MAX_NODES];
  bool jammedlist[MAX_NODES];
  bool NetReady = FALSE;
  enum {
    SEND_INTERVAL = 8192,
    TIMER_1 = 10000,
    TIMER_2 = 50000,
    TIMER_3 = 10000,
  };

  uint16_t getRssi(message_t *msg) {
    if(call PacketRSSI.isSet(msg))
        return (uint16_t) call PacketRSSI.get(msg);
    else
        return 0;
  }

  void Init(RssiRoot rssi[MAX_NODES][MAX_JAMMERS], bool list[], bool jammed[], uint16_t mean[]){
    uint8_t i;
    uint8_t j;
    uint8_t c;
    for (i=0;i<=MAX_NODES;i++) {
        for (j=0;j<=MAX_JAMMERS;j++) {
            rssi[i][j].rssi=0;
```

Figure A.3: RssiDetectionC.nc



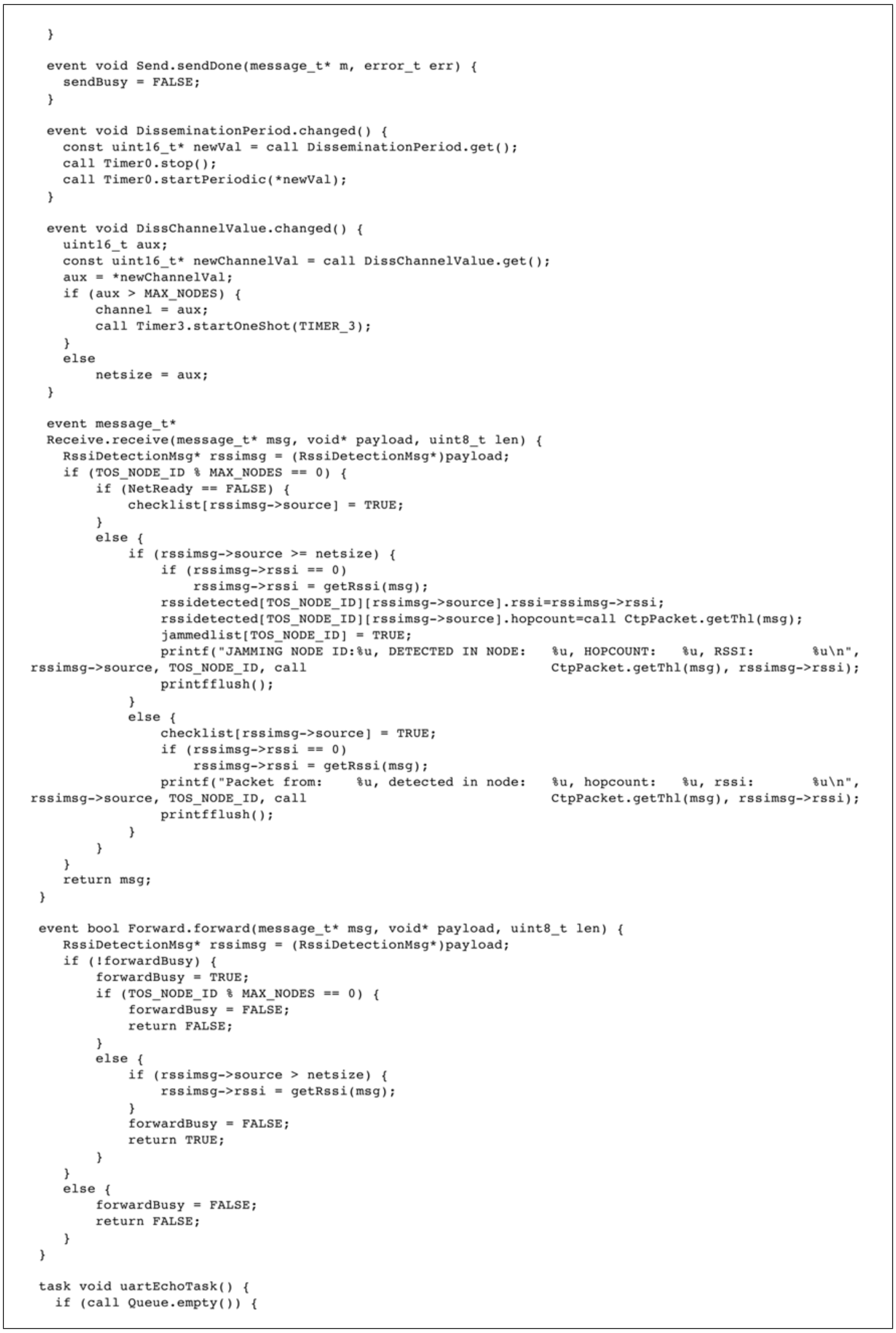
```
          rssi[i][j].hopcount=0;
      }
      list[i] = FALSE;
      jammed[i] = FALSE;
  }
  for (c=0;c<=MAX_CHANNELS;c++)
      mean[c]=0;
  printf("**********************************************\n");
  printf("All variables initialised! Current channel: %u\n", channel);
  printf("**********************************************\n");
  printfflush();
}

uint16_t incChannel(uint16_t oldchannel){
  uint16_t chan;
  if (oldchannel == 26)
      chan = 11;
  else
      chan = oldchannel+1;
  return chan;
}

void setChannel(uint16_t chan){
  error_t error;
  error = call RadioConfig.setChannel(chan);
  while ( error == EBUSY )
      error = call RadioConfig.setChannel(chan);
}

void Decision(RssiRoot rssi[MAX_NODES][MAX_JAMMERS], bool jammed[], uint16_t mean[]){
  uint8_t i;
  uint8_t j;
  uint16_t aux;
  uint16_t currentrssi;
  uint16_t jammednodes = 0;
  uint16_t newchannel;
  bool jammedchannel = FALSE;
  for (i=0;i<=MAX_NODES;i++) {
      aux = 0;
      currentrssi = 0;
      for (j=0;j<=MAX_JAMMERS;j++) {
          if (rssi[i][j].rssi != 0) jammednodes++;
          aux=rssi[i][j].rssi;
          currentrssi+=aux;
      }
      if (jammed[i] == TRUE) jammedchannel = TRUE;
      jammed[i] = FALSE;
  }
  mean[(channel-11)] = (currentrssi / jammednodes);
  if (jammedchannel == TRUE) {
      newchannel = incChannel(channel);
      channel = newchannel;
      call DissChannelUpdate.change(&channel);
  }
  call Timer3.startOneShot(TIMER_3);
}

event void ReadSensor.readDone(error_t err, uint16_t val) { }

event void Boot.booted() {
  call SerialControl.start();
}

event void SerialControl.startDone(error_t err) {
  call RadioControl.start();
}

event void RadioControl.startDone(error_t err) {
  if (err != SUCCESS) {
    call RadioControl.start();
  }
  else {
    call DisseminationControl.start();
    call RoutingControl.start();
    if (TOS_NODE_ID % MAX_NODES == 0) {
      call RootControl.setRoot();
      Init(rssidetected,checklist,jammedlist,meanrssi);
      call Timer1.startOneShot(TIMER_1);
    }
```

Figure A.4: RssiDetectionC.nc



```
      seqno = 0;
      call Timer0.startOneShot(call Random.rand16() & 0x1ff);
    }
  }

  event void RadioControl.stopDone(error_t err) {}

  event void SerialControl.stopDone(error_t err) {}

  void failedSend() {}

  void sendMessage() {
    RssiDetectionMsg* msg = (RssiDetectionMsg*)call Send.getPayload(&packet, sizeof(RssiDetectionMsg));
    am_addr_t parent;
    call CtpInfo.getParent(&parent);

    msg->source = TOS_NODE_ID;
    msg->seqno = seqno;
    msg->parent = parent;
    msg->rssi = 0;
    msg->channel = channel;

    if (call Send.send(&packet, sizeof(RssiDetectionMsg)) != SUCCESS) {
        failedSend();
        call Leds.led0On();
    }
    else {
        sendBusy = TRUE;
        seqno++;
    }
  }

  event void Timer0.fired() {
    uint16_t nextInt;
    call Leds.led0Toggle();
    nextInt = call Random.rand16() % SEND_INTERVAL;
    nextInt += SEND_INTERVAL >> 1;
    call Timer0.startOneShot(nextInt);
    if (!sendBusy)
        sendMessage();
  }

  event void Timer1.fired() {
    if (TOS_NODE_ID % MAX_NODES == 0) {
        uint8_t j;
        netsize = 0;
        for (j=0;j<=MAX_NODES;j++)
            if (checklist[j] == TRUE) netsize++;
        printf("*****************************\n");
        printf("TIMER_1 fired --> Netsize: %u\n",netsize);
        printf("*****************************\n");
        printfflush();
        call DissChannelUpdate.change(&netsize);
        call Timer2.startOneShot(TIMER_2);
        NetReady = TRUE;
    }
  }

  event void Timer2.fired() {
    if (TOS_NODE_ID % MAX_NODES == 0)
        Decision(rssidetected, jammedlist, meanrssi);
  }

  event void Timer3.fired() {
    if (TOS_NODE_ID < netsize) {
        setChannel(channel);
        if (TOS_NODE_ID % MAX_NODES == 0) {
            printf("*******************************************************\n");
            printf("TIMER_3 fired --> Network switching to best channel: %u\n", channel);
            printf("*******************************************************\n");
            printfflush();
            call Timer2.startOneShot(TIMER_2);
        }
    }
  }

  event void RadioConfig.setChannelDone(){
    call RadioControl.start();
    call Leds.led2Toggle();
```

Figure A.5: RssiDetectionC.nc

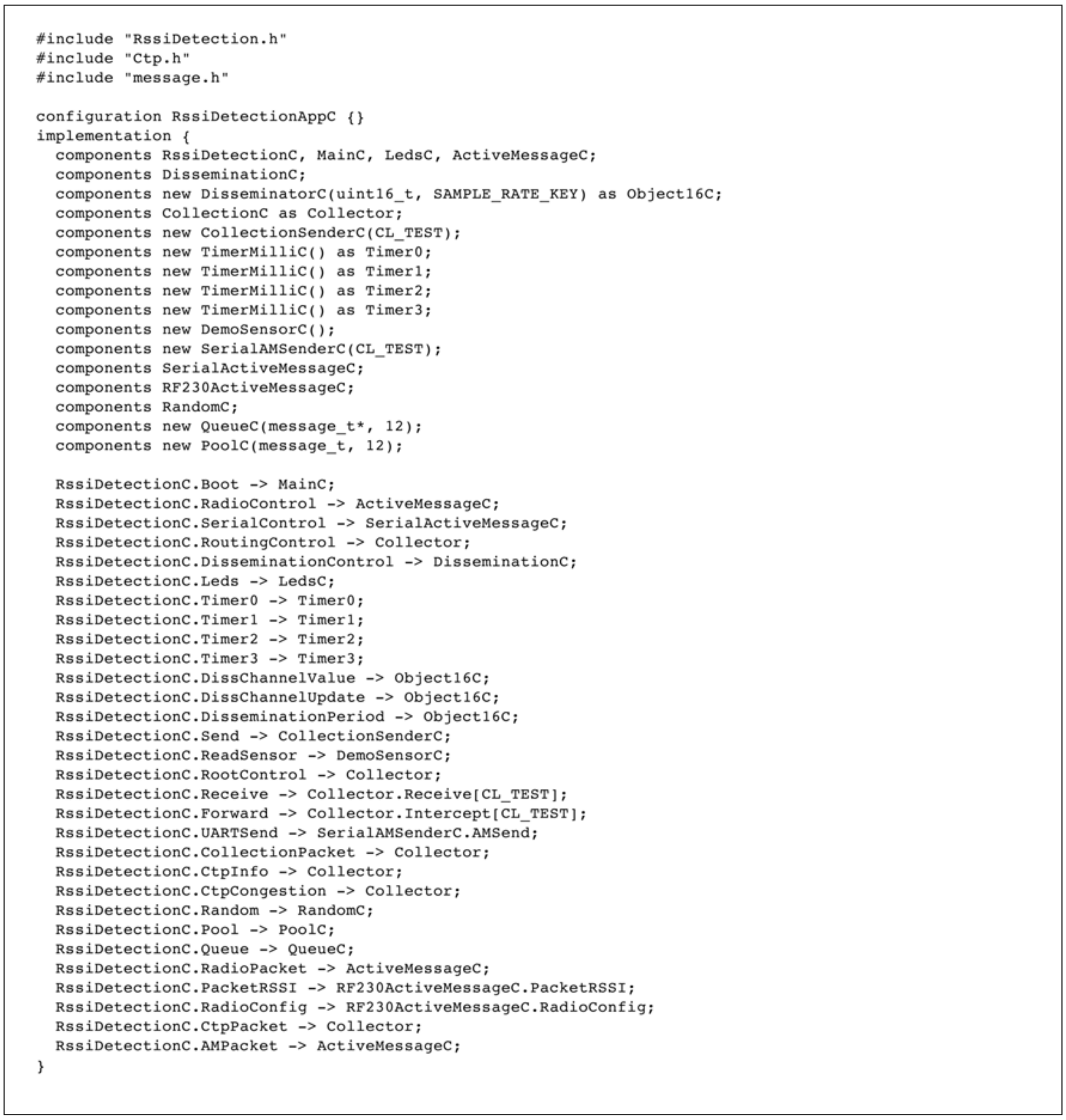


```
  }

  event void Send.sendDone(message_t* m, error_t err) {
    sendBusy = FALSE;
  }

  event void DisseminationPeriod.changed() {
    const uint16_t* newVal = call DisseminationPeriod.get();
    call Timer0.stop();
    call Timer0.startPeriodic(*newVal);
  }

  event void DissChannelValue.changed() {
    uint16_t aux;
    const uint16_t* newChannelVal = call DissChannelValue.get();
    aux = *newChannelVal;
    if (aux > MAX_NODES) {
        channel = aux;
        call Timer3.startOneShot(TIMER_3);
    }
    else
        netsize = aux;
  }

  event message_t*
  Receive.receive(message_t* msg, void* payload, uint8_t len) {
    RssiDetectionMsg* rssimsg = (RssiDetectionMsg*)payload;
    if (TOS_NODE_ID % MAX_NODES == 0) {
        if (NetReady == FALSE) {
            checklist[rssimsg->source] = TRUE;
        }
        else {
            if (rssimsg->source >= netsize) {
                if (rssimsg->rssi == 0)
                    rssimsg->rssi = getRssi(msg);
                rssidetected[TOS_NODE_ID][rssimsg->source].rssi=rssimsg->rssi;
                rssidetected[TOS_NODE_ID][rssimsg->source].hopcount=call CtpPacket.getThl(msg);
                jammedlist[TOS_NODE_ID] = TRUE;
                printf("JAMMING NODE ID:%u, DETECTED IN NODE:   %u, HOPCOUNT:   %u, RSSI:       %u\n",
rssimsg->source, TOS_NODE_ID, call                              CtpPacket.getThl(msg), rssimsg->rssi);
                printfflush();
            }
            else {
                checklist[rssimsg->source] = TRUE;
                if (rssimsg->rssi == 0)
                    rssimsg->rssi = getRssi(msg);
                printf("Packet from:    %u, detected in node:   %u, hopcount:   %u, rssi:       %u\n",
rssimsg->source, TOS_NODE_ID, call                              CtpPacket.getThl(msg), rssimsg->rssi);
                printfflush();
            }
        }
    }
    return msg;
 }

 event bool Forward.forward(message_t* msg, void* payload, uint8_t len) {
    RssiDetectionMsg* rssimsg = (RssiDetectionMsg*)payload;
    if (!forwardBusy) {
        forwardBusy = TRUE;
        if (TOS_NODE_ID % MAX_NODES == 0) {
            forwardBusy = FALSE;
            return FALSE;
        }
        else {
            if (rssimsg->source > netsize) {
                rssimsg->rssi = getRssi(msg);
            }
            forwardBusy = FALSE;
            return TRUE;
        }
    }
    else {
        forwardBusy = FALSE;
        return FALSE;
    }
 }

 task void uartEchoTask() {
   if (call Queue.empty()) {
```

Figure A.6: RssiDetectionC.nc

```
      return;
    }
    else if (!uartbusy) {
      message_t* msg = call Queue.dequeue();
      if (call UARTSend.send(0xffff, msg, call RadioPacket.payloadLength(msg)) == SUCCESS) {
        uartbusy = TRUE;
      }
    }
 }

  event void UARTSend.sendDone(message_t *msg, error_t error) {
    uartbusy = FALSE;
    call Pool.put(msg);
    if (!call Queue.empty()) {
      post uartEchoTask();
    }
  }

}
```

Figure A.7: RssiDetectionC.nc



```
#include "RssiDetection.h"
#include "Ctp.h"
#include "message.h"

configuration RssiDetectionAppC {}
implementation {
  components RssiDetectionC, MainC, LedsC, ActiveMessageC;
  components DisseminationC;
  components new DisseminatorC(uint16_t, SAMPLE_RATE_KEY) as Object16C;
  components CollectionC as Collector;
  components new CollectionSenderC(CL_TEST);
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  components new TimerMilliC() as Timer3;
  components new DemoSensorC();
  components new SerialAMSenderC(CL_TEST);
  components SerialActiveMessageC;
  components RF230ActiveMessageC;
  components RandomC;
  components new QueueC(message_t*, 12);
  components new PoolC(message_t, 12);

  RssiDetectionC.Boot -> MainC;
  RssiDetectionC.RadioControl -> ActiveMessageC;
  RssiDetectionC.SerialControl -> SerialActiveMessageC;
  RssiDetectionC.RoutingControl -> Collector;
  RssiDetectionC.DisseminationControl -> DisseminationC;
  RssiDetectionC.Leds -> LedsC;
  RssiDetectionC.Timer0 -> Timer0;
  RssiDetectionC.Timer1 -> Timer1;
  RssiDetectionC.Timer2 -> Timer2;
  RssiDetectionC.Timer3 -> Timer3;
  RssiDetectionC.DissChannelValue -> Object16C;
  RssiDetectionC.DissChannelUpdate -> Object16C;
  RssiDetectionC.DisseminationPeriod -> Object16C;
  RssiDetectionC.Send -> CollectionSenderC;
  RssiDetectionC.ReadSensor -> DemoSensorC;
  RssiDetectionC.RootControl -> Collector;
  RssiDetectionC.Receive -> Collector.Receive[CL_TEST];
  RssiDetectionC.Forward -> Collector.Intercept[CL_TEST];
  RssiDetectionC.UARTSend -> SerialAMSenderC.AMSend;
  RssiDetectionC.CollectionPacket -> Collector;
  RssiDetectionC.CtpInfo -> Collector;
  RssiDetectionC.CtpCongestion -> Collector;
  RssiDetectionC.Random -> RandomC;
  RssiDetectionC.Pool -> PoolC;
  RssiDetectionC.Queue -> QueueC;
  RssiDetectionC.RadioPacket -> ActiveMessageC;
  RssiDetectionC.PacketRSSI -> RF230ActiveMessageC.PacketRSSI;
  RssiDetectionC.RadioConfig -> RF230ActiveMessageC.RadioConfig;
  RssiDetectionC.CtpPacket -> Collector;
  RssiDetectionC.AMPacket -> ActiveMessageC;
}
```

Figure A.8: RssiDetectionAppC.nc