Minding the spectrum gaps

First steps toward developing a distributed white space sensor grid for cognitive radios

JAVIER LARA PEINADO

KTH Information and
Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Minding the spectrum gaps

First steps toward developing a distributed white space sensor grid for cognitive radios

Javier Lara Peinado

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

11 June 2013

Examiner: Professor Gerald Q. Maguire Jr.

Abstract

The idea that the radio spectrum is growing ever more scarce has become commonplace, and is being reinforced by the recent bidding wars among telecom operators. New wireless applications tend to be deployed in the relatively narrow unlicensed frequency bands, worsening the problem of interference for all users. However, not all frequency bands are in use in every location all the time, creating temporal and spatial gaps (also known as white spaces) that cognitive radio systems aim to take advantage of. In order to achieve that, such systems need to be able to constantly scan large chunks of the radio spectrum to keep track of which frequency bands are locally available any given moment, thus allowing users to switch to one of these unoccupied frequency bands once the current band becomes unusable (or less useful). This requirement of wideband sensing capabilities often translates into the need to install specialized radio components, raising the costs of such systems, and is often at odds with the focus on monitoring the current band as is done by traditional wireless devices.

The goal of this master's thesis project is to simplify cognitive radio systems by shifting the wideband sensing functionality to a specialized and inexpensive embedded platforms that will act as a white space sensor, thus freeing cognitive radio users from this task and making it easier to integrate dynamic spectrum management techniques into existing systems. To do that a wireless sensor gateway platform developed by a previous master's thesis has been repurposed as a prototype white space detector and tested against several wireless transmitters. The aim is to develop a standalone platform that can be deployed all around an area to collect data that can be used to create a geographical map of the use of the spectrum. Such a system should require as little maintenance as possible, thus auto-update and self-configuring features have been implemented in the detector, as well as a simple scanning protocol that allows for remote configuration of the wideband sensing parameters. Furthermore, a basic server has been developed to aggregate and display the data provided by the different sensors.

Sammanfattning

Tanken att radiospektrum blir allt knappare har blivit vardagsmat, och förstärks av de senaste budgivning krig mellan teleoperatörer. Nya trådlösa applikationer tenderar att sättas i de relativt smala olicensierade frekvensband, förvärrade problemet med störningar för alla användare. Men inte alla frekvensband som används i varje plats hela tiden, skapar tidsmässiga och rumsliga luckor (även känd som vita fläckar) som kognitiva radiosystem syftar till att dra nytta av. För att uppnå detta, sådana system måste hela tiden kunna scanna stora delar av radiospektrum för att hålla reda på vilka frekvensband är lokalt tillgängliga varje givet ögonblick, vilket gör omkopplaren när den nuvarande bandet blir obrukbar. Det här kravet på bredbands avkänning kapaciteter översätter ofta in behovet av att installera specialiserade radiokomponenter, höja kostnaderna för sådana system, och är ofta i strid med fokus på övervakning av strömmen band med traditionella trådlösa enheter.

Målet med detta examensarbete är att förenkla kognitiva radiosystem med wideband avkänning funktionalitet till en specialiserad och billig inbäddad plattform som kommer att fungera som ett vitt utrymme sensor, vilket frigör kognitiva radio användare från denna uppgift och gör det enklare att integrera dynamiskt spektrum förvaltning tekniker i befintliga system. För att göra det en trådlös sensor gateway plattform som utvecklats av ett tidigare examensarbete har apterat som en prototyp blanktecken detektor och testas mot flera trådlösa sändare. Målet är att utveckla en fristående plattform som kan sättas runt för att skapa en geografisk karta av användningen av spektrum och kräva så lite underhåll som möjligt, har automatisk uppdatering och självkonfigurerande funktioner implementerats i detektorn, samt som en enkel scanning protokoll som möjliggör fjärrkonfiguration av den bredbandiga avkänningsparametrarna. Dessutom har en grundläggande server utvecklats för att aggregera och visa uppgifterna från de olika sensorerna.

Acknowledgements

Writing a master's thesis is a lengthy business, especially for those that have a tendency to not see the forest for the trees. Therefore, first and foremost my thanks go to my supervisor, Professor Gerald Q. "Chip" Maguire Jr. for pointing the way out of the thicket every time and providing guidance and motivation, as well as a seemingly endless supply of hardware, when I needed it most. I would also like to thank Mats Nilsson, assistant director of the KTH's Centre for Wireless Systems, for allowing me to use his GSM equipment, as well as Professor Mark T. Smith for his advice on the idiosyncrasies of microcontrollers and of course Antonio Estepa for his constant support from the far south. I also thank Albert López and Francisco Javier Sánchez, on whose circuits this thesis stands and Václav Valenta, who generously allowed the reproduction of figure 1.2 on page 3 from [1].

Last but in no way least, I would like to thank my family for their constant support of my endeavors in the far north. They are the reason why I am here, in every sense of the word.

Contents

List of Figures

List of Tables

Acronyms

AGC Automatic Gain Control.

ASK amplitude-shift keying.

BNC Bayonet Neill-Concelman.

BOOTP Bootstrap Protocol.

BSL Bootstrap Loader.

CCS Code Composer Studio.

CR cognitive radio.

DHCP Dynamic Host Configuration Protocol.

DVGA digital variable gain amplifier.

FCC Federal Communications Commission.

FSK frequency-shift keying.

FTP File Transfer Protocol.

GFSK gaussian frequency-shift keying.

GSM Global System for Mobile Communications.

GSM-R Global System for Mobile Communications - Railway.

GUI Graphical User Interface.

HDF5 Hierarchichal Data Format.

IANA Internet Assigned Numbers Authority.

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

ISC Internet Systems Consortium.

ISM Industrial, Scientific and Medical.

JTAG Joint Test Action Group.

LC inductor-capacitor.

LED light-emitting diode.

LNA low-noise amplifier.

MAC Medium Access Control.

MCU microcontroller unit.

MSK minimum-shift keying.

OS operating system.

PAMR Public Access Mobile Radio.

PD Powered Device.

PoE power over Ethernet.

PSE power Sourcing Equipment.

PTS Swedish Post and Telecom Authority.

RAM Random Access Memory.

RF Radio frequency.

RoHS Restriction of Use of Hazardous Substances.

RSSI Received Signal Strength Indicator.

SMA SubMiniature version A.

SPI Serial Peripheral Interface.

SQL Structured Query Language.

SRAM Static Random Access Memory.

SRD short range device.

TCP transmission control protocol.

TFTP Trivial File Transfer Protocol.

TI Texas Instruments.

UDP User Datagram Protocol.

USB Universal Serial Bus.

USRP Universal Serial Radio Peripheral.

Chapter 1

Introduction

In this chapter we provide a brief overview of the objectives and general structure of this thesis, as well as a short description of the issues it aims to address.

1.1 Problem description

Radio spectrum is considered state property by most countries, and thus it is a tightly regulated resource. National governments usually have the exclusive right to decide how the available frequencies are used and, more importantly, who gets to use them. The most widely applied method of spectrum management is known as command and control[2], in which the radio spectrum is partitioned into bands that are then allocated to specific services such as fixed satellite or TV broadcast, often on a exclusive basis. This strategy ensures that authorized operators are able to take advantage of their allocation of the spectrum without fear of interference, but is also inherently inefficient and slow to adapt as it is a centralized top-down process, this leads to a low occupancy of the radio channels[3].

Another consequence of the aforementioned static allocation process is to reinforce the notion of spectrum scarcity. A quick glance at a spectrum allocation chart such as the one portrayed in figure 1.1 on page 3 conveys the idea of a crowded radio spectrum, with little elbow room left for new wireless applications. Each colored block in that chart represents an allocated band, one for which the regulator has already determined who is allowed to use it and what for, and it is clear to see that the radio spectrum is almost completely allocated. However, the picture changes when one considers that not all frequency bands are in use everywhere and all the time, that is to say, the spatial and temporal gaps or white spaces in the spectrum. For example, in 2002 the United States of America's Federal Communications Commission (FCC) Spectrum Policy Task Force issued a report[4] about the utilization of the radio spectrum, supported

by measurements in several urban areas, where it concluded that that there were widespread inefficiencies in many frequency bands. However, the report also stated that there was a rather large temporal and spatial variability in spectrum use, which led to its recommendation of encouraging geographically limited licenses and improving the time-sharing of the bands among multiple users. In the report's words, "*taking advantage of time and space*". Furthermore, it suggested restricting the command and control management method to a few critical bands and transitioning to more flexible models which would allow unlicensed users to transmit in a given band provided they took care not to interfere with the band's assigned primary user.

More recent studies reveal that spectrum utilization has not changed much since the FCC issued its recommendations. A measurement campaign performed in several European cities between 2008 and 2009[1] found significant reuse opportunities in many frequency bands, as well as a low overall utilization. That can be clearly seen in figure 1.2 on page 3, where the spectrum seems a calm sea of low-energy blue with occasional bursts of power here and there, mainly in the bands used by mobile services such as GSM. Not surprisingly, this European study also suggests spectrum sharing as a solution for underutilized frequency bands.

Therefore, instead of spectrum scarcity the real problem we have to contend with are the inefficiencies in spectrum utilization; for which spectrum sharing, either geographically or temporally, is the commonly proposed solution. This approach is known as spectrum-sensing cognitive radio (CR) and will be explained in more detail in section 2.1. However, sharing the spectrum efficiently requires that there be devices to periodically collect information about the ambient radio environment in order to keep track of which channels are occupied[2], as well as to compute statistics based on past observations of which are likely to be freed up in the near future. Given this information a the transmitter can make a fast handoff to an available frequency band if it needs to move out of its current one, such as it might occur because of activity by a licensed user.

The catch lies in that implementing those features would require specialized hardware and software, as well as distract the resources of the communication devices from their main tasks (supporting their user's communication needs). Doing so would increase their cost and complexity, thus slowing down the deployment of CR systems.

Additionally, there is the problem of the hidden listener - a listener who is near the potential transmitter who is listening to a signal from a remote transmitter using this frequency band. This issue will be addressed in a separate master's thesis by Julia Alba Tomo Peiro.

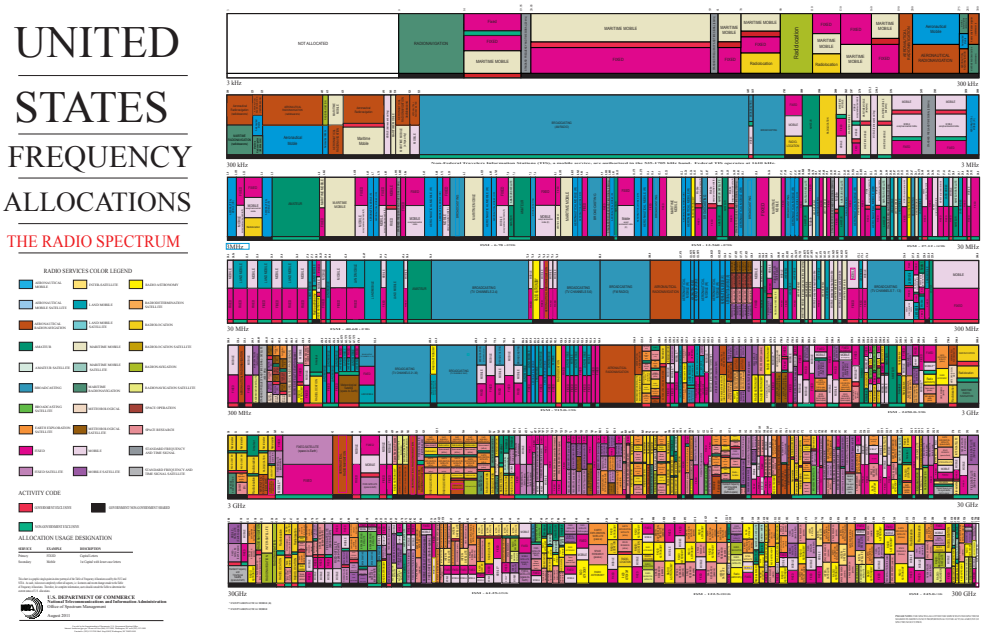


Figure 1.1: US radio spectrum allocations chart, 2011 edition[5]

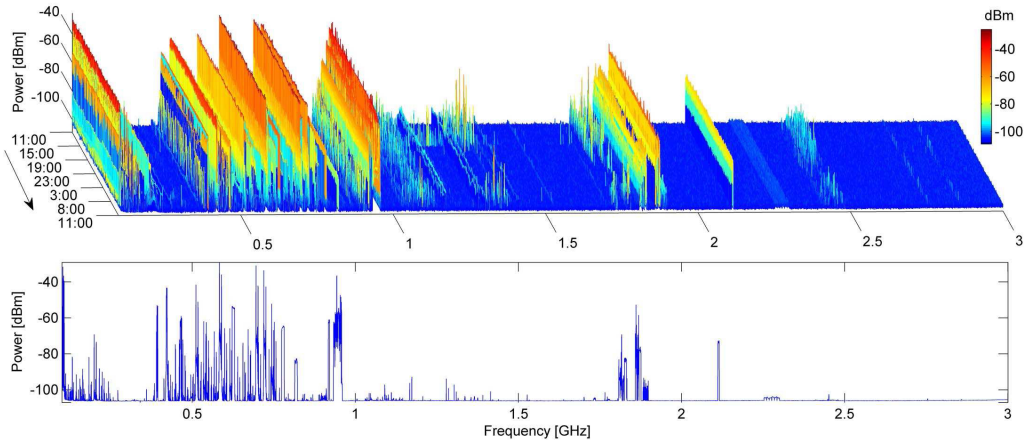


Figure 1.2: Plot of the 24-hour maximum spectrum usage in Brno[1]

1.2 Goals

The aim of this project is to facilitate the development of CR systems by offloading the task of scanning the spectrum and computing the availability statistics of each channel to a grid of specialized microcontroller platforms. This way CR devices could simply query a server which collected the information from the grid to get all the current information about the spectrum, thus reducing the complexity of the CR devices and extending their spatial sampling range. The grid would be comprised of "white space" sensors, capable of spotting the temporal and spatial gaps in the use of the frequency bands and reporting them. Therefore, the final objective of this project is to develop the software and hardware for a prototype white space sensor grid and test it to prove its feasibility.

1.3 Structure of this thesis

This thesis is divided into five chapters, which loosely mirror the development process of the master's thesis project itself. Chapter 1 gives a general overview of the problem that motivates this thesis, and chapter 2 sums up the information we had to collect to address it. Equipped with this new knowledge, chapter 3 explains the methods and tools we chose to achieve our goals and chapter 4 described how we tested the prototype to see if it sufficed to fulfill all our aims. As we did not manage to achieve all of them, chapter 5 summarizes our conclusions and describes what was left to do and what should be improved and suggests directions for further work in this area.

Chapter 2

Background

In this chapter we will provide a brief overview of some concepts useful to understand the rest of this thesis, as well as a survey of the related work in this area.

2.1 Introduction to Cognitive Radio

A cognitive radio (CR) is a transceiver capable of being aware of the characteristics of its ambient radio environment and of modifying its own transmission or reception parameters to adapt to changes in this radio environment and the tasks that the user wishes to achieve, with the aim of allowing as many simultaneous wireless conversations as possible in a given location and frequency band[6]. The concept was proposed for the first time by Joseph Mitola III and Gerald Q. Maguire, Jr. in their 1999 article[7], and has become a hot research topic within the field of dynamic spectrum management.

The ultimate goal of CR is to share the available radio spectrum as efficiently as possible. The most common approach is to allow unlicensed users to transmit in a given band provided that they do not interfere with the primary licensed, either by jumping to a gap in another band when the primary user resumes transmitting or by modifying the CR's modulation scheme, a process illustrated in figure 2.1. To achieve this adaptive operation a CR device has to go through a so-called *cognitive cycle*, which is comprised of the following steps[8]:

1. Spectrum sensing: In this step the device scans the different spectrum bands in search of gaps, detects them and stores their location information.

2. Spectrum analysis: After detecting spectrum holes, the device studies them in search of patterns and estimates their main traits.

3. Spectrum decision: Given all the information about the available white spaces in the spectrum, the device determines the best data rate, transmission mode, and bandwidth for each possibility, and chooses the most suitable based on the user quality of service requirements and the spectrum characteristics.

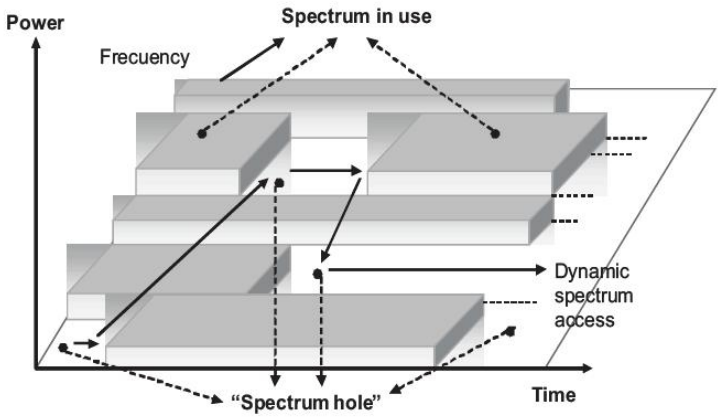


Figure 2.1: Spectrum gap concept

In this thesis project we will concern ourselves mostly with the spectrum sensing step, so we will elaborate a bit more on it. The main issue of this CR feature is how to detect whether a frequency band is in use or not. Many techniques have been developed to tackle that problem, which can be loosely grouped into two categories[3]:

- Transmitter detection: In this approach the cognitive radio classifies a band as occupied if it determines that the primary user is locally present in a certain spectrum. For such detection there are basically three different procedures:

  - Matched filter detection: A matched filter is obtained by correlating a known signal, or template, with an unknown signal to check if it contains the template. This is the optimal signal detection approach, and it has the advantage of also being the fastest, although it requires demodulation of the primary user's signal and maintaining a database of templates (typically preambles, pilots, synchronization words or spreading codes) for each frequency. Another drawback is that it would require the cognitive radio to realize a receiver for each primary user class.

  - Energy detection: This is the simplest of all three approaches, and also the most inefficient. It involves measuring the amount of energy detected in a particular band and checking if it exceeds a threshold, in which case we assume the channel to be occupied. However, it has many drawbacks, mainly that it has no way of distinguishing between noise, interference, and modulated signals; is vulnerable to noise and interference-induced changes in the threshold and is incapable of detecting spread spectrum signals; whose power levels all below that of the thermal noise. On the other hand, it is the easiest to implement and it requires no knowledge of the primary user's characteristics for each band, so we will use this method for the white space sensors of our grid.
  - Cyclostationary-feature detection: This technique attempts to detect a signal by looking for periodically repeated features, which are induced by the sine waves, pulse trains, spreading sequences, or cyclic prefixes with which they are generally coupled. To do that, the device has to compute the correlation between the signal and time-shifted versions of itself, and compare the result with a known pattern. This approach has the advantage of removing the effects of random noise and interference, which should exhibit no periodicity, and thus is more effective than a simple energy detector.

- Cooperative detection: In this scheme a number a sensors are distributed in different locations and share their spectrum scanning information. Each individual sensor can use any of the approaches stated above, but their collaboration makes it more likely that the primary user will be spotted. By distributing the sensors around a large area we compensate for the problem of individual sensors located in the shadow of the primary user's transmission. In this project we will use this approach to construct the sensing grid, while the individual sensors will use the energy detection technique.

2.2 TFTP

Trivial File Transfer Protocol (TFTP) is a simplified version of File Transfer Protocol (FTP) which was originally designed as a lightweight file transfer protocol that could fit in memory-constrained chips and thus be suitable for bootstrapping diskless workstations, i.e., computers that obtain all their software upon startup over the network using a small built-in program[9]. This is precisely

the functionality we are looking for, so TFTP is one of the main components of the network bootloader whose development is detailed in section 4.2.

Unlike FTP, TFTP has a limited amount of options and runs over UDP instead of transmission control protocol (TCP), which reduces the code complexity of its implementation, as it does not need to handle connections or transmission windows. Therefore, when we talk about ”connections” in TFTP we are referring to the logical flow of the conversation between client and server, without the requirement to establish a connection at the transport layer as is done with FTP. Regarding its architecture, TFTP is client/server based, and divides the process of transmitting a file into three major phases:

1. Initial connection: The TFTP client sends a connection request message from a pseudo-random port number to the server, which listens on the well-known port 69. In this message the client states the file name, whether it wants to perform a read or write operation and the desired transfer mode. In turn, the server replies with an acknowledgment, thus opening the connection, also sent from a pseudo-random port. These ports are used as identifiers of the conversation between client and server and avoid the need for an extra identification field in the TFTP message format, which reduces the protocol's overhead.

2. Data transfer: Once the conversation starts, the server transmits the file to the client in chunks of 512B of data, numbered sequentially starting from 1. Then end of the transfer is marked by the arrival of a message with less than 512B of data, indicating that the server has no more full chunks left to send. The client acknowledges each message as it arrives.

3. Connection termination: The conversation ends when the client acknowledges the last message, without requiring an explicit message.

As its name implies, TFTP is a really simple protocol. It offers virtually no error control, so any problem during the connection establishment will cause the whole process to start from scratch, a small sacrifice to maintain its lack of complexity. TFTP also includes a basic retransmission mechanism: if a packet gets lost in the network, its intended recipient will timeout and may attempt to retransmit the last packet (be it data or acknowledgment), which warns the sender of the missing message.

2.3 Related work

In this section we will provide a brief overview of the body of research on which this master's thesis is based, as well as the projects that share significant features

with ours.

2.3.1 Cooperative spectrum sensing

As stated in section 2.1 while discussing the different spectrum sensing schemes, our project will be based on cooperative spectrum spectrum sensing using the energy detection approach for the individual sensors. This is in no way a novel take on the subject, and there already exists a considerable body of research about the topic. A theoretical comparison of the leading detection mechanisms can be found in [10], and there are a host of other papers proposing a bewildering variety of detection schemes and optimizations, but only a few go beyond computer simulations to test their detection models in the field with real hardware. The most complete of those is [11], which uses Universal Serial Radio Peripherals (USRPs) as the sensor nodes and transmitters at 1.298GHz. In that paper a cooperative detection scheme with three sensor nodes individually using the cyclostationary feature detection is proven superior to just one sensor alone relying on the same approach. The cooperative scheme is simple but effective: choosing the largest out of all the spectral correlation densities reported by the sensors. It also verifies that the cyclostationary scheme can distinguish between two signals with the same frequency but with different modulations and it is robust to local oscillator frequency drifts.

However, to our knowledge no practical cooperative spectrum sensing grid has been implemented for the 790-928MHz range, the range in which our RF daughterboards operate, and we have not been able to find any implementation of cooperative schemes that used the energy detection approach. Therefore, this may add some new details to the topic.

2.3.2 White space databases

Besides detecting the spectrum gaps through our sensor grid, in this project we store such data in a central server, which allows us to compute the statistics about the geographical and temporal distribution of the white spaces. The idea is that any wireless application could request such information by querying our server. This is known as a “white space database”, a service that has received a lot of attention in recent years due to important changes of the spectrum regulations, particularly in the US.

In 2010 the FCC released a ruling that made available the TV white spaces for unlicensed broadband wireless devices[12]. To avoid transmitters interfering with the locally available TV stations, it required them to rely on a white space database. This way the ruling obviated the need for spectrum sensing at the device

level, placing the burden on the database instead, which should be kept up-to-date and certified by the FCC. This regulation has generated plenty of interest in the research and implementation of white space databases, with companies like Google[13] and Microsoft[14] developing their own prototypes.

Most of the implementations currently available use a Structured Query Language (SQL) database as the backend to store the white space data for ease of development: there are many mature SQL implementations that include the capability for remote querying. This way the transmitting devices willing to take advantage of the white spaces only need a simple SQL client to obtain the data. The disadvantage of this approach is that SQL is a relational database language, which is not really suited for handling large amounts of numerical data. This is the issue that motivated us to reject SQL in favor of the Hierarchichal Data Format (HDF5) data model, as will be described in section 4.4.1.

Chapter 3

Method

In this chapter we will set the specific objectives of this thesis and introduce the tools and techniques used to achieve and verify them.

3.1 Objectives

As we stated in section 1.1, the apparent spectrum scarcity can be mitigated if we take advantage of the temporal and spatial gaps in the use of the different bands. However, to do that efficiently, wireless devices would need wideband scanning capabilities to detect the available bands at a given moment, as well as a basic database with statistical information of past scans to make reasonable predictions about which bands would be free in the near future. Placing such requirements on already constrained platforms would increase their cost and complexity, while also taking away valuable resources from their main tasks. Therefore, in this thesis we propose another approach, where all the spectrum-sensing tasks are offloaded to a grid of dedicated white space sensors, which send the data to a central server that aggregates the sensor information into a temporal and geographical map of the spectrum. This system's topology is illustrated in figure 3.1. In this approach wireless devices only have to make query the server to learn all the spectrum data, which only requires them to have basic networking capabilities.

The aim of this master's thesis is to show that such an approach is reasonable and relatively easy to implement. In order to do that, we will repurpose an embedded platform, developed as a wireless sensor sniffer as part of a previous master's thesis project[15], to act as a prototype white space sensor, scanning the spectrum and conveying the information to a central server. A prototype of this server will also be developed during this project, and we will use it to coordinate a grid of sensor nodes and calculate the utilization statistics of the spectrum in a limited area.

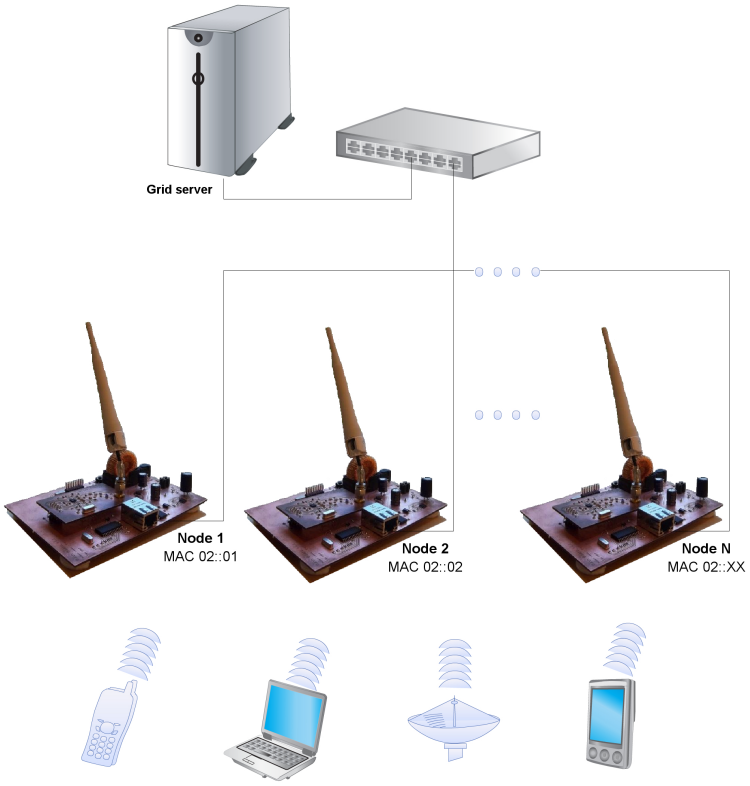


Figure 3.1: Topology of the white space sensor grid

As a proper sensor grid would imply deploying a large number of boards, the sensor nodes have been designed to be as plug-and-play as possible. They will get everything they need through the Ethernet cable, including power, code, network configuration and the spectrum scanning settings. The user simply plugs an Ethernet cable into the sensor node and forget about the sensors. Each sensor node which will receive code updates automatically via the network, avoiding the hassle of manually programming each device, thus adding a lot of flexibility to the grid. Additionally, this feature allows the devices to carry out new functions or to support changes in the radio receiver daughterboard.

To achieve that, in this project we will develop a network bootloader for a power over Ethernet (PoE)-capable embedded platform and a simple UDP-based protocol to convey the scan options and data, allowing the boards to be fully configured via the network from the central server. Furthermore, we will develop a graphical interface for the server to allow the user to easily change the sensor node's settings and display the information.

3.2 Description of the embedded platform

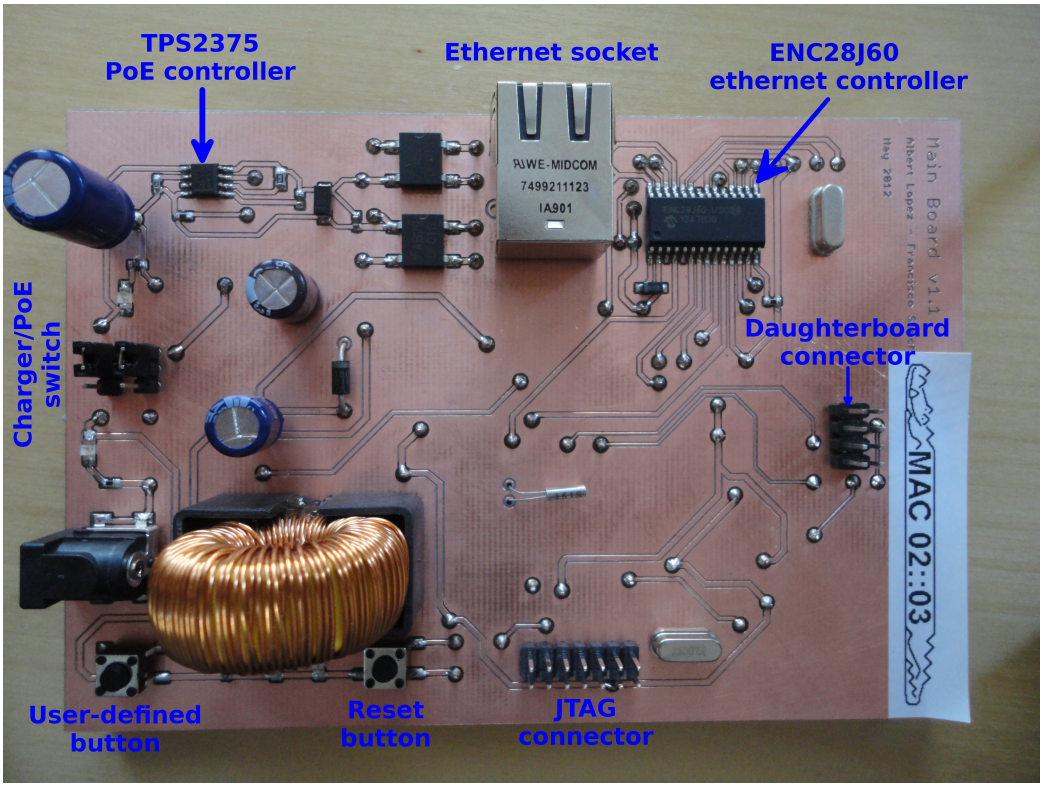
In this section we will provide a brief description of the embedded platform developed as a wireless sensor sniffer by a previous master thesis project[15]. We refer the reader interested in a more thorough explanation of this hardware to this earlier thesis. The platform is designed in a modular way, divided into a motherboard and a plug-in RF daughterboard, handling all the radio-related tasks. This provides a lot of flexibility, as the platform can change which frequency range it works in simply by switching the daughterboard to a compatible one. Additionally, because the motherboard communicates with the RF daughterboard via a serial interface, it is possible to have multiple radios on one daughterboard or to connect multiple daughterboards in a chain.

3.2.1 Motherboard

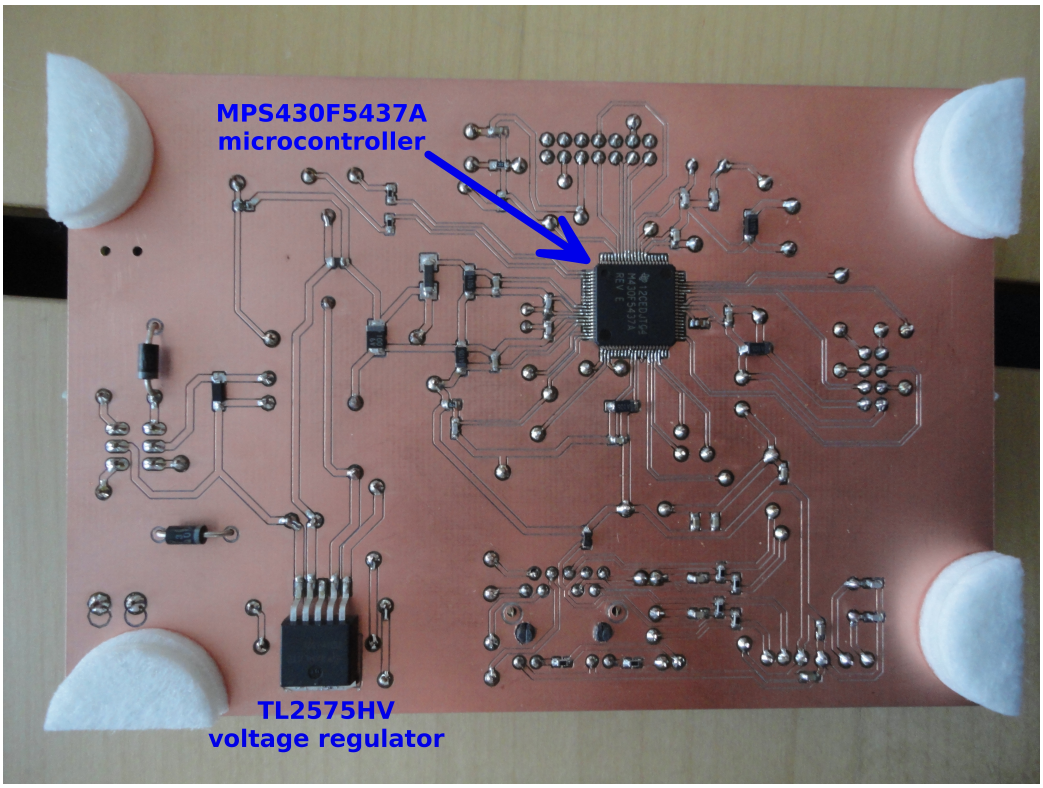
As illustrated in Figure 3.2, the motherboard hosts the powering, computing, and fixed networking sections of the embedded platform. This motherboard can be powered either by a external DC power supply or via PoE, and the user can select between the two options by changing the position of two jumpers. The voltage regulator circuit has a TL2575HV step-down converter[16], so the board can work with virtually any DC-supply that provides between 3.3 and 60V, which is the upper voltage limit of the converter chip. However, PoE is the more convenient energy source of the two, as only an Ethernet cable is needed for the board to work, so the user does not have to provide a power socket for each board or deal with bulky chargers. The disadvantage is that it requires the boards to be directly connected to power Sourcing Equipment (PSE), typically a switch or a router that is PoE-capable.

The signaling required for PoE is handled by the Texas Instruments (TI) TPS2375[17], a chip that fully supports the IEEE 802.3af Specification, the standard for PoE. This chip takes care of informing the PSE of the power requirements of the board using PoE power classes. Currently the motherboard is configured to advertise the board as a class 1 Powered Device (PD), the category that has the lowest maximum power(3.84W)[18], but still way more than the embedded platform requires, as all its chips were selected for their low-power characteristics.

Besides power, the other functions of the platform are overseen by the MSP430F5437A[19], a microcontroller from TI's MSP430 ultra-low power microcontroller unit (MCU) family. It provides many low-power modes to optimize its energy consumption by activating only the modules required at each moment, two independent Serial Peripheral Interfaces (SPIs) which we will use to connect it to the transceiver and the Ethernet controller, and two programming



(a) Front



(b) Back

Figure 3.2: View of the motherboard with the relevant components highlighted

interfaces: BSL and JTAG, although only the JTAG connector is included in the board, and therefore JTAG we will use to load the bootloader code.

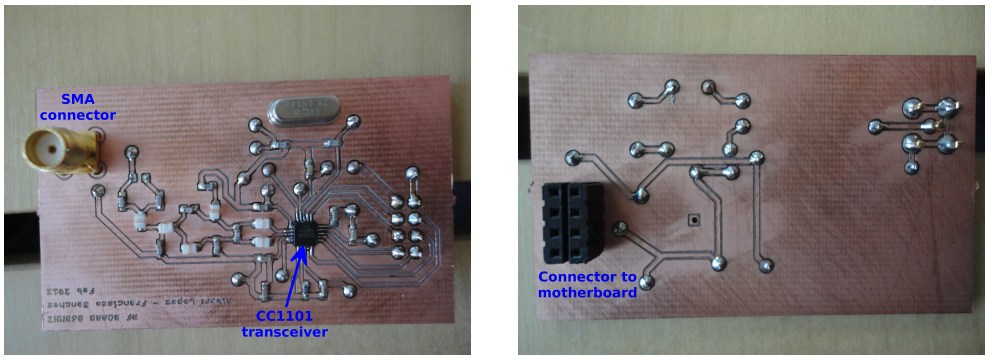
Regarding the processor's memory resources, this version of the chip has a relatively low amount of Static Random Access Memory (SRAM) (only 4KB), but plenty of flash (256KB), which means that we will be able to store large amounts of code but we must be very careful with this code's Random Access Memory (RAM) requirements. The MCU will configure the other chips and process their data, communicating with them via SPI, and also run the TCP/IP stack that enables networking.

The chip that is in charge of the actual transmission and reception of Ethernet frames is a Microchip ENC28J60 Ethernet controller[20]. This chip is connected to the MCU through a standard SPI, and has a 8KB transmit/receive buffer where it stores incoming frames from the SPI and the Ethernet port. This chip handles all the Ethernet layer functions, automatically encapsulating the IP packets received from the MCU via SPI into an Ethernet frame. It is important to note that this chip lacks a factory-defined unique Medium Access Control (MAC) address. This value should be configured by the MCU when initializing the device, and we have to take care to avoid MAC conflicts in our network. For our prototypes we have manually assigned MAC addresses to each of the nodes with the 02:00:00:00:00:XX format, which falls within the locally administered MAC range. A manufacturer would normally assign a fixed MAC address to each device it produces from a block of addresses bought from the Institute of Electrical and Electronics Engineers (IEEE).

3.2.2 RF daughterboard

The daughterboard contains all the wireless-related components: the transceiver, a passive inductor-capacitor (LC) filter and the SubMiniature version A (SMA) socket for the antenna. The transceiver is linked to the motherboard's MCU through a standard SPI, which simplifies the design of compatible daughterboards. Currently there is only one type of daughterboard available, designed around TI's CC1101 wireless transceiver[21], which is displayed in figure 3.3.

The CC1101 is a low-power RF transceiver manufactured by TI which is designed to operate in the 300-348 MHz, 387-464MHz, and 779-928MHz frequency ranges, which include the license-free short range device (SRD) and Industrial, Scientific and Medical (ISM) bands. However, the filter circuitry used in the RF daughterboard is optimized for the 779-928MHz range, so this version of the board will only be able to efficiently scan that frequency band. That choice was made because originally the board was intended as a sniffer for wireless sensor networks[15], which use the the SRD-800 band, located in the 863-870MHz range in Sweden and most of Europe[22].



(a) Front (b) Back

Figure 3.3: View of the daughterboard with the relevant components highlighted

To complete its functionality the daughterboard requires a suitable antenna. For this project we used Smarteq's MiniMag 1140.30SMA model[23], which is designed for GSM900 systems operating in the 824-960MHz range. This antenna covers most of the frequency bands of our current daughterboard's transceiver.

The motherboard can support any daughterboard compatible with its connector and whose transceiver has a SPI. In fact, the whole CC11xx/CC25xx RF transceiver family of TI, offers the same interface and pinout, and contains devices able to operate in a wide variety of frequency ranges, from SRD-800 to the WiFi, Bluetooth, or Zigbee bands. Furthermore, all of these devices present an almost identical programming interface, so switching from one RF daughterboard can be done with minimal changes to the code. Therefore, it would be reasonably straightforward to expand the sensor's scanning range with different RF daughterboards.

3.3 Hardware tools

In this section we will briefly introduce the hardware tools other than the sensor node platform that were useful for the development of this project.

3.3.1 JTAG USB debugging interface

As stated in section 3.2.1, our embedded platform has a JTAG connector for programming its MCU. This interface provides nice debugging functionalities, allowing the programmer to step through code and set breakpoints, which will be quite useful during software development. To connect this JTAG interface to a standard computer we use a MSP-FET430UIF debugger[24]. This is a flash

emulation tool that includes a USB debugging interface to program MCUs of the MSP430 family via JTAG or Spy Bi-Wire, and is well supported by the CCS Integrated Development Environment (IDE), introduced in section 3.4.1

3.3.2 GSM tester

Before deploying the sensor grid, we needed to verify that each node is capable of scanning the spectrum accurately, a task for which the HP8922M GSM tester[25] displayed the lower half of figure 3.4 came in handy. Though this device is mainly designed to check the functionality of GSM equipment, in this project we will employ its RF generator mode. In this mode the tester is capable of producing a wide range of RF signals from 10MHz to 1GHz with a resolution of 1Hz and amplitude up to -7dBm, so it covers the whole frequency range of our RF daughterboard. We will use this signal generator to make sure that the sensor nodes correctly detect known signals and to calibrate their frequency response, by connecting them to the GSM tester using a coaxial cable with Bayonet Neill-Concelman (BNC) connectors and a BNC-to-SMA adapter.

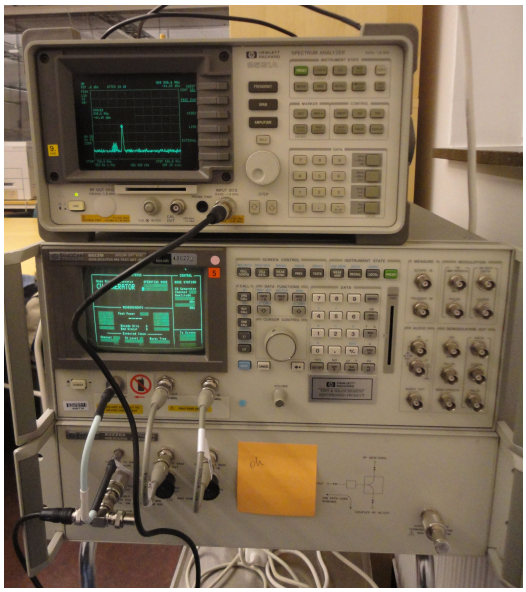


Figure 3.4: Spectrum analyzer and GSM tester rack used for this project

3.3.3 Spectrum analyzer

To compare the scan results of the board with that of a calibrated instrument, we will use the HP8591A spectrum analyzer shown in figure 3.4 (it is placed above

the GSM tester). This spectrum analyzer is capable of scanning the spectrum from 9kHz up to 1.8GHz, and will be used to check that the antennas connected to the boards work correctly and that the spectrum scans performed by the sensors make sense and are accurate. Additionally, this spectrum analyzer be used to display the test RF signals generated by the GSM tester so that they can be easily compared with the white space grid results.

3.3.4 Wireless thermometer

To check that our grid is capable of detecting real-life RF transmitters, we will take advantage of the wireless thermometer set studied in a previous master's thesis project[15], the TFA 'Wave' with radio-controlled clock(catalog part number 30.3016.54.IT)[26]. The wireless thermometer combines a display (on an alarm clock) and a wireless temperature transmitter that uses the 868MHz SRD band to convey its information. As this signal falls withing our RF daughterboard's range, we will use it to test the transceiver's sensitivity.

3.4 Software tools

In this section we will provide an overview of the main software tools we employed during this master's thesis.

3.4.1 CCS

To program our MCU, the MSP430F5437A, we will use the CCS IDE[24] from TI, the manufacturer selected for most of our embedded platform's chips, including the microcontroller. This software supports all TI devices of the MSP430 family, includes drivers for TI's USB FET debuggers (such as the MSP-FET430UIF described in section 3.3.1), provides useful code libraries, and has a rather intuitive and powerful debugging interface. This tool is based on the widely used Eclipse IDE, so anyone familiar with that will quickly get used to this tool. In short, CCS has everything required for easy one-click programming of an MSP430-based system, except for its price. The free versions of this software are either limited in code (up to 16KB) or in time (only usable for 180 days), so before committing ourselves to this IDE we had to check that our program would not exceed the code size limit to avoid getting stuck. As we wanted all the sensor nodes' software modules to be as small (in code size) as possible, we adopted this IDE.

3.4.2 Wireshark

This project involved a lot of networking, thus to troubleshoot the behavior of the sensor node's TCP/IP stack we relied on Wireshark[27]. Wireshark is one of the most popular packet analyzers available. It is free and open-source, and it was specially useful to verify the custom UDP protocol we developed (see section 4.3.1).

Chapter 4

Implementation and Analysis

In this chapter we will explain the steps taken to fulfill the objectives set in section 3.1, which involved a variety of programming. All the source code is made publicly available via a github repository located in `https://github.com/cazulu/mind-the-gaps.git`, whose structure is detailed in appendix A.The interested reader is referred to the code documentation in each part of the program for in-depth implementation details.

This chapter begins by describing in section 4.1 the limited TCP/IP that was developed for this project, specifically a UDP/IP stack with Bootstrap Protocol (BOOTP) and TFTP support. Section 4.2 describes the bootloader that was developed for the UDP/IP stack described in the previous section. The frequency scanning application is described in section 4.3. This frequency scanning code sends UDP packets to a grid server as described in section 4.4. Sections 4.5 describes the initital testing of the sensor nodes together with the grid server. Finally, section 4.6 describes some initial tests of the whole system with a number of sensors nodes.

4.1 TCP/IP stack porting

In order to convey the frequency scanning data to the server, providing networking capability to the board is a must. As mentioned in section 3.2.1, the embedded platform has the hardware required to handle a Ethernet socket, so we only need to add some code to process the raw Ethernet packets. The easiest way to achieve this is use a TCP/IP stack, a software package that implements the internet and transport layers of the TCP/IP model, and UDP. In this section we will describe the steps followed to port a TCP/IP stack to the embedded platform introduced in section 3.2

4.1.1 Choosing a stack

There is a wide variety of existing TCP/IP stacks intended for embedded systems, and we wanted to pick the most suitable stack, so we conducted a brief survey of the open source code options. The features that we were looking for were:

- Support for the hardware of our sensor: Ideally the TCP/IP stack should be designed to run on top of a MCU of the MSP430 family and provide a driver for the ENC28J60, the Ethernet controller chip used in the board. Failing that, the stack should be as hardware-independent as possible to ease porting.
- Support for Dynamic Host Configuration Protocol (DHCP), UDP, and TFTP: As we will see in the following sections, these protocols are needed for the different modules of the program, and we wanted to avoid having to implement them from scratch.
- Small memory and code footprint: The MCU used in our board, the MSP430F5437A, has only 256KB of flash memory and 16KB of SRAM, and the TCP/IP stack code size should be small enough to fit inside the flash and require as little RAM as possible. After all, networking is only one of the functions that the MCU will have to perform, so the TCP/IP stack should not hog all the available resources.
- Good documentation and online support.

For this survey we only considered open source and easy to tune stacks with an active developer community. Additionally, we tried to stay away from those that were only a part of a larger embedded operating systems (OSs). The only exception to this later constraint was Contiki[28], a lightweight OS built around the uIP TCP/IP stack, which we included in our comparison due to its popularity and small code size. The results of this survey are summarized in table 4.1. For the evaluation of the memory requirements of each option we used its developers' estimation of its size with TCP disabled, as our project only requires UDP for the transport layer.

Out of all the stacks evaluated, only Contiki and the stack developed by Microchip meet the RAM constraints of our platform. However, Contiki is a full-fledged OS, and that shows in its larger code footprint. Also, unlike the Microchip stack, Contiki does not contain a TFTP implementation. The main advantage of Contiki is that it provides support for our board's MCU, whereas the Microchip stack does not. However, in the end we concluded that porting the Microchip TCP/IP stack would be less troublesome, as it was developed and maintained by the same company that manufactured our board's Ethernet controller, so we chose it to realize our project's networking module.

Table 4.1: Comparison of the UDP/IP stacks for embedded systems

TCP/IP stack	MSP430 support	ENC28J60 driver	DHCP	UDP	TFTP	Code footprint	RAM requirements
Contiki	Yes	Yes	Yes	Yes	No	40KB	2KB
lwIP[29]	No	No	Yes	Yes	No	40KB	20KB
Microchip[30]	No	Yes	Yes	Yes	Yes	15KB	3KB
FNET[31]	No	No	Yes	Yes	Yes	30KB	20KB

4.1.2 Porting

Before loading the Microchip TCP/IP stack, we had to adapt its code to our platform's MCU, because the stack was developed to run on Microchip's PIC microcontroller family. However, the stack was also designed to be reasonably hardware-independent, which in practice meant that we only had to modify the Ethernet driver and the stack's timing module, while leaving the bulk of the code intact.

Also, it should be noted that we did not port all the features included in the stack, as we were only interested in its UDP/IP functionality. For example, we did not port the random number generator module it includes to provide support for TCP and some application-level protocols, and we also did not port the IPv6 version of the stack. Therefore, we ended up with a simple UDP/IP stack that only supports IPv4, which was exactly what we needed.

4.1.3 Testing

To check that the porting was successful, we created a simple program that requests an IP address using the stack's DHCP client, after which it goes into an endless loop where it blinks the light-emitting diodes (LEDs) and polls the stack. For this test we ran the latest stable version (4.2.5rc1) of Internet Systems Consortium (ISC)'s standard DHCP server[32] on a separate computer within the same subnet, and pinged the board after starting it. Figure 4.1 displays a wireshark capture of such test, where it can be seen that the board replies without any problem to the ICMP Echo requests sent by the server, indicating that the networking module of our platform is already up and running.

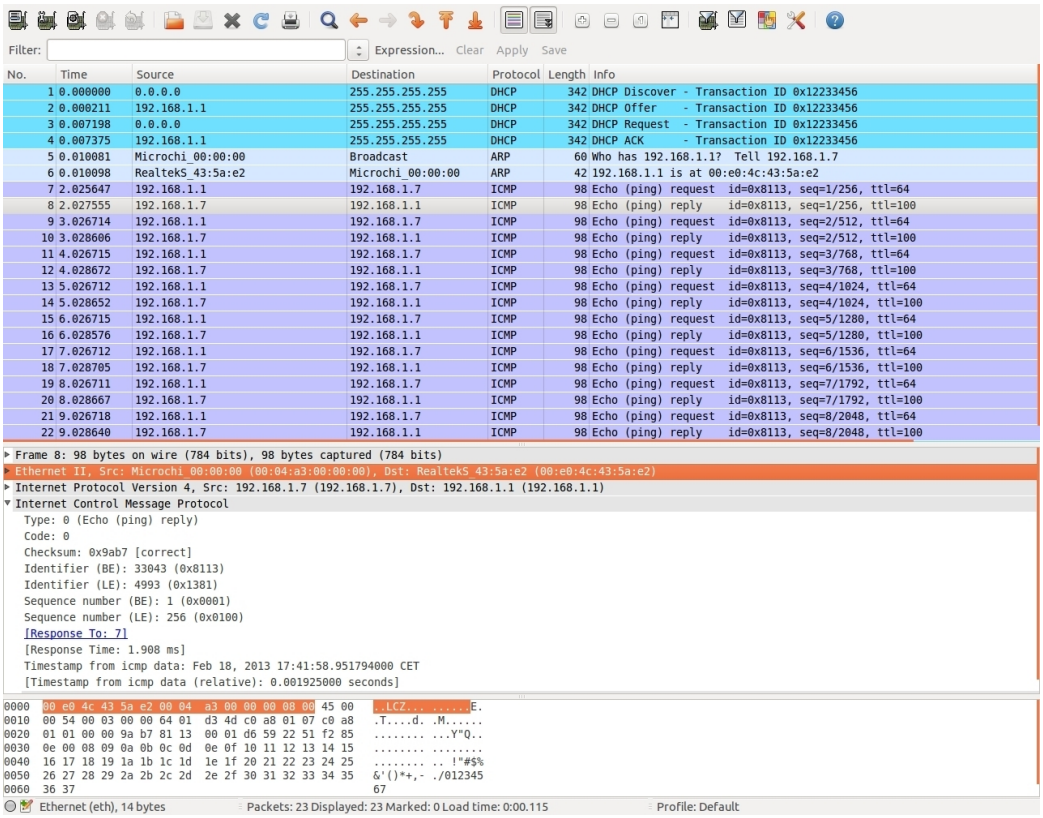


Figure 4.1: Wireshark capture of a ping test with the board

4.2 Network bootloader

As we mentioned in section 3.1, we wanted the nodes of the grid to be able to obtain all they required through the Ethernet cable, including their code. To achieve that, we needed to develop a network bootloader for our embedded platform, whose implementation process is described in this section.

A network bootloader is a piece of software that executes upon startup and obtains from the network the code required for the actual operation of the system. This way the boards need to be flashed by hand with the debugging interface described in section 3.3.1 only once, to install the bootloader. Subsequently the board can load any further software updates via the network. Using this approach saves the user the trouble of having to reprogram the whole grid manually if a new version of the code becomes available. Most importantly for the programming effort, the ability to simply boot the device again and have it get the latest version of the software that it was to accelerated the software development process.

4.2.1 Implementation

Now that we have checked that the TCP/IP stack works properly, we can start building on top of it. Our network bootloader relies on the protocols already provided by the for communication. This can be seen in the simplified flowchart of the network bootloader displayed in figure 4.2.

To keep the network bootloader as simple as possible, and yet compatible with the widely used infrastructures of DHCP and TFTP, we decided to base the bootloader on TFTP, a file transfer protocol often employed for this purpose. We provided a brief overview of this protocol in section 2.2. As explained in section 4.1.1, the Microchip TCP/IP stack already contained an implementation of this protocol. Our bootloader program takes advantage of that functionality to request a bootfile, i.e., the code to be executed, from a TFTP server.

However, in order to do this the bootloader needs to know the TFTP server's IP and the name of the bootfile to ask for. To reduce our program's dependence on a hardcoded configuration, the bootloader gets this information, as well as the board's IP address, via DHCP. For the bootfile name and the TFTP server's IP address we used DHCP's options 67[33] and 150[34], respectively. Furthermore, we also used DHCP's option 13 to transmit the size of the boot file as a multiple of 512B, which is useful to check if we have enough flash memory to store it before initiating the transfer. As none of these options were included in the Microchip TCP/IP stack implementation of the DHCP client, we had to extend the stack to add support for these extra options.

Once the bootloader obtains each chunk of the bootfile, it parses the file to extract the machine code and the memory positions where it has to store

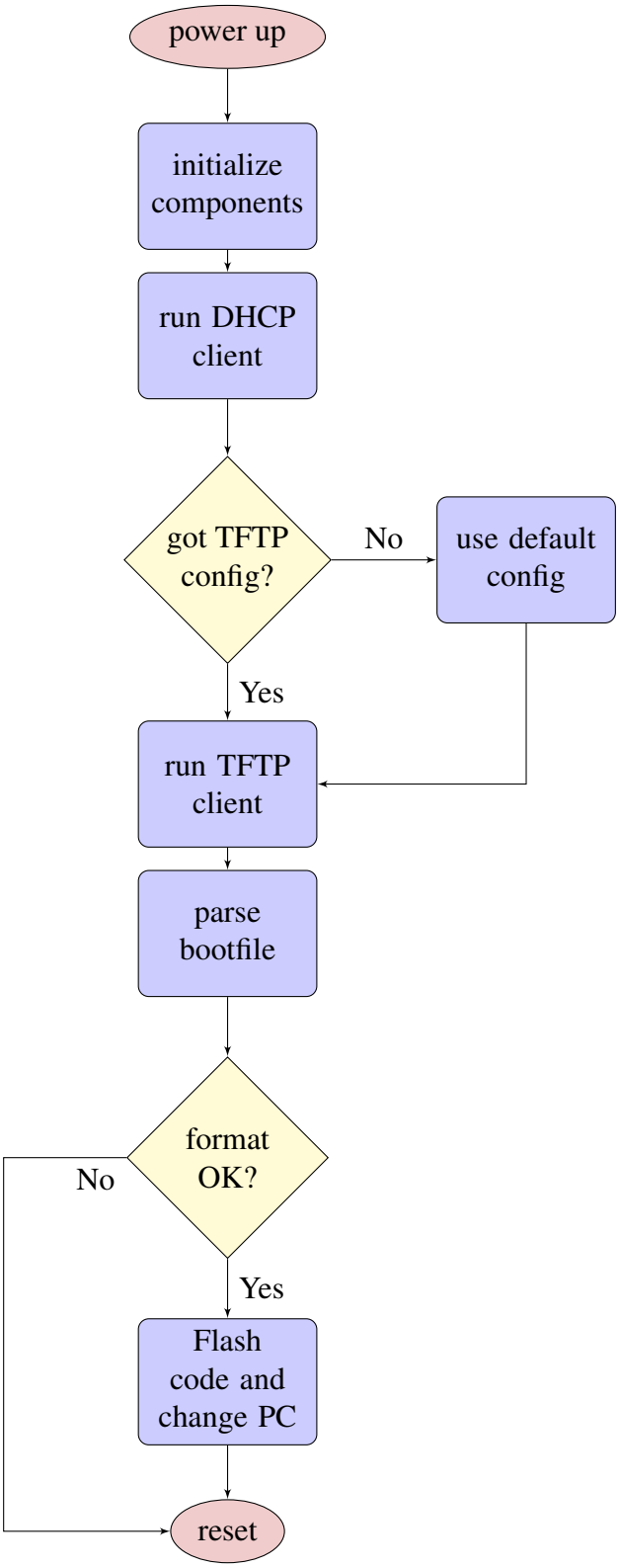


Figure 4.2: Flowchart of the network bootloader

this into the microcontroller's memory using the MCU's flash self-programming feature[35]. Currently the bootloader only supports the TI-TXT file format, as this format can be generated automatically by the CCS IDE after compiling the code. Also, due to the small amount of flash memory of our MCU (only 256KB), we cannot afford to first store the file and then parse it, as representing the binary data in the TI-TXT format expands it approximately three times from its original size[36]. To overcome this problem our parser processes the TFTP data chunks as they arrive, which only requires the storage of 512B (TFTP's default chunk size) at a time. This is where knowing the size of the bootfile as a multiple of 512B via the DHCP option 13 comes in handy, as the knowledge allows us to estimate the final size of the code in memory, as well as the number of chunks the parser has to analyze.

After a successful parsing of the bootfile, the bootloader triggers a software reset of the embedded platform, which launches the newly downloaded code. The user application can invoke the bootloader anytime after this by modifying the reset vector to point to the start of the bootloader and performing a software reset. Currently there is no mechanism for the TFTP server to inform the sensor grid that there is a new version of the code available, so for now the boards must poll the server by triggering the bootloader to check for updates.

To verify that the network bootloader works correctly, we used it to install a simple LEDs-blinking application to the board and we associated its user-defined button with a function that triggered the bootloader. The results of a Wireshark capture of such a test are displayed in figure 4.3, where it can be seen that the TFTP transfer was successful.

4.2.2 Caveats

An important detail to keep in mind when using the network bootloader is that the memory spaces assigned to the bootloader and the user application obtained via TFTP must never overlap. This restriction is necessary in order to prevent the bootloader's code from being overwritten. In its current version, the bootloader does not check if the newly arrived code fulfills that condition, so it is up to the user to make sure that his application falls within the acceptable range as shown in the memory map of figure 4.4. The bootloader takes up approximately 14KB, and is located at the start of the flash code memory space, which leaves around 248KB for the user application.

It is important to note that the user code cannot be arbitrarily located in the processor's address space, due to a quirk of the MCU's addressing system: the interrupt vectors, which store the memory address of the routine to be invoked when a particular interruption is triggered, are only 16 bits long, and that includes the reset vector. This implies that any memory address above FFFFh cannot be

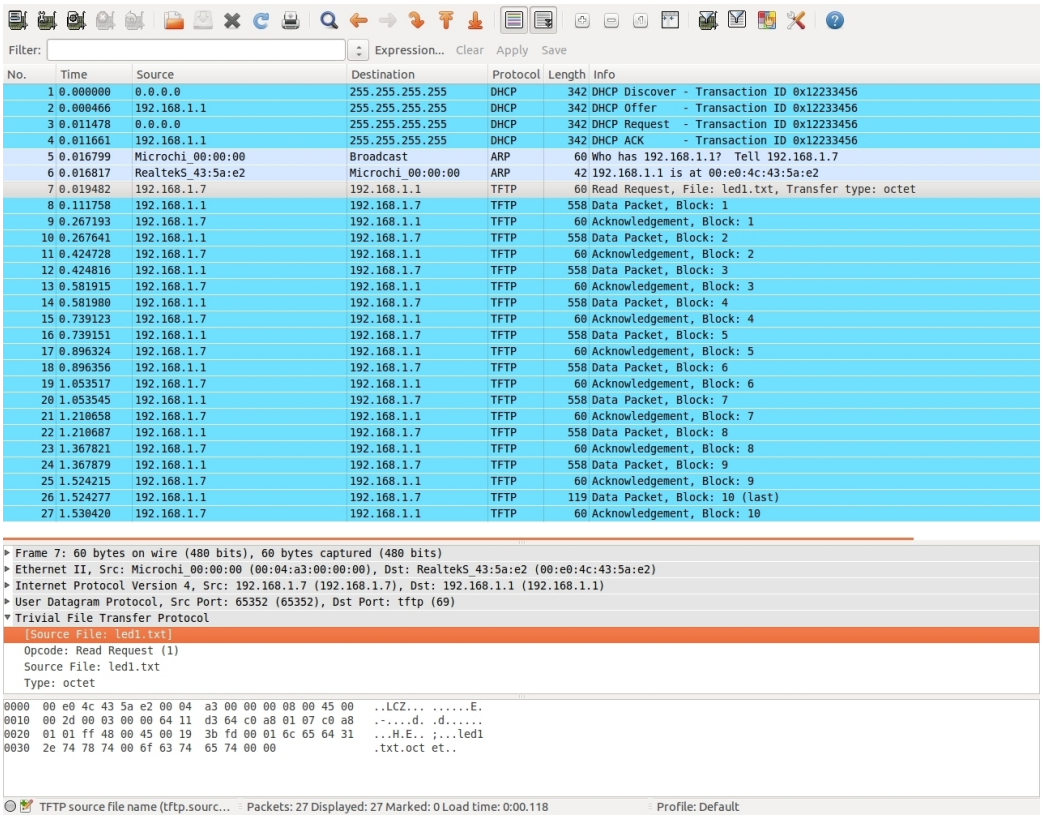


Figure 4.3: Wireshark capture of the network boot

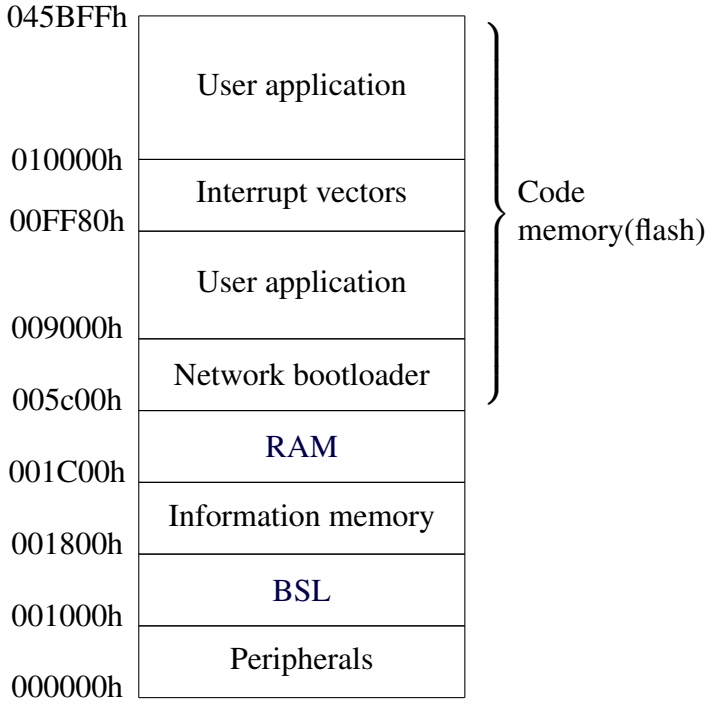


Figure 4.4: Memory map of the MSP430F5437A used for this project

stored in the reset vector.For this reason the user should make sure that at least the beginning of its application lies below that address in order for the switch from the bootloader via the reset vector to work correctly.

Finally, the user is advised to be very careful when updating the network bootloader. The MSP430F5xx MCU family, to which our board's microcontroller belongs, has a paranoid software feature which can prevent the microcontroller from being programmed via the JTAG interface: the JTAG Lock key[37]. If a value other than 0 or 0xFFFFFFFF is programmed anywhere between the 0x17FC and 0x17FF memory addresses, the JTAG interface locks itself. This feature is meant to protect code loaded into flash memory from competitors. An addressing bug while using the flash self-programming mechanism can easily lead to that situation, and indeed that happened several times during this project.

Fortunately, there's a way out: even with the lock activated, the microcontroller remains accessible through the BSL programming interface, which can be used to erase those memory locations and restore JTAG functionality. However, the BSL interface was not included in the design of the board, so we were forced to deploy a makeshift interface by attaching wires to some microcontroller ports, a process detailed in appendix B.

4.3 Frequency scanner

Now that the networking and code update modules are up and running, we can move on to implementing the main aspect of our sensor nodes: their frequency scanning capabilities. As we stated in section 3.2.2, the daughterboard of our embedded platform contains a RF transceiver, the CC1101, capable of working in any frequency band between 779 and 928MHz, which we will program to behave as a cheap and highly configurable spectrum analyzer.

4.3.1 Protocol

The CC1101 was designed as a transceiver, so it can only operate in one radio channel at a time, but it has a wide range of signal processing options, including many modulation schemes. However, that the feature interests us the most is its ability to report an estimate of the signal power level in the channel it is currently operating in, i.e., the Received Signal Strength Indicator (RSSI). According to the datasheet[21], this value is given in dBm with a $\frac{1}{2}$dB resolution in the $[-128, 0]$ dBm range, more than enough for our purposes.

Therefore, to turn the CC1101 into a flexible spectrum analyzer we need to determine the main configuration options affecting the RSSI and develop a simple protocol to set them via the network from the server, as well as a program to configure the transceiver from the MCU to sequentially scan all the desired bands. Looking at the datasheet and TI's Design Note 505[38], which deals with the RSSI interpretation and timing, we found that the main settings are:

- Base frequency of the frequency synthesizer: Controls the transceiver's carrier frequency and thus the center frequency of the band we are currently listening in. The CC1101 works with any value between 779 and 928MHz, with a minimum step size of about 400Hz.

- Receiver channel filter bandwidth: Sets the width of the band centered around the expected carrier frequency whose RSSI we will evaluate and therefore defines the resolution of the spectrum scan. This parameter is dependent on the crystal oscillator frequency and admits only 20 discrete values. In the case of a 26MHz crystal, which is the one used in our RF daughterboard, the valid channel bandwidth values lay between 58 and 812Hz.

- Amplifier gain setting: The CC1101 has three amplifier stages: two low-noise amplifiers (LNAs) and a digital variable gain amplifier (DVGA). In normal operation, the values of these three amplifiers are adjusted automatically by the Automatic Gain Control (AGC) module, whose

purpose is to ensure that they are do not saturate, so the demodulator can work correctly. The AGC behaves as a control loop that keeps the signal level in the demodulator constant and reports the RSSI. Though not recommended by the manufacturer, the AGC feature can be disabled to set the amplifier values manually[39].

- RSSI sampling time: The AGC module constantly reports the current RSSI, but until the control loop stabilizes such values are meaningless. Therefore we need to wait for the control loop to settle before reading the RSSI and moving on to the next band to scan. TI provides an statistical model in [38] to calculate the average waiting time by taking into account factors such as the signal baud rate and the channel spacing. In general, with the AGC enabled waiting 1ms is sufficient, and if we disable the AGC the sampling time goes down to 100us, meaning that we can speed up the scan but have a greater risk of saturating the amplifier. This can be useful if we are only interested in finding out if the channel is busy, instead of obtaining an accurate power level measurement.

- Modulation format: The expected modulation of the incoming signal. The CC1101 supports the amplitude-shift keying (ASK), 2-frequency-shift keying (FSK), 4-frequency-shift keying (FSK), gaussian frequency-shift keying (GFSK) and minimum-shift keying (MSK) formats.

To be able to change those parameters at runtime from the scanner server and to report the RSSI values from the nodes, we developed a simple custom protocol that runs over UDP. The frame structure is shown in figure 4.5. The frame header contains the board's identification (which includes the board's MAC address and is used by the server to tell the different sensor nodes apart), the scan options, and the RSSI data. Using the MAC as an identifier has the advantage of avoiding conflicts between boards with the same IP addresses located in different networks, but has the drawback of requiring a unique MAC per board. The board's Ethernet controller MAC has no factory-defined MAC, so its value has to be hardcoded and configured when initializing the chip.

Currently the protocol uses the an unassigned UDP port number (9930), though for the production version of the grid a Internet Assigned Numbers Authority (IANA)-assigned port number should be applied for. The protocol has the same frame format for both the server commands and the scanner nodes' replies. Once the node has completed a full scan of its configured spectrum range, each sensor node sends a frame to the server indicating its current scan parameters and the resulting RSSI array starting from the lowest frequency band. To change the scanning parameters, the server only has to send a frame with the desired configuration and an empty data field.

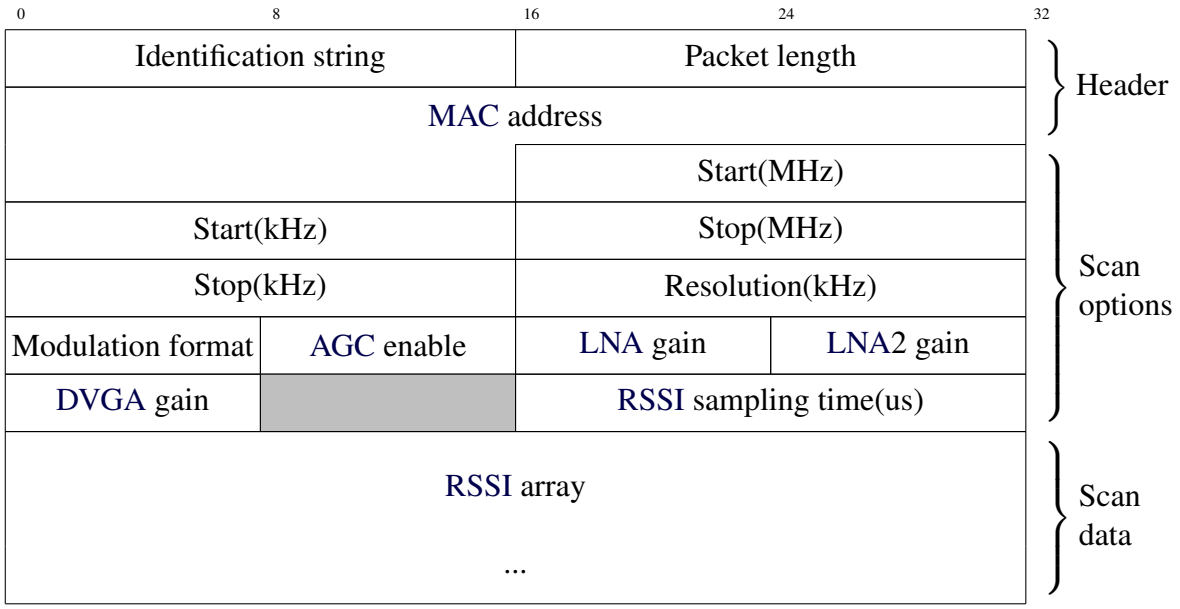


Figure 4.5: Custom UDP protocol to convey the scanning data and options

It should be noted that the current version of the scanning protocol is very simple and has more overhead than is needful. The protocol fields' lengths were chosen to reduce the amount of programming required for the sensor nodes, which contain a 16-bit MCU. This is the reason why the length of all the fields is a multiple of 8 bits, so there is quite a wide margin for optimization. Furthermore, the protocol lacks a version number field and another one to state the type of daughterboard in use, which should be added in future versions of the program.

4.3.2 Implementation

As for the frequency scanning program itself, the program was developed based on the information in TI's Design Note 508[40], which provides the pseudocode to extract the RSSI values of the CC1101 transceiver's whole frequency range, although with fixed settings. Therefore we modified the code to allow for the parameters to be configured and adapted the code to our MCU using TI's library for interfacing the CC11xx/CC25xx transceiver families and the MSP430 microcontrollers via SPI[41].

Like all the previous code, our frequency scanner builds on top of our ported UDP/IP stack. Upon startup each sensor node runs the stack's DHCP client to get an IP address, and from then on assumes that the DHCP server is also hosting the scanner server, and will direct all the spectrum scan results to it (future versions

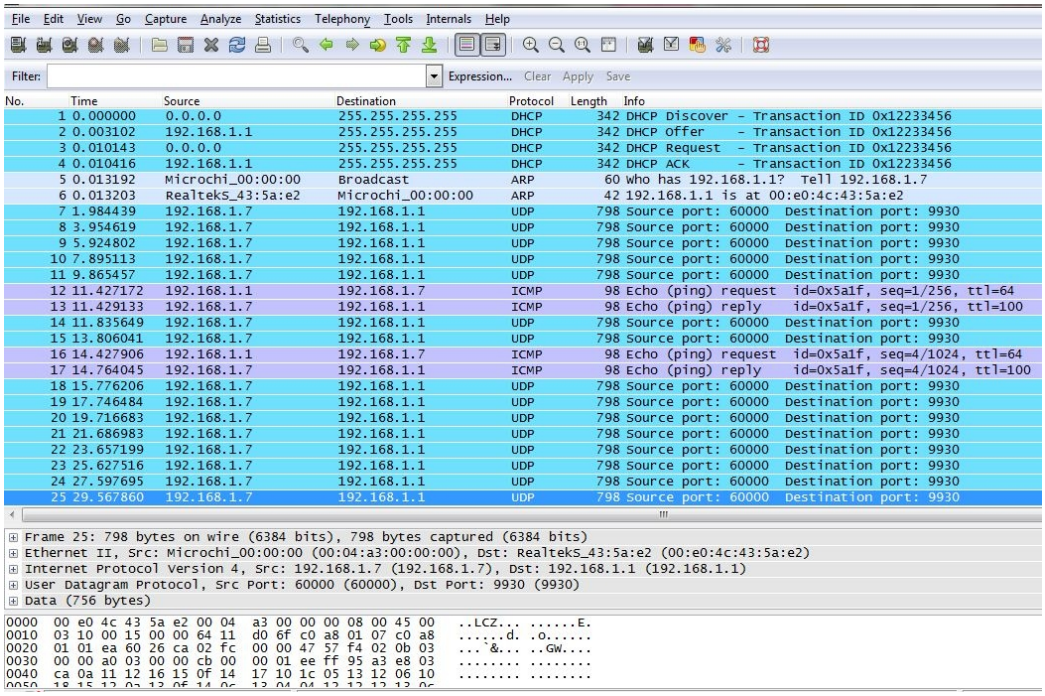


Figure 4.6: Wireshark capture of the UDP scan protocol

of the program should consider using DHCP Log Server option[33] to get around this limitation). This avoids hardcoding those IP addresses into the program and allows for a more flexible design. The only hardcoded value in the program is the node's MAC address.

After obtaining its IP address via DHCP, each node will start analyzing the spectrum and sending the results to the server, while polling the scanner's UDP port (9930) for configuration frames. If no UDP datagrams containing parameters are received, then each node will use the default settings for scanning. This default behavior can be seen in figure 4.6, which displays a wireshark capture of a board powering up and initiating a spectrum scan. Note that the address of the server is 192.168.1.1 in this example and that the node is able to reply to ICMP Echo requests sent by the server during the scan, indicating that the board remains responsive.

4.4 Grid server

Now that all the board-related code is in place, the only piece of software remaining to deploy the grid is the scanner server, which has to receive and aggregate all the spectrum information provided by the different nodes, store this

information into a database, and displaying it for the user. This program was written in Python for portability and ease of development. The program is divided into two separate modules: a backend that collects the data and a GUI that renders the data into graphs and allows the user to change the scanning parameters.

4.4.1 Backend

The goal of the backend is to store large amounts of sensor node data that originate in the grid in an easily readable format, so that this data will be available for later analysis, and to compute some basic spectrum statistics. To deal with the potentially huge dataset we opted to use the HDF5[42], a very flexible data model developed explicitly to manage large numerical data collections. Among its many nice features are the transparent handling of file compression and its table-like storage approach, which simplifies indexing and makes it particularly well suited to time series data such as the RSSI arrays we obtain from the sensors (each of which is associated with a period of time during which this data was collected).

Furthermore, HDF5 is a mature project with plenty of support, and a powerful Python wrapper is readily available: PyTables[43]. We used PyTables to handle all the HDF5-related parts of the backend. PyTables includes multiple compression formats in addition to those of vanilla HDF5, and is optimized to reduce memory and disk usage during operation. These optimizations are quite important for the long term operation of the server.

Taking advantage of these libraries, our backend receives data frames from the sensors through a UDP socket, extracts the scan data, and saves it in a compressed HDF5 file, along with a timestamp of the UDP packet's arrival time. To organize the data from all the nodes, the backend maintains a separate table for each board, labeled with the sensor node's identification data (MAC and IP addresses) and the current scan configuration. HDF5 allows POSIX-like /path/to/resource syntax for its resources, so once the data file is open, the tables of the sensors can be accessed as files under a common directory.

Additionally, the backend maintains statistics for the average, minimum, and maximum RSSI values for each sensor node. This way the GUI can display all the data simply by reading the HDF5 file. Additionally, this module can run without a graphical interface, so it is also suited for text-only servers and as input into other programs.

4.4.2 GUI

To simplify the monitoring of the grid, we developed a basic GUI that displays a selected set of the current scan data and allows the user to specify the scan options. As it can be seen in the snapshot of figure 4.7, the interface shows a list of sensor

nodes in the grid, plots the RSSI data of each board, and presents the different settings allowed by the scan protocol (as described in section 4.3.1). To change the set of sensor nodes data that is currently plotted, the user clicks on the desired item of the node list, which includes each board's MAC and IP addresses, and status. This way the state of a small grid can be grasped with a simple glance at the interface.

All the information displayed via the interface is extracted from the HDF5 file generated by the scanner backend. The access to this file is regulated with a simple file lock, to prevent corruption of the data as the PyTables module lacks good support for concurrency. Additionally, the GUI can start and kill the backend via its main window.

Regarding the programming details, the interface was developed using wxPython[44], a Python wrapper for the wxWidgets library[45], which is intended for creating GUIs for cross-platform applications. The data display functions were handled using matplotlib[46], a powerful plotting library for Python that provides out-of-the-box support for the wxWidgets toolkit.

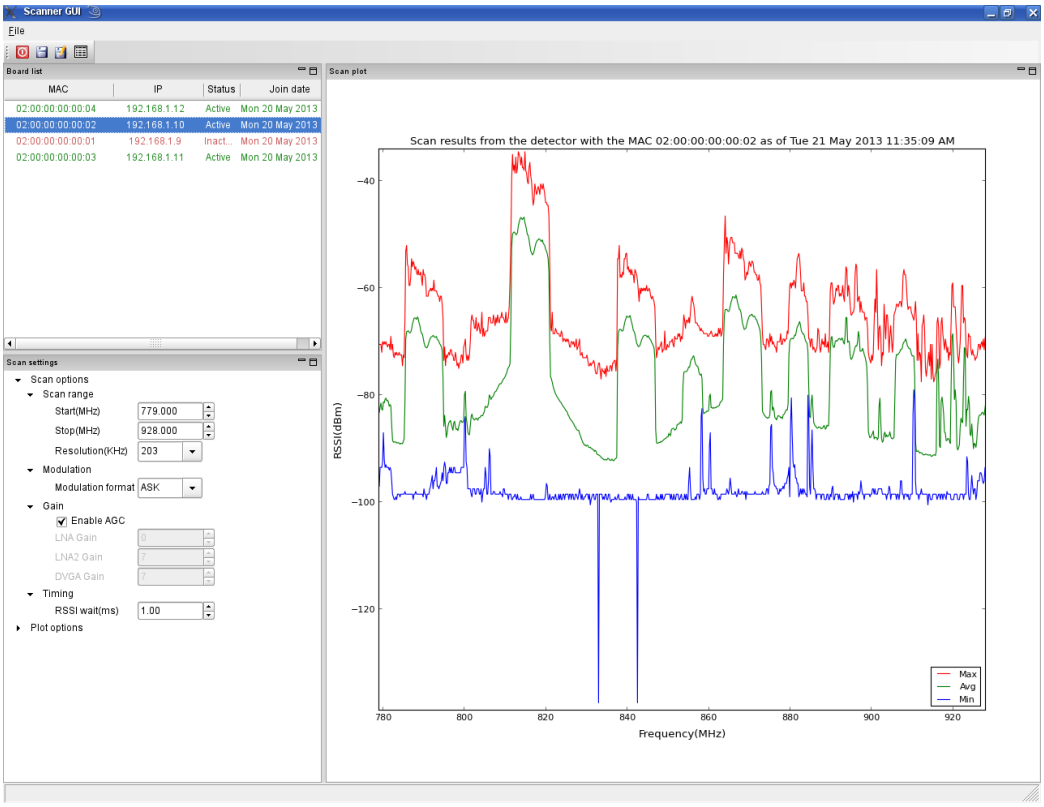
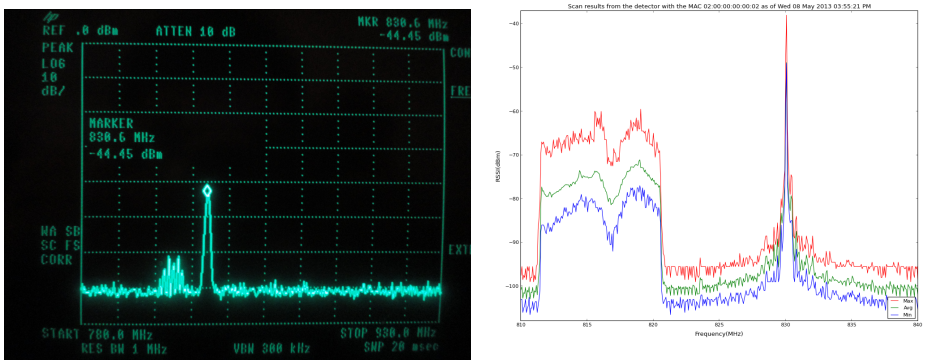


Figure 4.7: Screen capture of the grid GUI with four active sensors

4.5 Sensor node testing

All the software pieces are now already in place and the networking functionalities have been tested. However, before starting to deploy the grid, we need to make sure that the data we obtain from each individual sensor makes sense. To do that we compared the sensor's spectrum scan results with a known signal.

First we needed to rule out problems arising from the RF daughterboard circuitry itself, so we tested the sensor without an antenna. Using a coaxial cable and a 3dB power divider, we connected the output from the GSM tester described in section 3.3.2 to both the spectrum analyzer and, through a BNC-to-SMA adapter to the antenna connector of the daughterboard. This way the analyzer and the board receive the same signal, both attenuated by 3dB from the output created by the GSM tester. Using the GSM tester's RF generator mode, we generated sine waves with frequencies within the daughterboard's transceiver range, and verified that the board's scan results matched those of the spectrum analyzer for all the bands.



(a) Spectrum analyzer results (b) White space sensor results

Figure 4.8: Scan of a -40dBm sine wave from the GSM tester of section 3.3.2

The result of one of such tests is shown in figure 4.8. In this case we used a sine wave of 830MHz at -40dBm of amplitude. For the spectrum analyzer we used a 1MHz resolution bandwidth, and we set the board to scan between 810 and 840MHz with a 58kHz resolution and the AGC enabled. The spectrum analyzer reported -44.45dBm at that frequency, with the power divider and the connector loss accounting for -4.45dB attenuation, whereas the board detected -48dBm on average at the same band. The 3.55dBm disparity can be explained by inefficient impedance matching at the daughterboard's connector and the BNC-to-SMA adapter. This test showed that the board's scan results made sense, but that the numeric values of the signal required some calibration. Another detail is

the energy both plots show around 815MHz, which corresponds to the 800MHz mobile band downlink in Sweden[47]. This signal was picked up by the coaxial cable which acted as an inefficient antenna.

Now that we know that the transceiver is operating correctly, we tested it with an antenna and a known wireless transmitter: the wireless temperature sensor described in section 3.3.4. We knew from [15] that the transmitter's center frequency was 862.25MHz and that it used a 2-FSK modulation with 53kHz of frequency deviation and around -22dBm of peak output power as detected close to the transmitter. The board's scanning result is shown in figure 4.9, where it can be seen that it matches all those details about the transmitter, including the two peaks expected of the 2-FSK modulation. As the temperature sensor transmits data only once every 4 seconds, that short burst is only shown in the maximum RSSI plot, which justifies the usefulness of displaying the the maximum and minimum RSSI values along with each band's average.

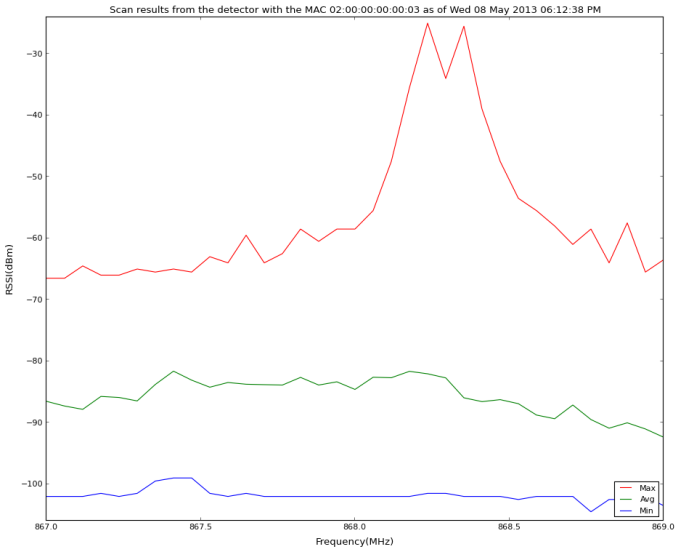


Figure 4.9: Spectrum scan 10 cm from the wireless thermometer of section 3.3.4

With these tests we concluded that the spectrum scan data reported by the sensor nodes made sense, and that they were only in need of some calibration to compensate from inefficient impedance matching of the RF daughterboard. Therefore, we could move on to deploying the white space sensor grid.

4.6 Grid deployment

Now that all the software modules have been developed and tested, we used the white space sensor grid to perform spectrum measurements for extended periods of time in order to showcase its potential. Unless otherwise stated, all the measurements were performed at a lab room on the 4th floor of the Electrum building at Kista, Stockholm.

4.6.1 Spectrum utilization

One of the main goals of the sensing grid is to give a picture of the spectrum usage at a particular location, as well as to inform clients of the grid server about the current and probable future gaps. To test this feature, we performed a 24-hour spectrum scan using 4 nodes during a weekday in our lab room. These results are displayed in figure 4.10. The labels for the usage of the different frequency bands were obtained from the 2012 Spectrum Orientation Plan[22] and the auction results[47] of the Swedish Post and Telecom Authority (PTS), the government agency responsible for regulating the radio spectrum in Sweden.

The overall RSSI statistics of figure 4.10(a) show that there are many underutilized frequency bands. As expected, the signal level is high in the bands dedicated to mobile communications, such as the GSM-900 band and the 800MHz one, though only in the downlink frequencies. This is due to the higher output power of the base stations compared to that of the mobile terminals, as well as the need of the stations to transmit constantly to advertise themselves. It should also be noted that the base stations for several of the operators are mounted on the Electrum building and those adjacent to it.

Figure 4.10(b) shows that there are also considerable temporal variations of the spectrum use, as illustrated by the overabundance of low-energy blue with some occasional columns of high-energy. Some downlink mobile channels maintain a consistently high output power, but those associated with the uplink mobile to base station channels show a markedly higher activity during the daylight hours, especially between 10:00 and 20:00, in comparison to the evening and night. These hours of higher activity match with normal working hours, which seems reasonable as the measurements were performed during a weekday. Also, some bands appear to be virtually unused all the time, such as the 800MHz-band centre gap, which according to PTS is license exempt and used for wireless microphones. It should be noted that no known classes or conferences were taking place in the building during this particular weekday (as this classroom and conference area of the building is closed for renovation). Hence it is not surprising that there were not wireless microphones operating in the building.

More surprising is the fact that even some downlink mobile channels appear

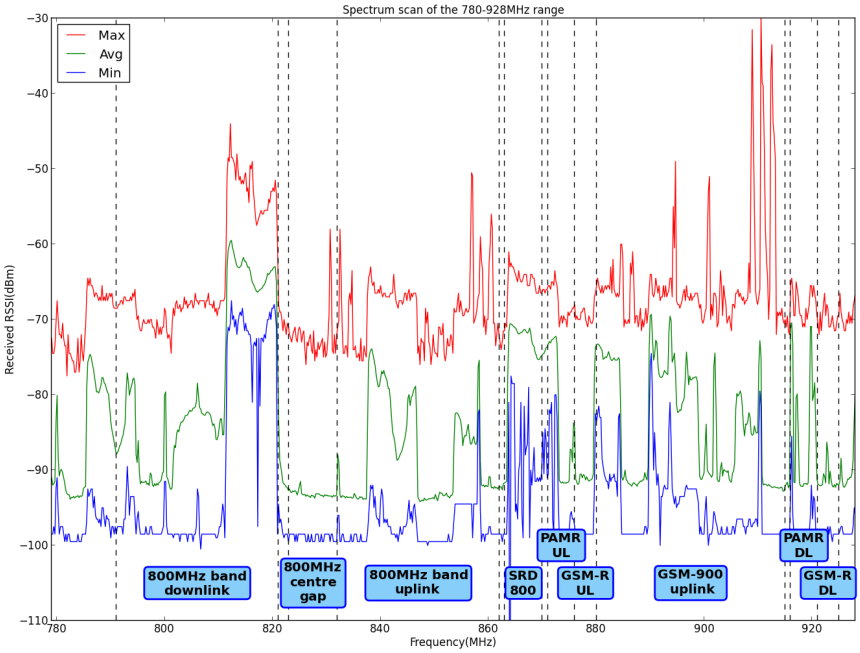
Table 4.2: Spectrum utilization measurements during a weekday

Spectrum band	Range(MHz)	Utilization(%)
Empty	790 - 791	57.77
800MHz downlink	791 - 821	69.36
Empty	821 - 823	21.42
800MHz centre gap	823 - 832	4.15
800MHz uplink	832 - 862	39.87
Empty	862 - 863	7.65
SRD-800	863 - 870	94.60
Licensed SRD	870 - 871	100
PAMR uplink	871 - 876	54.39
GSM-R uplink	876 - 880	18.55
GSM900 downlink	880 - 915	57.73
Empty	915 - 916	9.56
PAMR downlink	916 - 921	48.73
GSM-R downlink	921 - 925	14.97

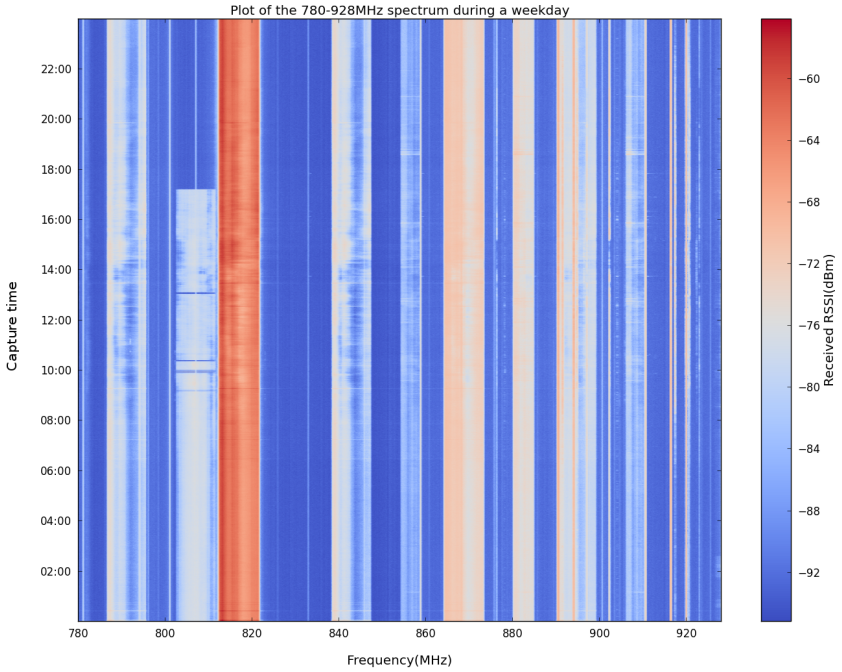
also vacant most of the time, such as the 796-801MHz band, which corresponds to the FDD2 block auctioned to HI3G Access AB[47]. This is probably due to the operator using those frequencies in adjacent cells or even not providing coverage where the lab room is.

To get a quantitative idea of how important those spectrum white spaces are, we computed a utilization value for each band. Following the same approach used in [1], we set a threshold of -90dBm and considered a channel occupied if the RSSI sample was above that level. Therefore, we calculated the utilization as the percentage of samples where we found each band to be busy. The results are shown in table 4.2, which confirms the impression of overall underutilization given by the predominance of blue in the waterfall plot of figure 4.10(b).

Almost none of the bands analyzed in table 4.2 is occupied all the time, hinting at the existence of plenty of temporal gaps which could be taken advantage of by opportunistic spectrum-aware transmitters. In fact, some of them appear vacant most of the time, such as the 800MHz centre gap band or the safeguard bands labeled as “empty”, so there is plenty of room for improvement in the spectrum utilization efficiency. Even within the relatively busy on average mobile bands there are also significant white spaces, as illustrated by tables 4.3 and 4.4, which detail the use of the different licensing blocks of the 800MHz and GSM900 mobile bands, excluding the uplink channels of GSM900, which fall outside our RF transceiver’s range. For example, as we already noted when analyzing the



(a) Plot of the overall RSSI statistics for the whole day



(b) Waterfall plot of the average RSSI per minute

Figure 4.10: Results of a 24-hour capture from the white space sensor grid

Table 4.3: Spectrum utilization of the 800MHz mobile band during a weekday

Frequency block	Downlink band(MHz)	Uplink band(MHz)	Utilization(%) *downlink*	Utilization(%) *uplink*	License holder
FDD1	791 - 796	832 - 837	66.97	6.02	HI3G Access
FDD2	796 - 801	837 - 842	19.10	87.73	HI3G Access
FDD3	801 - 806	842 - 847	63.07	66.57	TeliaSonera
FDD4	806 - 811	847 - 852	72.22	3.27	TeliaSonera
FDD5	811 - 816	852 - 857	99.80	51.56	Net4Mobility
FDD6	816 - 821	857 - 862	100	28.42	Net4Mobility

Table 4.4: Spectrum utilization of the GSM900 downlink band during a weekday

Frequency range(MHz)	Utilization(%)	License holder
880.0 - 885.0	95.88	HI3G Access
885.0 - 890.0	16.03	Swefour
890.0 - 897.5	96.89	Tele2
897.5 - 905.0	43.63	Telenor
905.0 - 915.0	42.02	TeliaSonera

waterfall plot of figure 4.10(b), the FDD2 downlink block is occupied less than 20% of the time.

Other frequency bands are not so promising in terms of temporal gaps, but have quite a lot of potential to be reused in different geographical positions. Such is the case for the Global System for Mobile Communications - Railway (GSM-R) bands. GSM-R is the standard used for mobile communications along the railway lines, and it appears as a relatively busy channel in our scan. We believe this is due to the proximity of our lab room to a train station and a major rail corridor. However, as GSM-R is by its own nature restricted to the railway, it will not have coverage in most of the territory.

Computing the utilization as we just did is *not* accurate for mobile systems such as GSM, where each downlink channel dedicated to voice transmission has a paired uplink one. This means that if we detect activity on one the downlink channel, this implies that its paired uplink is probably also in use, even though we might not measure a significant amount of energy in this uplink channel due to the relatively low power transmitted by user terminals. Such a discrepancy can be clearly seen in table 4.3, where most bands show a markedly higher utilization in

the downlink channels than in the uplink ones. Therefore, we should be careful when labeling a mobile uplink channel as ”vacant”, although gaps in these bands still present interesting reuse opportunities for low-power SRD that are unlikely to interfere with the user terminals.

Our measurements show significant spectrum utilization inefficiencies at our present location, which we described in section 1.1 as the issue that motivated this project. We have thus proven that there exist many interesting reuse opportunities, and that our sensor grid is capable of spotting and reporting them.

4.6.2 Radiolocation potential

The final test that we performed with our sensor nodes was to check its potential for radiolocation, that is to say, for providing the position of a transmitter based on the signal strength measurements from multiple sensors at various distances. To investigate this, we connected an antenna to the output port of the GSM tester of section 3.3.2 and we used it as a RF generator to continuously transmit a sine wave of -7dBm of amplitude at 868MHz, which falls within the license-free SRD range. Using a sensor board, we measured the RSSI at that frequency at different distances from the transmitter to get an idea of the propagation model of our lab.

The results of the propagation loss versus distance are shown in figure 4.11, where it can be seen that they do not follow a logarithmic decrease characteristic of free-space propagation, but rather the ups and downs of the fast fading model characteristic of indoors propagation[48]. This was to be expected, as the measurements were performed in a lab room with plenty of equipment laying about, which presents an ideal environment for multipath propagation and shadow zones. However, the most discouraging fact is the large variability in the RSSI measurements detected by the sensor nodes, clearly shown by the width of the boxplot whiskers in figure 4.11.

Taking into account those facts, we conclude that estimating of the distance to the transmitter indoors would require a complex model of the particular propagation characteristics of each node’s environment. This would require developing such a model before the deployment of the grid, which would be unfeasible and of limited use. However, a simpler and more practical approach would be locating the transmitters through the RSSI contour lines, assuming it to be at the center of the innermost contour line for its frequency, that is to say, the one with the highest RSSI. No matter how complex the propagation model, distance will always be a major factor. Due to lack of time, testing this approach was outside the scope of this project, but it is suggested as a direction for future work.

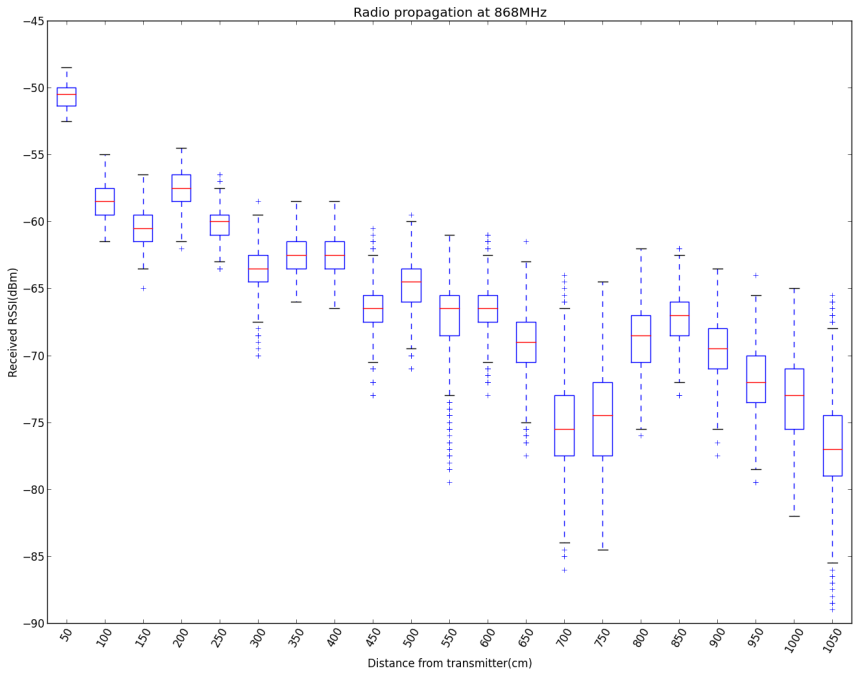


Figure 4.11: Boxplot of the path loss of a -7dBm sine wave at 868MHz

Chapter 5

Conclusions

This chapter lists the conclusions and insights reached during this master's thesis and suggests some directions for future work in this area.

5.1 General conclusions

The main goal of this master's thesis, as stated in section 1.2, is to ease the deployment of CR systems by developing and testing a white space sensor grid dedicated to detecting temporal and spatial gaps in the use of spectrum in an area. To achieve that aim we created a prototype grid comprised of the embedded nodes (described in section 3.2) which act as highly configurable and low cost spectrum analyzers due to the program explained in section 4.3, as well as a central server that coordinates these sensor nodes, and collects and interprets the data they report. The development of this central grid server detailed in 4.4.

The nodes of the grid were designed to be as plug-and-play as possible, so that the only concern of the user would be finding enough Ethernet cables to connect all the nodes in the grid. Through the Ethernet cable the nodes obtain power via PoE. Additionally, the nodes will automatically download the latest version of the code they will run using a network bootloader we developed (as explained in section 4.2). Hardcoded configuration was avoided as much as possible, and the nodes only require instructions from the server to change their scan settings to be different from the default parameters.

Furthermore, we have verified the validity of the nodes' spectrum scans with the tests described in section 4.5 and checked the robustness of the whole system by performing measurements for extended periods of time. The results of these experiments were described in section 4.6. These experiments allowed us to estimate the inefficiency of the spectrum utilization in our current location at the time of the measurements and to pinpoint some frequency bands that offer

potential for reuse.

This project involved the development and integration of many software modules running on a variety of platforms and written in a number of programming languages, while never losing sight of the embedded hardware on which it all depended. The constant juggling between software and hardware, low-level microcontroller routines and high-level Python scientific plotting libraries, has been an important lesson in flexibility and perspective. From now on I will never again waste time blaming the code when a PCB starts shooting up sparks, nor fear that having to get out a soldering iron is the end of the world.

With the benefit and bias of hindsight, there are many things I would have done differently in this project. The main change would be to try to get a clearer picture of how the different components to be developed should fit together before jumping ahead to the coding phase. Although it all worked out in the end, many parts of this project were developed as we went, and some had to be abandoned as dead ends due to not being able to see the forest for the trees. However, the main lesson is to never again attempt to develop a big system without having peer feedback always at hand, doing so in the echo chamber of one's head is a sure recipe for disaster and small-minded featuritis. One of the major reasons for this lack of feedback was that I did not write the thesis as I went along, hence it was not easy to show the overall state of what I was trying to do to anyone else.

5.2 Future work

Most of this project was devoted to designing, developing and setting up the software and hardware framework for the grid, but relatively little time was spent actually using the resulting grid. For now the grid server provides only basic graphs and spectrum usage statistics, but it could be enhanced to generate a true geographical map of the spectrum by combining the RSSI information from many sensor nodes with the knowledge of their position. Ideally it should be able to integrate the scanning information with a system such as the white space databases developed by Google[13] to provide spectrum utilization charts for wide areas, displaying the gaps and occupied bands much like weather maps show high and low pressure fronts.

Additionally, this project did not explore the potential of making predictions for the usage of a given band by taking into account a large body of spectrum data collected from similar situations, such as the same weekday over many weeks. Estimating the probability of a band being occupied in the future would be immensely useful, as would be checking these predictions against reality to see if the model is accurate.

One last area for future work would be improving the design of the current

embedded platform. As explained in section 4.2.2, the lack of a BSL programming interface is an important drawback of the current design, and future models should seriously consider including it. Also, right now there is only one RF daughterboard type available, so it would be very interesting to produce additional types of daughterboards making use of the different transceivers manufactured by TI, and thus expand the radio frequency scanning range of the sensor nodes.

5.3 Required reflections

This master's thesis project contributes towards the ease of sharing a limited public resource: available radio spectrum. By developing a sensor grid that reports the gaps in its use, we encourage the reuse of the existent frequency bands, instead of the wasteful allocation of channels to particular license holders with exclusive rights over them, regardless of whether they are actually transmitting in a given place and time. In that sense, we consider this master's thesis project to have a positive **social** effect.

Moreover, increasing the efficiency of the spectrum utilization would reduce the cost and accelerate the development pace of new wireless applications, as they would not require going through the expensive and slow spectrum auction process to acquire the rights for a given frequency. By deploying a sensor grid such as the one described in this project we might be able to relax the requirements on CR systems, as they could rely on the grid for the spectrum sensing functionalities, which would make them cheaper, one of the potential **economic** benefits of this thesis.

Furthermore, the embedded platforms used as sensor nodes in this thesis were developed with low-power parts compliant with the EU's Restriction of Use of Hazardous Substances (RoHS) regulation, so appropiate **environmental** considerations were taken into account during the design and implementation process.

Finally, no **ethical** issues were encountered during the course of this project. All the wireless transmission tests were performed in the license-free SRD-800 band with low power levels and complying with the regulations for that band. As for the spectrum scan itself, we only store the energy levels in the different frequency bands and do not attempt to decode any signal, thus there is no breach of privacy involved. However, there are some indications that in the future it may be possible to associate user movements with changes in the RSSI, however the system is currently far from being able to do this.

Bibliography

[1] V. Valenta, R. Marsalek, G. Baudoin, M. Villegas, M. Suarez, and F. Robert, "Survey on spectrum utilization in europe: Measurements, analyses and observations," in *2010 Proceedings of the Fifth International Conference on Cognitive Radio Oriented Wireless Networks Communications (CROWNCOM)*, Cannes, France, Jun. 2010. ISBN 978-1-4244-5885-1 pp. 1–5.

[2] P. S. M. Tripathi, A. Chandra, A. Kumar, and K. Sridhara, "Dynamic spectrum access and cognitive radio," in *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE)*, Mar. 2011. doi: 10.1109/WIRELESSVITAE.2011.5940911 pp. 1–5.

[3] D. Cabric, S. Mishra, and R. Brodersen, "Implementation issues in spectrum sensing for cognitive radios," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004*, vol. 1, Nov. 2004. doi: 10.1109/ACSSC.2004.1399240 pp. 772 – 776 Vol.1.

[4] Federal Communications Commision, Spectrum Policy Task Force Report, ET Docket No. 02-155, Nov. 2002. [Online]. Available: http://transition.fcc.gov/sptf/files/SEWGFinalReport_1.pdf

[5] "US frequency allocation chart," National Telecommunications and Information Administration, Oct. 2011. [Online]. Available: http://www.ntia.doc.gov/page/2011/united-states-frequency-allocation-chart

[6] B. Wang and K. J. R. Liu, "Advances in cognitive radio networks: A survey," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 1, pp. 5–23, Feb. 2011. doi: 10.1109/JSTSP.2010.2093210

[7] J. Mitola III and G. Q. Maguire, Jr., "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, Aug. 1999. doi: 10.1109/98.788210

[8] I. F. Akyildiz, W. Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: a survey," *Computer Networks*, vol. 50, no. 13, p. 2127–2159, 2006. doi: 10.1016/j.comnet.2006.05.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128606001009

[9] K. Sollins, "The TFTP protocol (revision 2)," IETF, RFC 1350 (Standard), Jul. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1350.txt

[10] J. Lai, E. Dutkiewicz, R. P. Liu, and R. Vesilo, "Comparison of cooperative spectrum sensing strategies in distributed cognitive radio networks," in *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012. doi: 10.1109/GLOCOM.2012.6503328 pp. 1513–1518.

[11] H. Murata, T. Ohno, K. Yamamoto, and S. Yoshida, "Implementation and field experiments of cognitive radio system with cooperative spectrum sensing," in *2010 International Symposium on Communications and Information Technologies (ISCIT)*, 2010. doi: 10.1109/ISCIT.2010.5665129 pp. 980–984.

[12] "Second memorandum opinion and order," Federal Communications Commision, FCC 10-174, Sep. 2010. [Online]. Available: http://hraunfoss.fcc.gov/edocs_public/attachmatch/FCC-10-174A1.pdf

[13] "Spectrum database – google," Jun. 2013. [Online]. Available: http://www.google.org/spectrum/whitespace/

[14] R. Murty, R. Chandra, T. Moscibroda, and P. Bahl, "SenseLess: a database-driven white spaces network," *IEEE Transactions on Mobile Computing*, vol. 11, no. 2, pp. 189–203, 2012. doi: 10.1109/TMC.2011.241

[15] A. Lopez Garcia and F. J. Sanchez Galisteo, "Exploiting wireless sensors : A gateway for 868 MHz sensors," Masters's thesis, Royal Institute of Technology, School of Information and Communication Technology, Stockholm, Sweden, Jun. 2012, TRITA-ICT-EX; 2012:110. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-98209

[16] "TL2575HV step-down(buck) converter," Texas Instruments, Datasheet SLVS638B, Jan. 2007. [Online]. Available: http://www.ti.com/lit/gpn/tl2575hv-adj

[17] "IEEE 802.3af PoE powered device controllers," Texas Instruments, Datasheet SLVS525B, Apr. 2008. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps2375.pdf

[18] "PoE explained," Veracity UK Ltd, White Paper 002, Dec. 2008. [Online]. Available: http://www.veracityglobal.com/media/27197/vwp-002%20poe%20explained.pdf

[19] "Mixed signal microcontroller," Texas Instruments, Datasheet SLAS655B, Oct. 2010. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f5437a.pdf

[20] "ENC28J60 stand-alone ethernet controller with SPI interface," Microchip, Datasheet DS39662C, 2008. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf

[21] "Low-power sub-1 GHz RF transceiver," Texas Instruments, Datasheet SWRS061H, May 2013. [Online]. Available: http://www.ti.com/lit/gpn/cc1101

[22] "Spectrum orientation plan," Swedish Post and Telecom Agency, Rev. 2012-06-11, Oct. 2012. [Online]. Available: http://www.pts.se/upload/Ovrigt/Radio/draft-orientation-plan-121011.pdf

[23] "MiniMag magnet mount antenna," Smarteq, Product No. 1140.30SMA, May 2013. [Online]. Available: http://www.smarteq.se/download/18.6018c17913483dc06428000159/MiniMag+1140.30SMA_550058E.pdf

[24] "Code composer studio (CCStudio) integrated development environment (IDE) v5," Texas Instruments, Nov. 2011. [Online]. Available: http://www.ti.com/tool/ccstudio

[25] "8922M/S GSM test set," Agilent Technologies, User's Guide 08922-90211, Jan. 1998. [Online]. Available: http://www.home.agilent.com/upload/cmc_upload/All/08922_90211.pdf?&cc=SE&lc=eng

[26] "'Wave' wireless thermometer with radio-controlled clock," TFA, Cat. No. 30.3016.54.IT, May 2013. [Online]. Available: http://tfa-dostmann.de/index.php?id=61&L=1

[27] "Wireshark homepage," May 2013. [Online]. Available: http://www.wireshark.org/

[28] "Contiki: The open source operating system for the internet of things," May 2013. [Online]. Available: http://www.contiki-os.org/#about

[29] "lwIP - a lightweight TCP/IP stack," May 2013. [Online]. Available: http://savannah.nongnu.org/projects/lwip/

[30] Nilesh Rajbharti, "The microchip TCP/IP stack," Microchip, Application Note AN833, 2008. [Online]. Available: http://ww1.microchip.com/downloads/en/AppNotes/00833c.pdf

[31] "FNET TCP/IP stack," May 2013. [Online]. Available: http://fnet.sourceforge.net/index.html

[32] "ISC DHCP server," Internet Systems Consortium, Version 4.2.5rc1, Mar. 2013. [Online]. Available: http://www.isc.org/downloads/

[33] S. Alexander and R. Droms, *DHCP Options and BOOTP Vendor Extensions*, ser. Request for Comments. IETF, Mar. 1997, no. 2132, published: RFC 2132 (Draft Standard) Updated by RFCs 3442, 3942, 4361, 4833, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc2132.txt

[34] R. Johnson, *TFTP Server Address Option for DHCPv4*, ser. Request for Comments. IETF, Jun. 2010, no. 5859, published: RFC 5859 (Informational). [Online]. Available: http://www.ietf.org/rfc/rfc5859.txt

[35] A. Muehlhofer, "MSP430 flash self-programming technique," Texas Instruments, Application Report SLAA103, Nov. 2000. [Online]. Available: http://www.gaw.ru/pdf/TI/app/msp430/slaa103.pdf

[36] P. Miller, "TI-TXT file format - man page," 2010. [Online]. Available: http://linux.die.net/man/5/srec_ti_txt

[37] "MSP430 programming via the JTAG interface," Texas Instruments, User's Guide SLAU320I, Nov. 2010. [Online]. Available: http://www.ti.com/lit/ug/slau320i/slau320i.pdf

[38] S. Namtvedt, "RSSI interpretation and timing," Texas Instruments, Design Note DN505, SWRA114B, 2007. [Online]. Available: http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=swra114

[39] "TI forum - amplifier gain manual settings," Aug. 2012. [Online]. Available: http://e2e.ti.com/support/low_power_rf/f/155/p/180664/730390.aspx#730390

[40] S. Johnsrud, "Frequency scanning using CC430Fx, CC110x, and CC111xFx," Texas Instruments, Design Note DN508, SWRA315, 2010. [Online]. Available: http://www.ti.com/lit/an/swra315/swra315.pdf

[41] K. Quring and W. Goth, "MSP430 interface to CC1100/2500 code library," Texas Instruments, Application Report SLAA325A, Jul. 2006. [Online]. Available: http://www.ti.com/lit/an/slaa325a/slaa325a.pdf

[42] "HDF5 homepage," May 2013. [Online]. Available: http://www.hdfgroup.org/HDF5/

[43] F. Alted, "PyTables - python wrapper for HDF5," May 2013. [Online]. Available: http://www.pytables.org/moin

[44] "wxPython homepage," May 2013. [Online]. Available: http://www.wxpython.org/

[45] "wxWidgets homepage," May 2013. [Online]. Available: http://www.wxwidgets.org/

[46] J. D. Hunter, "Matplotlib homepage," May 2013. [Online]. Available: http://matplotlib.org/

[47] S. Ibrahim, "Decision about licences to use radio transmitters in the 791–821/832–862 MHz frequency band," Swedish Post and Telecom Agency, Decision 10-10534, Mar. 2011. [Online]. Available: http://www.pts.se/upload/Beslut/Radio/2011/10-10534-desicion-assignment-800mhz.pdf

[48] Henri Bertoni (Ed.), "Coverage prediction for mobile radio systems operating in the 800/900 MHz frequency range," *IEEE Transactions on Vehicular Technology*, vol. 37, no. 1, pp. 3–72, 1988. doi: 10.1109/25.42678

Appendix A

Source code

All the source code involved in this master's thesis is publicly available via a github repository located in `https://github.com/cazulu/mind-the-gaps.git`. The folders of the repository correspond to the different sections of the program:

- The TFTPboot folder contains the code of network bootloader for the microcontroller platform.
- The blinker folder holds the code of a simple LED-blinking program, to test the board and for use as an example program to be downloaded by the network bootloader.
- The frequencyScanner branch stores the code of the scanner client for the board.
- The scannerGUI branch contains the Python scripts for the scanning server backend, the user interface, and a server test script that simulates a white space detector board.

Appendix B

BSL programming

As we mentioned in section 4.2.2, using the flash self-programming feature of the MSP430F5437A creates the risk of overwriting a memory position between 0x17FC and 0x17FF due to an addressing bug, and thus triggering the JTAG Lock feature. This would block access to the MCU via the JTAG interface, and the only way to recover the board would be using the BSL interface. Unfortunately, some of the BSL pins were not routed out in the board's design, and others are already in use for other purposes. In this appendix we will provide a brief description of how to hack together a makeshift interface and use it to recover the sensor node from the brink of disaster.

First, assuming your computer lacks a serial COM port, you'll need to fetch a USB-to-serial interface such as the one developed by Professor Mark T. Smith and shown in figure B.1. Next you have to connect this serial interface to the BSL pins. TI offers many other options in this link: `http://processors.wiki.ti.com/index.php/BSL_%28MSP430%29`.

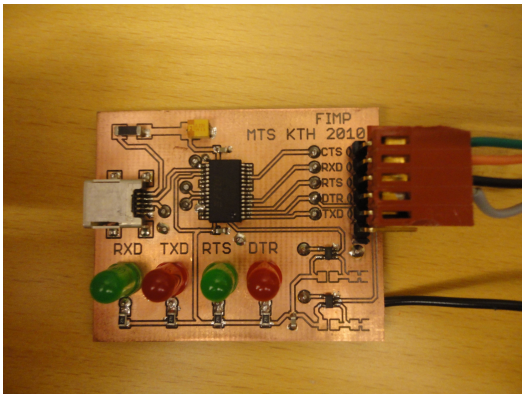


Figure B.1: USB-to-serial board used for BSL programming

Table B.1: Equivalence between BSL pins and those of the board

MCU pin	BSL function	Breakout location	Comments
$\overline{\text{RST}}$/NMI/SBWTDIO	Entry sequence(DTR)	JTAG $\overline{\text{RST}}$ pin	
TEST/SBWTCK	Entry sequence(RTS)	JTAG TEST pin	
P1.1	Data transmit(RXD)	P1.1 MCU pin	Not routed out!
P1.2	Data receive(TXD)	P1.2 MCU pin	Remove via wire
VCC	Power supply(VCC)	JTAG VCC pin	
VSS	Ground supply(GND)	JTAG GND pin	

After that you need to connect wires to the board pins listed in table B.1. The trickiest by far was pin P1.1, which was not routed out from the MCU, so you will need to be extra careful not to shortcircuit anything and use a really thin wire (for this we used 26 gauge wirewrap wire). Another important detail is that P1.2 is located on the lower side of the board and connected to the $\overline{\text{INT}}$ pin of the ENC28J60 through a via wire, you will need to take out the via wire in order for the BSL interface to work. This will prevent the ENC28J60 from interrupting the MCU, so do not forget to restore the via wire afterwards!

With all the BSL pins connected them to the appropiate pins of the USB-to-serial board, and plug it into your computer's USB port. From here on all the instructions are meant for a version of the Microsoft Windows OS, but alternative approaches exist for Unix-like systems. Before sending commands via BSL you will need the BSL scripter program from TI, which can be downloaded from this link: `http://www.ti.com/lit/zip/slau319`. Also, do not forget to check the settings for the USB-to-serial board driver of your OS. It should be configured for a baud rate of 9600, 8 data bits, 1 stop bit, even parity, 1.2ms of delay after sending a character and use COM port 1.

Now you are ready for the final step: executing a BSL script. To unlock the JTAG we only need to restore the memory locations between 0x17FC and 0x17FF back to 0, so a mass erase is sufficient. The script file required for that is a simple text file with the following contents:

```
MODE 543x_family COM1
MASS ERASE
```

To execute this script, open a Windows' command line prompt, go to the folder where you stored the BSL scripter and run it with the name of the script file as the only argument. That should be all that is necessary, good luck!

TRITA-ICT-EX-2013:102

www.kth.se