Towards Large-Scale and Collaborative Spectrum Monitoring Systems using IoT Devices

by

Roberto Calvo Palomino

in partial fulfillment of the requirements for the degree of Doctor in

Telematic Engineering

Universidad Carlos III de Madrid

Advisor: Domenico Giustiniano

July 2019

Towards Large-Scale and Collaborative Spectrum Monitoring Systems using IoT Devices

Prepared by:
Roberto Calvo Palomino, IMDEA Networks Institute, Universidad Carlos III de Madrid
contact: roberto.calvo@imdea.org

Under the advice of:
Domenico Giustiniano, IMDEA Networks Institute

Telematic Engineering Department, Universidad Carlos III de Madrid

This work has been supported by:

Unless otherwise indicated, the content of is thesis is distributed under a
Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA).

To María, my wife. For her endless love, encouragement and support.

To my parents, who worked hard and made the impossible to give me the life I have.

Thank you!

Acknowledgements

First things first, let me just remark that there is no way to reach an important milestone such as this thesis without the help of others. I would like to use the first pages of this thesis to really thank everyone that helped me during the last years along my Ph.D journey.

The first thank goes to my advisor, Dr. Domenico Giustiniano, who supervised and guided me during the full Ph.D. He was always providing constructive feedback, rigorous reviews of my papers and making me focus on the big picture of the research problem. Thanks to him, I have been involved in nearly 25 presentations (project meetings, seminars, workshops, conferences, etc) which gave me the confidence to be able to introduce and present a research topic in front of others. I really appreciate all his contributions, the late nights spent on reviewing my papers, seeking funding for my Ph.D and even pushing me to run two 10 km races.

I would also like to thank my "second" advisor, Dr. Vincent Lenders, who co-supervised my work, was always patient with me and provided very useful feedback. Thanks also for organizing the Cyber Alp Retreat workshop to which I was invited along 4 years. Thanks for hosting me during 6 months in armasuisse for my internship, I really learned during that period and enjoyed my stay in Thun.

I am very grateful to Dr. Fabio Ricciato who hosted me for 3 months during my internship in the University of Ljubljana. He really brought out the best in me during that period in which I felt a change in my way of thinking and affording research problems.

Let me also express my gratitude to Héctor Cordobés who helped me in the last part of my Ph.D providing a valuable theoretical vision in different areas (besides bad jokes).

Although he has not been involved in my Ph.D, I would like to thank Dr. Antonio Fernández Anta who convinced me to start this research journey at IMDEA Networks. Without his insistence I would not be writing these lines. Thanks for your always motivating and encouraging words.

Furthermore, I would like to thank my colleagues at IMDEA Networks. They really made my Ph.D an extraordinary experience. I had very good times with all of them. Let me start to thank my "two italians", Maurizio and Dario, who always had the time to explain me that *Roberto* is clearly an Italian name or how good the Italian coffee is.

Thanks to my old-desk mate Christian (even if he defines himself as the least lucky man in the world), and my non-old-desk mate Mohamed for dealing nicely with *my attacks*. Thanks to the ping-pong mates (Ander, Guillermo, Edgar, Maurizio and Oluwasegun) for the break times needed between papers (no Ander, you did not win!). Thanks to Noelia, for her stories and always having the drawer full of peanuts. By last, I would like to send a special thanks to everyone who ever said "yes" to Piratas!

I had the pleasure to work with people from different universities and companies during my Ph.D, and I would like to thank all of them. Thanks to the colleagues of armasuisse (Giorgio, Marc, Daniel and Dr. Vincent), KU Lueven University (Dr. Sreeraj, Dr. Bertold and Dr. Sofie), Zero Systems (the two Markus and Dr. Matthias), and Ljubljana University (Dr. Fabio, Blaz, Veljko and Tom).

Finally, I would like to thank all IMDEA Networks team (professors, IT, administrative, human resources, support, etc) for helping and making everything easier.

Published Content

This thesis is based on the following published papers:

[1] Roberto Calvo-Palomino, Domenico Giustiniano, Vincent Lenders, Aymen Fakhreddine. Crowdsourcing Spectrum Data Decoding. Published in *the 36th IEEE International Conference on Computer Communications (IEEE INFOCOM 2017)*, 1-4 May 2017, Atlanta, GA, USA. https://doi.org/10.1109/INFOCOM.2017.8057078

- This work is fully included and its content is reported in Chapter 6.
- The author's role in this work is focused on the design, implementation and experimentation with regarding of the concepts proposed in the paper.

[2] Roberto Calvo-Palomino, Domenico Giustiniano, Vincent Lenders. Measuring Spectrum Similarity in Distributed Radio Monitoring Systems. Published in *Tyrrhenian International Workshop on Digital Communications (TIWDC 2017)*, 18-20 September 2017, Mondello (Palermo), Italy. https://link.springer.com/chapter/10.1007%2F978-3-319-67639-5_16

- This work is partially included and its content is reported in Appendix A
- The author's role in this work is focused on the design, implementation and experimentation of the proposed methodology.

[3] Roberto Calvo-Palomino, Fabio Ricciato, Domenico Giustiniano, Vincent Lenders. LTESS-track: A Precise and Fast Frequency Offset Estimation for low-cost SDR Platforms. Published in *the 11th ACM Workshop on Wireless Network Testbeds, Experimental evaluation CHaracterization (ACM WiNTECH 2017)*, 16-20 October 2017, Snowbird, Utah, USA. https://doi.org/10.1145/3131473.3131481

- This work is fully included and its content is reported in Chapter 4
- The author's role in this work is focused on the design, implementation and experimentation of the proposed methodology.

[4] Sreeraj Rajendran, **Roberto Calvo-Palomino**, Markus Fuchs, Bertold Van den Bergh, Héctor Cordobés de la Calle, Domenico Giustiniano, Sofie Pollin, Vincent Lenders. ElectroSense: Open and Big Spectrum Data. *Published in IEEE Communications*

Magazine, 56 (1). pp. 210-217. ISSN 0163-6804, 1-4 May 2017, Atlanta, GA, USA. https://doi.org/10.1109/MCOM.2017.1700200

- This work is fully included and its content is reported in Chapter 2.
- The author's role in this work is focused on the design and implementation of the spectrum sensors and the API, as well as co-leader role in the writing of the full paper.

[5] Roberto Calvo-Palomino, Fabio Ricciato, Blaz Repas, Domenico Giustiniano, Vincent Lenders. Nanosecond-precision Time-of-Arrival Estimation for Aircraft Signals with low-cost SDR Receivers. Published in *the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2018)*, 11-13 April 2018, Porto, Portugal. https://doi.org/10.1109/IPSN.2018.00055

- This work is fully included and its content is reported in Chapter 5.
- The author's role in this work is focused on the designing, implementation, experimentation of the methodology proposed, as well as leader role in the writing of the paper.

[6] Roberto Calvo-Palomino, Héctor Cordobés de la Calle, Fabio Ricciato, Domenico Giustiniano, Vincent Lenders. Collaborative Wideband Signal Decoding using Non-coherent Receivers. Published in *the 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2019) in conjunction with CPS-IoT WEEK 2019*, 14-16 April 2019, Montreal, Canada. https://doi.org/10.1145/3302506.3310387

- This work is fully included and its content is reported in Chapter 7.
- The author's role in this work is focused on the designing, implementation and experimentation of the methodology proposed as well as leader in the writing of the full paper.

[7] Roberto Calvo-Palomino, Héctor Cordobés, Markus Engel, Markus Fuchs, Pratiksha Jain, Marc Liechti, Sreeraj Rajendran, Matthias Schäfer, Bertold Van den Bergh, Sofie Pollin, Domenico Giustiniano, Vincent Lenders. ElectroSense+: Empowering People to Decode the Radio Spectrum [Submitted to IEEE Communications Magazine, Minor Revision]. https://arxiv.org/abs/1811.12265.

- This work is fully included and its content is reported in Chapter 3.
- The author's role in this work is focused on leading the writing of the paper as well as responsible author for the design and implementation on the spectrum sensors.

[**8**] **Roberto Calvo-Palomino**. Radio-Signal Correlation for Collaborative Wideband Spectrum Monitoring System. MSc Thesis. Universidad Carlos III de Madrid, Spain. http://eprints.networks.imdea.org/1993/

- This work is partially included and its content is reported in Chapter 2.
- The author's role in this work is focused on leading the writing of the paper as well as responsible author for the design and implementation.

[**9**] **Roberto Calvo-Palomino**, Damian Pfammatter, Domenico Giustiniano, Vincent Lenders. Demo: A Low-cost Sensor Platform for Large-scale Wideband Spectrum Monitoring. Published in *the 14th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2015)*, 13-16 April 2015, Seattle, USA. https://doi.org/10.1145/2737095.2737124

- This work is fully included and its content is reported in Chapter 2.
- The author's role in this work is focused on building the full demo.

[**10**] Bertold Van den Bergh, Domenico Giustiniano, Héctor Cordobés de la Calle, Markus Fuchs, **Roberto Calvo-Palomino**, Sofie Pollin, Sreeraj Rajendran, Vincent Lenders. ElectroSense: Crowdsourcing Spectrum Monitoring (Demo). Published in *the 13th IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN 2017)*, 6-9 March 2017, Baltimore, MD, USA. http://doi.org/10.1109/DySPAN.2017.7920766

- This work is fully included and its content is reported in Chapter 2.
- The author's role in this work is focused on implementing the sensor side.

[**11**] Franco Minucci, Bertold Van den Bergh, Sreeraj Rajendran, Domenico Giustiniano, Héctor Cordobés de la Calle, Markus Fuchs, **Roberto Calvo-Palomino**, Vincent Lenders, Sofie Pollin. Demo: ElectroSense - Spectrum Sensing with Increased Frequency Range. Published in *the 15th International Conference on Embedded Wireless Systems and Networks (EWSN 2018), 14-16 February 2018, Madrid, Spain.*

- This work is fully included and its content is reported in Chapter 2.
- The author's role in this work is focused on implementing and integrating the embedded spectrum sensor to the expansion board.

Abstract

The Electromagnetic (EM) spectrum is well regulated by frequency assignment authorities, national regulatory agencies and the International Communication Union (ITU). Nowadays more and more devices such as mobile phones and Internet-of-Things (IoT) sensors make use of wireless communication. Additionally we need a more efficient use and a better understanding of the EM space to allocate and manage efficiently all communications. Governments and telecommunication operators perform spectrum measurements using expensive and bulky equipments scheduling very specific and limited spectrum campaigns. However, this approach does not scale as it can not allow to widely scan and analyze the spectrum 24/7 in real time due to the high cost of the large deployment. A pervasive deployment of spectrum sensors is required to solve this problem, allowing to monitor and analyze the EM radio waves in real time, across all possible frequencies, and physical locations.

This thesis presents ElectroSense, a crowdsourcing and collaborative system that enables large scale deployments using Internet-of-Things (IoT) spectrum sensors to collect EM spectrum data which is analyzed in a big data infrastructure. The ElectroSense platform seeks a more efficient, safe and reliable real-time monitoring of the EM space by improving the accessibility and the democratization of spectrum data for the scientific community, stakeholders and the general public. In this work, we first present the ElectroSense architecture, and the design challenges that must be faced to attract a large community of users and all potential stakeholders. It is envisioned that a large number of sensors deployed in ElectroSense will be at affordable cost, supported by more powerful spectrum sensors when possible. Although low-cost Radio Frequency (RF) sensors have an acceptable performance for measuring the EM spectrum, they present several drawbacks (e.g. frequency range, Analog-to-Digital Converter (ADC), maximum sampling rate, etc.) that can negatively affect the quality of collected spectrum data as well as the applications of interest for the community.

In order to counteract the above-mentioned limitations, we propose to exploit the fact that a dense network of spectrum sensors will be in range of the same transmitter(s). We envision to exploit this idea to enable smart collaborative algorithms among IoT RF sensors. In this thesis we identify the main research challenges to enable smart

collaborative algorithms among low-cost RF sensors. We explore different crowdsourcing and collaborative scenarios where low-cost spectrum sensors work together in a distributed manner. First, we propose a fast and precise frequency offset estimation method for low-cost spectrum receivers that makes use of Long Term Evolution (LTE) signals broadcasted by the base stations. Second, we propose a novel, fast and precise Time-of-Arrival (ToA) estimation method for aircraft signals using low-cost IoT spectrum sensors that can achieve sub-nanosecond precision. Third, we propose a collaborative time division approach among sensors for sensing the spectrum in order to reduce the network uplink bandwidth for each spectrum sensor. By last, we present a methodology to enable the signal reconstruction in the backend. By multiplexing in frequency a certain number of non-coherent low-cost spectrum sensors, we are able to cover a signal bandwidth that would not otherwise be possible using a single receiver.

At the time of writing we are the first looking at the problem of collaborative signal reconstruction and decoding using In-phase & Quadrature (I/Q) data received from low-cost RF sensors. Our results reported in this thesis and obtained from the experiments made in real scenarios, suggest that it is feasible to enable collaborative spectrum monitoring strategies and signal decoding using commodity hardware as RF sensing sensors.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACARS Aircraft Communication Addressing and Reporting System

ADC Analog-to-Digital Converter

ADS-B Automatic Dependent Surveillance - Broadcast

AGC Automatic Gain Controller

AIS Automatic Identification System

AM Amplitude Modulation

API Application Programming Interface

AWGN Additive White Gaussian Noise

BPPM Binary Pulse Position Modulation

BPSK Binary Phase-Shift Keying

CRC Cyclic Redundancy Check

DAB Digital Audio Broadcasting

DBPSK Differential Binary Phase Shift Keying

DFT Discrete Fourier Transform

DAS Dynamic Spectrum Access

ECDF Empirical Cumulative Distribution Function

EM Electromagnetic

FFT Fast Fourier Transform

FIR Finite Impulse Response

FM Frequency Modulation

GPIO General Purpose Input/Output

GPS Global Positioning System

GPSDO GPS Disciplined Oscillator

GSM Global System for Mobile communications

HDFS Hadoop Distributed File System

HTTP Hypertext Transfer Protocol

IoT Internet-of-Things

IP Internet Protocol

I/Q In-phase & Quadrature

JCR Journal Citation Reports

JSON JavaScript Object Notation

KF Kalman Filter

LO Local Oscillator

LoRa Long-Range

LS Least Squares

LTE Long Term Evolution

MLE Maximum Likelihood Estimator

MQTT Message Queue Telemetry Transport

MSE Mean Square Error

NAT Network Address Translators

NTP Network Time Protocol

OFDM Orthogonal Frequency Division Multiplexing

OSDaaS Open Spectrum Data as a Service

PCI Physical Cell Id

PLL Phase-Locked Loop

ppm Parts Per Million

PPS Pulse Per Second

PSD Power Spectral Density

PSS Primary Synchronization Signal

RF Radio Frequency

RMSE Root Mean Square Error

RSSI Received Signal Strength Indicator

RTC Real Time Communication

SDR Software Defined Radio

SNR Signal-to-Noise Ratio

SPR Sync Phase Reversal

SRTP Secure-Real-Time-Protocol

SSR Secondary Surveillance Radar

SSS Second Synchronization Signal

TCXO Temperature Controlled Local Oscillator

TLS Transport Layer Security

ToA Time-of-Arrival

UDP User Datagram Protocol

UI User Interface

USRP Universal Software Radio Peripheral

Part I

Introduction

"The secret of getting ahead is getting started."

Mark Twain (1864 – 1910)

1 Introduction

The Radio Frequency (RF) Electromagnetic (EM) spectrum is a scarce, precious and widely used resource for many different tasks and purposes in our society. For instance, the communication between the robots on Mars and the Earth is performed over the radio EM spectrum by sending radio waves. Location systems such as Global Positioning System (GPS), that is extensively used in aircraft, cars or smartphones, make use of the radio EM waves. Public broadcast transmissions such as radio/television or even everyday devices as Bluetooth peripherals, wearables and surveillance cameras use the radio EM waves to transmit and exchange information. Mobile data traffic, that increases every day, normally uses WiFi and Long Term Evolution (LTE) wireless communication technologies to provide content to the users. In other words, EM spectrum is used everywhere.

In the 1980s, the only concern for spectrum management was mostly about radio/television broadcasting and military communications. This is rapidly changing today. We are in the age of the Internet-of-Things (IoT) where more and more devices are connected to the Internet sending and exchanging information using the radio waves. We expect to see more than 20 billion inter-connected devices by 2020 [12] and more than 75 billion by 2025 [13]. All of them will make use of the EM spectrum by using different RF technologies such as WiFi, LTE, 5G, Bluetooth, Long-Range (LoRa), SigFox, etc. in order to be connected to the Internet and send information. In addition, mobile phones are becoming the main device used for multimedia content consumption which wireless data usage has increased very quickly. These mobile devices will take advantage of the fifth generation (5G) of cellular mobile communication, which is expected to provide high wireless network capacity, up to 100 times more than actual networks [14]. As a result, RF spectrum is becoming a limited resource and its use is fragmented, bursty and very diverse. Monitoring the spectrum becomes a complex task, as it requires a very dense information in frequency, time and space. Radio communications are and will remain essential for society, yet the traditional approaches for monitoring the spectrum use very expensive and bulky equipment. The latter disables the possibility to deploy large-scale deployments running 24/7 to get a better understanding of the EM space. A new spectrum

monitoring paradigm is needed to sense the EM waves, keep safe radio communications and make sure that RF technologies are regulation compliance.

Traditionally the governmental agencies and international organizations (such as the International Communication Unit, ITU) are responsible for regulating the usage of the radio EM spectrum. Each country creates and maintains its own national frequency allocation plan which describes how the EM spectrum shall be used [15]. Despite the EM spectrum is well-organized in terms of frequency allocation, technologies and organizations which can use it, its actual usage in different geographical places and times is not well-known at all. Today's spectrum measurements are mainly performed by governmental agencies and telecommunication companies which drive around using expensive and specialized hardware [16, 17]. This spectrum monitoring approach does not allow to create a well-scaled infrastructure to cover the pervasive deployment of wireless networks. Therefore the research community has recently focused on low-cost Software Defined Radio (SDR) for sensing the spectrum, such as the RTL-SDR [18]. A large-scale RF spectrum monitoring system in real time is desirable for a multitude of practical reasons. Storing spectrum data at large scale can provide a historical spectrum database that allows to get a better understanding and knowledge about how the EM spectrum is used. Analyzing the EM spectrum in real-time at large-scale and 24/7 is challenging, but can offer a wide-range of possibilities to build smart applications to detect anomalies or non-authorized transmitters in specific bands [19]. The upcoming smart and agile radios, cognitive radios [20, 21] could use spectrum data knowledge to detect what spectral resources are unused and exploit them to provide high throughput and better services. From the cyber-security point of view, it is important to monitor the spectrum and protect it from attacks whose aim is to destabilize the communications in a country and negatively impact its economical opportunities.

In the recent years, the idea of distributed spectrum monitoring has gained attention to monitor and capture the real-time usage of the RF spectrum at large geographical scale. Several platforms use expensive and specialized hardware for sensing the spectrum at large scale such as DARPA's Spectrum Challenge [22] and Microsoft Spectrum Observatory [16]. Other platforms such as Google Spectrum Database [23], BlueHorizon [24] or KiwiSDR [25] are application oriented (e.g. TV white spaces, amateur radio 0-30 MHz) and cannot be used for other purposes. Other works such as SpecNet [17], SpectrumSense [26] or RadioHound [27] propose the use of commodity hardware [28] for sensing the spectrum to analyze its usage, transmitter coverage estimation, etc. However, the previous works make use of expensive equipment to monitor the spectrum, and they are all limited to applications in which power spectrum measurements or spectrum usage are sufficient.

In this thesis we present ElectroSense, a crowdsourcing and large-scale system for monitoring and decoding the EM spectrum in real-time using a dense network of IoT spectrum sensors. In our vision, most of these IoT RF sensors will run on low-cost embedded boards, as they are affordable for most of users. In order to counteract the limitations of low-cost spectrum sensors, we propose to exploit the fact that a dense network of spectrum sensors will be in range of the same transmitter(s). These IoT spectrum sensors collect the spectrum information that is sent to the backend in a coordinated way to enable new *collaborative applications* as transmitter localization, anomaly detection, collaborative signal decoding, etc.

We envision a wide deployment of small-factor and low-cost sensors across cities monitoring and analyzing the spectrum in real time and reporting valuable information for users, telecommunication operators, regulators, and governments. This new paradigm for monitoring the spectrum presents new research challenges, which are presented in the next section.

1.1. Challenges

Monitoring the spectrum at large-scale, in real-time and making use of the crowdsourcing approach is challenging for several reasons ranging from the use of non-expensive and inaccurate hardware to fuse and reconstruct spectrum information in the backend. More details about the research challenges of this work are described below.

- *Real-time spectrum acquisition with low-cost IoT RF sensors:* Commodity and low-cost hardware has much more constraints in terms of computational calculation, memory and their RF front-end are limited in terms of sampling rate, frequency bandwidth, dynamic range which restrict the effectiveness of a spectrum monitoring system.

- *The need for a fine time and frequency synchronization:* A time/frequency fine synchronization is required in order to use In-phase & Quadrature (I/Q) data from multiple RF sensors for collaborative applications. This is not trivial to achieve with low-cost spectrum sensors that are distributively deployed and connected over the Internet.

- *Control of the uplink network bandwidth of the sensors:* In a crowd-sourced platform, the participation of the users is essential. Users may deploy the sensors at a location with a limited network bandwidth. The collection of I/Q (raw) spectrum information imposes a very large volume of data that most of the Internet connections may not handle properly. Smart algorithms and techniques are required to alleviate the network capacity needed for collaborative applications, and balance the uplink network load among RF sensors.

- *Spectrum data fusion using low-cost IoT RF sensors:* A large-scale deployment of sensors is an essential requirement for investigating collaborative strategies among them. Since the low-cost spectrum sensing devices have important hardware and computational limitations, fusing and reconstructing signals in the backend using spectrum data received from different RF sensors becomes challenging.

1.2. Contributions

The main contributions of this thesis have been published in 10 publications, of which 1 has been published in *IEEE Communication Magazine* (indexed in Journal Citation Reports (JCR)), and 1 is submitted to IEEE Communication Magazine and currently in *minor revision.* Other 3 publications have been published in tier-1 conferences (*IEEE INFOCOM* and *ACM/IEEE IPSN*) according to CORE2014[1] or ERA2010[2] datasets, 2 more publications have been published in workshops (*WiNTECH* and *TIWDC*) and also 3 demo papers have been presented in major conferences. In details,

Contribution 1. *Large-scale Spectrum Measurement Network.*
A large-scale framework for collaborative spectrum measurement has been investigated [4]. Crowd-sourcing networks, low-cost sensors and large-scale deployments are necessary conditions to enable the massive and worldwide spectrum monitoring in real-time. ElectroSense also allows users to remotely decode specific parts of the radio spectrum using a peer-to-peer communication system [7], widely used in well known applications such as Google Hangout and Skype. This spectrum sensing framework has been developed together with researchers and engineers from KU Leuven University (Belgium), Sero Systems (Germany) and armasuisse (Switzerland). The work made and explained in this thesis is focused on the full system design and the implementation of the spectrum sensor side.

- **Roberto Calvo-Palomino**, Héctor Cordobés, Markus Engel, Markus Fuchs, Pratiksha Jain, Marc Liechti, Sreeraj Rajendran, Matthias Schäfer, Bertold Van den Bergh, Sofie Pollin, Domenico Giustiniano, Vincent Lenders. ElectroSense+: Empowering People to Decode the Radio Spectrum *[Submitted to IEEE Communication Magazine, Minor Revision].*

- Sreeraj Rajendran, **Roberto Calvo-Palomino**, Markus Fuchs, Bertold Van den Bergh, Héctor Cordobés de la Calle, Domenico Giustiniano, Sofie Pollin, Vincent Lenders. ElectroSense: Open and Big Spectrum Data. *Published in IEEE Communications Magazine, 56 (1). pp. 210-217. ISSN 0163-6804.*

[1] http://portal.core.edu.au/conf-ranks/
[2] http://www.conferenceranks.com/

- **Roberto Calvo-Palomino**, Domenico Giustiniano, Vincent Lenders. Measuring Spectrum Similarity in Distributed Radio Monitoring Systems. Published in *Tyrrhenian International Workshop on Digital Communications (TIWDC 2017), 18-20 September 2017, Mondello (Palermo), Italy.*

Contribution 2. *Crowd-sourced collaborative Time-of-Arrival estimator.*
A collaborative, efficient and precise Time-of-Arrival (ToA) method estimator for aircraft signals has been developed [5] using low-cost SDR receivers. Having a nanosecond-precision ToA estimator allows to improve position estimation problems based on input ToA measurements [29].

- **Roberto Calvo-Palomino**, Fabio Ricciato, Blaz Repas, Domenico Giustiniano, Vincent Lenders. Nanosecond-precision Time-of-Arrival Estimation for Aircraft Signals with low-cost SDR Receivers. Published in *the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2018), 11-13 April 2018, Porto, Portugal.*

Contribution 3. *A precise and fast frequency offset estimation for low-cost SDR platform.*
An opportunistic frequency offset estimation has been developed [3] using LTE signals broadcasted by the base stations. One important performance aspect of the low-cost SDR receivers used in this work is related to the accuracy and stability of the internal local oscillator. Being able to precisely estimate the frequency offset of the internal clock becomes essential to enable collaborative methods and algorithms.

- **Roberto Calvo-Palomino**, Fabio Ricciato, Domenico Giustiniano, Vincent Lenders. LTESS-track: A Precise and Fast Frequency Offset Estimation for low-cost SDR Platforms. Published in *the 11th ACM Workshop on Wireless Network Testbeds, Experimental evaluation CHaracterization (ACM WiNTECH 2017), 16-20 October 2017, Snowbird, Utah, USA.*

Contribution 4. *Collaborative narrowband spectrum data for saving network bandwidth in the communication from the sensors.*
We have developed [1] a collaborative and distributed time-multiplexing mechanism to sample the spectrum in a coordinated fashion using several IoT spectrum sensors, and we have proposed techniques to identify and overcome errors in the timing information provided by sensors. The collaborative approach among low-cost SDR receivers alleviates the network bandwidth load used by each sensor by exploiting the similarity in the spectrum for nodes in the same coverage area.

- **Roberto Calvo-Palomino**, Domenico Giustiniano, Vincent Lenders, Aymen Fakhreddine. Crowdsourcing Spectrum Data Decoding. Published in *the 36th IEEE International Conference on Computer Communications (IEEE INFOCOM 2017), 1-4 May 2017, Atlanta, GA, USA.*

Contribution 5. *Increasing signal bandwidth by fusing spectrum information from different non-coherent receivers.*
The use of low-cost IoT RF sensors to have some important restrictions due to their low quality components. One of the major drawbacks is their limited sampling rate, which does not allow to decode wideband signals. We have proposed a methodology [6] to enable the signal reconstruction in the backend by multiplexing in frequency a certain number of non-coherent receivers (deployed in an area in range of the wideband signal transmitter), in order to decode a larger signal bandwidth that would not otherwise be possible using a single receiver. As such, it is equivalent to the reception of the signal by a high-end receiver.

- **Roberto Calvo-Palomino**, Héctor Cordobés de la Calle, Fabio Ricciato, Domenico Giustiniano, Vincent Lenders. Collaborative Wideband Signal Decoding using Non-coherent Receivers. Published in *the 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2019), in conjunction with CPS-IoT WEEK 2019, 14-16 April 2019, Montreal, Canada.*

1.2.1. Software released as Open Source

Several outcomes of this research work are pieces of software related with the ElectroSense architecture, algorithms and tools for SDR receivers. Publishing and releasing the software as open source allows to the community, researchers and practitioners to first, understand better the proposed solution and second, to make possible the replication and validation of the experiments. Table 1.1 shows the software released as open source along this thesis.

Table 1.1: Software released as Open Source.

Name	License	URL	Paper
es-sensor	GPLv3	github.com/electrosense/es-sensor	[4, 7]
LTESS-Track	GPLv3	github.com/electrosense/LTESS-track	[3]
dump1090-hp	GPLv3	github.com/openskynetwork/dump1090-hptoa	[5]
Mode S decoder	GPLv3	github.com/openskynetwork/modes-uplink-decoder	[6]

1.3. Outline of the thesis

The rest of the thesis is organized in different chapters detailing the contributions aforementioned in the previous subsection.

The ElectroSense framework is presented in detail in Chapter 2 where we propose a crowdsourcing approach together with low-cost SDR receivers to create a collaborative, dense, and distributed spectrum monitoring system that enables novel applications using RF spectrum data. Chapter 3 describes how to empower people to decode the radio spectrum by allowing them to remotely decode specific parts of the radio spectrum. In Chapter 4 we present a fast and precise frequency offset estimator for SDR platforms which uses LTE signals to characterize the inaccuracies of SDR receivers.

In Chapter 5 we present one of the first collaborative scenarios using the ElectroSense network, a collaborative ToA estimation for aircraft signals is proposed using low-cost IoT spectrum sensors. We propose two novel methods that provide superior results for real-world aircraft signals than the state-of-the-art.

Chapter 6 describes the methodology proposed for sampling the spectrum collaboratively by multiplexing in time several low-cost sensors. Different techniques have been studied and proposed to solve the time synchronization required among spectrum sensors using embedded hardware with low accurate oscillators and distributively connected over the Internet. In Chapter 7 we propose a solution for one of the major limitations of the low-cost SDR receivers, the signal bandwidth. Our approach is based on the idea of collecting spectrum information by different non-coherent receivers in a collaborative manner, and then reconstruct the original signal in the backend, regardless of the fact that its bandwidth is higher (e.g. twice) than the bandwidth of each SDR receiver.

Finally, Chapter 8 draws the most important conclusions of this research work.

Part II

ElectroSense: Large-Scale Spectrum Measurement Network

Over the past years, we have seen a tremendous increase in mobile data usage. To meet the data demands, more Radio Frequency (RF) bands have been used, cells become smaller, data rates increase and more users accommodated. In addition, novel technologies have being proposed that can coexist with legacy technology and waveforms, and short and long range Internet-of-Things (IoT) communications increase the variety of physical and medium access protocols connected to the Internet. As a consequence, RF spectrum use is fragmented and very diverse. Monitoring the spectrum is a complex task, as it requires a very dense sampling in time, frequency and space, resulting in a radio spectrum data deluge. Nevertheless, with the increasing spectrum usage complexity, knowledge and understanding of the spectrum usage patterns is becoming more and more critical to ensure continued effective use of this scarce resource. A large-scale spectrum measurement network is required since the RF signals using the Electromagnetic (EM) spectrum could vary considerably in nearby locations (due to interference, fading, shadowing, multi-path, etc)

ElectroSense is a crowdsourcing network that measures and analyzes the EM spectrum in real time using small-factor and low-cost IoT RF sensors, yet also supporting more expensive spectrum sensors such as Ettus boards. In this part, ElectroSense network and its capabilities are introduced in detail. In Chapter 2 we present the main architecture of ElectroSense that allows to collect spectrum information from the sensors and send it to the backend. Several algorithms are executed in the backend on the collected spectrum information providing data already processed to the user. In Chapter 3 we describe how users can remotely control IoT spectrum sensors of the ElectroSense network to decode specific parts of the radio spectrum (broadcast and control signals) in real time through the Internet. We also propose the use of a virtual accounting system to first incentivize users to host ElectroSense sensors and second, regulate the access to the sensors in a fair manner. Finally, in Chapter 4 we propose a fast and precise frequency offset estimation (integrated in ElectroSense), for Software Defined Radio (SDR) platforms that makes use of Long Term Evolution (LTE) signals as a reference to determine the inaccuracies of the low-cost SDR receivers.

"If you think that the internet has changed your life, think again. The Internet of Things is about to change it all over again!."

Brendan O'Brien (Aria Systems)

2 Open and Big Spectrum Data with IoT Sensors

Spectrum resource monitoring is important for end users, operators, spectrum regulatory organisms and also military applications. Each use case has its own specific needs and challenges. The grand challenge is how to design a cost-effective solution that meets the requirements of all potential end users. Users might be interested in electrosmog or optimization of their indoor WiFi network. Regulatory organisms might be keen on enforcing spectrum regulation. Operators might be concerned about coverage maps over time for optimizing their cell networks or refarming of their frequency bands. Military applications might be the most challenging, requiring the detection and positioning of any signal hidden on purpose. Novel operators might be interested in Internet-of-Things (IoT) cases, such as cooperative detection of signals transmitted by low-cost low-power transceivers. The key aspect of this chapter is to design a spectrum sensing network that meets all the aforementioned requirements and can be deployed quickly by synergistic cooperation between all stakeholders.

Even though a majority of wireless researchers, industries and spectral regulators are keen to develop a worldwide spectrum monitoring infrastructure, several attempts, the research community has not succeeded in deploying one. The multidisciplinary nature of the spectrum monitoring solution is one of the main challenges that prevents the realization of such a system, which in turn requires proper integration of new disruptive technologies. The infrastructure should flexibly address the variety and cost of the used sensors, the need for large spectrum data management and sensor reliability, and the security and privacy concerns, which can also target a wide variety of the use cases mentioned before.

A few spectrum monitoring solutions are proposed in the literature. Some examples include, Microsoft Spectrum Observatory [16] that allows to sense the spectrum using expensive sensors, Google spectrum [23] for measurements on TV white-spaces and the IBM Horizon [24] project that proposed a generic decentralized architecture to share IoT data. While Google Spectrum and Blue Horizon fail to cover a large part of the spectrum as they are application specific deployments, Microsoft observatory on the other hand fails

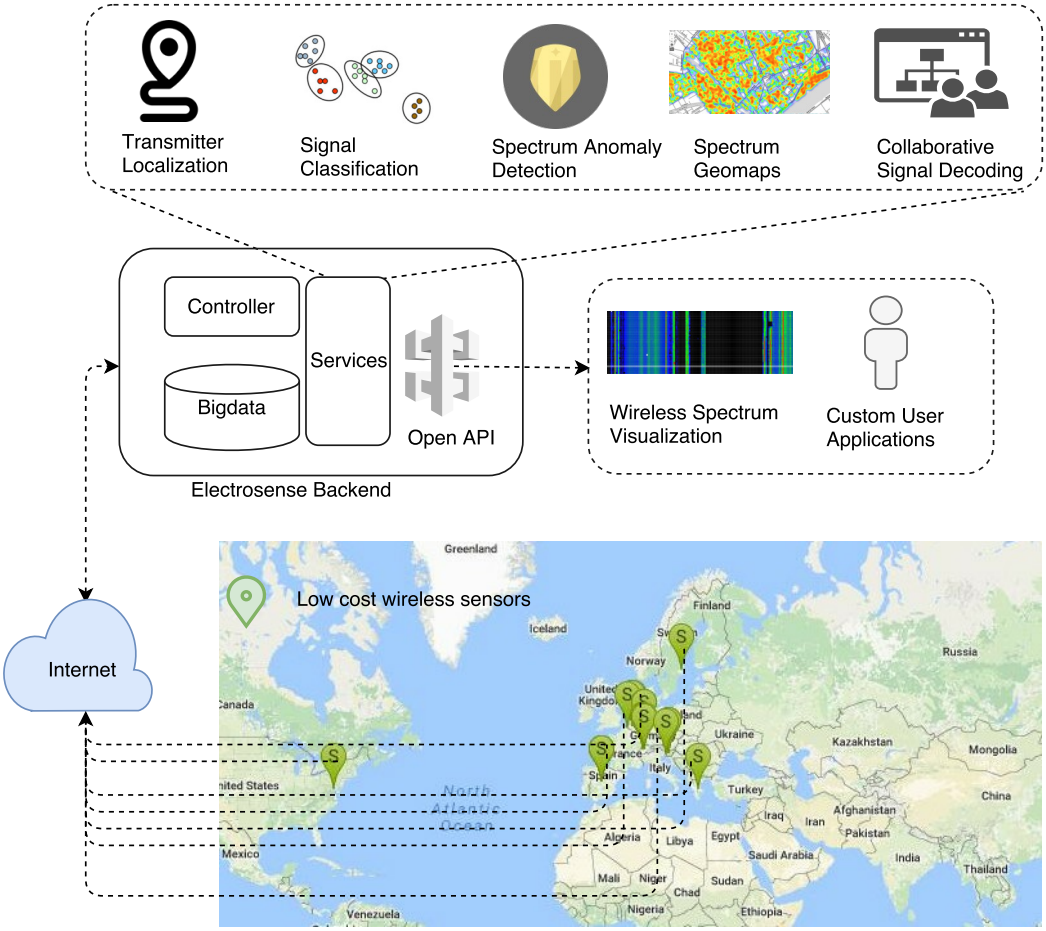


Figure 2.1: High-level overview of the ElectroSense network: Low-cost sensors collect spectrum information which are sent to the ElectroSense backend. Different algorithms are run on the collected information in the backend and the results of these algorithms are provided to the users as a service through an open API. Users can develop their own applications from the spectrum information retrieved using the API.

to enable a large scale sensor deployment mainly due to the cost of the sensing stations.

We introduce and design ElectroSense. ElectroSense follows the crowdsourcing paradigm to collect and analyze spectrum data using low-cost spectrum sensors as main device for sensing the spectrum. The main goal of this initiative, as shown in Figure 2.1, is to sense the whole spectrum in different regions of the world and provide the processed spectrum data to any user interested to acquire a deeper knowledge of the spectrum usage, enabling applications as the ones mentioned above [1, 30]. Worldwide deployments are plausible when the crowdsourcing paradigm is combined with low-cost sensors. ElectroSense sensors are designed using inexpensive and easily accessible Software Defined Radio (SDR) front-ends and embedded platforms like Raspberry Pi [31], to reduce the cost that spectrum data contributors have to bear. A low-cost down-converter

design[1] is also provided to extend the range of the low-end SDR platforms in two different frequency ranges: from 0 to 24 MHz and from 1.7 GHz to 6 GHz. Even though the aim of the ElectroSense project is to use low-cost hardware, high-end SDR devices can also be part of the network. Open source software modules for low- and high-end sensors are provided by ElectroSense for easy setup.

Crowdsourcing initiatives have been successfully applied in other sensing contexts, for example, distributed sensor networks collecting information about temperature, air quality, pollution or air traffic are now in widespread use today. Yet, the radio spectrum data collection poses novel challenges because of the sheer volume of information in the spectrum, which is several orders of magnitude higher than the data collected in typical sensing contexts as mentioned above. The amount of data produced by a single sensor typically varies between 50 Kb/s and 50 Mb/s depending on the sensor platform and configuration. Storing spectrum data received from each sensor can swiftly get any server machine out of storage space, for instance data from 60 sensors for a single month needs a storage space of 1 terabyte in 50 Kb/s mode. In order to handle this large volume of data in the backend and extract meaningful information over the entire spectrum, a flexible big data architecture is designed. This spectrum management architecture is responsible for the sensors' control, data storage, and algorithm deployment in the back-end for further processing.

ElectroSense also includes a complete framework for identifying, locating and deploying sensors securely through a consistent registration process and remote control framework of the sensors. The sensed spectrum data from different sensors can be retrieved from the ElectroSense backend using an open API [32] over the Internet. Data privacy concerns are addressed by enabling public, private and restricted data access permissions which restrict user data access with suitable time resolutions. Readily available spectrum aggregation tools in the back-end help the network users, to do easy spectrum data analysis as an additional incentive. The data access permissions and other applications are detailed in Section 2.1 and Section 2.3 of this chapter.

The rest of the chapter is organized as follows. A brief overview of the service model and the open API are presented in Section 2.1. Section 2.2 defines ElectroSense system design considerations and the proposed architecture. Section 2.3 details the spectrum data analysis tools and applications. Finally, conclusions and future work are presented in Section 2.4.

[1] https://github.com/electrosense/hardware

2.1. Spectrum Data as a Service

2.1.1. Service Model

In order to make the most out of the data, we introduce an Open Spectrum Data as a Service (OSDaaS) model in which the spectrum data can be used by several applications, each of them with unique requirements. This approach differs from a classical "Infrastructure as a Service" model used in other contexts where different applications cannot run at the same time due to conflicting requirements. The spectrum data inherently changes over time and thus allocating the node to a specific application would reduce the information and the amount of users interested in the same spectrum data. Therefore, the current framework prevents the users from launching their own measurement campaigns as they only need to access gathered data from one of the two pipelines of the sensor: Power Spectral Density (PSD) and In-phase & Quadrature (I/Q). In contrast, the flexibility of the OSDaaS model is an incentive to join and use the network and an opportunity for a wide engagement and participation of citizens. In addition, the spectrum data gathered in the past can be re-processed with a new application in mind or new spectrum data can be compared with older spectrum data. Users can join the ElectroSense network easily by adding their own sensor through the web interface [33]. Specifics about the sensor such as antenna details and sensor location can be specified during the registration process. The sensor status and location are readily visible in a global sensor map once it is registered.

2.1.2. Application Programming Interface (API)

Alongside with data processing, it is important to provide the community with easy access to the data. An open API serves this purpose for bulk or streaming data retrieval. The API also allows access to the algorithms' output running in the backend, for example results of the modulation classification algorithms or the anomaly detector. ElectroSense users can retrieve magnitude data along with the sensor details using the API. The API allows two data retrieval modes: aggregated and raw Fast Fourier Transform (FFT) data. Aggregated query type allows the user to request for a bulk of data with specified frequency and time resolutions, after applying a predefined aggregation function such as averaging or max-value. Raw requests permit to retrieve magnitude FFT data acquired by a sensor as such without any modifications. Raw FFT data from a sensor can be accessed only by its owner. In addition, as shown in Figure 2.3, there is an I/Q data pipeline which is enabled on demand for data testing or for algorithms that work on I/Q data in the backend. The I/Q data from a sensor is only directly available to its owner through the API, but conclusions from algorithms related to I/Q data will be made available to all users. Furthermore, real-time spectrum monitoring is possible through a streaming API

in the backend. A few examples for data retrieval can be found in the repository[2].

2.1.3. Security and Privacy

Wideband spectrum monitoring on a large scale typically raises critical security and privacy concerns. Security concerns include secure data transmissions from the sensor and proper sensor identification. Data privacy concerns in terms of data access restrictions should also be addressed. In addition, capture and storage of I/Q data, which can be decoded to reveal content, is not advisable especially in the military bands. ElectroSense's design addresses sensor identification via a proper registration process. The data from ElectroSense sensors are sent over secure Transport Layer Security (TLS) channels which guarantees data privacy and integrity and prevents the data getting modified by an attacker. The I/Q pipeline is only accessible by the ElectroSense backend and the data is deleted once the backend algorithms are done processing it. In addition, I/Q data is only made available to the users for their own sensors through the API. Furthermore, the highest time and frequency resolution that users can get from other sensors over the API are limited to 60 seconds and 100 kHz respectively. ElectroSense also allows to obfuscate sensor's publicly displayed location within a range of a few kilometers in the sensor map, thus protecting users' location privacy.

2.2. System Architecture

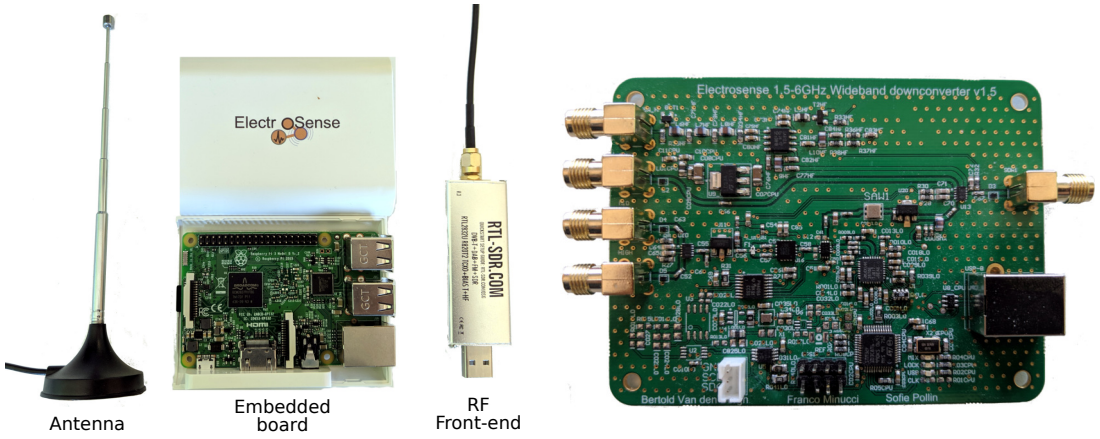
ElectroSense was designed with a clear vision in mind that translates into a list of design goals as described below. We present the actual architecture chosen to meet these requirements and give a more detailed explanation of its components.

2.2.1. Design Goals and ElectroSense Vision

ElectroSense is a crowdsourcing approach towards spectrum monitoring where volunteers play a central role. To keep the entry barrier as low as possible, large-scale deployment can only be achieved with *low-cost IoT sensors*. The backend should be flexible, to enable *wide-band monitoring* from sweeping low-end sensors, to *real-time scanning* with more capable SDR platforms or spectrum analyzers. In addition, the backend should be horizontally *scalable*, enabling to grow the network continuously. Scalability introduces even more complexity into the system, which makes it prone to failures. As such, the architecture must also be *fault-tolerant*. Finally, the backend should be able to process a large amount of data from any sensor with *low latency*.

With its main goal to serve researchers as a platform for spectrum analysis, the system should have a *central component to control* the sensors for specific measurement

[2] https://github.com/electrosense/api-examples



(a) IoT sensor: Raspberry Pi and RTL-SDR (b) RF converter to enable 0-6 GHz range.

Figure 2.2: ElectroSense IoT spectrum sensor.

campaigns. Some applications might need to give immediate response while others generate insights by inspecting the data set as a whole. Following these requirements, the system should support *low-latency stream processing* and *large-scale batch analyses* at the same time.

In order to address these goals and requirements, the ElectroSense architecture consists of three main components: (i) the sensors deployed by users collecting spectrum measurements, (ii) a centralized controller infrastructure to interact with the sensors and to administrate measurement campaigns and (iii) the backend which is responsible for collecting data from all the sensors and provide insights by applying algorithms. The following subsections describe these components in detail.

2.2.2. IoT Spectrum Sensor

The spectrum sensing nodes used in the ElectroSense network consist of small-sized, low-cost, software-defined embedded computing devices connected to a simplistic Radio Frequency (RF) front-end and a general purpose antenna [34] as shown in Figure 2.2(a). The sensors can measure the spectrum ranging from 0 MHz up to 6 GHz using an optional down-converter shown in Figure 2.2(b). Occasionally sensors may include an optional low-cost Global Positioning System (GPS) device to synchronize the time among them which in turn helps in enabling collaborative spectrum scanning and detection algorithms. For instance, in Chapter 6 the feasibility to recombine signals collaboratively from different sensors, using GPS as a reference clock, is analyzed in detail.

Two signal pre-processing pipelines are enabled on the sensor as illustrated in Figure 2.3. Each sensor can be configured in order to work with both PSD or I/Q pipeline. When the sensor is configured in the PSD mode, the spectrum sensed by the RF front-end is converted to the frequency domain using a FFT. In this mode only the averaged

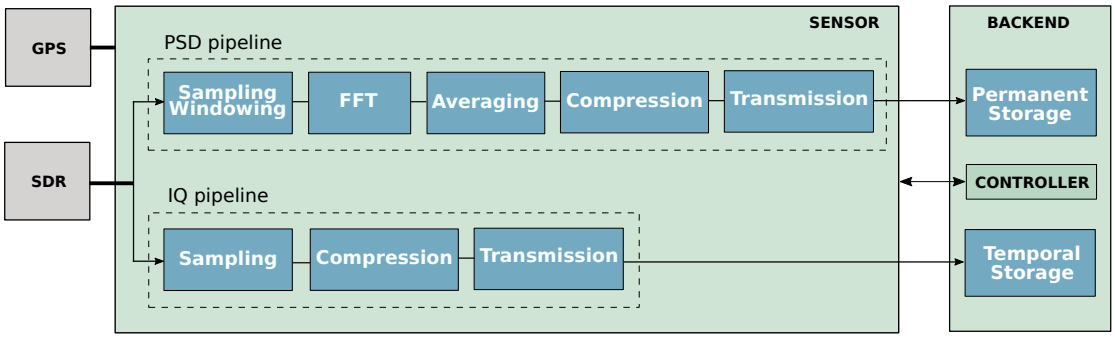


Figure 2.3: Spectrum sensor architecture: Sensor software contains two different pipelines for retrieving spectrum information.

squared magnitude FFTs are sent to the backend. Depending on the averaging and compression factors, the produced data is in the order of 50-100 Kb/s. This pipeline saves bandwidth and storage over the I/Q pipeline and thus contributes to the cost-efficiency of the whole network. Notice that we use advance spectrum estimation techniques such as Welch's method [35] to reduce the noise in the power spectrum estimations. The PSD mode aids spectrum data retrieval using a restricted network bandwidth which enables applications where phase information is not crucial to analyze data in terms of signal power (Section 2.3.2). In addition, wide-band scanning techniques are applied that prioritize frequency bands of bursty activities over bands which are stationary [34] to overcome the hardware-limitations of low-cost radios, for instance the limited sampling bandwidth of the low-end Analog-to-Digital Converters (ADCs) (e.g 2.4 MHz for RTL-SDR).

For applications which require phase and non-averaged information, the sensor implements an I/Q pipeline. The I/Q mode retrieves raw measurements (I/Q samples) from the RF front-end that are then compressed and sent to the backend. This pipeline easily produces up to 50 Mb/s and hence the data is only stored temporarily in the backend.

The software that runs in the ElectroSense spectrum sensors is released as open source[3]. In addition to this, a GNU Radio based software module, gr-electrosense[4], is provided to allow high-end SDR users to be a part of the ElectroSense network (see Appendix B).

The work presented in this thesis considers uncalibrated SDR receivers for all spectrum data collected and analyzed by both PSD and I/Q pipelines. SDR receiver calibration is out of the scope of this thesis.

[3]https://github.com/electrosense/es-sensor
[4]https://github.com/electrosense/gr-electrosensexamplese

2.2.3. Controller

Apart from continuous wide-band monitoring, it is interesting to have a more detailed look at particular parts of the frequency spectrum. This is accomplished with the help of a command-and-control layer as depicted in Figure 2.4. In order to start and control such in-depth measurements, the controller communicates with sensors directly and influences their scanning strategy, some examples include, the frequency range the sensor should scan, the sensor frequency hopping strategy or the sensor sampling rate.

The core of the controller infrastructure is a Message Queue Telemetry Transport (MQTT) broker system which consists of a set of several independent machines forming a cluster. This ensures scalability and fault-tolerance in case of a hardware failure. MQTT is a publish-subscribe-based messaging protocol and has a wide adoption in IoT applications [36]. The brokers have a permanent connection to the sensors over secure TLS channels as well as to a machine called *master controller*. The master offers an interface where administrators can start and stop *measurement campaigns*. It is responsible for executing these commands on different sensors using the MQTT connection. Spectrum measurements collected by the sensors are passed to the backend over secure TLS channels as shown in Figure 2.4. The control layer is a part of the ElectroSense backend and works in a closed loop with the data processing path for easy response validation.

2.2.4. Backend

All data collected by the spectrum sensors is sent to the *backend*. In order to fulfill the design goals, in particular scalability, the system is built upon a Lambda architecture [37] with exclusively horizontally scalable system components. The Lambda architecture is a three-layered architecture consisting of *batch*, *speed* and *serving* layer. An overview of the whole backend architecture can be found in Figure 2.4. The batch layer is responsible for executing bulk analysis tasks on large data sets. These tasks, mostly involve complex long-running algorithms, whose results might take several minutes to hours to complete. To overcome this gap and, furthermore, provide the capability for real-time monitoring with sub-second delay, there is the need for a speed layer. It is possible to run different or identical applications on each of these layers simultaneously.

In addition, the ElectroSense backend has an *ingestion* layer to support high availability. It consists of a distributed message queue that stores multiple replicas of each incoming measurement. Implementation and functionality of each layer is detailed in the subsequent sections.

2.2.4.1. Ingestion Layer

Sensor measurements eventually reach the *collector* as shown in Figure 2.4. The collector inserts received data as messages into a distributed queuing system. Using a

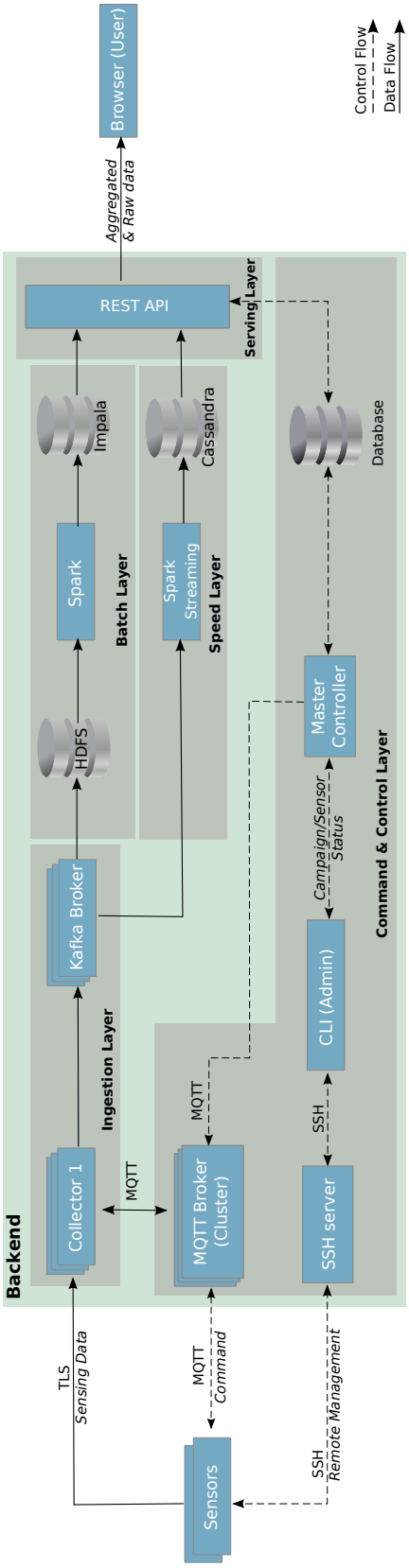


Figure 2.4: Sensors are managed by the control layer which sets the scanning parameters for a specific campaign. The configured sensors send the spectrum information to the backend, where the data is processed and distributed over the API to the users.

distributed queuing system, we decouple data ingestion from processing in the batch and speed layer, allowing an asynchronous mode of operation. Furthermore, the ingestion layer serves as an intermediate buffer for incoming data, allowing maintenance of batch and speed layers without data loss. Apache Kafka [38], a distributed messaging system which features replication, partitioning and data retention is used as the queuing system. Following the ingestion layer, spectrum data is processed on two different paths: in batch and speed layer simultaneously.

2.2.4.2. Batch Layer

The batch layer is responsible for storing all incoming raw data as an immutable master dataset and executing long-running workloads. The batch layer facilitates processing of huge amounts of historical data, ensuring no loss of information in case of algorithmic errors. The master dataset is stored in a binary format on a distributed file system. The Hadoop Distributed File System (HDFS) [39] is employed allowing data to be stored in files which are distributed and replicated over a cluster of servers.

Apache Spark [40], which allows parallel computation by evenly distributing workload over several computing nodes, is employed for data processing.

For accessing data stored on HDFS, a fast and scalable query engine Cloudera Impala [41] which understands a subset of the SQL language is deployed. The latency of the batch layer as a whole is in the order of minutes or even hours.

2.2.4.3. Speed Layer

The ElectroSense speed layer is based on Spark Streaming. It is an extension of Spark which computes its results continuously on a parallel cluster over a small window of recent data, with the current default window length set to 5 seconds. Spark Streaming meets the design requirements as the spectrum data is high in volume and algorithms do not demand sub-second latency.

As Impala is not suitable for near real-time storage requirements, the speed layer persists its results in another database. We use Apache Cassandra [42], a hybrid between a key-value and column-oriented distributed storage system for this real-time database as it represents well the data structure of spectrum data.

2.2.4.4. Serving Layer

As seen before, the batch and speed layers store their results differently. The serving layer takes care of data fusion to provide query results to the user while hiding the complexity of the multi-layer architecture. It offers the open API which serves as the endpoint for any user or application to retrieve data. The serving layer handles queries on different views and combines results from batch and speed layer if necessary. In case

results are available in both layers, batch layer results are preferred as they provide more accurate data. In ElectroSense, the serving layer is a custom component which offers a RESTful web service over Hypertext Transfer Protocol (HTTP).

2.3. Spectrum Data Processing and Analysis

To substantiate the value of the data generated by the ElectroSense framework, a few applications are presented in this section. We envision some of these applications to be integrated in ElectroSense's batch and speed layer in the backend, but the goal of ElectroSense is that users can start implementing their own applications by using the open API.

2.3.1. Live and Historical Spectrum Visualization

We have developed a web front-end to enable easy visualization of the spectrum data[5]. The spectrum visualizer allows interactive sensor selection and varying frequency resolution for detailed spectrum viewing. The visualization tool allows to look at historical spectrum data from the sensors, since the sensor registration. For this purpose, the batch layer pre-computes spectrum data at various temporal and frequency aggregation levels and stores this information in queryable tables for low-latency access over the web. For live spectrum updates with 5 second delays, a streaming display is incorporated to the web interface directly from the speed layer.

A real example about how useful is to store historical data of spectrum measurements is shown in Figure 2.5. Using an ElectroSense spectrum sensor located in Leganés (40.33,-3.77), we were able to detect (on June 2015) when some DVB-T channels stopped transmitting (800-860 MHz)[6] to make room for 4G networks[7].

Another real example is shown in Figure 2.6. Thanks to the real time 24/7 spectrum sensing that ElectroSense platform provides, we are able to go back to specific moments in time and check how the Electromagnetic (EM) spectrum was at that time. Figure 2.6 shows a spectrum screenshot corresponding to the 5th of June of 2018 in Leganés (40.33,-3.77), the city of Madrid. We can observe some anomalies around 09:40 where some channels stopped the transmission (2.6 GHz), and others started the transmission (780 MHz). We can also note certain interference in some of the Long Term Evolution (LTE) band frequencies (1.8 GHz and 2.2 GHz). A power outage in the region of Madrid was reported at that time which is most likely the cause of these interference and anomalies above-mentioned.

[5]https://electrosense.org/app.html
[6]Frequency reallocation - *Dividendo Digital* - https://es.wikipedia.org/wiki/Dividendo_Digital
[7]Spectrum data measured available in http://doi.org/10.5281/zenodo.168066

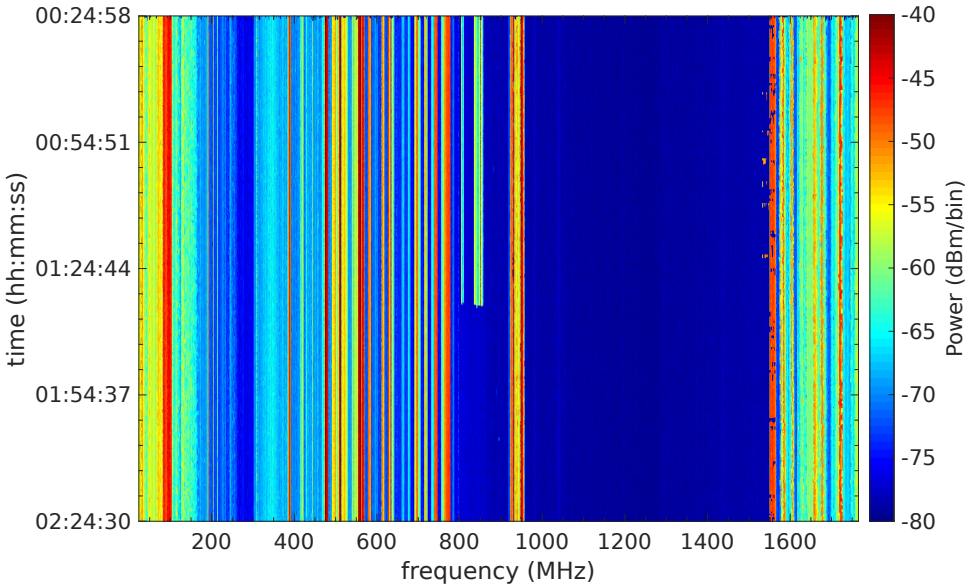


Figure 2.5: Some DVB-T channels stopped transmitting (800-860MHz) to make room for 4G networks (31/03/2015)

2.3.2. Dynamic Spectrum Access

Dynamic Spectrum Access (DAS) enables a technology model to manage spectrum availability and needs in terms of frequency, time, geography, quality and cost of the service. Cooperative and distributed spectrum sensing approaches for cognitive radios can solve hidden primary user problems and reduce the non-detection probability and false alarms remarkably [43]. Advanced collaborative sensing strategies along with detailed sensor density analysis is required to address the complete cognitive cycle. A white space detection study is done to validate the feasibility of the framework in this direction.

The term *white spaces* usually refers to the unused broadcasting frequencies in the TV band (400-800 MHz). The availability of white spaces is highly region dependent, for instance the white spaces in Europe in the 470-790 MHz band is found to be less than in the USA [44]. Using the ElectroSense infrastructure and the sensors configured in the PSD pipeline, it is feasible to detect the white spaces in a certain region. Figure 2.7 shows the spectrum occupancy in the broadcasting TV band (400-800 MHz) of two different cities in Europe with a frequency resolution of 1 MHz and 60 seconds time resolution. Similar analysis can be easily performed by retrieving the aggregated data from the ElectroSense backend through the API.

2.3.3. Spectrum Cop

Spectrum enforcement is challenging mainly due to the effort involved in analyzing, detecting and locating anomalies occurring in the spectrum. Even with a large

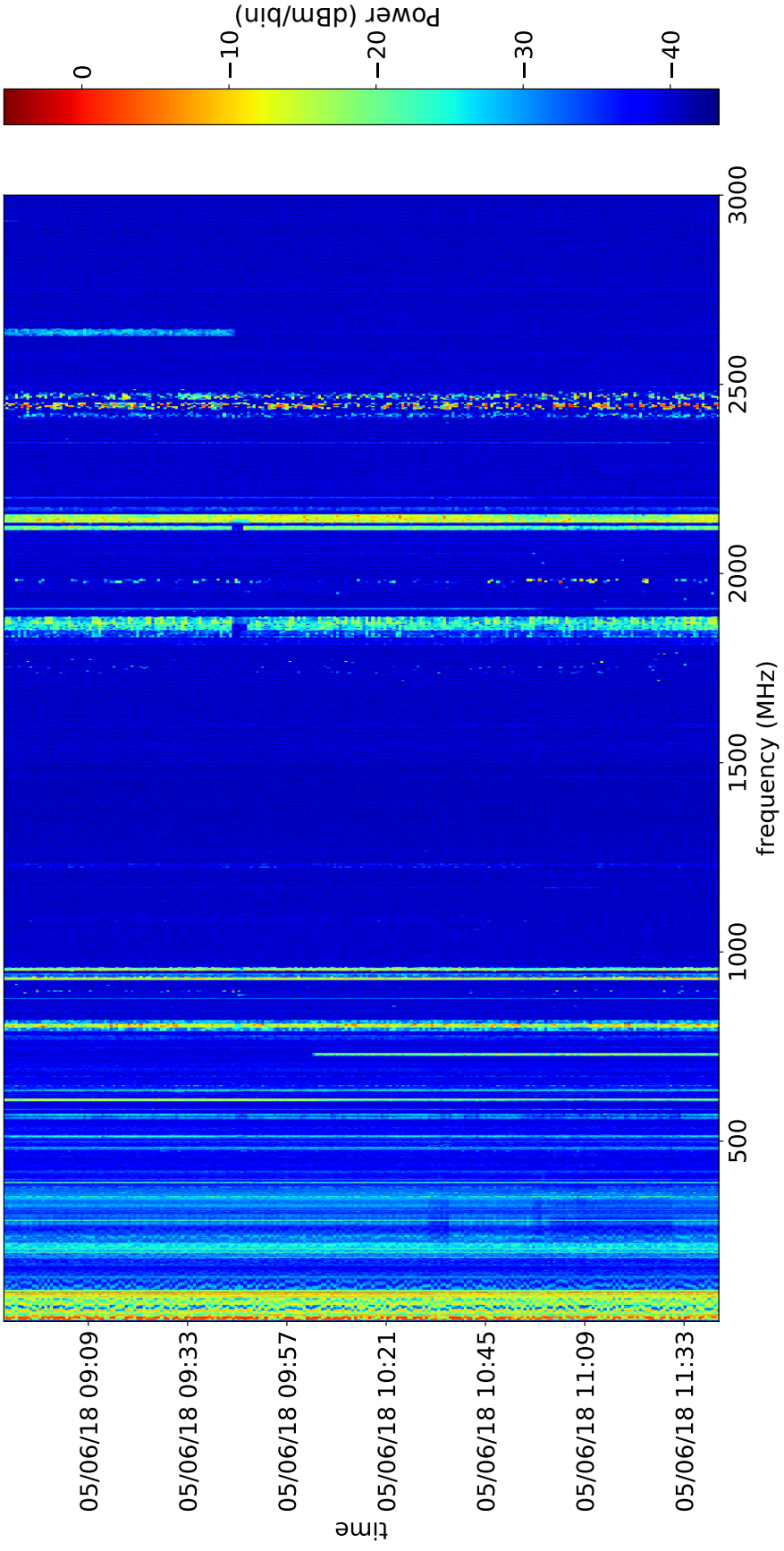


Figure 2.6: EM spectrum screenshot taken by a ElectroSense sensor

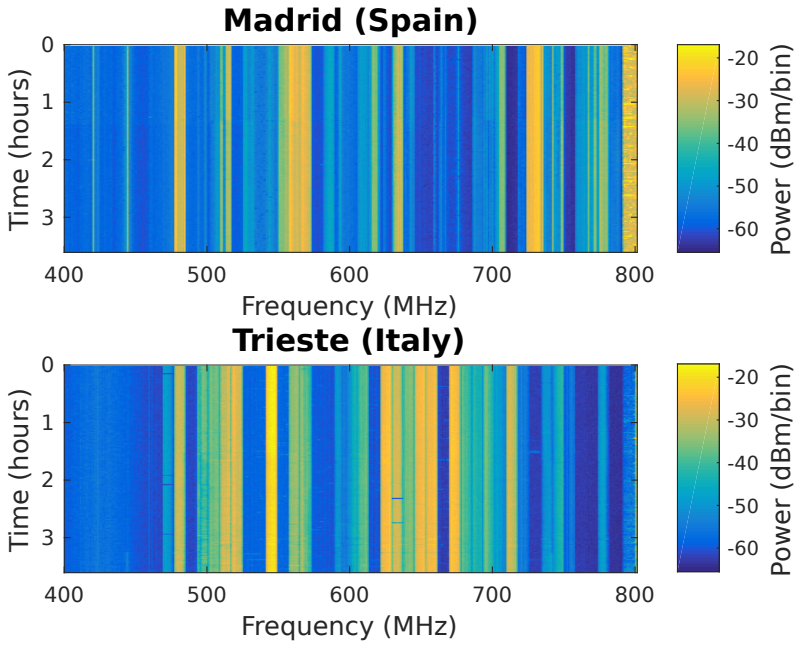


Figure 2.7: TV band occupancy in Madrid and Trieste.

scale deployment of spectrum scanners, the anomaly detection process is demanding. Automated systems to detect pirate Frequency Modulation (FM) stations, fake Global System for Mobile communications (GSM) towers, unauthorized transmissions hindering normal functioning of meteorological radars and even transmissions at higher power levels than the permitted levels of operation are fundamental for effective enforcement. A good anomaly detector should first detect whether a particular transmission type is allowed in the frequency of interest. Moreover, it should fingerprint normal spectrum transmissions by analyzing the temporal and spatial features of the transmitters in terms of power levels and spectral occupancy. The former can be achieved to some extent using a modulation classifier and the later by learning spectral occupancy distributions over time. A drastic variation of any of the aforementioned features can be classified as an anomaly.

To validate modulation classification using ElectroSense, a time domain deep learning modulation classifier is used in the backend which works effectively with gr-electrosense using the I/Q pipeline [45, 46]. The deep learning model takes I/Q samples as input giving out the probability of the data belonging to a particular modulation class. The model is trained to learn from the modulation schemes' time domain amplitude and phase information, without requiring expert features, such as higher order cyclic moments. Analyses show that the proposed model yields an average classification accuracy of over 90% at varying Signal-to-Noise Ratio (SNR) conditions ranging from 0 dB to 20 dB, independent of channel characteristics. In future a computationally efficient binarized version of the model will be deployed on low-cost sensors reducing the data overhead of

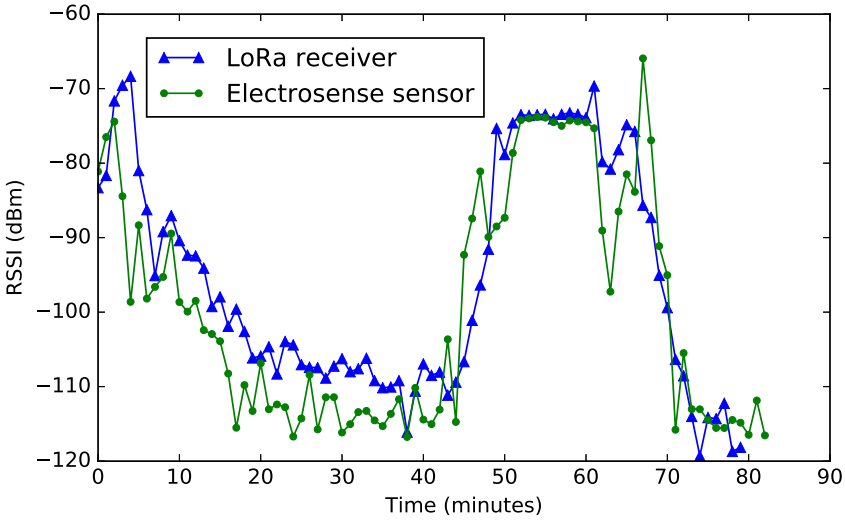


Figure 2.8: RSSI measured using a LoRa and ElectroSense sensor.

sending I/Q samples to the backend [46].

2.3.4. Localization

Transmitter localization is a promising application used for generating both automated transmitter maps and transmitter fingerprinting. Research on Received Signal Strength Indicator (RSSI)-based indoor and outdoor localization has gained interest in recent years [47]. However, the accuracy of the RSSI information drawn from the ElectroSense sensors must be verified to validate localization. Such a sensor is calibrated and then the system gain, consisting of the antenna gain, cable losses and front-end gain, is measured for absolute power measurements. RSSI measurements are made in the 435 MHz ISM band using standard Long-Range (LoRa) sensors. A splitter is employed to ensure an equal antenna signal reaches both a LoRa receiver and a low-cost ElectroSense IoT sensor. A mobile LoRa transmitter is configured to send packets every three second over a period of 80 minutes. Their GPS logged and time-stamped locations are sent to the receiver. The collected RSSI measurements from both the LoRa and ElectroSense sensors are depicted in Figure 2.8. This plot shows that accurate RSSI information can be obtained from properly calibrated ElectroSense sensors. The RSSI dips in the sensor data are attributed to the lengthy frequency scan of the sensor, which is over 40 seconds, causing it to miss some packets. Existing algorithms presented in the literature [47] can be deployed using the sensed spectrum data retrieved through the open API or implemented directly in the backend.

2.4. Discussion

We have introduced ElectroSense, a global initiative that seeks a more efficient, safe and reliable use of the electromagnetic space by improving the accessibility of spectrum data for the general public. ElectroSense as a crowdsourced network provides solutions to the problems associated with large-scale spectrum monitoring and the resulting data deluge. The network users receive incentives in terms of free data storage, readily available applications and an open API.

As the first version of the ElectroSense network is fully functional, the next step will be in the direction of enabling usable applications in the backend. The feasibility of various applications were already presented in the section 2.3 of this chapter. Research will be continued on the application space, to develop new algorithms for spectrum prediction and anomaly detection. Furthermore, algorithms to extract useful analytics from the spectrum data to overcome data storage constraints will also be explored in future. Detecting fake or forged data from malicious users, by enabling temporal spectrum comparisons among co-located sensors and user sensor ranking, is another active area for future research. Extensive studies will be also done to understand how complex algorithms can be disintegrated for large-scale deployment on these cheap sensors. ElectroSense can be the first step to democratize the access to the spectrum data to everyone. The age of spectrum data democratization has arrived and it could help to increase the transparency and the knowledge about spectrum usage.

"Alone we can do so little; together we can do so much."
Helen Keller (1880 – 1968)

3 Empowering People to Decode the Radio Spectrum

The idea of web-based distributed radio applications has recently gained interest [4, 48, 49], motivated by the diversity in space of the spectrum and the wide range of services benefiting from it. Multiple crowdsourcing initiatives have been proposed using various spectrum sensors ranging from low-end hardware to expensive spectrum analysers. They monitor the spectrum in a distributed way and provide applications that target specific communities.

Some of the major initiatives for analyzing the entire wireless electromagnetic spectrum are ElectroSense [4], Microsoft Spectrum Observatory [16], Google TV White Space [23] and IBM Horizon [24] and SpecNet [17]. Other initiatives focus instead on more specific monitoring applications over a limited frequency range, such as for example remote radio monitoring stations in OpenWebRX [50] and WebSDR [48], live air traffic control (ATC) broadcasts from air traffic control towers in LiveATC [49] or aircraft monitoring systems such as OpenSky [51]. Airspy [52] provides a sensor client-server architecture to operate Software Defined Radios (SDRs) remotely, but it relies on the computational power of the client-side to decode the signals, and high network bandwidth to send In-phase & Quadrature (I/Q) data stream to the client.

All the above initiatives have severe drawbacks, such as limited use cases (e.g., focus only on FM radio decoding or spectrum analysis), limited incentives to host a sensor (dynamic spectrum access and anomaly detection do not attract the large audience), require expensive SDRs or dedicated hardware (such as the Microsoft Observatory), poor scalability and complicated process to run measurement campaigns or access the data (sensors are busy), or high network requirements for sending I/Q data to the client.

Our vision is that people are the primary operators of Internet-of-Things (IoT) spectrum sensors. We aim at empowering people implementing a global spectrum system which let them connect to any spectrum sensor in the network and decode any publicly decodable radio spectrum part, such as broadcast and control messages, in real time through the Internet. In our system, spectrum analysis, or applications such as dynamic spectrum access and anomaly detection become secondary tasks, being active only if the

sensor is not used by people. The overarching goal is to support low-cost and software-defined IoT spectrum sensors devices and provide incentives for people to participate and host those sensors at their homes or organizations, enhancing the mission of building a crowdsourcing spectrum monitoring system.

In this Chapter we introduce ElectroSense+, an improved version of its predecessor (ElectroSense [4]) which was introduced in Chapter 2. The contributions are:

- We propose a novel radio spectrum decoding architecture where the primary operators are the sensors' owners. The architecture provides a transparent system to remotely decoding the spectrum on the IoT embedded sensors, and makes use of real-time peer-to-peer communication to send the information already decoded to the users.

- We implement the decoding process on the IoT spectrum sensors in an efficient way alleviating the processing load in the client, reducing the network bandwidth used, and adding a security-privacy layer since no raw data (I/Q) is sent to the users.

- We introduce an incentive for users to be part of the radio crowdsourcing community based on digital coins called Electrocoins. We propose a rewarding system for users that also helps to regulate the access rights of the users to the sensors in a fair manner.

- We evaluate the architecture proposed in real scenarios with 6 different decoders: Frequency Modulation (FM)/Amplitude Modulation (AM) radio, Automatic Dependent Surveillance - Broadcast (ADS-B), Automatic Identification System (AIS), Long Term Evolution (LTE), and Aircraft Communication Addressing and Reporting System (ACARS).

3.1. Design Goals

Past web-based spectrum monitoring initiatives are either application-specific [48, 50] or do not scale well for remote signal decoding [4]. Scalability is challenged by the large data volumes needed for wideband spectrum monitoring, much higher than needed by typical IoT applications. But, even with a larger bottleneck, we experienced that motivating users to deploy sensors and keep their sensors operational is the main hurdle for the wide-spread deployment of crowdsourced spectrum monitoring. The main reason is that the most interesting services for stakeholders that need to monitor the spectrum, such as governmental organizations and telecom providers, are orthogonal to the interests of the vast majority of users.

In this work, we propose a novel radio monitoring architecture that addresses the main limitations of previous systems:

General purpose decoding. The system architecture allows to decode any public decodable wireless technologies that is within range of the deployed sensors. As the spectrum is used by many different wireless technologies and new technologies are emerging constantly, we support the integration of open source spectrum decoders developed by the community. The system architecture thus defines open interfaces and Application Programming Interfaces (APIs) to allow easy integration of various decoder types.

Peer-to-peer architecture. As the system is expected to support a large number of concurrent users and spectrum data is very large in nature, a centralized approach is unfeasible as it would cause a data deluge to the backend or large latency from the sensor to the consumer. In order to support real-time applications and scale well, ElectroSense+ supports peer-to-peer communication between the IoT spectrum sensors and the users.

User incentive. Since in crowdsourcing initiatives, people are expected to acquire and run a spectrum sensor on their own, good incentives are needed to foster participation. This includes rewards for hosting a spectrum sensor but also to provide valuable spectrum services that they will get in return. In ElectroSense+, spectrum services are provided to the users in the form of *spectrum apps* and users receive *Electrocoins* for the time their sensors are online and used by the community.

Security and privacy. Spectrum data can contain private information and there should be limitations on some specific frequencies about the information type that users can listen to. For example, the system should not allow users to listen to private voice or other text conversations. It can instead decode broadcast and control messages. To this end, ElectroSense+ does not ship raw I/Q data to the users but only *aggregated* spectrum data and *filtered* decoded data. That way, ElectroSense+ keeps full control over the data that users will receive by enforcing strict integration policies on which decoders are allowed to run on the sensors and data is filtered.

3.2. Architecture

The ElectroSense+ architecture is depicted in Figure 3.1. The main system components are the IoT spectrum sensor, the client, and the backend. While these components were all present in the original ElectroSense design (described in Chapter 2), the novelty is to enable direct peer-to-peer connections between the sensor and clients in order to provide direct decoding services (apps) for users, and to account for the usage patterns in order to reward sensor operators. In this section, we focus on the new required architectural components for these enhancements.

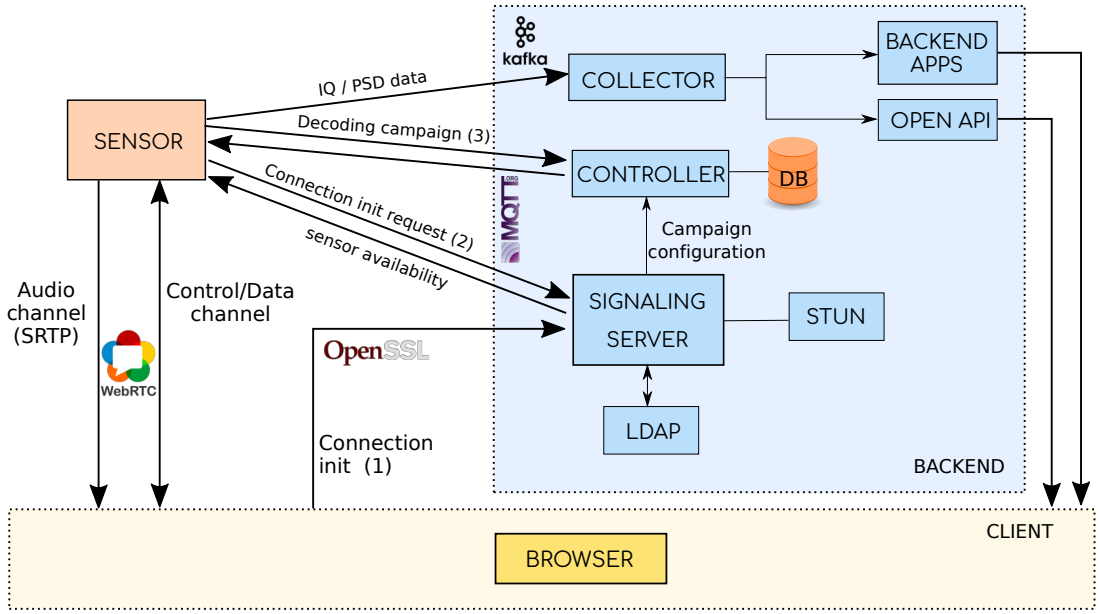


Figure 3.1: Full overview ElectroSense+ architecture

3.2.1. Signaling and Controlling

The sensors are managed by the ElectroSense+ backend over secure messaging via the Message Queue Telemetry Transport (MQTT) [53] protocol. The client-sensor connection is handled over the WebRTC protocol suite. WebRTC provides web browsers with Real Time Communication (RTC) without using dedicate plugins for this task. At an initial state of the communication, client and sensor need to exchange meta-data to coordinate the communication using the *signaling server*. The STUN server allows to find the public Internet Protocol (IP) address of the client and sensor in order to provide a direct connection between them, even if they are located behind firewalls or Network Address Translators (NAT).

When a client wants to connect to a sensor, it signals a request through the signaling server and establishes a direct connection to the sensor (without passing though the ElectroSense+ backend). Then, two different channels are created: Control/Data channel and Audio channel (as Figure 3.1 shows). The Control/Data channel is a bi-directional channel used to send the spectrum information and data decoded from the sensor to the client, and to command the sensing parameters from the client to the sensor (frequency, gain, etc.). The Audio channel is exclusively used to stream audio in real time from the sensor to the client using Secure-Real-Time-Protocol (SRTP). This peer-to-peer communication minimizes the network delays between the client and sensor, providing a fast and scalable access to the data from the spectrum sensor.

When there are no users that are connected to a particular sensor, the sensor is handled

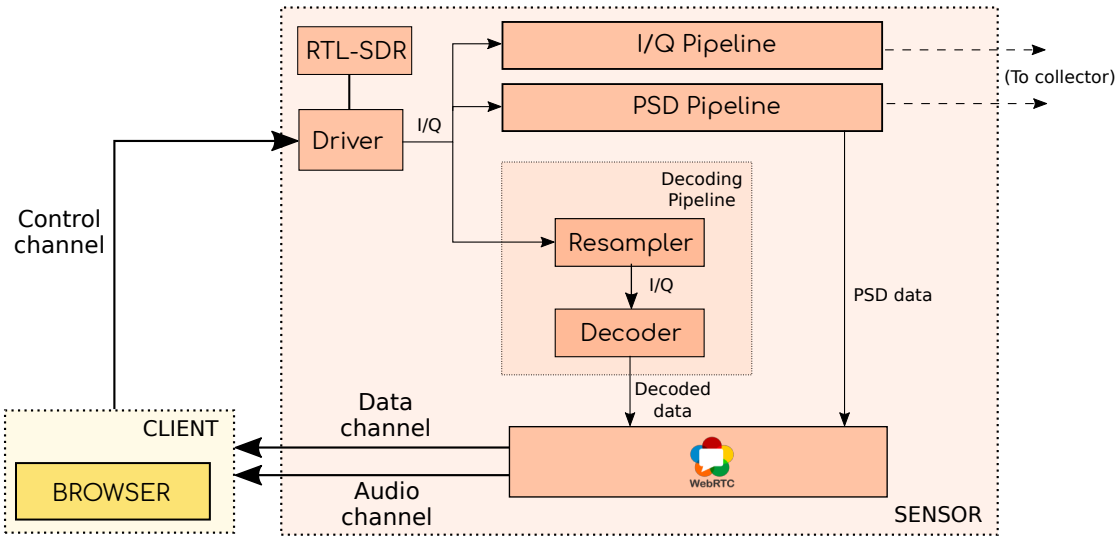


Figure 3.2: User-Sensor communication diagram.

by the controller in the backend instructing it to sweep the spectrum or launch a specific campaign. As soon as a client connects to a sensor the peer-to-peer communication is established. Then, the client takes over the control of the sensor and selects the sensing parameters (frequency, gain, etc.). Clients can remotely tune to any radio frequency and thus influence which decoder is activated in the sensor, e.g., if the Radio Frequency (RF) front-end is tuned to the FM radio band, the FM decoder will be active. If the client tunes to a frequency which has no associated decoder on the spectrum sensor, the client only sees a real-time waterfall diagram of the Power Spectral Density (PSD) data at the selected frequency band.

3.2.2. IoT Spectrum Sensor

The software of the IoT spectrum sensor is designed to run on low-cost embedded computing platforms such as Raspberry Pi [31] devices for the signal processing and RTL-SDR USB [54] dongles as radio front-end. The sensor architecture supports 3 spectrum data pipelines: I/Q , PSD and decoding pipeline as Figure 3.2 shows. The first 2 pipelines have been presented in Chapter 2. ElectroSense+'s architecture allows to run PSD and decoding signal processing pipelines in parallel. Both pipelines are reading the same I/Q samples from the RTL-SDR, but they process the data in a different way.

The spectrum analysis pipeline computes an aggregated PSD representation of the signal using the Fast Fourier Transform (FFT). The PSD data is sent to connected clients and the backend. At the client, the PSD data is useful for the user to visually analyze the spectrum in the frequency domain in real time, and to identify parts of the spectrum with

ongoing transmissions. In the backend, the PSD data is stored for historical inspection of the spectrum and to understand the evolution of spectrum activities over time.

The decoding pipeline is used to locally demodulate and decode the signals at the sensor. We implement data decoding in the spectrum sensor as it largely reduces the amount of data sent to the user. In addition, it avoids security and privacy concerns as no I/Q data is sent directly to the users which would allow them to decode any wireless signals, even those that may contain sensitive personal data. The sensor has multiple decoders to decode different parts of the spectrum. So far, we have integrated existing open source decoders including FM radio, AM radio, ADS-B / ACARS (air traffic signals), AIS (automated tracking system of ships), and LTE cell broadcasting. In the future, we plan to integrate many more technologies such as e.g., Digital Audio Broadcasting (DAB), DVB-T, Global System for Mobile communications (GSM) cell broadcasting, Long-Range (LoRa), Sigfox, etc. Given the limited resources of the embedded computing platform of the sensor, only one decoder is active on each sensor at the same time.

To facilitate the integration of existing and future decoders that are provided by the open source community, we have defined standard input and output interfaces. The input interface (used between the sensing software and the decoder) is a Unix User Datagram Protocol (UDP) socket over which the decoder can read the sampling parameters and I/Q data in chunks. For the output interfaces (used between the decoder and web-browser), we differentiate between a message and a streaming interface depending on the type of data that is decoded. If the data is text, the data is sent as a JavaScript Object Notation (JSON) object over a UDP socket. For audio and video, the data is streamed over a Unix UDP socket in a raw audio/video format.

Since the sampling rate that the decoders expect may be different from the configured sampling rate of the RTL-SDR, a re-sampler is implemented in front of the decoder to provide the sampling rate that the decoder needs. The re-sampler is also useful to decimate the I/Q data for performance reasons as some decoders consume too much CPU processing power on the limited hardware of the sensors when the sampling rate is too large.

3.2.3. Electrocoins

We propose a virtual accounting system based on our crypto-currency named Electrocoins, that provides two main features in our system. First, it helps to regulate the access to the sensors and distribute the rights among users in a fairly manner. And second, it is used as an incentive for people to host a sensors (they will be rewarded for deploying and hosting sensors). In addition, we plan to build Electrocoins on top of Ethereum [55] using digital tokens. In that case, Electrocoins would be directly compatible with any other contract that uses the Ethereum standards.

Our rewarding model consists of issuing Electrocoins to users which operate

ElectroSense+ sensors. Sensor operators are rewarded with Electrocoins for each sensor they operate. To avoid abuse, we check regularly the sensor quality in the backend which is determined by how often users connect to one sensor. The latter is most likely a good indicator that the sensor is deployed at a good location with a good antenna. This approach could also be combined with more intelligent algorithms in the backend based on signal learning capabilities [46] and anomaly detection [56].

Electrocoins are needed to directly connect to sensors and listen to the spectrum. Hence we expect that users deploy sensors in order to get Electrocoins, and then use them to connect to other sensors in order to consume spectrum decoded data.

3.2.4. Security and Privacy

While the ElectroSense+ architecture allows in principle to decode any type of wireless signals that fall in the frequency range of our sensors, we enforce a strict policy on the allowed decoders in the sensors in order to prevent from disclosing personal information to ElectroSense users. The decoders and their operational frequencies are set by ElectroSense+ to make sure that any decoded data provided by sensors does not violate any private information. Although the user can propose, implement and even integrate new decoders on the sensor side, the backend will not allow untrusted decoders avoiding privacy leaks.

By allowing users to listen to the content of such communication would violate the privacy of the persons using those devices. Our policy is thus to integrate only decoders for broadcast communication systems and for public control signalling messages. For example, in this work, we have implemented and integrated decoders for FM/AM radio, ADS-B and AIS (broadcasting systems). These decoders are integrated in the decoding pipeline on the sensor (see decoder block in Figure 3.2) where they read IQ samples as input and send the decoded data to the user through the data channel. The implementation of every decoder depends on the signal to be decoded. For communication systems such as LTE or ACARS, we only decode the signalling and management messages which are sent broadcast over the channel.

3.2.5. Spectrum Applications

Spectrum services are provided to the users by means of spectrum apps that give valuable information. Various users are interested in different aspects of the wireless spectrum, and the ElectroSense+ architecture is scalable and versatile enough to empower all use cases. The most widespread use of the wireless spectrum is however to broadcast information, and a primary set of spectrum apps focuses on the decoding of such broadcast information. As this broadcast information is intended for the general public, and not encrypted, hence there are no privacy concerns.

Table 3.1: Decoders, operational frequencies/bandwidths and open source projects.

Decoder	Frequencies	Bandwidth	Open Source Project
AM radio	153 kHz - 30 MHz	60 kHz	SciPy.org
FM Radio	88-108 MHz	240 kHz	SciPy.org
acars	129-136 MHz	2.4 MHz	acarsdec
AIS	162.025 / 161.975 MHz	1.6 MHz	rtl-ais
ADS-B	1090 MHz	2.0 MHz	dump1090
LTE-Cell	700 - 3500 MHz	1.92 MHz	LTE-Cell-Scanner

A first generation of ElectroSense+ spectrum apps focuses on the decoding of broadcast signals that are of interest to a broad audience, such as AM and FM signals, but also ADS-B, AIS or ACARS messages and LTE cells (see Table 3.1). These applications make use of the lower frequency bands, making it easier to achieve good coverage with a limited number of ElectroSense+ sensors. As the system scales to higher deployment densities, also higher frequency signals can be added. We note that the frequencies for AM signals are not covered by a standard RTL-SDR dongle, but with the ElectroSense expansion board [4] also lower (and higher) frequencies ranging from DC to 6 GHz can be covered.

3.3. User Interface

The user interface plays an important role in this work to empower people to use the system for decoding radio signals of the electromagnetic spectrum. Using standard technologies that execute in web browsers allows us to reach a great number of users which do not have any signal processing knowledge.

An example of the user interface of ElectroSense+ is shown in Figure 3.3. In this case, the user has selected the FM radio decoder and tunes the SDR receiver at 105 MHz. The user can visually inspect the spectrum in that band by checking the PSD data represented in the PSD plot. Although the decoder focuses in a narrow band for decoding the FM radio channel (180 kHz), we show the spectrum information of a wider band (2.4 MHz) for a better understanding of the spectrum by the user. The user can distinguish where transmissions are going on by checking the power in different frequencies. The user can click on the interested channel and he/she will start listening the current FM radio station. This audio streaming is sent using the direct audio channel between the sensor and the client. The user also can set different sensing parameters as the DC gain or the volume used by the decoder, or even the power scale to identify better the EM spectrum.

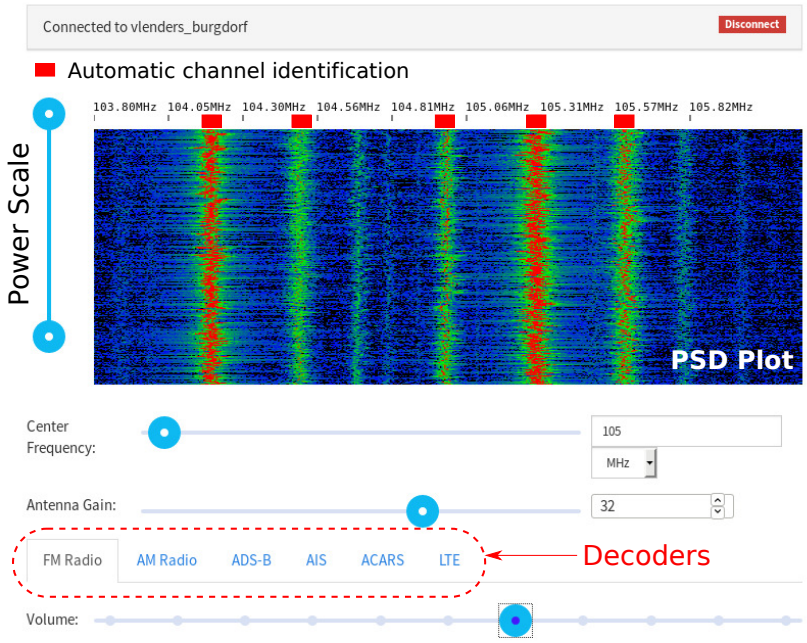


Figure 3.3: User interface for visualizing the spectrum and decoding FM radio.

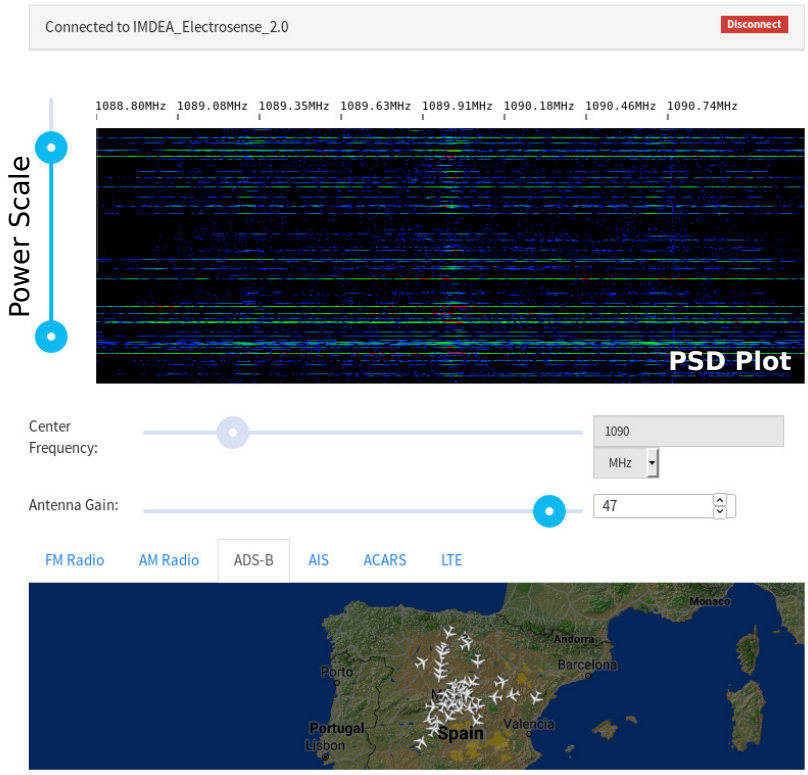


Figure 3.4: User interface for decoding ADS-B aircraft messages.

In order to help users to identify where the transmissions are, we provide an automatic channel identification for FM radio bands that it is executed in the client side. The browser runs a channel identification algorithm for detecting the transmissions using the PSD data, and identifies where FM radio channels are located. A red rectangle is shown in the web interface for every FM radio channel detected (see Figure 3.3), which makes it easier for the user to select another FM radio channel.

Figure 3.4 shows the interface when the user configures the sensor to decode ADS-B messages (aircraft continuously broadcast their position and other control information). Usually aircraft broadcast their position and other control information to other aircraft and/or ground stations (at 1090 MHz). Using the data channel, the sensor sends the PSD data to the client and also sends the information decoded on the sensor. In this case, the ADS-B decoder decodes the messages that aircraft send with information about their location. All this information is collected by the sensor and shown to the user in a map in the web interface.

3.4. Evaluation

In this section, we evaluate the performance of ElectroSense+ as a real-time system for decoding the spectrum remotely. The sensors are based on the Raspberry Pi-3B+ [31] and the RTL-SDR radio receiver [54]. The Raspberry Pi-3B+ has a Cortex-A53 processor and 1 GB of RAM, representing a typical low-cost IoT embedded device.

3.4.1. CPU Load and Throughput

The software executed on the Raspberry Pi model 3B+ is split in three main components: (1) sensing, which is responsible to execute the different pipelines to provide PSD data to the client and I/Q data to the decoders (including also the resampler); (2) WebRTC, which manages the communication between the sensor and the client; and (3) signal decoders. Table 3.2 shows the CPU load for the three software components and for every implemented decoder. The minimum CPU load of the sensing component (11%) occurs when only the PSD data is computed. When one of the decoder is enabled, the sensing process also needs to execute the resampler process, increasing the CPU load (14-24%). The load depends on the bandwidth and sampling rate that is expected by the decoder (see Table 3.1). The WebRTC CPU load is very low (0.1-0.2%) since the only task is to manage the communication of the data/audio channel. For the case that FM/AM decoders are enabled, the WebRTC component also needs to resample the audio stream to make the audio compatible with the expected audio input of the browsers. This increases the CPU load up to 6% for this component. The CPU load for every decoder and the total CPU is also shown in Table 3.2. The decoders who use the CPU the most are FM radio and LTECell, but still the Raspberry Pi has enough resources to properly

Table 3.2: CPU Load on the ElectroSense+ sensor and Throughput for every decoder.

Decoder	CPU Load sensing (%)	CPU Load Web-RTC (%)	CPU Load decoder (%)	CPU Load Total (%)	Throughput (kb/s)
PSD	11	0.1	-	11.1	120-140
FM radio	24	6.5	24.6	55.1	40-50
AM radio	22	6.2	10	38.2	40-50
ADS-B	14	0.2	4	18.2	190-200
AIS	20	0.1	6.2	26.3	50-60
acars	23	0.1	8.1	31.2	90-100
LTECell	21	0.2	48	69.2	10

process and deliver the decoded data in real time. While the FM/AM radio decoders have a constant CPU usage, other decoders such as LTECell only consume CPU resources for a specific amount of time (≈ 10 sec.) to compute the information that is delivered to the client. It is important to mention that the CPU usage of decoders such as ADS-B, AIS, or ACARS depends on the number of incoming messages over the channel.

The network bandwidth is also shown in Table 3.2. The data throughput for AM/FM decoders together with PSD is less than 200 kb/s, which is reasonable for most of the broadband internet connections nowadays. Other existing solutions like Airspy [52] (based on rtl_tcp) make a more intensive use of the network reaching 300-1000 kb/s for the same AM/FM decoding purpose. The maximum network throughput is used by the ADS-B decoder which together with the PSD data reaches 350 kb/s, still a data rate affordable for most home Internet users.

3.4.2. Real Time Response

Since the new ElectroSense+ architecture is built to provide a user experience close to real time for decoding signals, it is important to measure the time delays for the main tasks that can be performed by the user in the User Interface (UI). We want to measure the response time when the user accesses the sensor for the first time and the time between the moment a user selects a sensor until the first batch of PSD data flows in the web-browser. The average delay across the sensors attached to ElectroSense+ for the user to access the sensor for the first time is 0.97 seconds. This includes the time to establish the WebRTC connection between the browser and the sensor (see Figure 3.1), tuning the radio frontend to the frequency of operation, collect the I/Q stream, process the first batch of I/Q data through the PSD pipeline (see Figure 3.2) and deliver the first batch of processed PSD data through the data channel to the client (browser).

We have also evaluated the response time for the decoders, meaning that we measure

the average delay between the moment a user selects a decoder until the first decoded payload arrives to the web-browser. Every time a new decoder is selected by the user, the sensor requires to stop the previous decoder, re-tune to a new frequency, and start the new decoder (among other tasks). For radio FM/AM decoders, the average waiting time until the user starts to receive and listen to the audio stream is 2.6 seconds. Once the FM/AM decoder is set, the retuning by the user is much faster since there is no need to start the decoder again. For the rest of the decoders that stream a data flow encoded as JSON (e.g. ADS-B), the average waiting time for the user is 1.5 seconds.

3.4.3. Automatic Channel Identification

Using the PSD data sent by the spectrum sensor, the web-based client infers where active channels are located in the spectrum by applying a power-based channel identification algorithm. The idea of this feature is to support the user to identify interesting parts of the spectrum which are utilized and can potentially be decoded. We have evaluated the automatic channel identification algorithm performance over FM radio bands. As Table 3.3 shows, using an averaging window of 5 seconds over the PSD stream data, our model is 84% accurate with a precision of 0.88 (low false positive rate). Applying an averaging window over the PSD data has two positive effects: 1) reducing the noise and thus improving the performance of our algorithm, and 2) requiring less computation power on the client side.

3.4.4. Scalability

ElectroSense+ architecture takes advantage of peer-to-peer communication between the spectrum sensor and the user. The system scalability also depends on the traffic load of the signaling server (see Figure 3.1) which handles the control messages (connection request, keep alive, etc). These messages represent less than 2 kb/s per sensor, meaning that one instance of the signaling server with a 50 Mb/s symmetric network can manage more than 25K spectrum sensors at once.

Table 3.3: Automatic channel identification performance (FM Radio)

	TP	TN	FP	FN	Accuracy	Precision	Recall	F1
no-avg	14	31	2	9	0.80	0.88	0.61	0.72
avg-5sec	14	33	2	7	0.84	0.88	0.67	0.76

3.5. Discussion

We have presented ElectroSense+, a system that allows users to remotely decode specific parts of the radio spectrum using low-cost IoT devices as radio sensors. ElectroSense+ is built on top of its predecessor [4], and provides a new peer-to-peer communication among clients and sensors to exchange information and make the system scalable (a centralized approach would cause large latency and high load resources in the backend). The system architecture allows to decode any wireless signal that is within range of the sensors. We have integrated several publicly available decoders that are not intrusive to the privacy of the wireless users. Our decoders operate on the sensor-side and have optimized their computational performance to run in embedded and low-cost IoT devices. The decoders that are currently implemented are AM and FM for radio; ADS-B, ACARS and AIS for tracking systems; and LTECell for LTE signals. We manage to keep the average CPU load of the IoT sensors below 40% in most of the cases, even when the PSD and decoding pipelines are executed on the sensor at the same time. The communication channel is also implemented in an efficient way which allows to keep the network bandwidth low between the sensor and the client. For streaming a single audio channel to the user the bandwidth needed is 50 kb/s, while for sending data (e.g. generated by the ADS-B decoder) the bandwidth used can go up to 200 kb/s. In both cases the network bandwidth is low, allowing the users to connect to the system using conventional home Internet connections and WiFi hotspots. We have implemented a friendly user interface based on a web-browser (platform independent) for users to interact with the sensors. ElectroSense+ provides the opportunity for individuals to gain better knowledge and understanding of the spectrum utilization, by offering remotely signal decoding capabilities and direct incentives to deploy own sensors.

"The universe does not allow perfection."

Stephen Hawking (1942 – 2018)

4 Characterizing the Frequency Offset for SDR Platforms

In the previous chapters ElectroSense has been introduced as a spectrum monitoring platform that uses low-cost Software Defined Radio (SDR) as Radio Frequency (RF) receivers. The field of SDR is becoming increasingly popular among academics and practitioners. The popularity of SDR was unleashed by the combined availability of low-cost SDR hardware and free open-source SDR software. The so-called RTL-SDR dongle devices are nowadays among the most popular in the SDR community and are largely used in crowd-sourced projects [26, 33, 57].

Generally speaking, low-cost devices may be expected to have much higher Local Oscillator (LO) instability than other classical receivers, possibly limiting some potential applications. For example, in collaborative tasks amount sensors such as signal decoding, receivers may need to compensate for frequency offset effects. A time-varying LO offset might impede the correct estimation of time-of-arrival or time-difference-of-arrival, as relevant e.g. in time-based localization, since timing information is ultimately derived from LO. Also, it might impede the correct estimation of Doppler shifts [58]. In all such application categories, the software designer should then decide whether to include more or less sophisticated frequency correction methods into the SDR code to counteract the LO frequency deviations and fluctuations.

In general, understanding LO offset near real-time is essential to take the most appropriate actions. Low measurement delay is important for two reasons. First, it allows to swiftly evaluate short-term frequency fluctuations of the device under test using a recorded dataset. Second, it can be used as an ancillary tool serving other SDR applications that requires periodic re-estimation of absolute LO offset. For instance, low-cost RTL-SDR receivers scanning the spectrum may periodically re-tune their center frequency to some common reference signal (e.g. Long Term Evolution (LTE) or Digital Audio Broadcasting (DAB)) in order to estimate and correct their LO offset. This procedure should be as fast as possible to minimize any outage in the measurement campaign. For all theses reasons, we propose a fast and precise frequency offset estimator for SDR platforms that is integrated in ElectroSense platform.

The contribution of this work are three-fold:

- We present LTESS-track, a LO frequency offset evaluation tool that allows SDR practitioners to determine the frequency offset of their SDR devices without the need to acquire additional laboratory equipment, such as high-end signal generators or other methods. LTESS-track leverages the ubiquity of LTE coverage: it exploits the Primary Synchronization Signal (PSS) that is continuously broadcast by LTE base stations as a reference signal of opportunity. In principle, LTESS-track can work with any SDR front-end capable of tuning to LTE frequencies (see Appendix B). Our method is designed to deliver a frequency offset estimation with sub-ppm resolution and *maximum measurement delay below 1 second* in embedded environments such as Raspberry Pi. This is a particular feature of LTESS-track, not present in other existing LO offset estimation methods [59–61].

- We compare LTESS-track against three other popular software methods for low-cost LO offset estimation, namely the rtl_test [59], Kalibrate-RTL [60] and LTE-Cell-Scanner [61]. We show that previous works have under-exploited the potential of cellular signaling for frequency offset estimation and we demonstrate that our architecture design allows to achieve higher performance. As we show in our work, one common limitation of previous methods is that they are computationally expensive for running in embedded boards. As a result, they have a high measurement delay, up to 12 seconds (Kalibrate-RTL) or even several minutes (rtl_test). Furthermore, we show that some of these other tools present occasionally large errors (LTE-Cell-Scanner), or simply do not work in the presence of large LO offset (Kalibrate-RTL).

- We use our method to assess the actual LO performance of two very popular RTL-SDR models, namely the "Silver" and "Blue" models, respectively with and without Temperature Controlled Local Oscillator (TCXO). We consider both normal and harsh environments, with device temperatures exceeding 50 degrees Celsius. The results show how the new generation of RTL-SDR with TCXO, despite its low cost, has an exceptional LO stability in changing temperature environments.

LTESS-track implements several key mechanisms not presented in other methods, such as initial frequency offset compensation, up-sampling, sampling of data only in time proximity to the expected synchronization signal to reduce the computational cost and linear regression of samples. Our method will further contribute to the “popularization" of low-cost SDR development and related crowd-sourced SDR projects.

4.1. Problem statement

4.1.1. Receiver model

The available specifications do not provide the exact details in all the levels of the RTL-SDR hardware architecture [62]. For our work we have assumed the general architecture depicted in Figure 4.1 with a single LO that feeds both the Down-Conversion (DC) and the Sampling (S) stage by means of two distinct clock distribution networks.

We introduce the following notation:

- f_{LO} the *nominal* LO frequency.
- $\Delta f_{LO}(t)$ the difference between the *actual* LO frequency at time t and its nominal value, i.e., the *absolute* frequency offset of LO.
- $\gamma(t) \stackrel{\text{def}}{=} \frac{\Delta f_{LO}}{f_{LO}}$ the instantaneous *relative* frequency offset of LO at time t.
- f_D the *nominal* tune-in frequency.
- $\Delta f_D(t)$ the difference between the *actual* and nominal tune-in frequency at time t, i.e., the *absolute* frequency offset at the down-conversion stage.
- f_S the *nominal* sampling rate.
- $\Delta f_S(t)$ the difference between the *actual* and nominal sampling rate at time t, i.e., the *absolute* frequency offset at the sampling stage.

In general, we can assume that the *relative* frequency offset at the down-conversion and sampling stage are equal or anyway very close to the LO one, formally:

$$\frac{\Delta f_S(t)}{f_S} \approx \frac{\Delta f_D(t)}{f_D} \approx \frac{\Delta f_{LO}(t)}{f_{LO}} = \gamma(t). \tag{4.1}$$

The relative frequency offset $\gamma(t)$ is a dimension-less parameter. The specifications typically provide an indication of the maximum LO frequency tolerance ϕ expressed in Parts Per Million (ppm). For example, a relative frequency tolerance of 30 ppm means a maximum time offset of 30 microseconds in one second. The tolerance value represents an upper bound on the maximum relative deviation that may be expected, i.e. $|\gamma(t)| \leq \phi$.

4.1.2. Design goals

Our goal is to develop a generic method to estimate and evaluate the frequency offset of the low-cost RTL-SDR devices with the following features:

- *Reliable:* The method should report a reliable estimate of the LO frequency offset, with estimation error below 1 ppm.

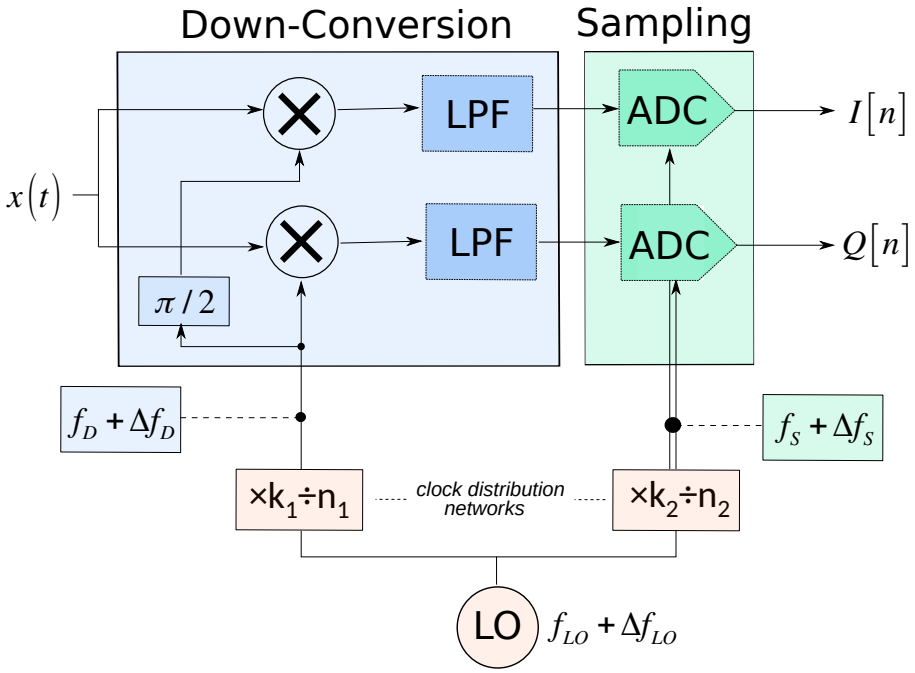


Figure 4.1: Reference receiver architecture.

- *Fast*: The method should be fast to provide new estimates with a maximum delay of 1 second, in order to minimize any outage in the spectrum measurement campaign.

- *Flexible:* The method shall be flexible enough to work with different RTL-SDR devices (TCXO and non-TCXO models), possibly with large LO offset values (several tens of ppm).

- *Efficient:* The method should be executed in small-factor embedded architectures such as Raspberry Pi.

4.2. LTESS-track

In this section, we detail the proposed methodology to estimate the LO offset of SDR devices. Our method relies on the availability of LTE signals that are captured by the SDR devices.

4.2.1. LTE signal model

We first briefly review a few fundamental concepts about LTE. Typically in LTE networks, the user needs to get the cell id of the base station and the frame synchronization to perform more complex operations. The first step in order to get the proper time and frequency synchronization is to search for the PSS and Second Synchronization Signal

Table 4.1: LTE Parameters

T_F	10 ms	Nominal period of LTE frames.
f_S	1.92 MHz	Nominal sampling frequency.
T_S	520 ns	Sampling period (f_S^{-1})
f_D	806 MHz	Nominal center frequency of the LTE cell *

* We have tested three different LTE cells at different frequencies: 796 MHz, 806 MHz and 816 MHz. All results were very similar. In this work we present only the results for the 806 MHz cell.

(SSS) which have a band of 1.4 MHz [63, Chapter 7] [64]. LTE defines two structures called frame and subframe [63]. Each frame has a duration of 10 ms and contains 10 subframes of 1 ms each. The PSS and SSS signals can be found in subframes 0 and 5 of every frame. The PSS is a frequency-domain Zadoff-Chu [64] 128 bits long sequence and encodes the layer identity of the cell. The SSS encodes the cell identity and is modulated using Binary Phase-Shift Keying (BPSK). For our purposes, we consider only the PSS signal and its periodicity (twice every 10 ms) to design a frequency offset estimation method.

The choice of LTE synchronization signals as absolute clock reference is motivated by the very high precision and stability of such signals: in fact, LTE base stations must meet strict requirements in terms of frequency stability with maximum tolerance below 0.05 ppm [65], i.e., much smaller than the expected tolerance of RTL-SDR devices currently on the market.

We consider the LTE parameters as shown in Table 4.1. The input stream of complex baseband In-phase & Quadrature (I/Q) samples at the sampling rate f_S will be denoted by $x[n] \stackrel{\text{def}}{=} x(t)|_{t=nT_s}$. We shall index in $k = 0, 1, \ldots$ consecutive LTE frames. Without loss of generality, we fix the time origin at the (true) arrival time of the PSS signal for the first frame $k = 0$. For the generic k-th frame, we denote by $y[k]$ the true (unknown) PSS arrival time, and by $\hat{y}[k]$ the corresponding *measured* value, as obtained with the measurement procedure detailed later in Section 4.2.2. Two distinct sources of errors affect the measured value $\hat{y}[k]$: the clock error ρ_k and the measurement noise e_k, i.e.

$$\hat{y}[k] = y[k] + \rho_k + e_k = kT_F + \rho_k + e_k. \tag{4.2}$$

The measurement noise term e_k is modeled by a sequence of i.i.d. random variables with zero-mean and variance σ_e^2. The clock error accumulated until the k-th frame is given by the integral of the instantaneous (relative) frequency deviation $\gamma(t)$ and can be developed

as

$$\rho_k = \int_0^{kT_F} \gamma(t)dt \approx \gamma k T_F + \int_0^{kT_F} \sum_{n=1}^{\ell} \beta_n t^n dt, \tag{4.3}$$

wherein the last term represents the time-varying component of the LO frequency and is modeled (approximated) by a polynomial of sufficiently high degree ℓ. From Eq. (4.3) we derive the general signal model for time-varying LO frequency:

$$\hat{y}[k] = (1+\gamma) \cdot kT_F + \sum_{n=2}^{\ell+1} \alpha_n k^n + e_k \tag{4.4}$$

with $\alpha_n = \beta_n T_F^n / n$. For a short observation interval we can neglect the time-varying component ($\alpha_n = 0$, $n = 1, \ldots, \ell$) and consider a *static* scenario with fixed frequency offset $\gamma(t) = \gamma$. In this special case the model simplifies as:

$$\hat{y}[k] = (1+\gamma) \cdot kT_F + e_k. \tag{4.5}$$

Consider an observation window of duration W seconds embedding $N \stackrel{\text{def}}{=} \lfloor \frac{W}{T_F} \rfloor$ frames. The vector of measurements collected in said window will be denoted by $\hat{\boldsymbol{y}} \stackrel{\text{def}}{=} \{\hat{y}[k],\, k = 1, \ldots, N\}$. The choice between the static model Eq. (4.5) and the dynamic model Eq. (4.4) depends on the duration W of the observation window and on the temporal stability of the LO. For short windows of a few seconds, we can neglect temporal variations and resort to the simpler static model Eq. (4.5).

4.2.2. Estimation of PSS arrival times

In this section, we describe the method implemented to obtain a precise estimate of the (sequence of) PSS arrival times $\hat{y}[k]$. The overall scheme is depicted in Figure 4.2. The PSS tracking stage is preceded by an initial acquisition stage.

The core block of the PSS detection process is a correlation filter: a chunk of $L = 128$ samples from the input I/Q stream starting at position m is correlated with the known ZC-128 template $z[n]$. However, we introduce the following refinements:

- Frequency offset compensation: in order to counteract the effect of frequency offset in the down-conversion stage, we consider the following frequency-adjusted template

$$z'[n] \stackrel{\text{def}}{=} z[n] \cdot \exp^{-j2\pi\hat{\gamma}_0 \frac{f_D}{f_S}} \tag{4.6}$$

  wherein $\hat{\gamma}_0$ denotes a coarse initial estimate of the frequency offset, obtained during the initial acquisition phase.

- Up-sampling: in order to achieve sub-sample resolution for the individual estimate $\hat{y}[k]$ we up-sample the I/Q stream by a large factor U. Unless differently

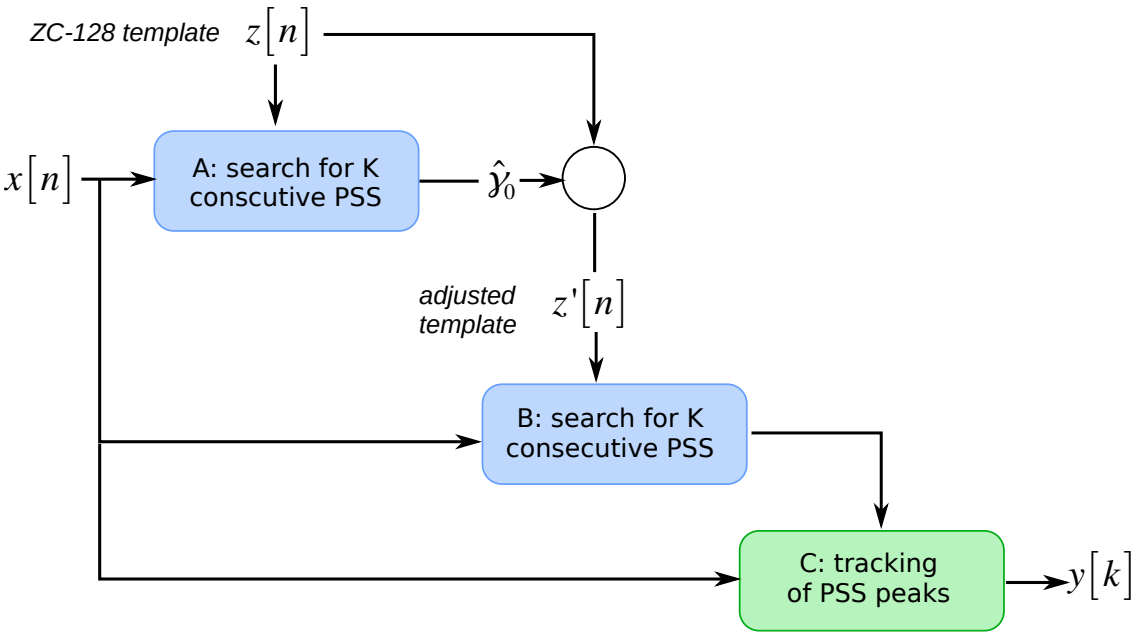


Figure 4.2: Overview of our PSS tracking algorithm.

specified we used $U = 40$. For more details on the principles of re-sampling and up-sampling refer to [66].

- To speed-up the computation process, in the tracking stage correlation and up-sampling are applied only to the portion of the incoming I/Q stream in the neighborhood of the expected PSS position as predicted from the previous frame, and specifically in a search window of $\pm N_{search}$ samples centered at $\hat{y}[k-1] + T_F$, with $N_{search} << T_F f_S$.

Hereafter we elaborate on the need to consider the frequency-adjusted template Eq. (4.6). Generally speaking, for a continuous-time signal $s(t)$, the ambiguity function $A(\tau, \nu)$ is given by the cross-correlation of $s(t)$ with a copy of the same signal delayed in time by τ (sec) and shifted in frequency by ν (Hz), formally:

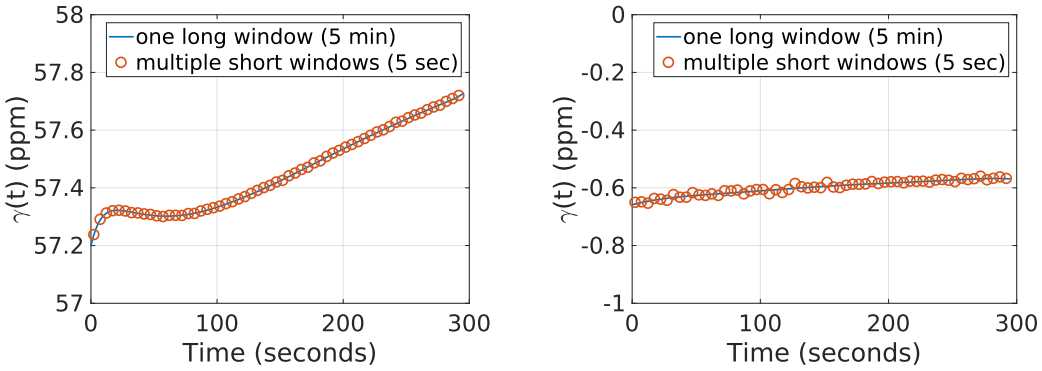
$$A(\tau, \nu) \stackrel{\text{def}}{=} \left| \int_{\infty}^{\infty} s(t) \cdot s^*(t-\tau) \exp^{+j2\pi\nu t} dt \right| \tag{4.7}$$

4.2.3. Estimation of instantaneous frequency deviation

The overall estimation process is split into two stages:

- Estimation of PSS arrival times $\hat{\boldsymbol{y}}$ from the stream of I/Q samples, as presented in the previous subsection.

- Estimation of the instantaneous frequency offset $\hat{\gamma}$ (or $\hat{\gamma}(t)$ for the dynamic case) from the vector of PSS arrival times $\hat{\boldsymbol{y}}$.



(a) non-TCXO “blue"

(b) TCXO “silver"

Figure 4.3: Short-term fluctuations: the red circles represent estimates obtained with short windows of 5 seconds and linear regression. The continuous line represents the result of higher-order polynomial regression on the total window of 5 minutes

The latter is detailed in the present subsection.

If the observation window W is sufficiently short, we can neglect higher-order variations of the instantaneous LO frequency (i.e., frequency drift) occurring within the observation window and consider the fixed-frequency (static) model in Eq. (4.5). In this case, from the vector of N measurements $\hat{\boldsymbol{y}}[k]$ we obtain an estimate $\hat{\gamma}$ simply by linear regression. The higher the precision of individual measurements (i.e., the lower σ_e), the faster a reliable estimate of $\hat{\gamma}$ can be achieved. In case of longer observation windows (larger W), we must consider the dynamic clock error model in Eq. (4.4). In this case, we apply higher-order polynomial regression in order to estimate the coefficients $\hat{\gamma}$ and $\hat{\alpha}_n$'s, and from the latter compute the $\hat{\beta}_n$'s. The collection of such parameters represents the full trajectory of the LO frequency within the (long) observation window.

To illustrate, in Figure 4.3 we present the estimated profile of $\hat{\gamma}(t)$ obtained with real devices during an observation window of 5 minutes. The continuous line was obtained by processing all the data from the whole long window of $W = 5$ minutes in a single batch, with regression to an high-order polynomial. The red circles represent the estimates obtained by splitting the dataset into short sub-windows of $W = 5$ seconds, with simple linear regression based on the static model from Eq. (4.5). As expected, the two approaches lead to very similar estimates. Unless differently specified, in the remainder of this work we will adopt the short-window approach with linear regression in order to reduce complexity and computational time.

4.3. Evaluation

In this section, we detail the testbed used and explain the comparison made by LTESS-track against three open source tools for frequency offset estimation.

4.3.1. Testbed

We deploy an outdoor testbed to perform the evaluation of the frequency offset for different RTL-SDR devices. We rely on the Raspberry Pi as the main board (Figure 4.4(a)) to execute the different tools in a Linux environment. We setup a set of Raspberry Pi in a small container with TCXO RTL-SDR devices (Figure 4.4(b)) and non-TCXO RTL-SDR devices (Figure 4.4(c)) attached to enable the comparison among the different frequency estimation methods. In addition to that, we also measure the ambient temperature and the temperature of the RTL-SDR case (with commercial temperature sensors) in order to study their relation with the frequency offset.

4.3.2. Evaluated tools

The following three existing tools have been considered for this study in addition to the newly proposed method:

- **rtl_test** [59]. This benchmark tool is part of the rtl-sdr software. The simple approach used for rtl_test is to count the samples read by the RTL-SDR device and compare it with the nominal sampling rate.

- **Kalibrate-RTL** [60]. This tool allows to scan and find Global System for Mobile communications (GSM) base stations in a frequency range and therefore use them to estimate the frequency offset of the rtl-sdr local oscillator.

(a) Raspberry Pi (b) TCXO “Silver" (c) Non-TCXO “Blue"

Figure 4.4: Hardware

- **LTE-Cell-Scanner** [61]. This tool performs a LTE base station search in a given frequency range. Once the base station is detected the tool reports the cell id and the frequency offset estimated using the PSS and SSS defined in LTE structure [63, Chapter 7] [64].

We evaluate each tool described above against our LTESS-track. As it is detailed below, we found some common limitations among those tools in terms of coarse time granularity, long processing time and, in some cases, gross estimation errors.

Comparison with rtl_test. The main limitation of the rtl_test tool is the coarse temporal resolution. This tool computes the frequency offset based on the difference between the actual number of I/Q samples collected in each time-bin of duration ω interval $[0, \omega]$ and the expected number thereof based on the nominal sampling frequency. This method is affected by errors in the determination of the reference interval ω in the absence of an accurate reference clock. rtl_test introduces a new source of error which is the frequency offset of the internal clock of the computer where the measurement is performed. In order to mitigate this error, two possible approaches can be taken: (1) choose a large value for W, and (2) average k subsequent measurements. The temporal resolution of the measurements is therefore $\tau \stackrel{\text{def}}{=} k \cdot \omega$. Figure 4.5 shows the estimated LO frequency offset values reported by rtl_test after τ seconds based on the average of k subsequent measurements in window of size W, for different values of the latter. The measurement obtained with our method is also plotted as reference. It can be seen that even with the most favorable setting ($W = 60$ sec), it takes more than 3 minutes for rtl_test to approach the LO offset value as determined by LTESS-track after 1 sec. Furthermore, with $W = 5$ sec the output values appear to be diverging. We conclude that rtl_test cannot be used to evaluate frequency offset variations at timescales smaller than a few

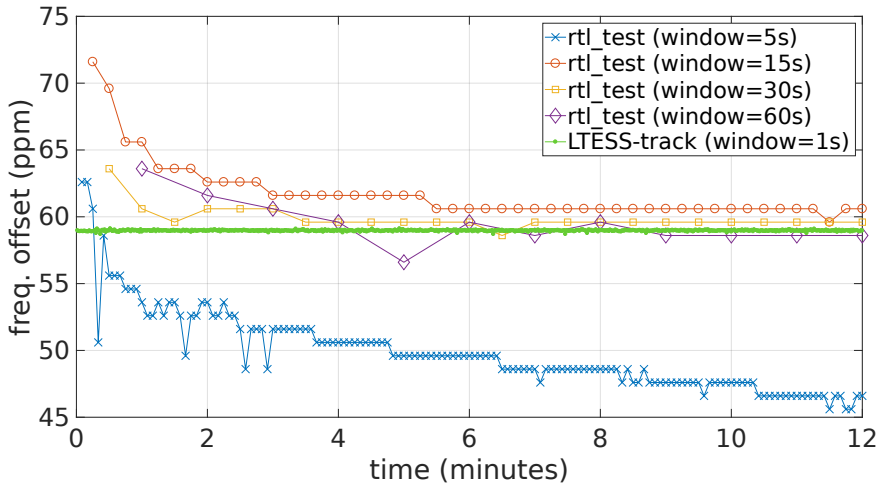


Figure 4.5: Comparison with rtl_test (non-TCXO).

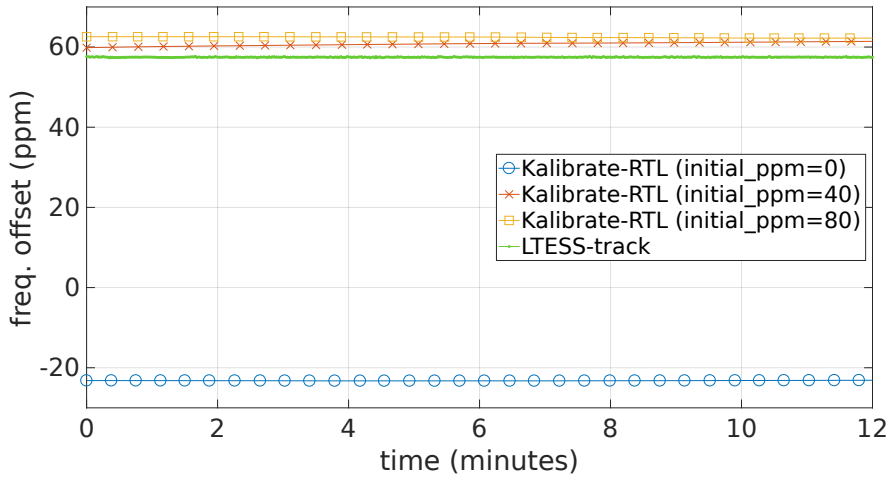


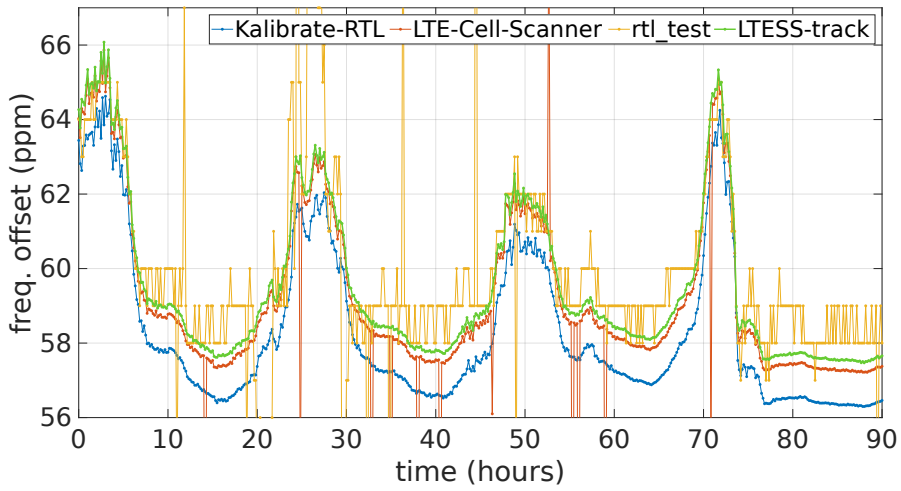
Figure 4.6: Comparison with Kalibrate-RTL (non-TCXO).

minutes.

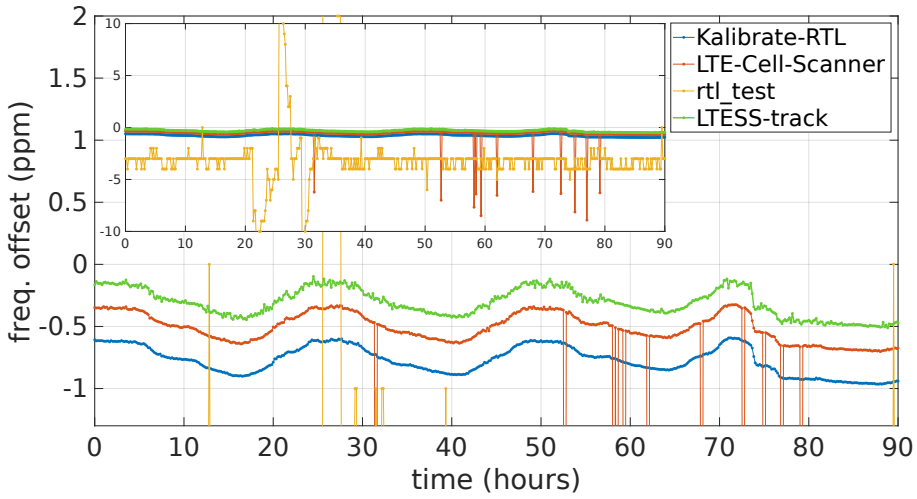
Comparison with Kalibrate-RTL. We observed that this tool delivers grossly erroneous results when used with devices affected by large frequency offsets. For example, for the "blue" non-TCXO dongle, it was reporting an estimated value of $\hat{\gamma} = -22$ ppm while all other tools where consistently reporting values around $\hat{\gamma} = +59$ ppm. The inaccuracy of this tool when applied on devices with LO offsets in excess of about 20 ppm is a known issue[1]. The problem can be mitigated by providing a good initial guess of the LO frequency offset as input to the tool. That means Kalibrate-RTL can be used only to refine the initial estimate obtained by other means. In Figure 4.6, we report the estimated values obtained with Kalibrate-RTL for different initial guess values given as input. It should be noted that even with proper initialization, the reported output value is sensitive to the exact input value.

Comparison with LTE-Cell-Scanner. Next, we tested LTE-Cell-Scanner. Similarly to our new tool, also LTE-cell-track relies on LTE signals as reference. This tool uses a fixed observation window of 160 ms and this is the value that we used in our tests. For most of the measurement timebins, the reported value were very close to the one estimated by our method—a clear indication of the precision of both tools. However, in less 1% of the timebins we observed occasional large errors (see Figure 4.7(a) and Figure 4.7(b)). Another limitation of this tool is the heavy computation: on a Raspberry Pi 3 it takes approximately 1 minute to process the data and report a frequency offset estimation. Due to such limitations, it is not possible with this tool to observe frequency offset fluctuations at small timescales.

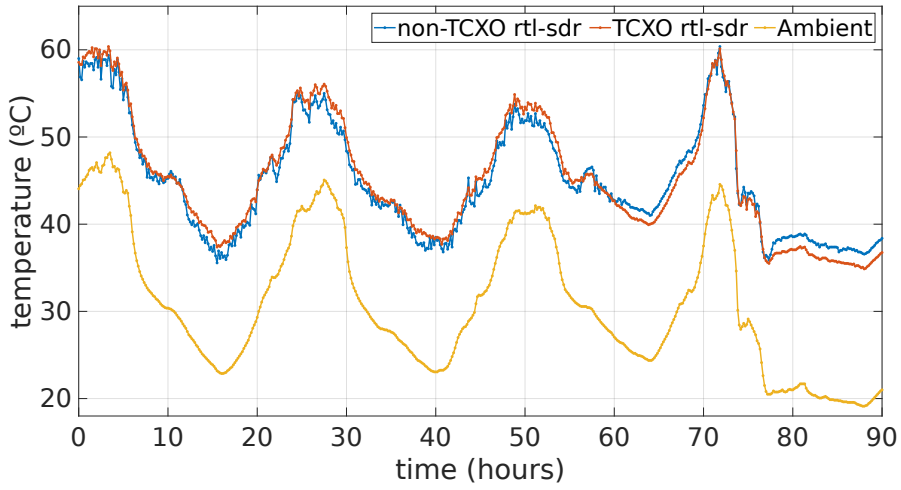
[1] https://github.com/steve-m/kalibrate-rtl/issues/8



(a) Non-TCXO “Blue": Frequency offset vs time

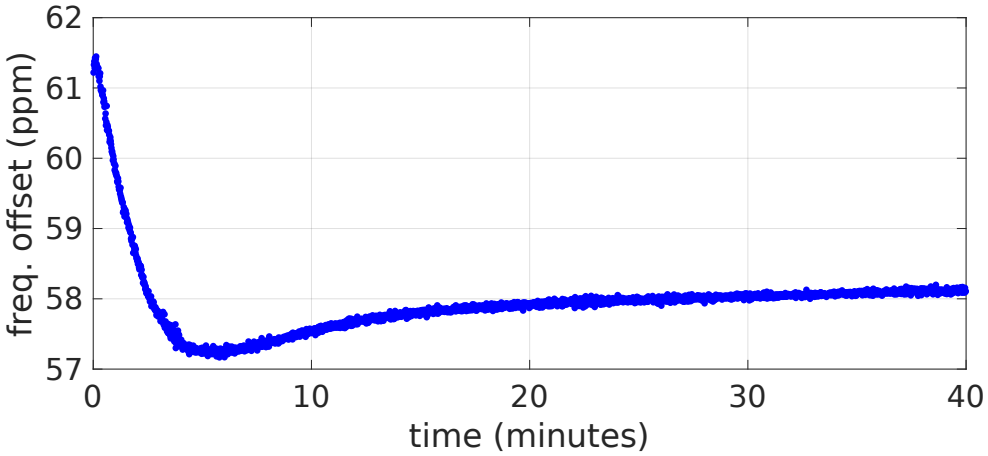


(b) TCXO “Silver": Frequency offset vs time

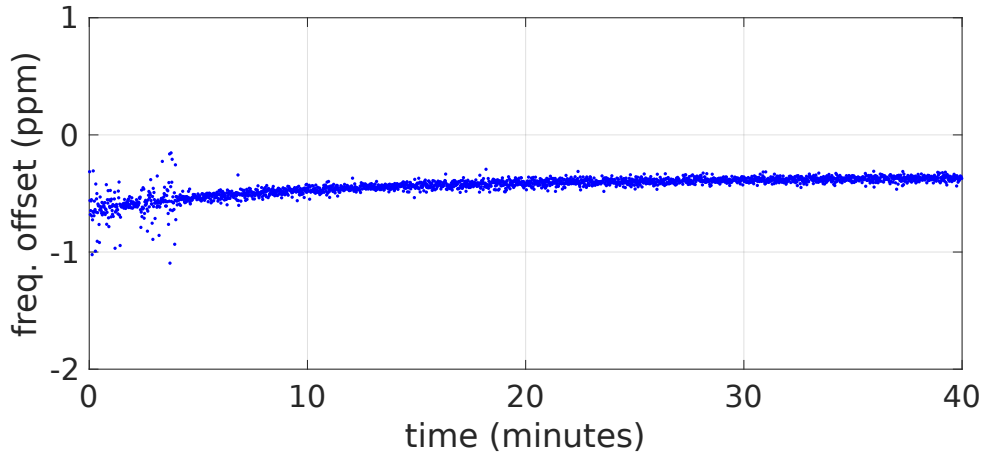


(c) Temperature vs time

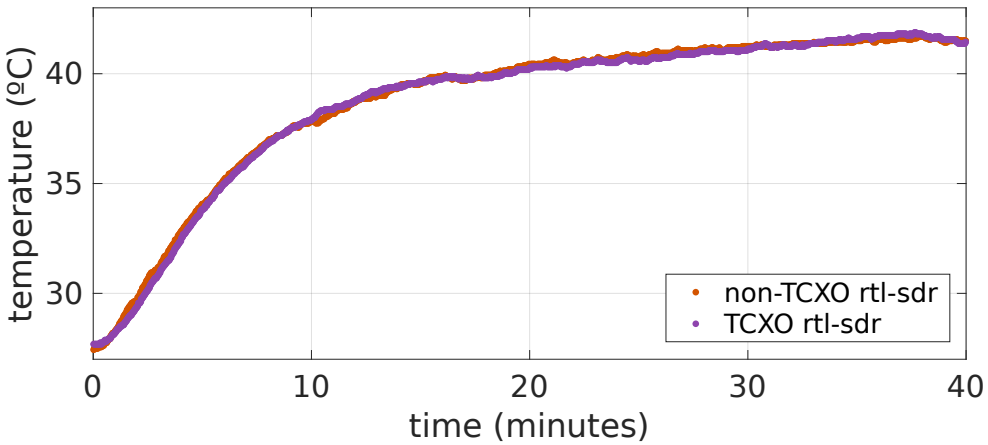
Figure 4.7: Long-term frequency offset analysis.



(a) Non-TCXO “Blue"



(b) TCXO “Silver"



(c) Sensors temperature

Figure 4.8: Short-term frequency offset variations for Non-TCXO (top) and TCXO (middle) RTL-SDR devices. Sensors temperature are shown on the bottom.

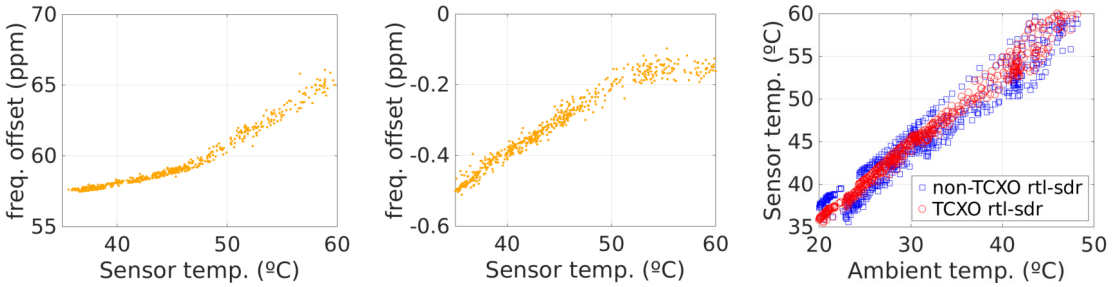
4.3.3. Short-term variations

In Figure 4.8(a) and Figure 4.8(b), we plot the evolution of the LO frequency offset estimated with LTESS-track. We run our method during the first 40 minutes of operation (starting from a cold state of the devices), respectively, for the blue and silver dongles. Figure 4.8(c) shows the device temperature. An initial transitory state is clearly in place, with steeper LO frequency excursion due to initial heating. After approximately 20 min the device temperature stabilizes (around 40°C) and so does the LO frequency. For the non-TCXO dongle, we observe a maximum excursion of 4 ppm within the first 5 minutes. For the TCXO device, we observe a smaller variation of 0.8 ppm during the first 5 minutes. After this initial warm-up, the LO offset excursion remains contained within 0.2 ppm, well within the declared specifications of 1 ppm [62].

4.3.4. Long-term variations

We have conducted a set of long-term measurements to evaluate LO offset variations of the RTL-SDR devices over a long period and across different temperatures, and at the same time to perform a long-term comparison of the output of different tools. The different tools were run on the same device in a round-robin fashion, with cycles of 10 minutes during a measurement period of 90 hours. In order to perform a fair comparison we have configured each tool to work in the most favorable conditions. More specifically: 1) rtl_test runs during $\tau = 4$ minutes and averaging the cumulative frequency offset estimation every $\omega = 1$ minute; 2) Kalibrate-RTL executes taking as input the initial offset estimation as computed by LTESS-track in the previous time-bin; 3) LTE-Cell-scanner executes with default configuration (recall that the computation time is about 1 minute in the Raspberry Pi). 4) LTESS-track is configured with an observation window of $W = 1$ second.

Figure 4.7(a) shows the frequency offset estimates reported in each cycle of 10 minutes by the different tools for the non-TCXO "Blue" device. The first observation is that the frequency offsets in these devices are strongly depending on the temperature: the higher the temperature, the higher the instantaneous LO frequency, as can be seen more clearly in Figure 4.7(c). LTESS-track and LTE-Cell-Scanner report very similar values except that the second one occasionally estimates the frequency offset with a large error. Kalibrate-RTL shows a similar trend of the frequency offset estimated but with a gap in the order of 1-2 ppm. Recall from Figure 4.6 that the output values of this tool are somewhat dependent on the initial guess provided in input. It seems that the precision of this tool is somewhat limited to 1-2 ppm. By last, rtl_test reports very inaccurate frequency offset values even using the most favorable settings, with observation intervals of 4 minutes. Notice that the resolution of the estimates provided by rtl_test is 1 ppm.



(a) Non-TCXO “Blue"

(b) TCXO “Silver"

(c) Sensor temp. vs Ambient temp.

Figure 4.9: Frequency offset and temperature analysis.

The long-term results for the TCXO "Silver" RTL-SDR device are shown in Figure 4.7(b). The rtl_test tool again reports grossly inaccurate offset values (peaks of ± 10 ppm and average around -4 ppm). that the minimum resolution is 1 ppm. However the other 3 tools (Kalibrate-RTL, LTE-Cell-Scanner and LTESS-track) report similar values < 1 ppm. Kalibrate-RTL shows a higher gap compared to our method (0.5 ppm), but as said above, the precision of this tool is anyway coarser than 1 ppm. LTE-Cell-Scanner shows again occasional large errors (maximum peaks of -9 ppm). Besides those, we observe a small and systematic gap of 0.2 ppm between the estimates delivered by LTE-Cell-Scanner and LTESS-track that could be caused by minor differences in the computation details between the two tools.

In Figure 4.9(b), we plot the frequency offset of the TCXO RTL-SDR device as reported by LTESS-track versus the device temperature. The plotted data points span the whole measurement period of 90 hours. We can conclude that the stability of the TCXO device is well within the specifications (< 1 ppm). Notice that up to 50°C there is an approximately linear relation between frequency offset and temperature while in the range of 50-60°C the frequency offset remains constant. On the other hand, non-TCXO RTL-SDR devices (Figure 4.9(a)) shows an absolute frequency offset of several tens of ppm (50-70) with daily fluctuations around ±5 ppm depending on the temperature. By last, Figure 4.9(c) shows the linear relation between the temperature in the ambient and the temperature on the RTL-SDR device during operation. Both TCXO and non-TCXO devices seem to be 15°C above the temperature of the environment.

4.3.5. Computation performance

We have evaluated the execution time of every tool by measuring the time required to compute one single LO offset estimate. The tests are performed in a Quad-Core i5 laptop. LTE-Cell-Scanner reads samples for 160 ms and then computes a single frequency offset measurement in 15 seconds, due to the heavy computations needed for the PSS and

SSS detection. However, LTESS-track is optimized for LO estimation and is able to provide a frequency offset measurement every second (reading samples for 0.5 seconds). LTESS-track is also 10 times faster than Kalibrate-RTL which performs each frequency offset measurement every 10 seconds. The evaluation of the rtl_test performance is not relevant since the performance depends on the the observation window, and the latter is 4 minutes long for this tool (besides the frequency offset estimated is not reliable).

4.4. Discussion

We have introduced a precise and fast frequency offset estimator for low-cost SDR devices connected to embedded boards. LTESS-track exploits the synchronization signals broadcasted by LTE base stations to determine the LO offset of the RTL-SDR devices. LTESS-track implements several key mechanisms not presented in other methods such as initial frequency offset compensation, up-sampling, sampling of data only in time proximity to the expected synchronization signal to reduce the computational cost and linear regression of samples. Our method is 10 times faster than the best open-source tools currently available, and is able to provide a new estimate every second. Therefore, our method allows to analyze and evaluate short-variations in time of the frequency offset. We have evaluated the two most common RTL-SDR devices in the market, the ones with TCXO integrated and the ones without. We have demonstrated that the frequency offset of the LO can be highly temperature dependent. Thanks to LTESS-track we can conclude that the maximum fluctuation in the TCXO "silver" device is around 0.2 ppm, while the non-TCXO "blue" device reports daily fluctuations of $\pm$ 5 ppm around an average value that can be in the order of 50-70 ppm. LTESS-track can also work with any SDR front-end capable of tuning to LTE frequencies. The advantages of our approach become significant in crowd-sourced scenarios where LO frequency offsets need to be estimated quickly and compensated for a massive number of RTL-SDR devices deployed over a wide area. In Chapter 7 we will show the direct benefits of estimating and correcting the frequency offset of the SDR receivers.

The MATLAB implementation of LTESS-track is released as open-source[2].

[2]https://github.com/electrosense/LTESS-track

Part III

CrowdSourcing Scenarios for IoT Spectrum Receivers

We are experiencing a democratization of the spectrum monitoring with the emergence of low-cost Software Defined Radio (SDR) boards. Spectrum measurements are now affordable with commonly available general-purpose low-cost hardware. A real example of this idea is ElectroSense, that has been introduced in the previous chapters.

In this third part we introduce and describe crowdsourcing and collaborative scenarios for Internet-of-Things (IoT) spectrum sensors in the context of the ElectroSense network. Thanks to the high scalability and reconfigurability of the ElectroSense system, collaborative approaches among sensors are feasible to provide higher capabilities than what it is possible with one single spectrum sensor.

In Chapter 5 we focus on new Time-of-Arrival (ToA) estimation methods that can run on low-cost SDR receivers which are used for widely deployed sensor networks for collecting air traffic control messages such as the OpenSky Network, FlightAware or FlightRadar24. We compare our proposed methods with the state-of-the-art using real scenarios and real air traffic signals. The results obtained show that our algorithm can deliver ToA estimations with nanosecond-level precision even using the cheapest SDR hardware that is currently available.

In Chapter 6 we focus on the collaborative approach among IoT spectrum sensors in order to enable collaborative narrow-band decoding capabilities considering the limited network uplink bandwidth of the sensor's connection. We describe a distributed system architecture for collaborative radio signal monitoring and decoding that builds on SDR receivers, propose a time multiplexing approach for sampling the spectrum and address the strict time synchronization required among sensors to efficiently optimize the network bandwidth usage and reconstruct the signal.

In Chapter 7, the last of this part, we present a methodology to enable the wideband signal reconstruction in the backend using non-coherent receivers. We propose a method for collaborative wideband signal decoding that exploits the idea of multiplexing in frequency a certain number of non-coherent receivers in order to cover a higher signal bandwidth that would not otherwise be possible using a single SDR receiver.

"The most effective way to do it, is to do it."
Amelia Earhart (1898 – 1937)

5 Collaborative Time-of-Arrival Estimation for Aircraft Signals

Aircraft and unmanned aerial vehicles continuously transmit wireless signals for air traffic control and collision avoidance purposes. These signals are either sent as responses to interrogations by Secondary Surveillance Radar (SSR) or automatically on a periodic basis (Automatic Dependent Surveillance - Broadcast (ADS-B)). Both types of signals are transmitted over the so-called *Mode S* data link [67] on the 1090 MHz radio frequency.

Over the last few years, sensor network projects have emerged which collect those signals using a crowd of low-cost Software Defined Radio (SDR) receivers such as e.g. the OpenSky Network [51], Flightaware [68], Flightradar24 [69] and many others. These sensor networks can leverage the Time-of-Arrival (ToA) of Mode S signals for various kinds of applications, including aircraft localization [51, 70], air traffic data verification [57, 71–74], and self-localization [75]. In those applications, a set of cooperating receivers measure *locally* the ToA of the arriving signals and then send these data to a central computation server. By joint processing the ToA of the same signal arriving at different receivers, the central server is able to estimate the location of the transmitter, the location of the receivers, or the exact time when the signal was transmitted.

The accuracy of these applications heavily depends on the precision of the ToA estimation, and in order to estimate positions up to a few meters it is necessary to estimate the ToA with nanosecond precision. The goal of this work is to provide a method for the ToA estimation of Mode S signals that delivers nanosecond-level precision even with low-cost SDR receivers, such as the widespread RTL-SDR dongle [62]. We show that existing ToA estimation approaches based on a cross-correlation with a reconstructed signal template are sub-optimal in the particular context of Mode S signals. In fact, the loose tolerance margins allowed by the specifications on the shape and position of each individual symbol within the packet (up to $\pm$ 50 ns) adds uncertainty to the reconstruction of the whole packet waveform at the receiver.

We propose two alternative methods that improve the precision and at the same time reduce the computational load. We test different variants of ToA estimation on real-world signal traces captured with RTL-SDR, which is currently the cheapest SDR device on the

market and widely used by crowdsourced sensor networks. Our results show that the best proposed method delivers ToA estimates with a standard deviation error of 1.5 ns. We further identify the limited dynamic range of the RTL-SDR device (less than 50 dB with 8-bit Analog-to-Digital Converter (ADC) and fixed Automatic Gain Controller (AGC)) as the main performance bottleneck, and show that sub-nanosecond precision is achievable for signals that are not clipped due the limited dynamic range of the device.

5.1. Background

This section provides background on aircraft signals which we rely on to estimate the ToA, and the limitations of classical ToA estimation methods.

5.1.1. Mode S signal format

Hereafter we briefly review the physical-layer format of SSR Mode S [76] reply and ADS-B messages transmitted by aircraft on the 1090 MHz channel. Mode S transmissions contain some information like higher resolution altitude, aircraft capabilities and identification, while ADS-B transmissions (which are considered Mode S extended messages) contain aircraft Global Positioning System (GPS) position and velocity. Both packet formats consist of a preamble of 8 μs plus a payload of 112 or 56 bits (only for other SSR Mode S replies) sent at 1 Mbps rate, for a total duration of 120 μs or 64 μs, respectively. The information bits are modulated with a simple Binary Pulse Position Modulation (BPPM) scheme as illustrated in Figure 5.1: the symbol period of 1 μs is divided into two “chips" of 0.5 μs, and the high-to-low and low-to-high transitions encode bits “1" and “0", respectively. It is clear from Figure 5.1 that the BPPM modulation produces two types of pulses of different duration, denoted hereafter as “Type-I" and “Type-II". Type-I pulses have a nominal duration of one chip period and are produced by the bit sequences “00", “11" and “10". The preamble consists of four Type-I pulses. On the other hand, Type-II pulses have a nominal duration of two chip periods and are produced exclusively by the “01" sequence. On average, we expect approximately $112/2 = 56$ Type-I and $112/4 = 28$ Type-II pulses for a payload of 112 bits.

The real-valued baseband signal is then modulated on the 1090 MHz carrier frequency and transmitted over the air. On the receiver side, the decoding process relies exclusively on the signal *amplitude*, since in BPPM the signal *phase* carries no information.

5.1.2. Limitation of standard Time-of-Arrival methods

The standard “course book" approach to ToA estimation in the Additive White Gaussian Noise (AWGN) channel is a correlation filter [77]: the received signal is cross-correlated with a known template corresponding to the source signal, and the point in time maximizing the cross-correlation module is taken as ToA estimate.

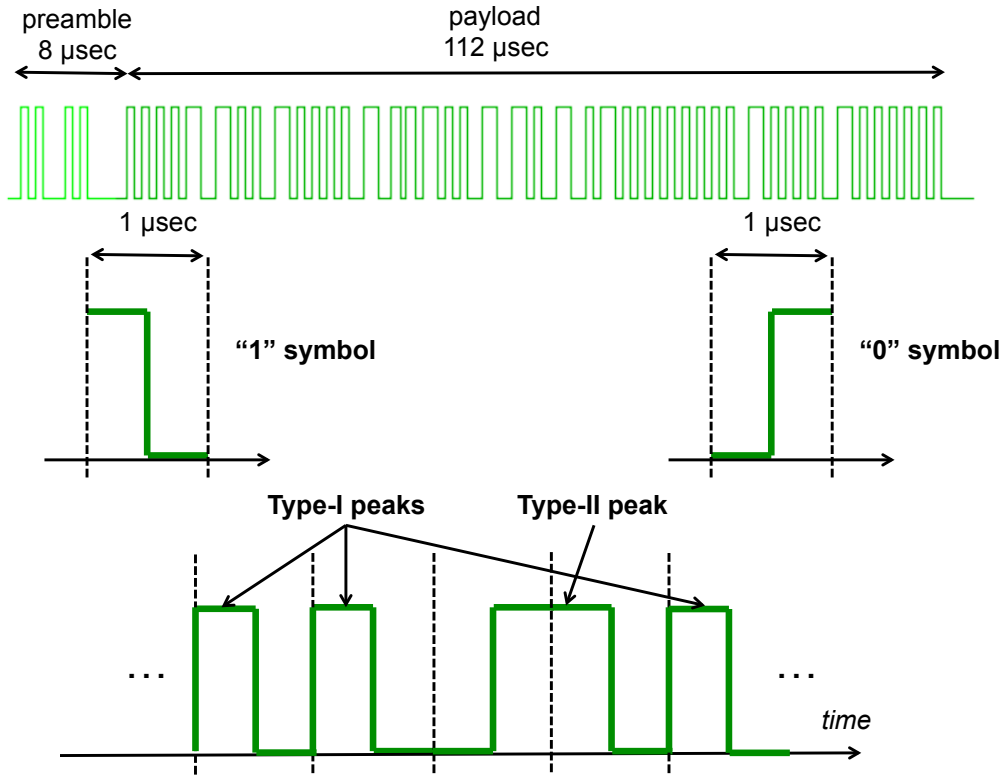


Figure 5.1: Mode S packet structure with a BPPM.

The correlation method relies on the assumption that *the source signal can be reconstructed very precisely* at the receiver, based on the signal specifications and knowledge of the payload bits $\boldsymbol{p}_m$. Under this assumption, the correlation method represents the Maximum Likelihood Estimator (MLE) [77]. However, this assumption is problematic in the particular case of *real-world* Mode S signals.

In fact, the standard specifications tolerate up to ±50 ns jitter in the *position* of each individual pulse within the packet: such high tolerance value is practically negligible for the decoding process, but not for the task of determining the ToA with nanosecond precision. As to the *shape* of each pulse, tolerance values of 50 ns are allowed for the pulse *duration* and *rise time* and up to 150 ns for the *decay time*, while pulse amplitude may vary up to 2 dB (approximately 60%). Such loose tolerance margins introduce uncertainty in the prediction of the shape and position of the pulses in the source signal. Considering that Mode S signals are typically received with high Signal-to-Noise Ratio (SNR), such an uncertainty might well prevail over the effect of additive noise. Consequently, the correlation-based approach with a known packet template is no longer guaranteed to be optimal, motivating the quest for alternative, more precise methods.

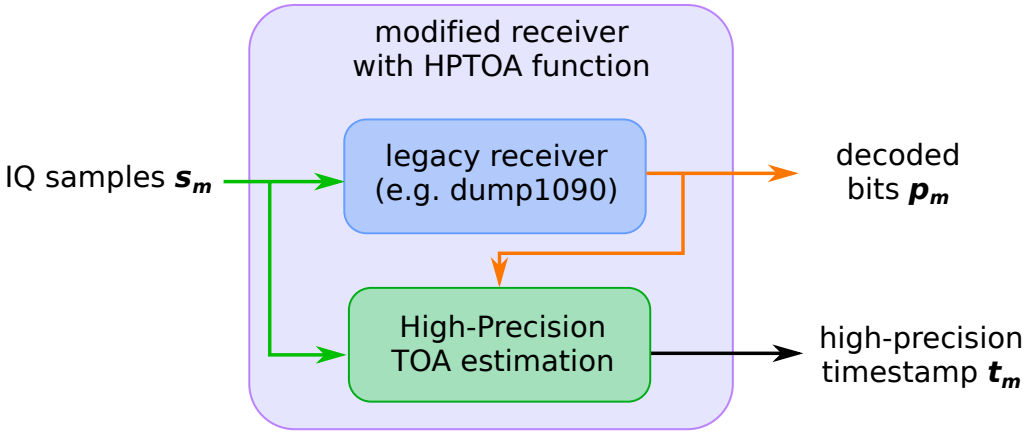


Figure 5.2: Block diagram of improved receiver with high-precision Time-of-Arrival estimation.

5.2. Our Time-of-Arrival estimation methods

In this section we describe the general approach to ToA estimation based on the decoded payload and received signal samples, and then present the different ToA estimation algorithms that were tested.

5.2.1. Signal acquisition architecture

In the software domain, the high-precision ToA estimation process can be seen as an additional function that is optionally called within the receiver and remains independent from the main decoding process. As such, it can be implemented on top of any legacy receiver, including but not limited to the widely adopted open-source tool `dump1090` [78]. The overall block diagram of the proposed scheme is exemplified in Figure 5.2. The legacy receiver takes as input a stream of complex in-phase and quadrature (In-phase & Quadrature (I/Q)) samples collected at sampling rate f_s (for RTL-SDR hardware $f_s = 2.4$ MHz). The legacy receiver seeks to detect and decode the incoming packet and, if successful, provides as output the decoded bit sequence $\boldsymbol{p}_m$ along with the indication of the leading I/Q sample of the detected packet.

Denote by $\boldsymbol{s}_m$ the sequence of complex I/Q samples corresponding to the whole packet. The sequence includes approximately 300 samples since we also pick a few samples immediately before and after the packet in order to mitigate edge effects. The sample vector $\boldsymbol{s}_m$ and the decoded bit vector $\boldsymbol{p}_m$ represent the input to our ToA estimation block.

5.2.2. Proposed methods: *CorrPulse* and *PeakPulse*

Hereafter we describe two novel ToA estimation algorithms specifically developed for Mode S signals. For a generic packet m we shall denote by K_m the total number of pulses

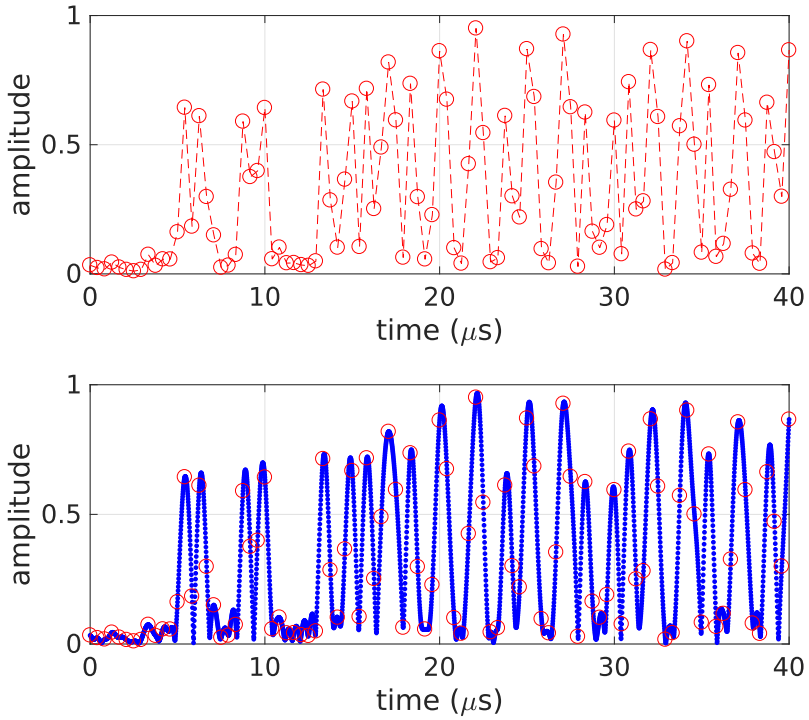


Figure 5.3: Example of received signal amplitude corresponding to the preamble and initial payload of a real ADS-B packet. Original samples at $f_s = 2.4$ MHz (top, red circles) and corresponding upsampled version (bottom, blue line).

in the whole packet (preamble and payload). The input vector of complex samples $\boldsymbol{s}_m$ is preliminary upsampled by a factor N and transformed into vector $\boldsymbol{s}'_m$ (for a review of upsampling process see e.g. [66]).

To illustrate, Figure 5.3 plots an excerpt of the amplitude of both vectors, namely $|\boldsymbol{s}_m|$ (top plot) and $|\boldsymbol{s}'_m|$ (bottom plot), for a generic packet found in a real-world trace.

The key aspect of the proposed algorithms is that the actual temporal position $\hat{\tau}_k$ of the generic kth pulse within the packet is estimated *independently* from other pulses, with no need to reconstruct a template for the whole packet. For each pulse $k \geq 2$, we compute the individual shift $\Delta\tau_k \stackrel{\text{def}}{=} \hat{\tau}_k - \tau_k$, i.e., the difference between the estimated and nominal pulse position relative to the (estimated) position of the first pulse. Finally, the pulse shifts are averaged in order to obtain the final ToA estimate:

$$\hat{t} = \hat{\tau}_1 + \frac{1}{K_m - 1} \sum_{k=2}^{K_m} \Delta\tau_k \tag{5.1}$$

The two proposed variants differ in the way individual pulse position estimates are obtained, and which type(s) of pulses are considered. In the first variant, labeled *CorrPulse*, each pulse position is determined through pulse-level cross-correlation of the

upsampled vector $\boldsymbol{s}'_m$ with the corresponding nominal pulse shape. Both Type-I and Type-II pulses are considered in the final averaging.

In the second variant, labeled *PeakPulse*, individual pulse positions are determined by simply picking the local maximum point value within the pulse interval, with no cross-correlation operation. In this variant only Type-I pulses are considered, while Type-II pulses are ignored. This is motivated by the fact that Type-II pulses have lower curvature, hence their local peaks cannot be identified as reliably as for Type-I pulses.

5.3. Evaluation Methodology

This section describes how we evaluate our new methods. First, we introduce the other competing methods taken as reference for the comparison. Then, we present the testbed setup with commercial low-cost hardware. Finally, we provide details on the procedure adopted to empirically assess the precision of the ToA measurement methods in the given setup.

5.3.1. Other methods for comparison

5.3.1.1. Correlation with whole-packet template: *CorrPacket*

This is the canonical cross-correlation method with a known signal template. For every packet m, the whole packet template is reconstructed from the decoded bits $\boldsymbol{p}_m$ and then cross-correlated with the amplitude of the incoming signal. Here also, upsampling by a factor N is adopted to achieve sub-sample precision. Within the template, the kth pulse is positioned at the nominal time τ_k. As to the pulse shapes, we have tested two different variants: “Rectangular" (R), and “Smoothed" (S). The two versions will be denoted by *CorrPacket/R* and *CorrPacket/S*. The rectangular pulses have a nominal duration of 0.5 μs and 1 μs for Type-I and Type-II pulses, respectively, and zero rise/decay times. The rectangular pulse mask is represented exclusively by “0" and “1" values, hence multiplications with another vector reduce to element selection, which saves on computation load. The “Smoothed" shape corresponds to the output of a low-pass filter with passband of 2.4 MHz—matched to the bandwidth of the RTL-SDR receiver—when the input signal is a nominal Type-I/Type-II pulse with the minimum decay/rise time of 50 ns as per specifications [79].

5.3.1.2. Existing dump1090 based implementations

We also evaluate the precision of the timestamp reported by the mutability fork of the open-source tool dump1090 [78]. Furthermore, we test on our traces also the method adopted by Eichelberger et al. in a ACM SenSys'17 paper [75] which is also based on dump1090. Code inspection revealed that this method is based on a cross-correlation

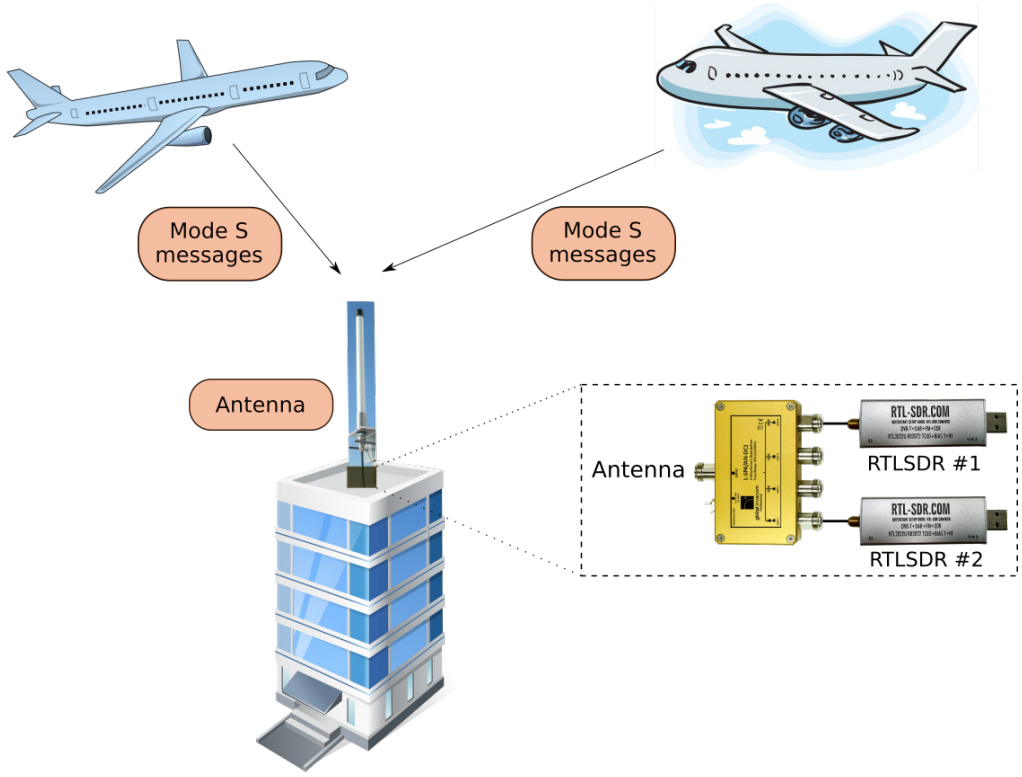


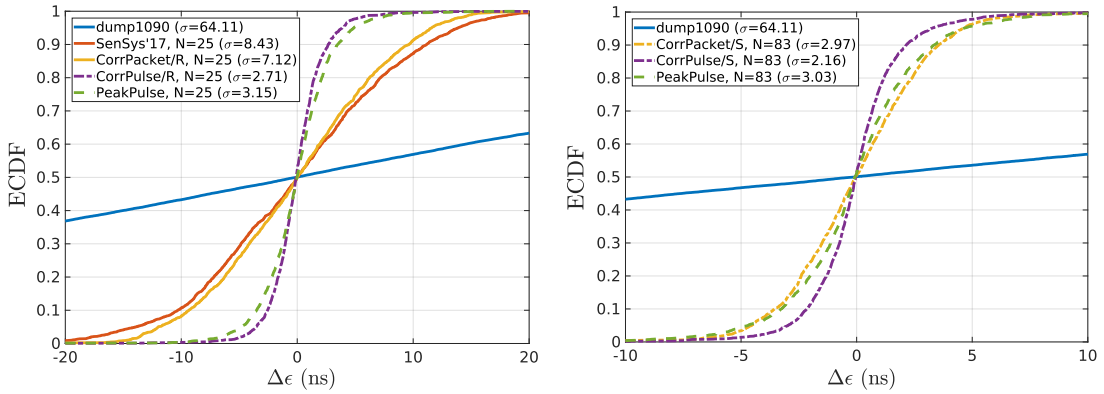
Figure 5.4: Experimental setup. Two identical receivers connected to the same antenna via a splitter are collecting Mode S messages sent by aircraft.

(implemented in frequency domain) with a partial packet template consisting of the preamble plus one quarter of the payload, with rectangular pulses and upsampling factor $N = 25$.

5.3.2. Testbed setup

The experimental setup consists of two identical sensors connected to a single antenna through a power splitter and cables of identical length. The sensors are located on the roof of a building as Figure 5.4 shows.

Every sensor consists of one RTL-SDRv3 “Silver" model [62] attached to a Raspberry Pi-3 [31]. The AGC gain is set to a fixed value, manually tuned to maximize the packet decoding rate. The sampling rate was set to $f_s = 2.4$ MHz, the maximum value that our setup is able to acquire with sample losses. Every I and Q sample is represented with 8 bit. The full stream of I/Q samples are recorded one and processed multiple times offline. Our results are based on a sample trace of 5 minutes collected in Thun (Switzerland) on 02-Aug-2017 at time 09:41. The number of ADS-B packets that are correctly decoded *at both sensors* by the `dump1090` open-source tool [78] amount to 26445 from 59 different aircraft.



(a) Low upsampling factor

(b) High upsampling factor

Figure 5.5: ECDF of $\Delta\epsilon$ residuals.

5.3.3. Evaluation Metrics

In this section, we briefly describe the methodology adopted to assess the precision of the different ToA estimation methods. The problem is not trivial, since our receivers are not synchronized and the "true" ToA is unknown. Therefore, we developed an evaluation method which allows us to quantify the ToA precision without a ground truth.

Denote by $t_{m,i}$ the *true* absolute arrival time of packet m to receiver i and by $\hat{t}_{m,i}$ the corresponding *measured* ToA (by the method under test). In general, the measured value $\hat{t}_{m,i}$ is affected by two distinct sources of error, namely clock error and measurement noise:

$$\hat{t}_{m,i} = t_{m,i} + \xi_i(t)|_{t=t_{m,i}} + \epsilon_{m,i}. \tag{5.2}$$

The term $\xi_i(t)$ models the *clock error* between the receiver clock and the absolute time reference, and can be modeled by a *slowly-varying* function of time. Its magnitude depends on the *hardware* characteristics of the device, and specifically on the stability of the local oscillator.

The term $\epsilon_{m,i}$ represents the measurement noise in the ToA estimation process and is modeled by a *random variable* with zero mean and variance σ^2_{TOA}. The *precision* of the TOA estimate, defined as the reciprocal of the noise variance, is *independent* of the clock error. The goal of the present study is to reduce σ^2_{TOA}.

The problem of counteracting the clock error component remains outside the scope of the present contribution. Here it suffices to mention that the clock error can be mitigated by adopting receivers with GPS Disciplined Oscillator (GPSDO), or it can be estimated and compensated in post-processing [29, 80, 81].

Hereafter we illustrate the methodology to experimentally quantify the empirical ToA standard deviation $\hat{\sigma}_{\text{TOA}}$ notwithstanding the presence of a non-zero clock error component. First, we need to get rid of the unknown *true* absolute arrival time $t_{m,i}$ in

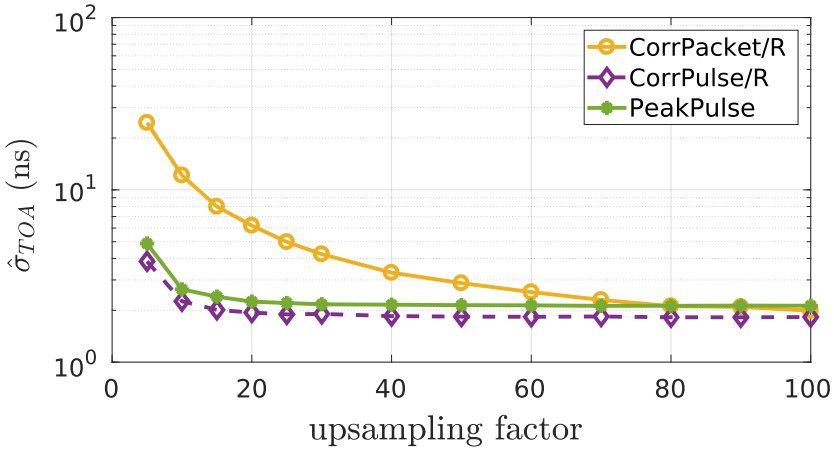


Figure 5.6: TOA standard dev. error vs. upsampling factor N

Equation (5.2). Since we use two identical receivers attached to the same antenna, we can set $t_{m,1} = t_{m,2} = t_m$ and subtract the ToA measurements at the two sensors to obtain the corresponding time difference:

$$\hat{\Delta}t_m \stackrel{\text{def}}{=} \hat{t}_{m,2} - \hat{t}_{m,1} = \Delta\xi(t_m) + \Delta\epsilon_m \tag{5.3}$$

wherein $\Delta\xi(t) \stackrel{\text{def}}{=} \xi_2(t) - \xi_1(t)$ denotes the compound clock error, and $\Delta\epsilon_m \stackrel{\text{def}}{=} \epsilon_{m,2} - \epsilon_{m,1}$ the compound measurement error with variance $\sigma^2_{\Delta\epsilon} = 2\sigma^2_{\text{TOA}}$.

At short time-scales, within the coherence time of the process $\Delta\xi(t)$, the clock error represents a systematic error, i.e. a *bias* term that can be estimated and removed in order to estimate the error *variance* $\sigma^2_{\Delta\epsilon}$. We do so by modeling the slowly-varying function $\Delta\xi_i(t)$ by a polynomial whose coefficients are estimated by standard order-recursive Least Squares (LS) (refer to [77, Chapter 8] for details). After removing the estimated clock error component, we obtain a set of residuals $\{\Delta\epsilon\}$. Their Mean Square Error (MSE) represents an empirical estimate of twice the ToA variance $MSE_{\Delta\epsilon} = 2 \cdot \sigma^2_{\text{TOA}}$. Accordingly, their Root Mean Square Error (RMSE) provides a direct empirical estimate of the ToA error standard deviation, formally:

$$\hat{\sigma}_{\text{TOA}} = \frac{1}{\sqrt{2}} RMSE_{\Delta\epsilon} \approx 0.7 \cdot RMSE_{\Delta\epsilon}.$$

5.4. Numerical Results

We now present the results on the precision of the different ToA estimation methods as evaluated in our testbed.

Table 5.1: Empirical estimates of TOA error standard deviation $\hat{\sigma}_{\mathrm{TOA}}$.

Estimation method	$\hat{\sigma}_{\mathbf{TOA}}$ **[nanoseconds]**			
	all packets	L	M	H
legacy dump1090	45.20	44.94	45.19	45.43
SenSys'17, $N = 25$	5.90	6.11	5.88	5.78
CorrPacket/R, $N = 25$	4.98	5.48	4.85	4.94
CorrPacket/R, $N = 83$	2.14	3.04	1.78	2.35
CorrPacket/S, $N = 83$	2.07	3.00	1.68	2.275
CorrPulse/R, $N = 25$	1.89	2.75	1.56	1.86
CorrPulse/R, $N = 83$	1.63	2.72	1.04	1.77
CorrPulse/S, $N = 83$	1.51	2.60	**0.79**	1.77
PeakPulse, $N = 25$	2.20	3.36	1.70	2.23
PeakPulse, $N = 83$	2.12	3.44	1.62	2.17

5.4.1. Error distribution

In Figure 5.5 we plot the Empirical Cumulative Distribution Function (ECDF) of the residuals $\Delta\epsilon$'s obtained with different ToA estimation methods for all the packets in the test trace. The corresponding values of the ToA error standard deviation $\hat{\sigma}_{\mathrm{TOA}}$ are reported in the leftmost column of Table 5.1.

For those applications where the computation load is of concern, it is relevant to investigate the performance of the different methods with moderate value of the upsampling factor ($N = 25$). For *CorrPacket* and *CorrPulse*, we consider the rectangular pulse shape with binary 0/1 values, due to lower computation load. Referring to Figure 5.5(a), we observe that the proposed *PeakPulse* algorithm achieves a $RMSE_{\Delta\epsilon} = 3.15$ ns, less than half the value of the canonical *CorrPacket/R* method. It is remarkable that such good result was obtained with no cross-correlation operation. Figure 5.6 shows $\hat{\sigma}_{\mathrm{TOA}}$ for different values of the upsampling factor N. We observe that the precision of the proposed methods *PeakPulse* and *CorrPulse/R* improves faster than *CorrPacket/R* with increasing N. These results indicate that *PeakPulse* should be preferred when computation load is at premium.

Next we consider applications that enjoy abundant computation power, for which the main goal is to maximize precision and computation load is not of concern. For these, it is convenient to consider higher upsampling factors ($N = 83$ in our case) and, for the cross-correlation methods, the more elaborated “Smoothed" pulse shape. The latter

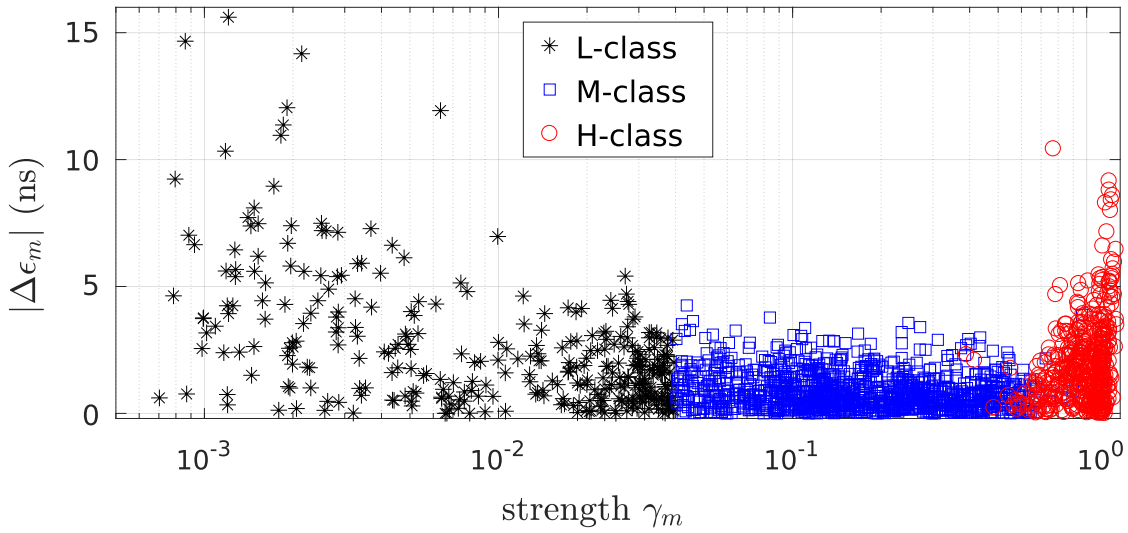


Figure 5.7: Absolute error $|\Delta\epsilon_m|$ vs. packet strength γ_m.

matches more closely the pulse shape passed through the RTL-SDR front-end, leading to slightly higher precision than the simpler “Rectangular" shape, as can be verified from Table 5.1. The ECDF of the residuals $\Delta\epsilon$'s for these methods are plotted in Figure 5.5(b). It can be seen that the proposed *CorrPulse/S* method is more precise than the classical *CorrPacket/S* method, and achieves $RMSE_{\Delta\epsilon} = 2.16$ ns corresponding to $\hat{\sigma}_{\text{TOA}} = 1.51$ ns.

5.4.2. Error vs. signal strength

In the following, we investigate the impact of signal strength on the ToA error obtained with the most precise method, namely *CorrPulse/S with* $N = 83$. For a generic packet m and sensor i, we denote by $\gamma_{m,i}$ the average of the squared pulse height over all pulses — an indicator of the arriving packet strength. Furthermore, we denote by $\beta_{m,i}$ the number of pulses that are clipped in the receiver due to one or more of the corresponding I/Q samples saturating the ADC. In Figure 5.7, we plot for each individual packet m the absolute value of the residual error $|\Delta\epsilon_m|$ obtained with *CorrPulse/S* ($N = 83$) against the mean signal strength between the two sensors $\gamma_m \stackrel{\text{def}}{=} \frac{\gamma_{m,1}+\gamma_{m,2}}{2}$. Each packet is classified into one of three classes: packets with $\gamma_m \leq 0.04$ are labeled by “L", packets with $\min_{i=1,2} \beta_{m,i} \geq 10$ are labeled with “H", and all remaining packets are labeled with “M". The three classes are marked respectively with black, red and blue markers in Figure 5.7. The estimated ToA error standard deviation obtained by each method for each class are reported in Table 5.1. On one extreme, timing estimates for “L" packets with lower strength are impaired by quantization noise. On the other extreme, packets received with high strength are subject to ADC clipping, a form of distortion that clearly degrades timing precision. As expected, these two classes yield higher error with all methods.

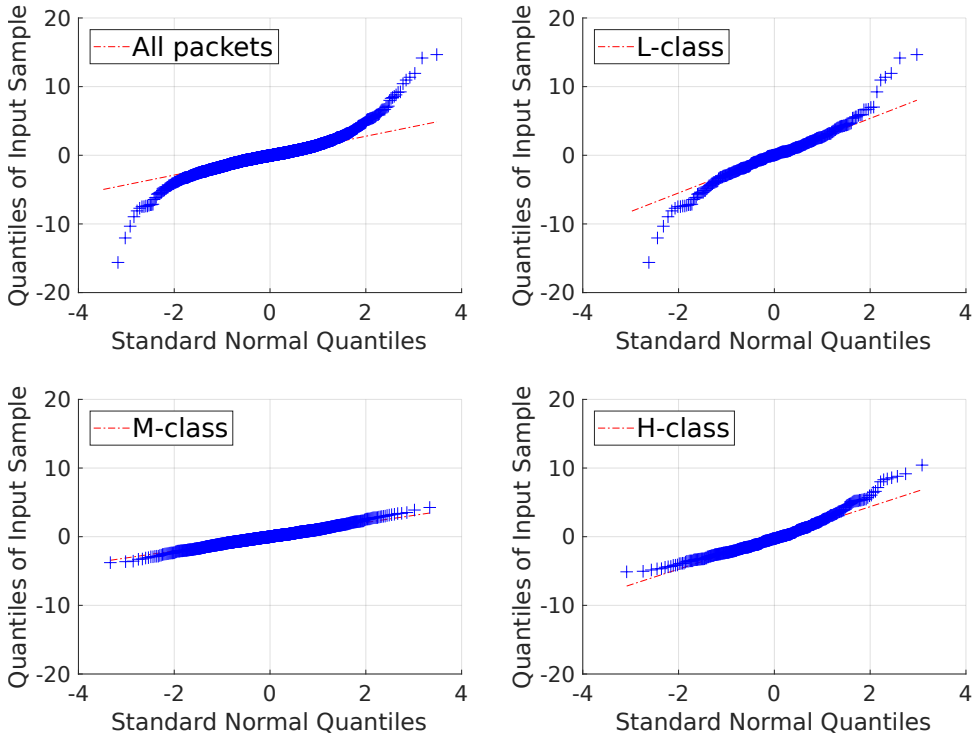


Figure 5.8: Quantile-quantile plot of empirical errors $\Delta\epsilon$ vs. normal distribution.

Between the two extremes, the strength of “M" packets fits well the dynamic range: for these, the proposed method achieves $\hat{\sigma}_{\mathrm{TOA}} = 0.79$ ns.

In our traces, less than 60% of all packets fall into class “M". With better hardware, and specifically with more ADC bits and larger dynamic range, it would be possible to tune the AGC gain so as to increase the fraction of packets falling in this class, thus improving the overall precision.

The above results indicate that the received packet metrics $\gamma_{m,1}$ and $\beta_{m,i}$ can be used to provide, *for each individual ToA measurement* $\hat{t}_{m,i}$, also an indication of the expected precision, i.e., of the error variance $\hat{\sigma}^2_{m,i}$ affecting each individual measurement. In this way, algorithms that take ToA measurements as input (e.g., for position estimation) have the possibility to *weight* optimally each individual input measurement, as done e.g. in Weighted Least Squares [82] methods.

Finally, we find that *within each class* the empirical error distribution is very well approximated by the Gaussian distribution, as seen from the normal Q-Q plots in Figure 5.8. This justifies the adoption of LS methods for position estimation problems based on input ToA measurements [29], since for normally distributed input errors the LS solution coincides with the MLE.

5.5. Discussion

We have presented two variants of a novel ToA estimation method for Mode S signals that does not rely on long cross-correlations with the template of a full packet. The most precise variant, namely *CorrPulse/S*, involves only short cross-correlation operations on individual pulses. The other variant, namely *PeakPulse*, is lighter to compute, involves no cross-correlation operation and works well also with moderate upsampling factors. We have shown that such algorithms can achieve TOA estimates with nanosecond-level precision even with real-world signals captured with the cheapest SDR hardware that is currently available, namely RTL-SDR. A closer look at the test results reveals that the main limiting factor for the achievable ToA precision with RTL-SDR is the limited dynamic range, resulting in a large fraction of packets being clipped or drowned into quantization noise. For packets that are received with signal strength well within the dynamic range of the receiver, the *CorrPulse/S* achieves sub-nanosecond precision. It can be expected that precision can be further improved with better hardware. The *PeakPulse* method has been implemented in C, integrated in the dump1090 receiver and is released as open-source[1].

[1] http://github.com/openskynetwork/dump1090-hptoa

"It is the long history of humankind that those who learned to collaborate and improvise most effectively have prevailed"

Charles Darwin (1809 – 1882)

6 Collaborative Narrowband Spectrum Data Decoding

We are experiencing a democratization of the spectrum monitoring with the emergence of low-cost software-defined radios such as RTL-Software Defined Radio (SDR) [30, 34] and GnuRadio devices [83]. Spectrum measurements are now affordable with commonly available general-purpose low-cost hardware. This has led researchers to the idea of building a networked and distributed infrastructure connected over the public Internet [16, 17, 84], crowdsourcing the spectrum data collection to users with Internet connection. While these systems represent a huge improvement over how the spectrum is monitored today, they are all limited to applications in which simple power spectrum measurements are sufficient. In this work we aim to make a step ahead, crowdsourcing the collection of raw digitalized In-phase & Quadrature (I/Q) radio samples in a frequency band over time and decode the information in the backend. Decoding of radio spectrum data in the backend poses various challenges that do not exist in classical spectrum monitoring solutions:

Challenge 1: Signal acquisition with low-cost IoT spectrum sensors. Low-cost spectrum sensors are much more constrained in terms of sampling rate, frequency bandwidth, sweep time, dynamic range and sensitivity than their high-end counterpart, limiting the effectiveness of a spectrum monitoring system. At the extreme end, we envision to use commodity hardware such as USB dongles for radio signal acquisition that cost not more than $12 using software available as open-source [30, 34]. The spectrum sensor used in this work is shown in Figure 6.1.

Challenge 2: The network bandwidth problem. The second challenge is that the system requires the participation of users that may deploy the Internet-of-Things (IoT) spectrum sensors at a location (such as their homes) with a limited network bandwidth. This bandwidth is not a concern for most crowdsourcing initiatives which collect averages of the power spectrum. Instead the collection of raw I/Q samples is necessary to decode the original information in the backend. This imposes a large volume of data which is several orders of magnitude higher than the power spectrum as collected in typical sensing

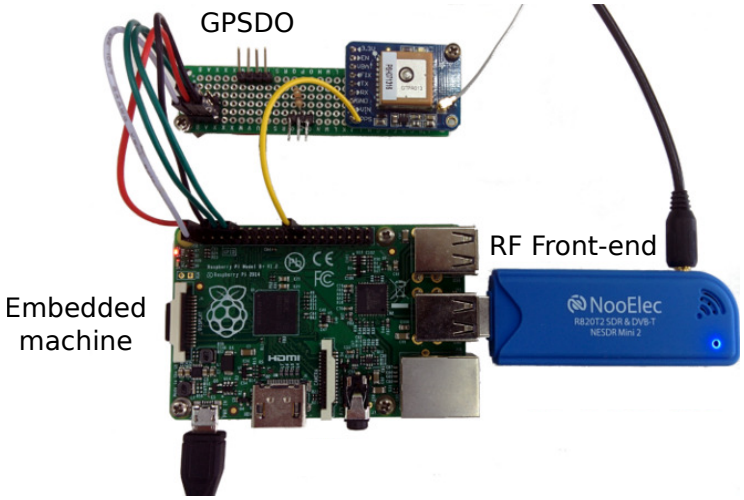


Figure 6.1: IoT RF sensor for Challenge 1.

contexts. The core idea we investigate in this work to address Challenge 2 is to *exploit the similarity of the spectrum* in the time domain in order to identify spectrum sensors that are in the coverage range of the same transmitters (as shown in Figure 6.2). The principle is to instruct from the backend the spectrum sensors to listen to the same frequency band of interest in separate time slots, thus alleviating the amount of I/Q samples to be sent from each sensor (and user). However, there is a fundamental problem in this approach that brings to the challenge discussed next.

Challenge 3: The need for a fine time synchronization. It is not trivial to apply a collaborative approach to the time domain analysis. In fact, collaboration for decoding a signal transmitted over the air has much stricter timing requirements than signal energy detection and occupancy map studies of independent sensor entities, each responsible to monitor the spectrum in a given area and with fully autonomous decisions for signal detection and radio occupancy such as in the focus of related works [17, 85–88]. The approach we propose requires *a time synchronization in the order of the sampling rate of the signal to be decoded and up to the frequency bandwidth of the Radio Frequency (RF) frontend* (corresponding to sub-microseconds synchronization for our RF frontends).

Our contributions. We present and implement a system architecture able to collect the digitalized radio samples (output of the Analog-to-Digital Converter (ADC) of each sensor) of different sensors and to reconstruct and decode the signals in the backend. The collaborative approach alleviates the network bandwidth load used by each sensor by exploiting the similarity of the spectrum of sensors in the same coverage area. We provide the following key contributions:

- We introduce a distributed architecture for radio signal monitoring and decoding using the digital samples collected by low-cost spectrum sensors.

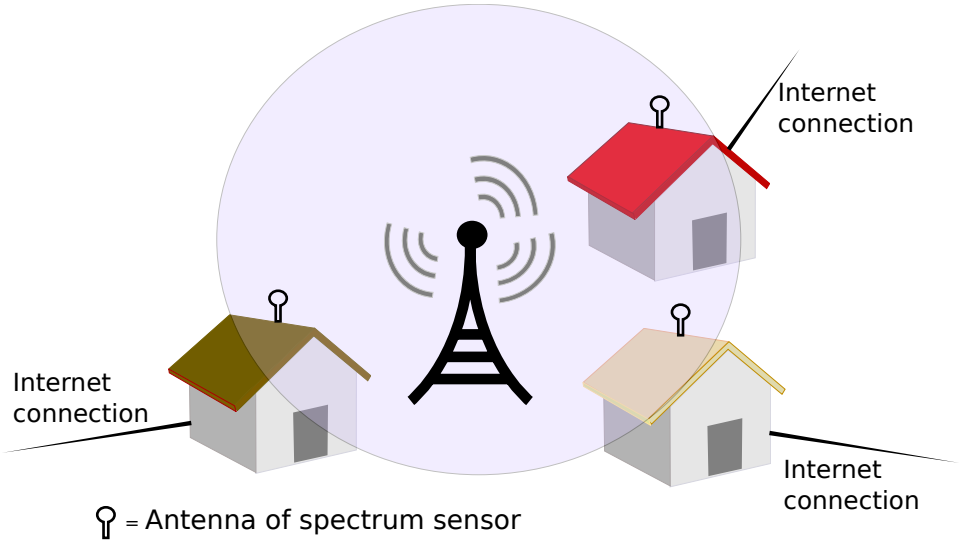


Figure 6.2: IoT RF sensors in coverage range of the same transmitter for Challenge 2.

- We propose different techniques to solve the problem of timing synchronization among the sensors even when it is not possible to use Global Positioning System (GPS) for signal acquisition synchronization.

- We present a distributed time multiplexing mechanism and introduce a Kalman filter model that operates in the backend. It is used to estimate and correct the relative time offset between pairs of sensors with a limited amount of uplink bandwidth usage.

- We evaluate the system with a proof-of-concept using real data from Long Term Evolution (LTE) base stations.

- We make the sensing spectrum software available as open source[1].

The reminder of this chapter is organized as follows. We discuss the system components and the principles for the time division approach in Section 6.1. We present our findings and related techniques to solve the problem of timing synchronization in Section 6.2. We present the distributed time multiplexing mechanism that uses a Kalman filter model to estimate the offset with a limited amount of data in Section 6.3. We then evaluate the platform and the algorithms in Section 6.4. Related work is provided in Section 6.6. Finally, we draw the conclusions in Section 6.7.

[1] http://github.com/electrosense/sensing/

6.1. Architecture

This work considers the overarching goal of collecting digitalized radio samples of encoded signals transmitted over the air and decoding them in the backend. Next, we present the proposed architecture which is based on the main ElectroSense architecture explained in Chapter 2 but we also add a GPS device in order to study and evaluate the time synchronization among IoT spectrum sensors.

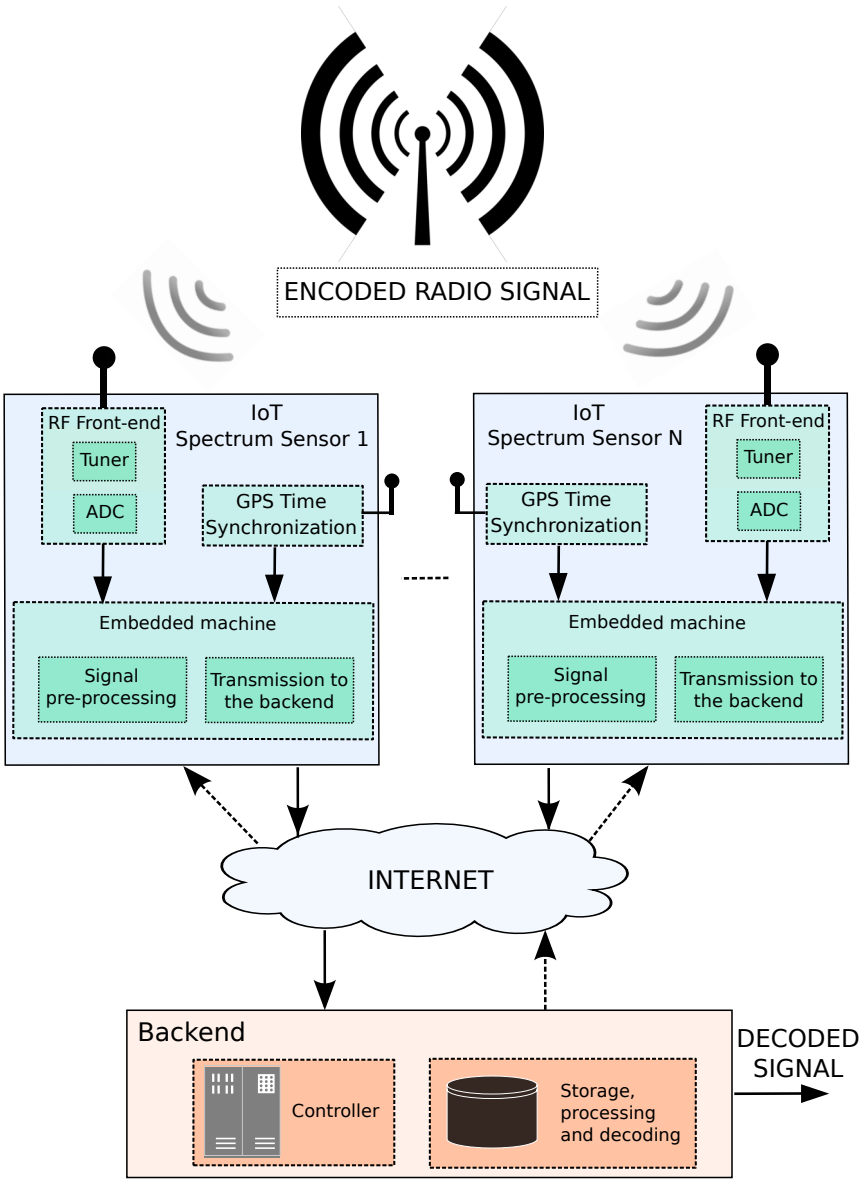


Figure 6.3: High-level overview of the distributed system architecture for collaborative radio signal monitoring and decoding.

6.1.1. System components

A high-level overview of the distributed system architecture for crowdsourcing radio signal monitoring and decoding is presented in Figure 6.3. We can distinguish two main components, the sensors and the backend.

IoT spectrum sensors. We consider sensors such as the ones shown in Figure 6.1. We rely on a low-cost software defined-radio USB dongle for signal acquisition and a commodity embedded machine for signal pre-processing and transmission to the backend. Our hardware components are commercial off-the-shelf and among the cheapest currently on the market (total sensor cost of less than 100 $).

Our *radio signal acquisition hardware* is a RTL-SDR based software-defined radio that acts as RF frontend, providing raw I/Q samples to the embedded machine over USB. The USB dongle can sample any radio frequency between 24 and 1766 MHz and send it to the embedded machine at a maximum rate of 2.4 MS/s without any sample losses.

The *embedded machine* is a Raspberry Pi, model B+ with a CPU clocked at 700 MHz and 512 MB of RAM. A dedicated Raspberry Pi software module accesses the RF frontend through the RTL-SDR library (*librtlsdr* from the OsmocomSDR project[2]) to retrieve raw I/Q samples in a configurable frequency band. The Raspberry Pi then processes, compresses (loss-less compression making use of the zlib data compression library [34]) and sends the samples to the backend through Internet using the on-board Ethernet interface.

Backend. The backend is a server or set of servers with sufficient large storage and computation capabilities. It contains the two main blocks shown in Figure 6.3. The controller block sends commands to the spectrum sensors and instructs them about the configuration scheme for data collection. The storage, processing and decoding block provides sufficient storage to save all I/Q samples and collaboratively recombines and decodes spectrum data.

6.1.2. Spectrum similarity for the network bandwidth problem

Determine the spectrum similarity among sensors can be done by correlating in frequency a small amount of data (c.f Appendix A). Reconstruct and decode the signal in the backend requires a more complex architecture, which is described in this Chapter.

Transmitting raw I/Q samples to the backend requires a large volume of data. In order to alleviate this problem, we consider that *low-cost sensors can allow a pervasive deployment*, with nodes belonging to users in the same neighborhood and observing a similar spectrum in the frequencies of interest.

Example. We consider two sensors with similarity in the spectrum. The controller in the backend (c.f Figure 6.3) assigns time slots to each sensor to collect I/Q samples

[2] http://osmocom.org

alternating the sensors in subsequent time slots (*sensor 1* in time slot 1, *sensor 2* in time slot 2, *sensor 1* in time slot 3, etc.). Samples are sent by *sensor 1* and *sensor 2* to the backend. After signal processing and storing, the original signal is recombined and decoded. Considering slots of equal duration, this mechanism *alleviates the network bandwidth load of each user by a factor equal to the number of sensors in range of the same transmitter.*

Recombining the original signal with partial data from multiple sensors requires a tight time synchronization among sensors. Otherwise, the decoder will not be able to correctly decode the signals transmitted over the air. However, precisely collecting I/Q samples in each sensor during the time slots assigned to is a difficult task given our software-defined and low-cost distributed sensor network architecture. In the next section, we study this problem in details.

6.2. Timing Synchronization Analysis

In order to apply the time division approach presented in Section 6.1.2, the backend requires a precision close to the sampling rate of the signal to be decoded in order to align the I/Q samples. This precision can be up to the frequency bandwidth of the RF frontend, which corresponds to *sub-microseconds time synchronization* for our RF frontends. In this section, we study the techniques we apply and the problems we have to solve to guarantee this high precision of the time synchronization.

6.2.1. Precision with GPS disciplined oscillator

We embed a low-cost GPS Disciplined Oscillator (GPSDO) to improve the precision of the timing synchronization between pairs of spectrum sensors (see Figure 6.1). The GPSDO works as a stable time reference with nanosecond accuracy, and it is directly connected to the Raspberry Pi (RPi) as a global time reference using a General Purpose Input/Output (GPIO) pin. Using this pin, the GPSDO sends a Pulse Per Second (PPS) signal every second. In this way, the Raspberry Pi knows when a second starts and can correct any possible local clock drift. Local timestamps are then appended by the Raspberry Pi to each single I/Q sample using the time reference provided by the GPSDO module.

Evaluation. We integrate the GPSDO module in two sensors and evaluate their relative time offset in two scenarios. In the *GPSDO+RPi* scenario, the two sensors are scheduled to execute a command (local timestamp acquisition) at a given absolute time. The offset is computed as the difference of the local timestamps acquired in each sensor. In the *GPSDO+RPi+RF* scenario, the sensors are scheduled to start the RF sensing command to tune to the same central frequency at the same absolute time. The offset is computed by detecting the time shift between the signals captured by the two sensors

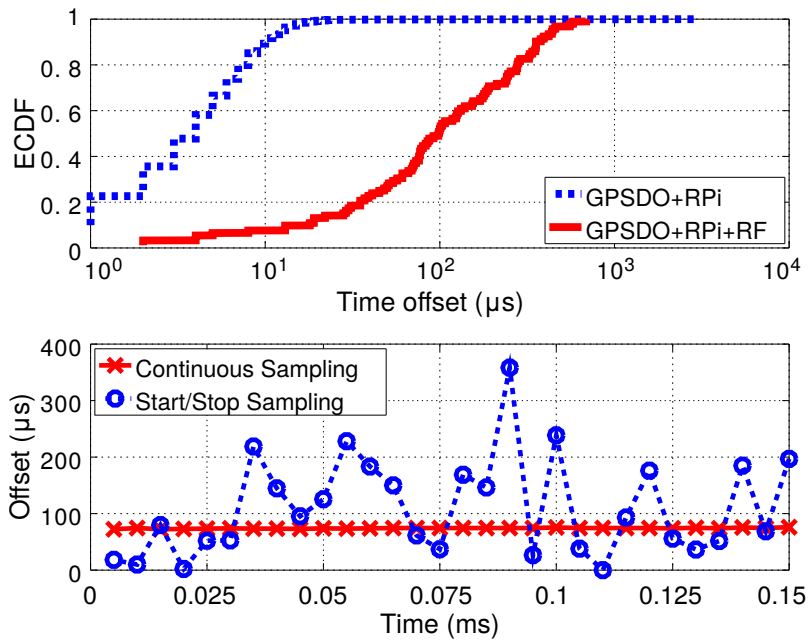


Figure 6.4: GPSDO+RPi: time offset between spectrum sensors computed at the system level; GPSDO+RPi+RF: time offset for the acquisition of samples via the radio frontend (top). Comparison of the time offset, using continuous sampling strategy in both sensors and the start/stop sampling strategy in both sensors (bottom).

(the methodology to detect the time shift is introduced in Section 6.2.3). The results are plotted in Figure 6.4 (top). We observe that the GPSDO does not suffice to achieve high precision (1 μs or less) between sensors in both scenarios.

Results analysis. There are three fundamental reasons imposed by our low-cost sensor hardware for the results in Figure 6.4 (top). First, the minimum resolution of the *embedded machine software clock* (i.e. of the RPi) is 1 μs. Even with the GPSDO as a time reference signal, Figure 6.4 (top) shows that the 80th-percentile of the time offset between two sensors (GPSDO+RPi) remains up to 8 μs. It follows that this software clock is subjected to significant noise.

Second, the impossibility to directly discipline the Local Oscillator (LO) of the RF front-end by using the GPSDO. Low-cost RF front-ends, such as the one shown in Figure 6.1, do not allow an input signal clock to discipline their internal LO without making any custom modification. As a result, it is not possible to synchronize the RF front-ends at nanosecond level in the signal acquisition stage.

Third, the software-based signal acquisition transmits I/Q samples over the USB interface of the spectrum sensor. This *USB interface introduces significant jitter* which manifests itself in large sampling offsets between signals that are acquired by different sensors. Figure 6.4 (top) shows that the 80th-percentile time offset of two signals acquired

by different sensors (GPSDO+RPi+RF) is about 278 μs. Considering a sampling rate of 2.4 MS/s, an offset of 278 μs results already in a signal misalignment of more than 600 samples.

6.2.2. Continuous sampling methodology

The results in the previous section demonstrate a significant relative time shift of the I/Q samples collected by independent spectrum sensors. We now instruct the two sensors to start and stop sampling at the same time and we compute the offset by detecting the time shift of the signals acquired. Figure 6.4 (bottom) shows that the time offset between sensors varies over time with ranges between 0 and 400 μs (0-800 I/Q samples). Such a large variation of the offset is undesired for the time division approach presented in Section 6.1.2.

While a time-division approach intuitively suggests to apply a start-stop approach as the one studied above, we propose instead to continuously collect samples from the RF interface in each spectrum sensor. In other terms, we start the sampling process at the booting time of the board and do not stop it. In this method, *the sensor is sampling continuously, but it just sends the I/Q samples of its slot time to the backend and dumps the rest of samples.* We show the resulting improvement in Figure 6.4 (bottom). While this approach is still affected by a time offset between sensors (in the experiment, approximately 78 μs $\sim$ 156 I/Q samples), the offset drift is largely reduced. We observe a variation range of 8 I/Q samples over a short time, 100 times less than using the start/stop method.

There are two advantages of this methodology. First, it inherently reduces any delay caused by the main board for processing the requests to start and stop (embedded machine software clock). Second, it significantly reduces the drift caused by the communication between the main embedded board and the USB interface of the RF frontend. The approach of continuously sampling the radio can be applied also to the RPi B+ used in this work (an entry-level model), which features only one CPU core. In fact, the CPU speed is sufficiently faster than the sampling process, and it is capable of executing other tasks such as compressing and transmitting I/Q samples to the backend.

6.2.3. Technology independent offset computation

A longer trace over time of the continuous sampling approach is shown in Figure 6.5(top) where we can observe how the offset increases over the time. This implies that the internal clock of one of the RF frontend tuners works faster than the other one. In order to compute the relative offset, we have used a beacon signal detection mechanism (Primary Synchronization Signal (PSS) signal of LTE, see also Section 6.4.1). However, this method has the drawback of being *technology dependent.* In fact, a beaconing signal

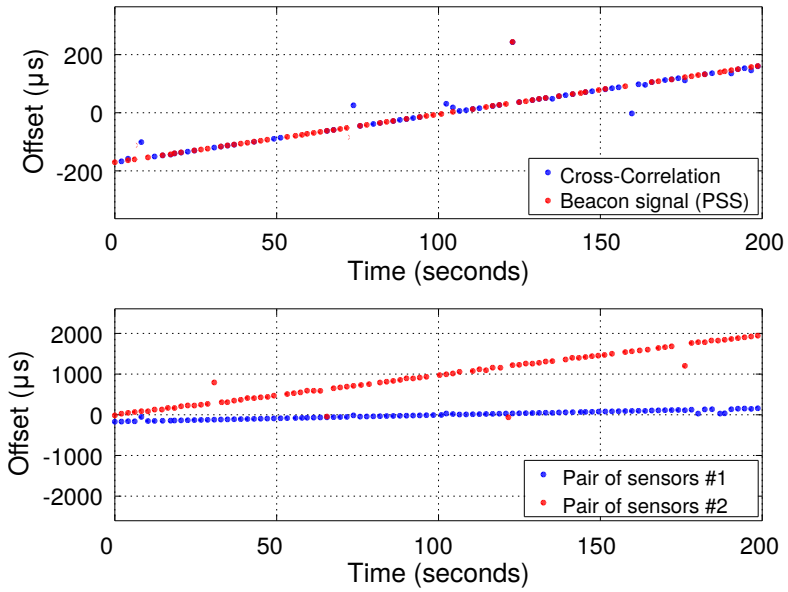


Figure 6.5: Time offset computed using cross-correlation between signals and beacon signal detection (top). Each pair of RF frontend tuners has a different drift (bottom).

is not always available, which calls for an approach which is *technology independent*. We then propose to compute the relative offset between sensors using the cross-correlation between the I/Q samples collected by each sensor. Let $\mathbf{x}$ and $\mathbf{y}$ be the complex vector of I/Q samples of two different sensors and $r_{\mathbf{x},\mathbf{y}}[m]$ denote the cross-correlation of both vectors with a lag m. At time k, we then compute the time offset μ_k as the maximum cross-correlation:

$$\mu_k = \underset{m\in\{-I,\ldots,I\}}{\arg\max}\ r_{\mathbf{x},\mathbf{y}}[m] \tag{6.1}$$

where I is the maximum lag. As shown in Figure 6.5(top), we obtain similar offset values using both techniques. It follows that *we can rely on the maximum cross-correlation metric to compute the time offset between spectrum sensors.*

Figure 6.5 (bottom) shows that each pair of RF SDR receiver has a different, but stable, drift over time. From Figure 6.5 (bottom), we can infer that the offset largely depends on the pairs of RF SDR receivers. Not shown in Figure, when changing the RPi boards and using the same pair of RF SDR receivers, we observe a similar drift. This confirms that the offset depends on the pair of RF SDR receivers, but not on the main embedded boards. For "pair of sensors 2", the measured offset results in a relative drift equal to $\approx 73\,\text{kHz}$. This is consistent with the reported high frequency instability (up to 50 Parts Per Million (ppm)) of the low-cost on-board crystal oscillator of each RTL-SDR USB dongle [89].

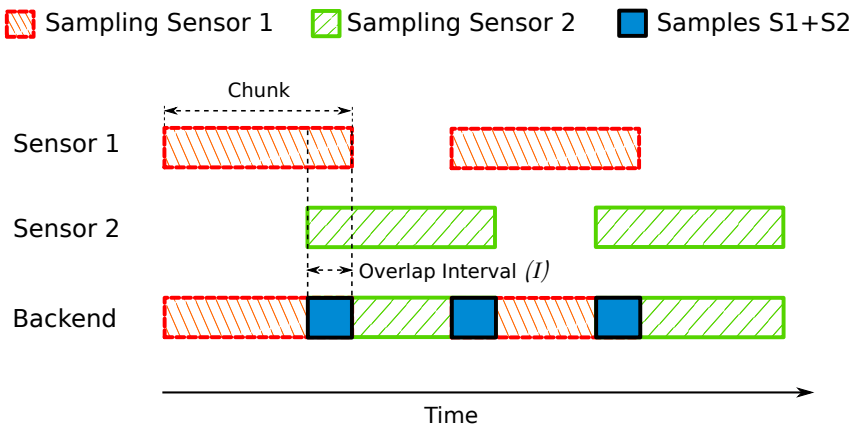


Figure 6.6: Time multiplexing mechanism. The whole frequency band is covered by collecting I/Q samples from multiple sensors. Cross-correlation analysis is enabled by overlap intervals.

6.3. Distributed time multiplexing

As explained in Section 6.1.2, we want to assign different time slots to different IoT spectrum sensors for I/Q sample collection. We have also seen that low-cost sensors impose significant limitations that affect the time synchronization among sensing nodes and that the correlation analysis provides a robust methodology to estimate the offset with continuous sampling (c.f. Section 6.2). However, the following problems emerge:

- Assigning continuous and independent slots to different sensors means that there is no spectrum data that can be used for correlation analysis.
- Even if we would be able to correctly estimate the time offset between sensors, correcting this offset in the sensor would imply to stop the sampling process. In turn, this would affect the quality of gathered I/Q samples, as shown in Figure 6.4 (top).

In order to solve these problems, we present in what follows our distributed time multiplexing mechanism to collect I/Q samples from different spectrum sensors for data decoding.

6.3.1. Overlap interval

We define time slots, called *chunks*, that include an *overlap interval* I, in which I/Q samples are sent from more than one sensor to the backend. Figure 6.6 shows a schematic of the distributed sampling technique. *Our idea is to use the overlap interval I to perform correlation analysis as in Eq. (6.1) and compute the offset.* A fundamental trade-off exists in the design of the overlap interval I:

(a) It should be sufficiently large to ensure that the peak of the cross-correlation (time offset) can be found.

(b) It should be sufficiently small to not waste the uplink network bandwidth usage.

6.3.2. Estimation of the offset between pairs of sensors

We derive a model to estimate the offset and provide the correct alignment among sensors. Our model is based on a Kalman Filter (KF) that allows us to estimate the offset and drift more precisely that the one based on a single noisy measurement. The model supports the empirical observation that the offset increases as a linear function over time due to the drift between RF frontend tuners (cf. Section 6.2).

The state vector that we want to estimate using the KF is $\mathbf{x} = \begin{bmatrix} O & D \end{bmatrix}^T$ where O represents the offset and D the drift, following this state model:

$$\mathbf{x_k} = \mathbf{F} \cdot \mathbf{x_{k-1}} + \mathbf{w_{k-1}}, \tag{6.2}$$

To derive $\mathbf{F}$ and $\mathbf{w_k}$, let O_k and $D_k = \dot{O}_k$ denote respectively the offset and the drift at time k. The second-order model is described by $\ddot{O}_k = n_k$ where $n_k \sim \mathcal{N}(0, \sigma_n^2)$ is an Additive White Gaussian Noise (AWGN). We assume that the drift is constant for this model since the offset shown in Figure 6.5 is linear over a sufficient large time period[3]. Thus, the choice of a very low value of $\sigma_n = 10^{-2}$. We derive the parameters of the state model as follow:

$$\dot{O}_k = \dot{O}_{k-1} + \Delta T_k \cdot n_{k-1}, \tag{6.3}$$

$$O_k = O_{k-1} + \Delta T_k \cdot \dot{O}_{k-1} + \frac{\Delta T_k^2}{2} \cdot n_{k-1}, \tag{6.4}$$

$$\begin{bmatrix} O_k \\ D_k \end{bmatrix} = \begin{bmatrix} O_{k-1} + \Delta T_k \cdot D_{k-1} \\ D_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta T_k^2}{2} \\ \Delta T_k \end{bmatrix} \cdot n_{k-1} \tag{6.5}$$

$$\mathbf{x_k} = \begin{bmatrix} O_k \\ D_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta T_k \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} O_{k-1} \\ D_{k-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta T_k^2}{2} \\ \Delta T_k \end{bmatrix} \cdot n_{k-1} \tag{6.6}$$

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta T_k \\ 0 & 1 \end{bmatrix}; \quad \mathbf{w_k} = \begin{bmatrix} \frac{\Delta T_k^2}{2} \\ \Delta T_k \end{bmatrix} \cdot n_k, \tag{6.7}$$

where $\Delta T_k = t_k - t_{k-1}$ and t_k represents the absolute time at iteration k.

[3]Effects such as the temperature can vary the offset, however they tend to occur at larger time scales than what we consider here, and can then be easily tracked with a small value of σ_n.

For the KF measurement model, we have:

$$\mathbf{z_k} = \mathbf{G} \cdot \mathbf{x_k} + \mathbf{v_k}, \tag{6.8}$$

$$\mathbf{z_k} = \begin{bmatrix} \mu_k \\ \Delta\mu_k \end{bmatrix} = \begin{bmatrix} O_k \\ \Delta T_k \cdot D_k \end{bmatrix} + \mathbf{v_k} \tag{6.9}$$

$$\mathbf{z_k} = \begin{bmatrix} \mu_k \\ \Delta\mu_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \Delta T_k \end{bmatrix} \cdot \begin{bmatrix} O_k \\ D_k \end{bmatrix} + \mathbf{v_k} \tag{6.10}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 \\ 0 & \Delta T_k \end{bmatrix} \tag{6.11}$$

where μ_k is the output (lag) of the maximum cross-correlation analysis in presence of continuous sampling (cf. Eq. (6.1)).

$\mathbf{w_k} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q_k})$ and $\mathbf{v_k} \sim \mathcal{N}(\mathbf{0}, \mathbf{R_k})$ represent respectively the process noise and the measurement noise that follow Gaussian distributions with autocovariance matrices $\mathbf{Q_k}$ and $\mathbf{R_k}$. According to Eq. (6.7), we derive the following formulation of $\mathbf{Q_k}$:

$$\mathbf{Q_k} = \mathbb{E}[\mathbf{w_k}\mathbf{w_k}^T] = \begin{bmatrix} \frac{\Delta T_k^4}{4} & \frac{\Delta T_k^3}{2} \\ \frac{\Delta T_k^3}{2} & \Delta T_k^2 \end{bmatrix} \cdot \sigma_n^2 \tag{6.12}$$

For the autocovariance matrix $\mathbf{R_k}$ of the measurement noise, we use an adaptive estimation based on the covariance matching method originally introduced in the context of GPS positioning. This method calculates the residuals $\hat{\boldsymbol{\nu}}_k = \mathbf{z_k} - \mathbf{G} \cdot \hat{\mathbf{x}}_\mathbf{k}$ that are the differences between the observation vectors $\mathbf{z_k}$ and their corresponding estimated values $\mathbf{G} \cdot \mathbf{x_k}$ consistent with their theoretical values ($\hat{\mathbf{x}}_\mathbf{k}$ is the corrected state vector). Then, it calculates $\mathbf{R_k}$ based on these residuals computed in the last w iterations [90].

As the KF assumes that the measurement noise is Gaussian distributed, any measurement outlier can negatively affect the filter. We propose a method to determine if the current measurement μ_k can be used to correct the prediction of the KF and improve the estimation. Figure 6.7 shows that the new offset estimation is based on the *prediction+correction* steps only if the new measurement is accepted. Otherwise, only the *prediction* step is used to estimate the offset. The current measurement is accepted if it is not an *outlier*. This evaluation method is based on the median calculation using the last estimates. Measurement values higher than a certain threshold are considered as *outliers* and discarded.

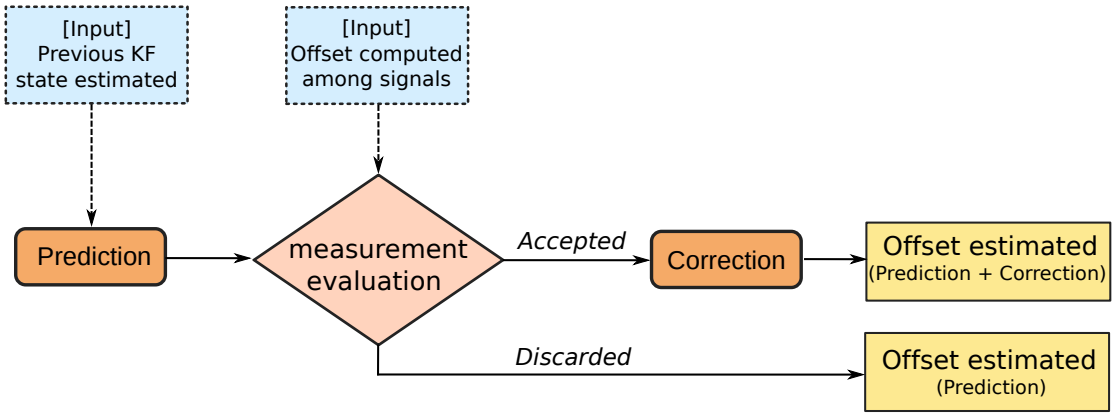


Figure 6.7: The KF correction step is applied only if the measurement μ_k is not discarded (it is not an outlier). Otherwise the time offset estimated for the next iteration will be computed using exclusively the prediction step.

6.3.3. Signal reconstruction and decoding

Figure 6.8 shows the workflow to reconstruct the signal and decode it using I/Q samples received from a set of sensors. The first step is to align the signals in time using the overlap interval I. In case of two sensors, the overlap interval contains samples from both sensors. Using the raw I/Q samples of this overlap interval, we compute the *cross-correlation* for different lags to determine the time offset μ_k between pairs of sensors. This offset is used in the data decoding step for iteration k and time slots assignment for iteration $k+1$.

Data decoding for iteration k. At each iteration k, the backend runs a test to verify if signal reconstruction is feasible. We distinguish the following cases:

- *Low spectrum similarity.* If μ_k is lower than a given threshold, spectrum sensors are too far apart (they are not listening to the same transmitter), or the overlap interval I_k is too small to mitigate the misalignment of the signals.

- *High spectrum similarity.* If μ_k is equal or above the threshold, we can proceed with signal combining and decoding.

In the latter case, we align the sequences of I/Q samples from pairs of sensors using the value μ_k. While this technique can be performed independently at each iteration, it is affected by noisy samples and the cross-correlation computation may erroneously declare the lag with the highest correlation value. In order to increase the robustness of the offset estimation, we also consider an estimator that uses the last N iterations of overlap intervals $\{I_{k-N-1}, I_{k-N-2}, \ldots, I_k\}$, and gives as output the *sequence of drift estimations*. Since the drift is approximately constant over a sufficiently short period (cf. Figure 6.5), using the last N subsequent overlap intervals, the *aligned+median* technique

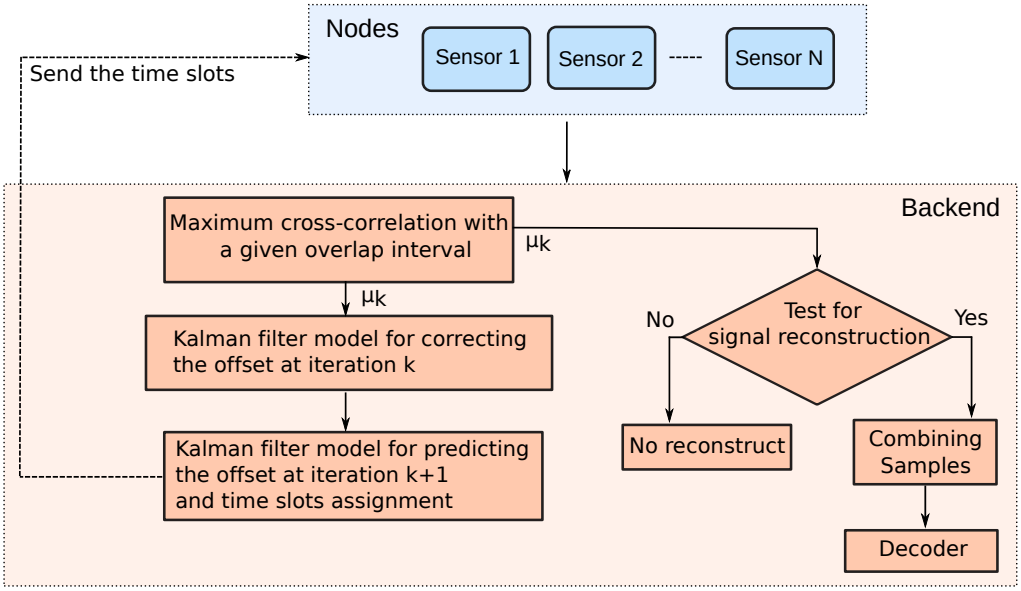


Figure 6.8: Workflow for collaborative signal monitoring and decoding.

takes as output the median of the sequence of drift estimations and converts it to the offset between pairs of sensors. Finally, the backend aligns the sequences of I/Q samples received from different spectrum sensors and it combines them in one sequence for data decoding. The decoder is unaware that the sequence of I/Q samples has been received with inputs from multiple spectrum sensors, and decodes the signal using standard demodulation and decoding techniques.

Time slots assignment for iteration $k+1$. Each spectrum sensor continuously collects I/Q samples, and the controller compensates their relative offset by anticipating or delaying the sequences of retrieved I/Q samples from each sensor accordingly. We start with μ_k as input (measurement model) of the KF model introduced in Section 6.3.2. The output of the KF gives the corrected time offset and drift $\hat{\mathbf{x}}_\mathbf{k}$ between pairs of spectrum sensors. As studied in Section 6.3.2, the KF correction step is applied only if μ_k is not an outlier. Otherwise this step is bypassed. Finally, the KF model predicts the offset for the iteration $k+1$ using the prediction step of the KF model, and uses it to assign the time slots in each sensor to compensate for their relative offset.

6.4. Evaluation

We evaluate our approach using LTE signals at 806 MHz in Madrid. Our analysis combines both simulation and real scenarios. The simulation scenario focuses on the impact of the offset among signals and how the noise affects the signal reconstruction. On the other hand, the practical analysis focuses on the evaluation of the different sampling strategies and the effect in the performance when the offset is estimated and corrected.

6.4.1. LTE

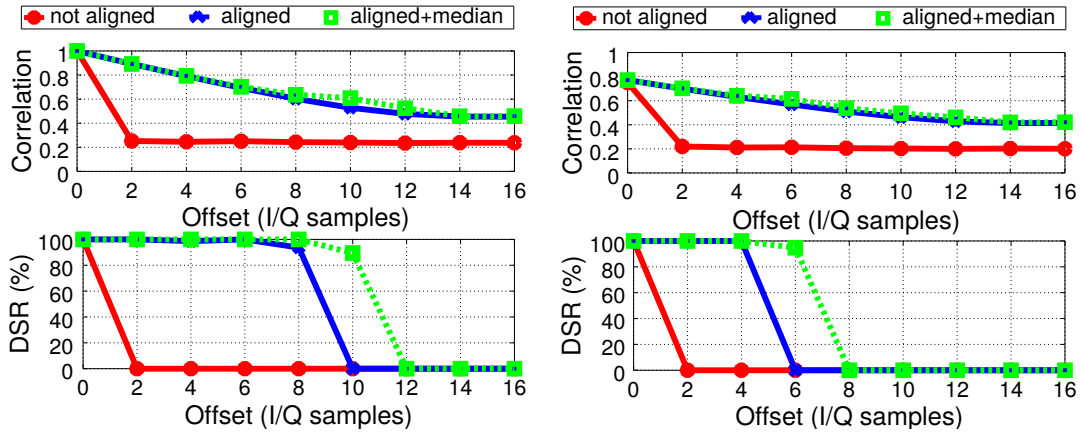
We first briefly review the main LTE concepts that are necessary for the evaluation. LTE defines two structures: *frame* and *subframe*. A *frame* is a structure represented in the time domain with a duration of 10 ms. Each *frame* contains 10 *subframes* of 1 ms duration and each *subframe* contains 7 Orthogonal Frequency Division Multiplexing (OFDM) symbols. An OFDM symbol corresponds to a variable number of samples depending on the system bandwidth. The UE (*User Equipment*) needs to get a cell id, a time slot and a frame synchronization in order to perform any more complex operation in a given network. The first step for the UE is to scan different frequencies and search for the PSS and Second Synchronization Signal (SSS), which have a band of 1.4 MHz. The PSS and SSS are located in subframes 0 and 5 of every frame. Since each subframe is 1 ms long, this means the UE can synchronize every 5 ms. Once the PSS is detected, the SSS is always located one OFDM symbol earlier. The PSS is a frequency-domain Zadoff-Chu [64] sequence and provides the *layer identity* (0 to 2). The SSS codes the *cell identity* as 1 out of 168 pseudo random sequences which are Binary Phase-Shift Keying (BPSK) modulated. Decoding the PSS and SSS properly, we obtain the Physical Cell Id (PCI) as: $\text{PCI} = 3 \times (\text{cell\_identity}) + \text{layer\_identity}$. To decode the PCI, the minimum sampling rate is 1.92 MS/s, which implies that an OFDM symbol is 128 I/Q samples long. We use the LTE-Cell-Scanner[4] to decode the LTE channel cell id.

6.4.2. Emulation with real data

In this experiment, spectrum data is collected from one single spectrum sensor that scans continuously with a center frequency of 806 MHz. With this data, our simulation environment creates two datasets using the following configuration: chunk_size = 100 I/Q samples and overlap_interval = 20 I/Q samples. We introduce artificial time offsets and add Gaussian noise to one copy of the signal in order to understand the impact of sensor synchronization errors and noise on the signal recombining and decoding process. We set the threshold correlation value to determine if there is sufficient similarity or not for signal reconstruction equal to 0.65 (cf. Section 6.3.3). We recombine the signal in time using these two datasets and finally decode the control information of the LTE channel.

Figure 6.9(a) shows the cross-correlation value and decoding success rate for different techniques and different offsets. The *aligned* technique can decode the signals in the simulation environment with a high success rate as long as the offset is not higher than 40% of the overlap interval. In addition, the *aligned+median* technique can decode the signals as long as the offset is not higher than 50% of the overlap interval. These results outperform the *not-aligned* technique that simply assumes there is no time offset between signals and computes the cross-correlation without shifting on the overlap area of both

[4] https://github.com/Evrytania/LTE-Cell-Scanner



(a) No noise.

(b) SNR = 10 dB

Figure 6.9: Average correlation (top) and decoding success rate (DSR) applying different artificial offsets and several techniques to reconstruct the signal (bottom) (chunk_size = 100 I/Q and overlap_interval = 20 I/Q).

datasets (therefore, it only works with an offset equal to 0).

We also evaluate the impact of the noise. In order to conduct this experiment, we add AWGN to one of the signals. The results are shown in Figure 6.9(b). For the evaluation, we set the signal-to-noise ratio equal to SNR = 10 dB. The *aligned* and *aligned+median* techniques reduce their success rate, but it is still possible to decode the signal when the offset is lower than 20% and 30% of the overlap_interval, respectively. We finally evaluate the robustness of the different techniques in presence of various levels of noise and offset = 10 I/Q. Figure 6.10 shows that the *aligned-median* technique clearly outperforms the other two methods.

6.4.3. Real scenario

In this experiment, two IoT spectrum sensors are located three meters from each other, scanning in the center frequency of 806 MHz (wavelength is 0.37 meters) with a sampling rate of 1.92 MS/s (complex samples). In our envisioned crowdsourcing scenario, sensing nodes may be located farther apart but this reference scenario serves as a baseline to understand the decoding performance when spectrum sensors receive almost identical signals. As in the previous section, we compare the decoding success rate using different sampling techniques. In these experiments, we compare the sampling process using the start/stop approach and the continuous approach introduced in Section 6.2.2. This experiment is executed 300 times. Each time, the generated dataset contains between 15 and 18 PSS-SSS signals, corresponding to a total of more than 4000 PSS-SSS.

As explained in Section 6.2.2, the start/stop sampling approach introduces a large and unpredictable offset. This implies that the overlap areas are misaligned and the correlation

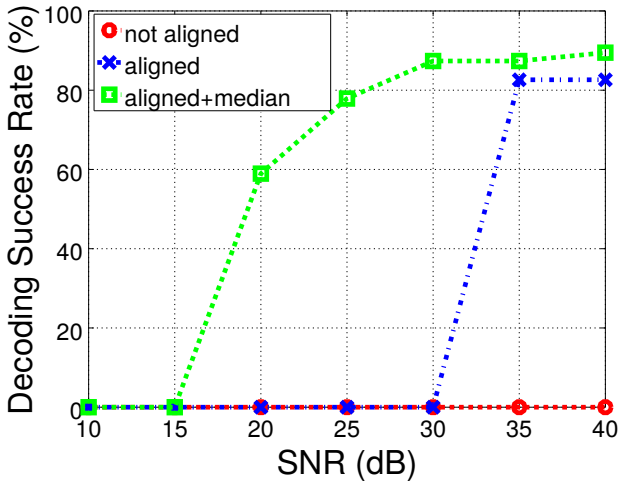


Figure 6.10: Decoding success rate applying different SNR values and evaluating the signal reconstruction techniques (offset = 10 I/Q, chunk_size = 100 I/Q, overlap_interval = 20 I/Q).

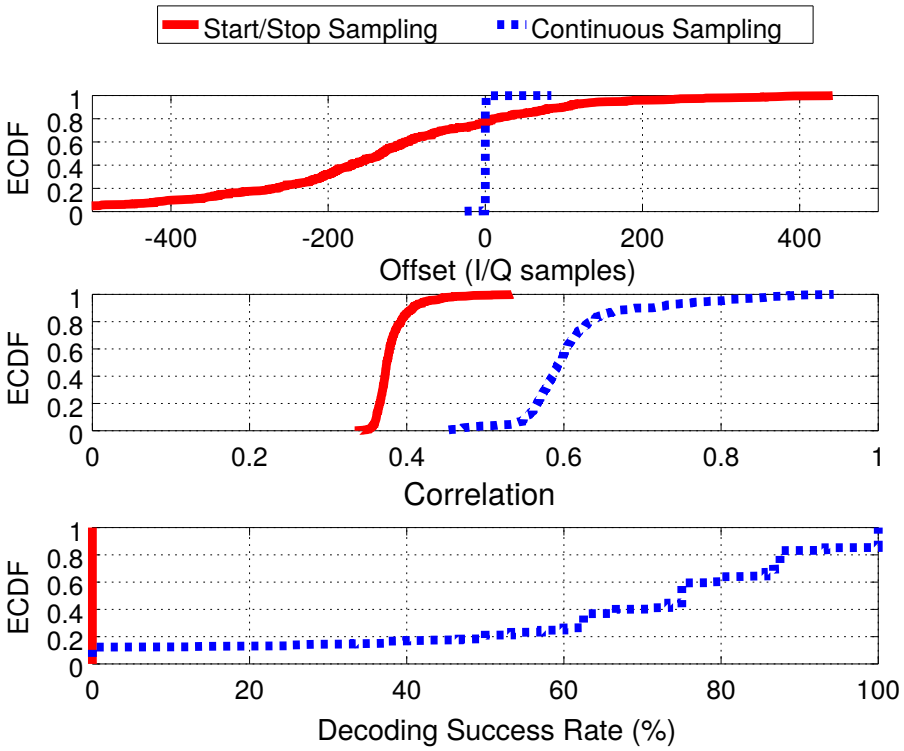


Figure 6.11: Evaluation of the offset (top), correlation (middle) and decoding success rate (bottom) between the start/stop and continuous scanning strategies using real spectrum data coming from two different spectrum sensors (chunk_size = 100 I/Q, overlap_interval = 20 I/Q).

value is low. Figure 6.11 (top) shows how the offset spans a very large set of values, and therefore the correlation value is not enough to obtain reliable decoding information. However, the continuous sampling approach allows to keep the overlap interval aligned thanks to the KF model proposed in this work. As shown in Figure 6.11 (middle), the correlation value increases when applying the *continuous sampling* technique since the overlap intervals are aligned. We finally evaluate the impact on the decoding success rate in Figure 6.11 (bottom). While the correlation for start/stop sampling does not guarantee reliable decoding (with values close to zero), the decoding success rate is highly improved with the proposed continuous sampling approach, making the system functional.

6.4.4. Evaluation of the Kalman filter model for offset estimation

We study the accuracy of the KF model using the same setup scenario of Section 6.4.3 and we present the results in Figure 6.12. First we consider the case of a long overlap interval (10,000 samples) which guarantees that the maximum of the offset can be easily found with the *technology independent* cross-correlation analysis. The plot in Figure 6.12 (top) for 10,000 I/Q samples ("estimated offset (10000)") shows that the filter has the same performance as the real offset (computed using the *technology dependent* approach in Section 6.2.3).

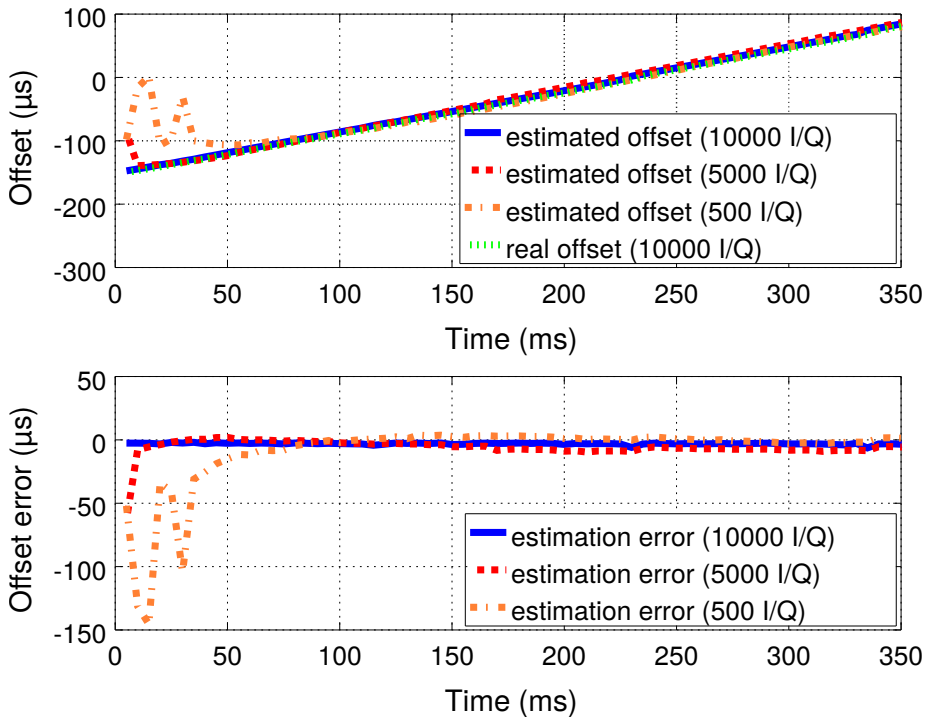


Figure 6.12: Estimated offset (top) and estimation error (bottom) using the proposed Kalman Filter model.

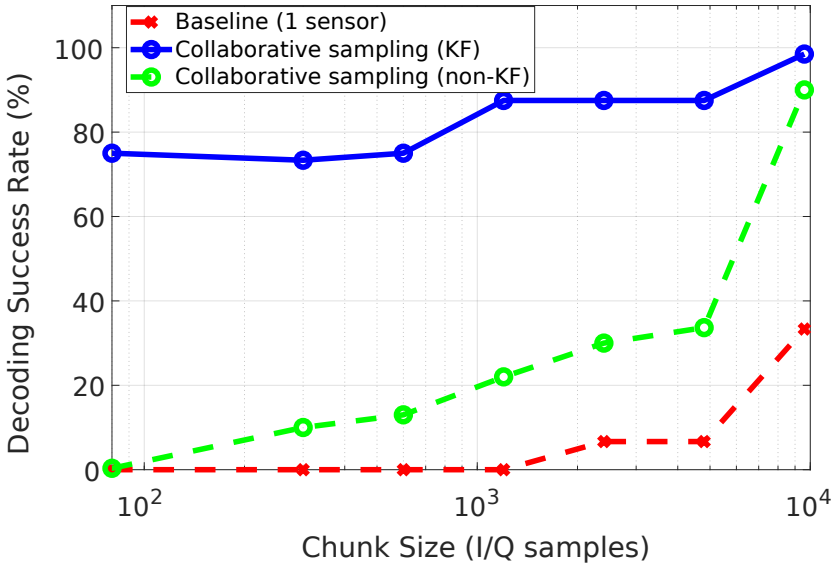


Figure 6.13: Decoding success rate of cell id (overlap_interval = 20 I/Q).

As explained in Section 6.3.1, a smaller overlap interval should be preferred to reduce the load of the uplink bandwidth of users. We then evaluate the accuracy of the filter with smaller overlap intervals. As expected, the convergence is slightly slower. Yet, as shown in Figure 6.12 for 500 and 5,000 I/Q samples, the filter is always able to converge to the true offset. The KF model adapts well to different conditions. For instance (not shown in figure), it estimates an average standard deviation of the offset for 500 I/Q samples that is more than 15 times larger than for 10,000 I/Q samples. This shows that the estimated autocovariance matrix $\mathbf{R_k}$ of the measurement noise can deal with more noisy data observed with a smaller overlap interval I, and it automatically assigns a standard deviation that tends to increase when the overlap interval I decreases.

We then study the decoding success rate for different chunk sizes and using a small overlap interval (20 I/Q). For a fair comparison in terms of network bandwidth, the baseline is represented by one spectrum sensor scanning in its assigned time slots. We present the results in Figure 6.13. In the case that the KF model is not applied, the decoding success rate decreases considerably for almost all chunk size values. Here, the overlap intervals are misaligned and the predicted offset is not corrected. For all the different chunk sizes, the collaborative sampling approach using the KF model provides a high decoding success rate of the signals recombined from the I/Q samples of two spectrum sensors.

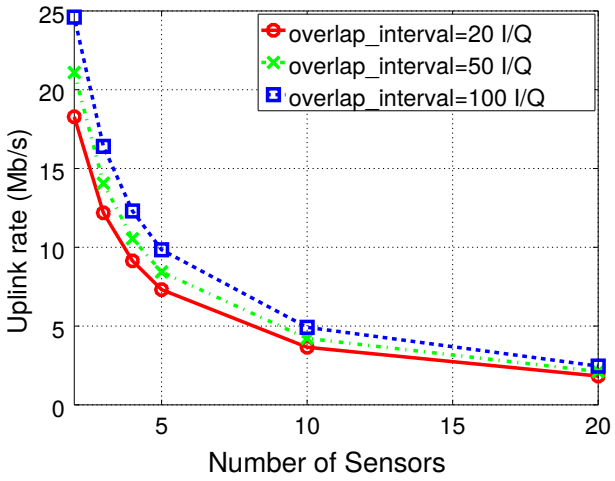


Figure 6.14: Uplink bandwidth used by each IoT spectrum sensor.

6.4.5. Uplink network bandwidth

One of the important benefits to use the collaborative approach for signal reconstruction and decoding is that it can reduce the uplink bandwidth used by each single IoT RF sensor. One single node scanning continuously a LTE channel at 1.92 MS/s would need an uplink bandwidth of about 36.8 Mb/s to send all I/Q samples to the backend using a compression factor of 70 %. Depending on the number of spectrum sensors and the sampling configuration parameters of the system (chunk size and overlap interval), the uplink network bandwidth for each IoT spectrum sensor can be reduced. Figure 6.14 shows in simulations how the uplink network bandwidth decreases considering a set of spectrum sensors in range of the same transmitter. The crowdsourcing approach clearly allows to relieve the network bandwidth per user.

6.5. Extended Evaluation

We have extended the evaluation of the signal recombination methodology using other different type of signal. In this case, we evaluate the decoding success rate of Mode S downlink channel. This communication channel is used by aircraft to send air-traffic messages (refer to Section 5.1.1 of previous chapter to details). We rely on dump1090 decoder [78] to obtain the decoding success rate after the recombination. Figure 6.15 shows the decoding success rate of Mode S packets when a small overlap interval is used (20 I/Q). As we also saw in the previous evaluation, for all different chunk sizes evaluated the collaborative approach using the KF model provides a better decoding success rate of the packets. We can also observe that for small chunk sizes (lower than 10^3 samples) the estimated offset by the KF results much more beneficial for the decoding success rate

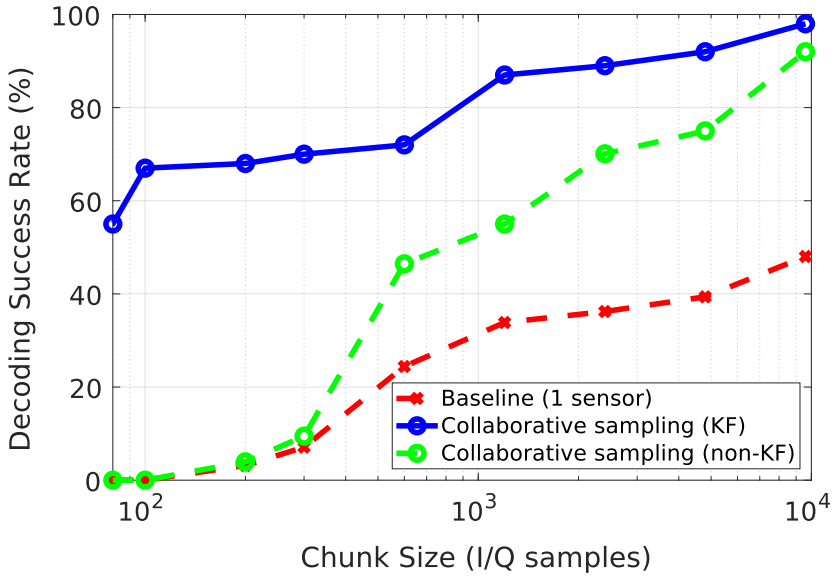


Figure 6.15: Decoding success rate of Mode S downlink messages (overlap_interval = 20 I/Q).

in comparison with the non-KF scenario. If the KF is not applied to estimate properly the offset between signals, the decoding success rate decreases dramatically. The reason for this is because there is no tracking of the offset between signals, and overlap interval only contains a few IQ samples. As a consequence, the correlation is more sensitive to noise in the overlap interval.

6.6. Related work

Large corporations have shown interest in the topic of large-scale spectrum sensing. Google has lunched the Spectrum Database [23], a joint initiative between Google, industry and regulators to make more spectrum available by using a database to enable dynamic spectrum sharing in TV white spaces. They provide an access to the data to query white space availability based on time and location. Microsoft Spectrum Observatory [16] is a platform with a high cost (approximately 5000 dollars per node). Using data collected with the Spectrum Observatory, [88] proposed a system that identifies transmitters from raw spectrum measurements without prior knowledge of transmitter signatures. SpecNet [17] is a platform developed by Microsoft Research that the scientific community can use to remotely schedule spectrum measurements in real-time in order to study the spectrum usage. ElectroSense [4, 7] and BlueHorizon [24] are crowdsourced architectures that allow to share spectrum information. In contrast to the works above, the sensors in our system work collaboratively with the overarching goal of reconstructing and decoding a radio signal transmitted in the backend.

[87] described the implementation and evaluation of a real-time, centralized spectrum monitoring and alerting system. Their analysis is conducted using binary vectors, by comparing each power value in the received power vector to a user-defined threshold. [91] introduced the idea of cooperative sensing where a certain frequency spectrum is monitored distributively with different sensing nodes. [92] proposed to use a cooperative environment to distinguish between an unused band and deep fade due to shadowing or fading effects. That work has been studied only by means of simulations. [93, 94] employed correlation techniques in different environments, but the study is based on simulations only. No system architecture problems were studied in these works for their actual implementation. In addition, unlike classical collaborative decoding [84] and cooperative diversity schemes [95], the sensors in our work are not performing the physical-layer decoding, but just provide interleaved measurements of raw I/Q samples which are stored and decoded in the backend. SpecInsight was introduced in [85] and is a system for acquiring 4 GHz of spectrum in real-time using Universal Software Radio Peripheral (USRP) radios with tens of MHz in 7 locations in the US. Because of the high-end platform used in their work, there are little opportunities for pervasive deployments. [34] proposed different frequency hopping strategies to overcome the hardware limitations of low-cost radios. These systems considered spectrum data in the frequency domain, while we consider data in the time domain (I/Q samples), posing new challenges, as discussed in this chapter.

6.7. Discussion

We have studied the problem of crowdsourcing spectrum data decoding using low-cost software defined radios. We have addressed the main challenges and proposed a distributed approach and several techniques and strategies for sampling the spectrum collaboratively. We have proposed sampling methods on the sensor side and synchronization techniques on the backend side in order to align, at the sub-microsecond level, the signals received from multiple sensors connected over the Internet when traditional approaches, such as discipline the LO of the RF frontend, are not an option due to the hardware limitations. Our approach can reconstruct the signal based on raw I/Q samples received by different low-cost sensors, all in range of the same transmitter. We have provided an evaluation with real LTE and Mode S signals and shown the feasibility to reconstruct and decode signals in a crowdsourcing scenario with low-cost sensors.

"Coming together is a beginning, staying together is progress, and working together is success."

Henry Ford (1863 – 1947)

7 Collaborative Wideband Spectrum Data Decoding

We are in the age where crowdsourcing systems are helping the society to solve complex problems by collecting data, using low-cost devices massively deployed around the world and exploiting collaborative techniques among them. These systems can measure different indicators as pollution, temperature, solar radiation, and air-traffic signals. More recently, we are experiencing an important growth of interest for crowdsourced sensing of the electromagnetic spectrum and democratize the access to the community. Several projects such as ElectroSense [4, 7] rely on volunteers that host low-cost Internet-of-Things (IoT) spectrum sensors to sense the electromagnetic spectrum at large scale. Having IoT Radio Frequency (RF) sensors deployed massively enables new strategies of spectrum analysis. Yet, as Analog-to-Digital Converter (ADC) are costly, said Software Defined Radio (SDR) receivers are affected by a low sampling rate. The consequence is that they do not fulfill the Nyquist-Shannon theorem for several types of wideband signals and are not able to decode them.

In particular, what has strongly gained attention in the research community is the ability to sense the spectrum using low-cost SDR receivers, such as the RTL-SDR [62]. These receivers are used in large deployments such as RadioHound [27], ElectroSense [4,7] (for sensing radio spectrum) and OpenSky [51] (for collecting air-traffic signals). RTL-SDR receivers have an acceptable performance but obviously due to their low quality components they have serious limitations in terms of ADC resolution, frequency range or maximum signal bandwidth, and lack calibration. The maximum bandwidth of low-cost RTL-SDR usually is 2.4 MHz [62], meaning they cannot decode signals such as Long Term Evolution (LTE) (10 MHz), Wi-Fi (20 MHz), WiMAX (20 MHz) or air traffic signals such as Mode-S uplink (4 MHz) [67].

In order to solve this problem, we envision a scenario where low-cost RF spectrum sensors deployed distributively in an area work collaboratively to cover a bandwidth larger than the one of a single SDR receiver. As a crowdsourced approach, we use such a distributed sensor arrangement to maximize the signal coverage of the system, even if some sensors may suffer reception issues in the band of interest (higher-capability receivers are

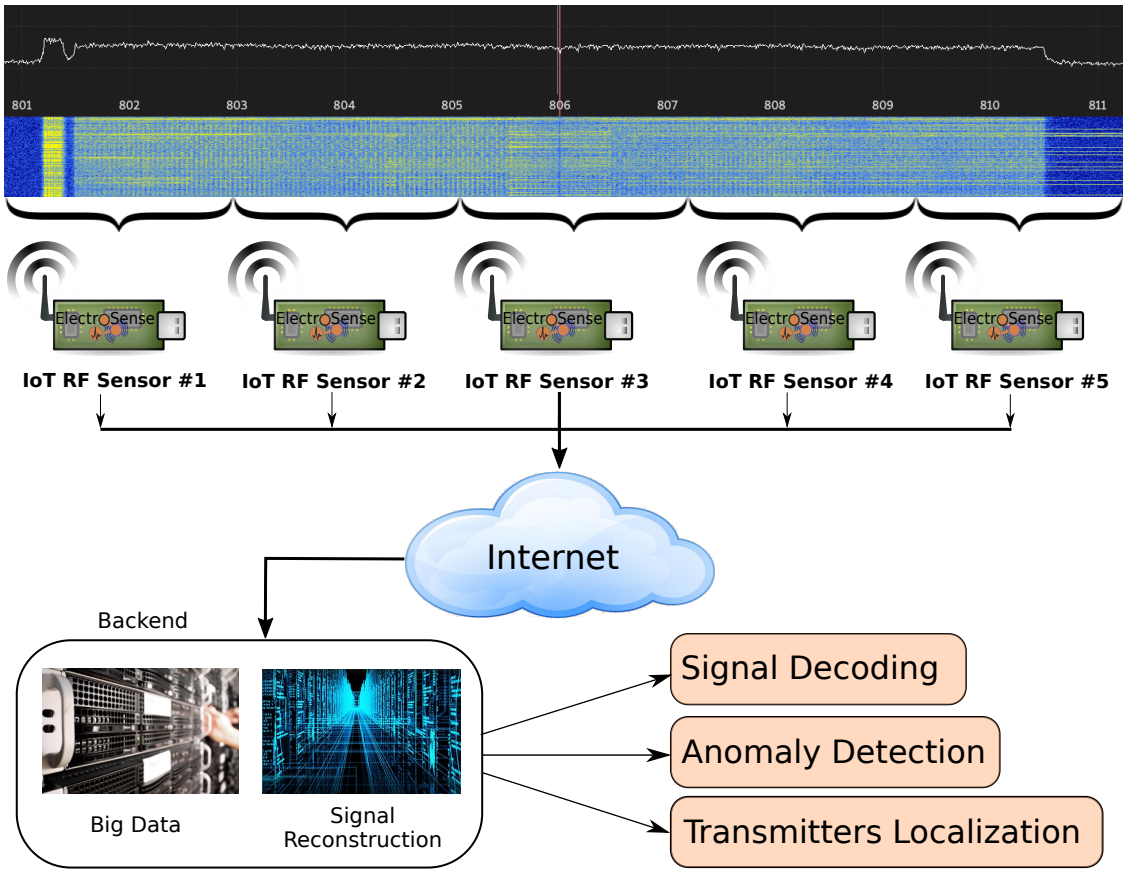


Figure 7.1: Several IoT RF sensors distributed in a geographical area cover a bandwidth larger than the single spectrum sensor. By letting each spectrum sensor sample different adjacent portions of the wideband signal, we can reconstruct and decode a signal, which would not be possible using one single receiver.

less suited due to the increased cost of geographical scalability). A pictorial representation is given in Figure 7.1. Several IoT spectrum sensors located in different places are multiplexed in frequency, and take care of a specific portion of the signal. The spectrum data obtained by the sensors is sent to the backend where the signal is reconstructed. This scenario presents important challenges among non-coherent SDR receivers that must be solved in order to guarantee the signal integrity and the reconstruction in the backend using the partial view of the signal seen by each sensor. The signal reconstruction in the backend is performed in an *agnostic* way, i.e., the system is not aware of the specific modulation scheme. As such, the system must deal with the fundamental signal information (amplitude, frequency, phase...) provided by each sensor to make the signal recombination feasible in the backend.

In this chapter we aim to provide a generic methodology and architecture to enable the signal reconstruction in the backend by multiplexing in frequency a certain number of IoT spectrum sensors in order to cover a signal bandwidth that would not otherwise

be received using a single sensor. Our contributions are as follows:

- We propose a distributed frequency multiplexing mechanism for covering bandwidths higher than what a single SDR receiver can.
- We identify the main challenges in signal processing to solve in order to enable the collaborative signal reconstruction.
- We present an architecture for collaborative signal reconstruction performed in a common backend that does not use information from the modulation scheme.
- We evaluate our solution in a real scenario using low-cost IoT RF sensors for reconstructing and decoding collaboratively air traffic signals (Mode-S [79]) in the backend and we compare our solution against high-performance receivers.

7.1. Motivation

The collaborative and distributed signal decoding using non-coherent and low-cost SDR receivers is the main scope of the work described in the Chapter. Decoding signals of larger bandwidth than a single receiver in a distributed network is the main application that is presented in this work. These sensors are independently deployed by users in a given area and connected to the backend over the internet. Yet, the ability of decoding signals larger than what a single spectrum sensor can do is a *primitive* that can enable other key applications, a foundation that exploits the crowdsourcing and the distribution of costs among all participants to now be able to do more with a sensor, and can provide incentives for people in the same neighborhood to join the initiative. The applications below are presented for completeness.

- Anomaly detection. Anomaly detection can be more powerful if part of the data can be labelled (e.g. signal-modulation classification [46]). Yet, if the signal is of wider bandwidth than the SDR receiver, the backend cannot decode the data and label it properly, limiting the applicability of methods already proposed [19] to detect anomalies in the spectrum.

- Localization. For signals of larger bandwidth than the SDR receiver, the ability to decode the signal can be considered as a sanity check that the different IoT spectrum sensors are synchronized. By exploiting the position of known transmitters, the IoT spectrum sensors can then solve the delay from the transmitter to each spectrum sensor and from the spectrum sensor to the backend. Also, the fact that a subset of IoT spectrum sensors yields the best collaborative decoding rate of a known signal may be used as an indicator that this group of sensors could be the optimum to localize the transmitter.

- Selection of sensors. Usually crowdsourced and collaborative systems face the problem of choosing the most favourable set of sensors to achieve a specific goal as in

spectrum monitoring, transmitter localization or detection of anomalies in the spectrum. Choosing the closest sensors not always ensures the best performance, due to interference or antenna location. IoT spectrum sensors can be instead better grouped depending on how good they perform in the process of decoding collaboratively a known signal. The problem of choosing the best subset of sensors for this task is not trivial. If one of those sensors reports bad information it may taint the final goal. Besides the great value and contribution of collaboratively decoding a signal using different low-cost sensors, this method can help in ranking and grouping sensors in crowdsourced spectrum monitoring networks.

- Enhanced diversity. By having IoT spectrum sensors located away from each other we can implement a diversity schema to increase the signal-to-noise ratio. The methods exposed in this work can be used to align signals, so they can be further processed in equal-ratio, maximal-ratio or selection combination.

7.2. Decoding wideband signals with distributed sensors

The objective of this work is to use different low-cost SDR receivers to cover a high bandwidth that would not otherwise be possible using one single receiver given its limited ADC sampling rate. The first problem to address is what type of receiver to use, and two options are possible. On the one hand, coherent-receivers [96] know the type of signal that has been transmitted, and typically use calibration and synchronization techniques in order to synchronize in time and phase and compensate for impairments in the wireless channel. On the other hand, non-coherent receivers do not have instead any knowledge of the signal properties. Spectrum sensors have the capabilities of collecting In-phase & Quadrature (I/Q) samples at different frequencies (in the case of the RTL-SDR, with low ADC sampling rate, the range is between 24 and 1766 MHz [34]), and they do not need to interact with the transmitters (no strict time requirements to process the incoming I/Q data). Therefore we propose to use non-coherent receivers to *divide the wideband signals in different sub-channels*, each of them monitored by one single receiver, and then reconstruct *said wideband signals* in the backend. In other terms, we apply a *frequency division approach* to multiplex the signal. The advantage of non-coherent receivers is that they do not need to be close to each other and share internal components (e.g., local oscillators), but the disadvantage is that the signals provided by non-coherent receivers are not synchronized.

7.2.1. Common signals in the frequency domain

In typical non-dense frequency-division multiplexing, the spectrum of each channel is independent. Instead, in order to solve the issue of synchronization among non-coherent receivers, we propose that the sub-channels have a small overlap area. We use this common

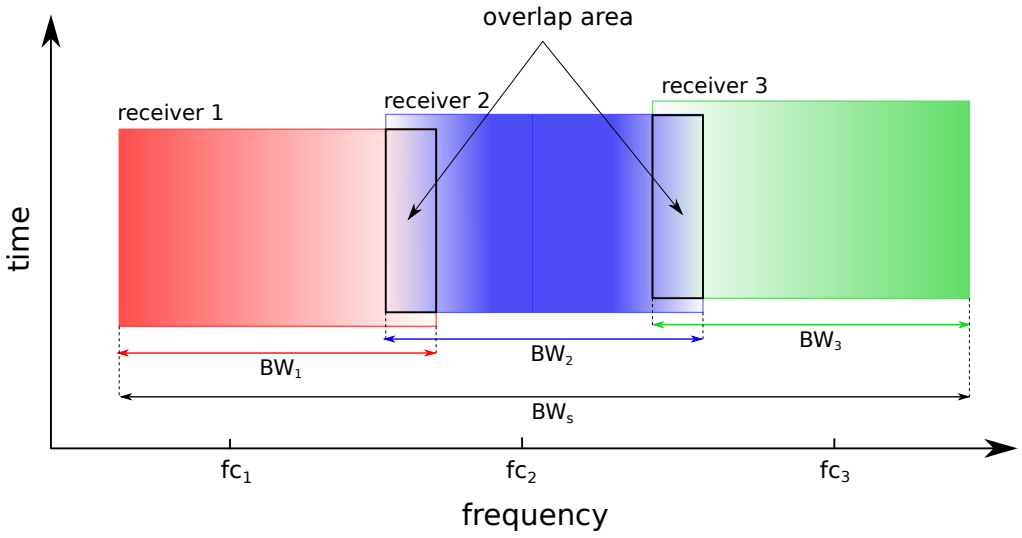


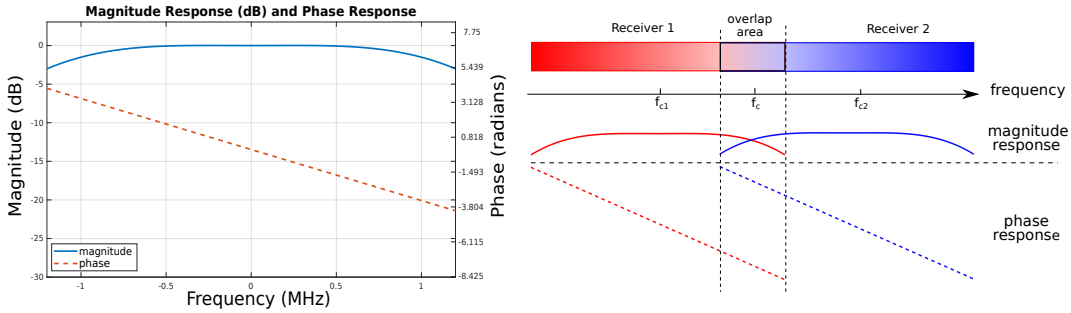
Figure 7.2: High-level representation of the frequency division approach to deal with wideband signals and low-cost spectrum sensors. The collected samples in the overlap areas are used to synchronize and equalize the signals in the backend.

area, named as *overlap area*, to extract the estimators needed to synchronize the signals. As Figure 7.2 shows, IoT spectrum sensors are located in nearby frequencies in such a way there is a overlap frequency area among them. This overlap area (O) contains common information that will be used to reconstruct the signal as it is explained later in Section 7.3. Notice that the overlap area size does not depend on the final aggregated bandwidth. The overlap area size can be set depending on the use case or the scenario conditions, e.g. if the noise in the signal is too high the overlap area bandwidth can be increased until enough data for a clean merge is available.

More formally, a given signal $s(t)$ with a specific bandwidth BW_s and center frequency f_{cs} is to be reconstructed and decoded by using N receivers, configured with certain bandwidths (BW_1, BW_2, ..., BW_n) and center frequencies (f_{c1}, f_{c2}, ..., f_{cn}) in such a way that the complete bandwidth of the signal $s(t)$ is covered. The signal reconstruction using the frequency division approach can be exploited as long as the sum of the bandwidths covered by the receivers is strictly greater that the bandwidth of the original signal as Eq. 7.1 shows:

$$\sum_{k=1}^{N} BW_k > BW_s \tag{7.1}$$

Notice that in the above equation the terms BW_k already include the overlap area bandwidth (c.f. Figure 7.2). Therefore, the overlap area between two consecutive receivers, which is the key of the signal reconstruction, given that the frequencies $f_{c(n)}$



(a) FIR filter of the RTL-SDR v3.

(b) FIR filter applied in 2 different receivers.

Figure 7.3: Low-pass filter of the RTL-SDR and how is applied in the overlap area.

are in ascending order, can be expressed as follows:

$$O_{(n,n-1)} = \left(f_{c(n-1)} + \frac{BW_{n-1}}{2}\right) - \left(f_{c(n)} - \frac{BW_n}{2}\right) \tag{7.2}$$

7.2.2. Challenges

The usage of distributed non-coherent receivers implies that we need to process the signals in such a way that we can cope with the signal impairments with respect to a pure ideal scenario. The following challenges must be solved in order to reconstruct the signal collaboratively by applying the proposed approach that exploits overlap areas.

Relative time delays between signals. As each signal traverses a different path (different location, antenna, receiver sampling time...), most likely a delay from one signal to the other will be found. The main root causes are: i) *Different time-of-arrival in multi-antennas scenarios*: the electromagnetic signal may reach each antenna at different times, mainly depending on the different path observed by different antennas. On a more generic set-up, the collaborative reception may be performed from distant receivers, so there may appear a time delay between the different received signals because of this distance; ii) *Different receiver architectures*: if using different receivers, the delay from the RF interface to the final acquisition point may differ. Also, as the sampling clock is a synthesized signal, the sampling instant can be additionally delayed from one signal to the other on a sub-sample quantity.

Frequency alignment. The internal clocks used in radio circuitry are affected by tolerances and environmental factors, so we need to handle these effects when combining signals. We have to bear in mind that this is an agnostic reconstruction without any knowledge of the signal properties, so the overall reconstructed signal does not guarantee a specific center frequency better than the clock tolerances. This is not a problem, as in typical low-cost SDR there is no Phase-Locked Loop (PLL) synchronization to the carrier [97], and demodulators cope with this effect. In fact for contributions from receivers

which are not receiving a carrier, performing phase-lock could be unfeasible. The main source of misalignment is the *offset in the local oscillators*. As a simplification on the signal flow, let us consider without loss of generality that on each receiver there is only one local oscillator to down-convert the radio transmission into an complex base-band signal. This oscillator may be affected by an offset in its reference clock (such as a quartz crystal, where tuning during manufacture and operating temperature may affect its nominal frequency). Additionally this type of oscillators is generated from a chain of integer multipliers and dividers of the clock frequency, so an additional granularity error may appear as well.

Signal power at the receiver. There could be many different factors that could impact the power of the sampled signal among different receivers (radio channel differences, antenna gain, amplifier configurations, etc.), so it will be needed to equalize the signal power from each spectrum sensor so that the reconstruction does not distort the original signal.

Receiver frequency response equalization. The receivers apply an antialiasing filter prior to sampling the signal. As an example, Figure 7.3(a) shows the final magnitude and phase response of the FIR (Finite Impulse Response) filter applied in the RTL-SDR-v3 [62] receiver in the frequency range of interest (using the maximum sampling rate available). In the overlapping ranges (Figure 7.3(b)) we aim to compare and fuse contributions from different receivers, so it will be necessary to invert the filter response to better reconstruct the signal. In case the receiver implements a Finite Impulse Response (FIR) filter the phase response can be considered just a delay in the time domain. If not, a complementary phase correction at this stage will make sure that the final phase response is linear with the frequency.

Phase matching. Once the receiver filter response in the receiver has been equalized, and the signals received in the overlap areas from the non-coherent receivers have been corrected in time and frequency, the phases of the signals in this overlap areas have to be matched. The phase difference may be caused by different reasons. As the final synthesized signal from the local oscillators is generated by a series of multipliers and dividers, without a PLL on the received signal, we cannot guarantee the value of the local oscillator phase, even in shared-local-clock scenarios. So we will not have a *phase synchronization of the local oscillators*. This will be seen as a relative phase rotation between the signals (a constant phase difference in the same frequency components from the signals). We may also find a *subsample-delay phase change*, as we cannot control the exact sampling instant among all the signals (even with synchronized clocks). Because of that, we may still find some additional subsample delay that can be interpreted as a linear phase change. Also some *different phase contribution during combination* may appear. If there is still some linear-phase response in the chain (some delays unaccounted for, other filters, etc.) that has not been completely corrected, as the final phase integration of the

contributions reflect different bands, we may find an additional separation of the phases. To understand this difference, we can see the distance between the phase responses (red and blue straight lines) in the overlap area in Figure 7.3(b). This height is proportional to the separation of the tuning frequencies (f_{c1}, f_{c2}).

7.3. Collaborative Signal Reconstruction

In this section we present our methodology to collaboratively reconstruct a signal in the backend using partial representations captured by different receivers.

The proposed methodology only assumes the following signal properties to achieve a correct collaborative signal reconstruction in the backend: (1) The Signal-to-Noise Ratio (SNR) must be good enough in order to detect a potential packet on the overlap area. (2) The signal must allow a clear cross-correlation maximum computation in the overlap area to estimate the coarse time synchronization. (3) The signal must occupy a continuous portion of the spectrum with no gaps bigger than the overlap area, allowing the merge using that portion of the signal. (4) The channel response in the overlap area must be similar for both receivers.

The overview of the signal reconstruction methodology is shown in Figure 7.4. In the previous section, we discuss the case of two signals, as the case of more signals can be regarded as an inductive extension (adding any additional received signal to the previously reconstructed one). Our solution does not make any assumption on the signal besides the frequency range and maximum expected length of the packets. The architecture consists of four different blocks: Calibration, Correction, Estimation and Recombination. The *calibration block* computes the intrinsic parameters of the receivers such as the absolute frequency offset (Δf_a). The *correction block* is responsible for modifying the

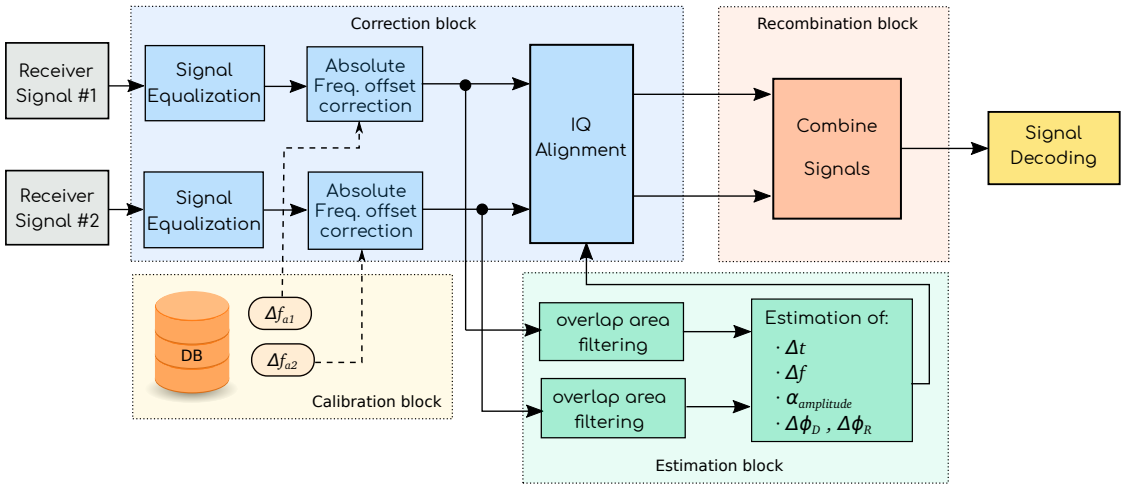


Figure 7.4: Signal reconstruction workflow.

signals according to the parameters provided by the estimation block. The *estimation block* computes the parameters to synchronize both signals as time offset (Δt), relative frequency offset (Δf), amplitude ratio (α) and phase offset ($\Delta\phi$). All these parameters are computed using only the overlap area, where the received signals share a common frequency range. Lastly, the *recombination block* takes two signals already synchronized and fuses them to generate a signal with the specific bandwidth required.

For the simplest model of two different receivers (more receivers could be considered sequentially once the first two portions have been integrated), let us consider the signal received on the overlapping area. From a complex signal transmitted originally as

$$x(t) = \Re\{s(t)e^{j2\pi ft}\} \tag{7.3}$$

the received signals $x_r(t)$ can be modeled like this to account for time, amplitude and phase differences in the receivers, assuming that the frequency deviation Δf is actually an error induced by the local oscillator, and the channel transfer function is flat.

$$x_1(t) = \Re\{s(t)e^{j2\pi ft+\phi_1}\} + \sigma_1(t) \tag{7.4}$$

$$x_2(t+\Delta t) = \alpha[\Re\{s(t)e^{j2\pi(f+\Delta f)t+\phi_2}\}] + \sigma_2(t) \tag{7.5}$$

We will define $s_r[n]$ as the complex baseband, discrete-time signal provided by the receiver r, which have been obtained by tuning the receivers to the frequencies $f_{c,r}$ and a sampling rate $f_{s,r}$ in accordance with the receiving bandwidth BW_r.

As the original signal $s(t)$ is not available, our goal is to find the parameters so we can correct the *differences* between both signals in the overlapping area, and subsequently apply these parameters to correct the whole signals $s_r[n]$.

7.3.1. Calibration Block

There are some intrinsic parameters in the receivers that do not change substantially over time. One of these parameters is the frequency offset on the receiver caused by the internal oscillator. Nowadays, most receivers integrate a Temperature Controlled Local Oscillator (TCXO) to mitigate the fluctuations of the internal oscillator due to temperature changes.

This block is responsible for estimating, in a calibration process, the absolute frequency offset (Δf_a) of the receiver using a known and precise signal. We rely on LTESS-Track [3] in order to estimate the frequency error of the receiver. LTESS-Track takes advantage on the synchronization signals transmitted by LTE base stations as reference in order to provide a frequency offset estimation with sub-ppm (parts-per-million) accuracy. The

absolute frequency offset (Δf_a) for each receiver will be used to compensate the frequency error between the receiver and the nominal frequency of the transmitter, providing the *coarse frequency synchronization* of the system. Notice that this calibration process is run only once per receiver and therefore the Δf_a value can be used as long as the receiver is not replaced.

Since Δf_a may vary slightly due to the tolerance error of the local oscillator, the variance on the calibration, environment factors [3], and the intrinsic uncertainty of real oscillators, a *fine frequency synchronization* is needed as is explained in Section 7.3.3. The calibration process is needed since two given receivers may have a significant frequency error (for instance 100 Parts Per Million (ppm)). As the reconstruction process matches the frequency components between the sensors, but not to the original f_c, the final reconstructed signal will show the same frequency error shift.

7.3.2. Correction Block

The correction block receives I/Q data directly from the receivers. The Analog-to-Digital Converters make use of an antialiasing [98] filter prior to obtaining the samples that will be the discrete representation of the signal. As an example, the magnitude and phase response of the FIR filter applied in the RTL-SDR v3 is shown in Figure 7.3. The first step is to equalize the signal to correct the possible changes made by this filter, in such a way that the rest of the reconstruction steps can use a signal without distortions (or reasonably minimized). It comes without saying that for this process to work the filter response must be known for all the different devices involved in the reconstruction task. Recalling that to perform the signal recombination we will use the overlap frequencies between the two receivers, we see the band of the common area in every receiver has been modified by a different band of the filter, as Figure 7.3 shows. This difference makes it clear why this effect must be handled and minimized.

The next action is to correct both signals in frequency given the $\Delta f_{a,n}$ provided by the calibration block. The goal is to obtain a signal as close to the raw ideal as possible so to perform a precise synchronization in the *estimation* block.

The *I/Q alignment* block then performs the operations needed over one of the signals using the estimated corrections provided by the *estimation* block that is disclosed in the next subsection.

7.3.3. Estimation Block

The *estimation* block receives the signals already equalized as input. The discrete-time signals present in the overlap areas (o_1 and o_2) are extracted from the received signals (s_1 and s_2) by extracting the higher and lower frequencies of the signals respectively as Figure 7.4 shows. The overlap area represents the common frequencies in both receivers

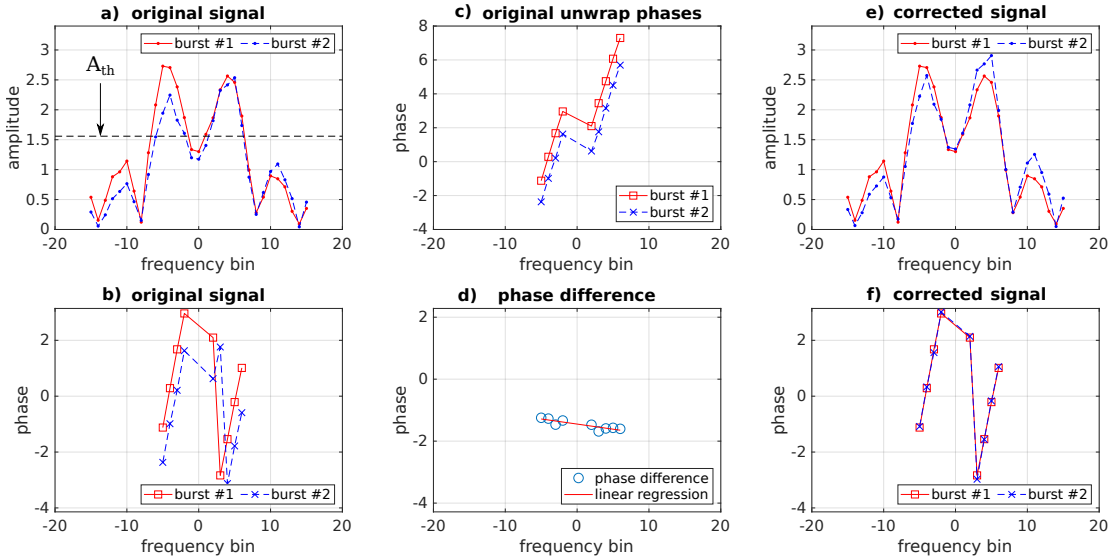


Figure 7.5: Amplitude and phase representation of the original bursts (left), phase difference estimation (center) and bursts already aligned (right).

where all the synchronization operations will take place. The width of the area is directly related to the receiver bandwidth (BW_r) and the tuning frequencies of the receivers (f_c, r).

First, the **coarse time synchronization** (Δt) is computed by cross-correlating the overlap areas of both signals in the time domain using *chunks* of data of certain duration as Eq. 7.6 shows. As at this point we cannot guarantee that the I and Q components from both signals are matched (as there might be a phase rotation between them, ϕ_n values are unknown), this operation is calculated over the norms. Considering Δt in number of samples:

$$\Delta \hat{t} = \underset{\tau}{argmax}(|o_1| \star |o_2|)[\tau] \tag{7.6}$$

We assume the signals s_r have a SNR high enough to detect the presence of a transmission. Other characteristics from $s(t)$, such as the modulation scheme, are not needed to estimate the corrections. For every *chunk* of data already synchronized in time an energy detector is applied in order to get the interesting time ranges of the signal called *bursts*. Then, a *burst* is defined as a time range of the signal, considered in the overlap area, where a potential packet (P) can be and therefore where the synchronization parameters will be estimated. A couple of bursts (b_1 and b_2) are shown in Figure 7.5-a). At this point is where the synchronization parameters needed for this potential packet (P) can be estimated, in order to combine and match the signals.

Secondly, the **power normalization** of the bursts is done by computing the scale factor α. The original energy in the overlapping area is nominally the same, so we can

estimate the α parameter as the square root of the power ratio of the two signals:

$$\hat{\alpha} = \sqrt{\frac{E[|b_2|^2]}{E[|b_1|^2]}} \tag{7.7}$$

Third, the **relative frequency shift** (Δf) is estimated providing the *fine frequency synchronization* in the system. To compute the frequency misalignment between both signals we may use the discrete Fourier transform of the signals. Given that the signals have not been distorted we may use their frequency representations to find the best match. As the values for ϕ_r are still unknown, once again it is needed to use the norm of the functions to calculate cross-correlation. Given that $b_r[n] \xrightarrow{DFT} B_r[k]$, and by cross-correlating the overlap areas in the frequency domain:

$$\Delta\hat{k}_r = \underset{\theta}{argmax}(|B_1| \star |B_2|)[\theta] \tag{7.8}$$

It is worth noting that the bursts have been extracted so to contain some guard time before and after the packet P. This helps to reduce spurious components in the Discrete Fourier Transform (DFT), as it behaves as a windowing in time. The relation with the real frequency is then, given the length of the DFT as M:

$$\Delta\hat{f}_r = \frac{f_s \Delta\hat{k}_r}{M} \tag{7.9}$$

From this point forward, let us define $B_2'[n]$ as the frequency-corrected equivalent of $B_2[n]$.

Fourth, we perform the **phase difference estimation**. In order to have a coherent recombination process of the signal, the phases of both bursts must be matched. This step is important since phase alignment represents both a matching of I and Q components from each $s_r[n]$ (neutralizing the relative phase rotation between them), and a fine (subsample) time synchronization. Figure 7.5-b) represents the phase of the bursts in the frequency domain. Notice that we only take those bins whose amplitude values are above a certain threshold (A_{th}) since low amplitude values usually introduce noise in the phase component that can lead to an erroneous phase estimation.

After that, we can study the phase of the bursts visually, by unwrapping the phases to get a continuous representation of the phase over the frequency bins as Figure 7.5-c) shows. It is clear that phases are not equal and therefore we compute the phase difference over all the frequency components of both bursts and then we obtain the phase difference shown in Figure 7.5-d). This difference should take care of the local phase conditions ϕ_n, possible fine time adjustments and phase rotations. As we saw in section 7.2.2 the expected difference is a linear function of the frequency.

Thus we will first calculate the phase difference:

$$\Delta\phi[k] = \angle B_1[k] - \angle B_2'[k] \tag{7.10}$$

And given that the k elements are ordered such to keep the DC component in the center, then the linear regression over $\Delta\phi$ could be expressed as:

$$\Delta\hat{\phi}[k] = \Delta\hat{\phi}_R + k\Delta\hat{\phi}_D \tag{7.11}$$

where $\Delta\phi_R$ can be regarded as a rotation that applies to all the frequency components, and the slope $\Delta\phi_D$ as a time delay, with a subsample resolution.

Figure 7.5-d) shows the value of the burst phase difference and its linear regression. Figure 7.5-e) and Figure 7.5-f) show the bursts normalized in amplitude and synchronized in phase after applying the correction values, $\hat{\alpha}$ and $[\Delta\hat{\phi}_D, \Delta\hat{\phi}_R]$ respectively.

We note that the estimation block does not have the notion of *packet*. In particular, this block performs a signal-level synchronization only using the bursts (portion of signal in the overlap area). The packet-level synchronization and its detection are performed by the decoder ("Signal decoding" block in Figure 7.4), which is independent from the signal reconstruction scheme.

7.3.4. Recombination Block

The *recombination* block takes as input the signals $s_r[n]$ of bandwidth BW_r, which were tuned to in $f_{c,r}$. The first step is to resample both signals to increase the sampling rate. The minimum upsampling is that which obtains a sampling frequency enough to contain the full reconstructed signal. After that, both signals are relocated so their components of f_c match the equivalent f_c of $s(t)$, by shifting $f_c - f_{c,1}$ and $f_c - f_{c,2}$ respectively. The frequency components on the overlapping area are halved, as that area will receive the energy from the two signals. After that, both signals are summed and downsampled if needed to match the specific bandwidth desired and therefore the reconstructed signal is complete as Figure 7.4 shows.

7.3.5. Decoding

The last step of the process is to decode the signal that has been reconstructed. Decoding the signal provides a good indicator that the reconstruction is done properly in the backend. It is worth mentioning that the *signal decoding* block (Figure 7.4) is not aware about the origin of the signal, e.g., the *decoder* does not know if the signal either has been reconstructed by our method or has been collected by a regular receiver.

7.4. Evaluation

This section describes how we evaluate our methodology for signal reconstruction in real scenarios. We use *real signals* to evaluate our collaborative signal decoding, specifically we use Mode-S uplink [67, 79]. These signals are sent by aircraft and uses 1030 MHz frequency for transmissions and 4 MHz of signal bandwidth. The receivers used for the signal reconstruction are the well-known RTL-SDRv3, whose maximum available signal bandwidth is 2.4 MHz. Therefore this is a very complete set-up for testing and evaluating our collaborative signal reconstruction methodology in a real-world and complex scenario (amplitude-phase modulation, signal sent from moving transmitters, different SNRs, etc.). In the following subsections, Mode-S and the experimental setup are explained and after that the most important evaluation results are shown.

7.4.1. Real use case: Mode-S uplink

Aircraft are constantly sending air traffic signals that contain useful information for the safety and optimization of the flight routes. Several initiatives as OpenSky [51], Flightradar [69] or Flightaware [68] have already deployed low-cost sensors based on RTL-SDR devices for collecting avionic signals whose signal bandwidth is smaller than 2.4 MHz, e.g., Mode-S downlink channel. All those platforms could get benefit of our contribution of using non-coherent receivers for decoding signals with a higher bandwidth than 2.4 MHz.

Mode-S [67, 79] is a Secondary Surveillance Radar (SSR) process that allows selective interrogation of aircraft according to the unique address assigned to each aircraft named *ICAO*. The ICAO is a 24-bit field used to identify every aircraft uniquely worldwide. Mode-S uses two different channels, one for interrogating (uplink channel, 1030 MHz / 4 MHz) and another for answering (downlink channel, 1090 MHz / 2.4 MHz). The aircraft replies to interrogations initiated from either a ground station or other aircraft. The aircraft interrogation is performed using the uplink channel and has the capability for selective interrogation of individual Mode-S transponders. This functionality allows to interrogate a specific aircraft as needed, avoiding the response from nearby aircraft and therefore an unnecessary saturation of the channel. The messages sent by the uplink channel are the ones that cannot be decoded currently by one single RTL-SDR due to the signal bandwidth limitation.

Mode-S messages can be 56 or 112 bits long. As Figure 7.6 shows, Mode-S uplink messages consist of a preamble part (used for message detection), a Sync Phase Reversal (SPR) (used for time synchronization), and the payload encoded using Differential Binary Phase Shift Keying (DBPSK) where the ICAO of the addressed aircraft is indicated. Other important piece of information encoded in the payload is the uplink format, reply length, acquisition special or designator id [76,79]. It is worth mentioning that our generic solution proposed in Section 7.3 and the evaluation performed in Section 7.4.3 are not

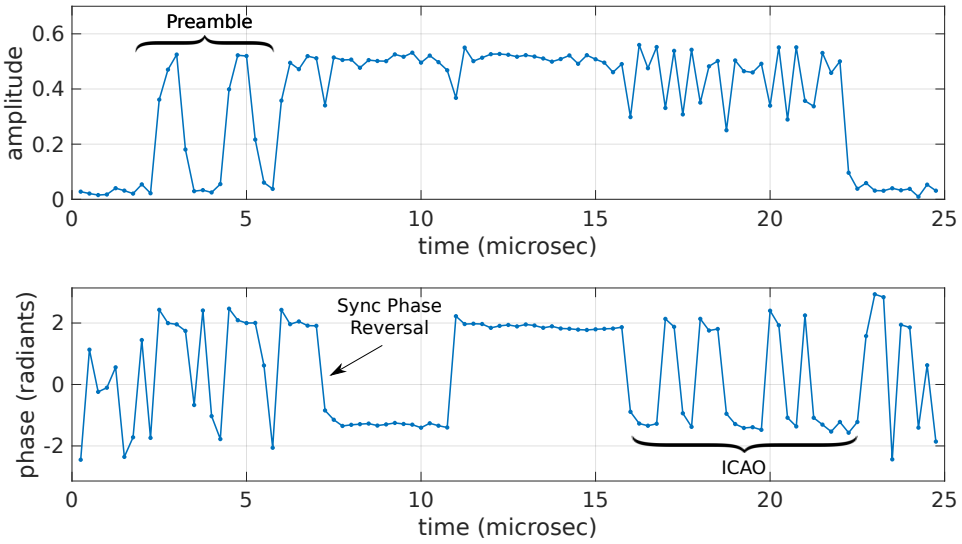


Figure 7.6: Amplitude representation (top) and phase representation (bottom) of a Mode-S uplink packet captured at 1030 MHz @ 4 MHz by an USRP.

aware about the signal specifications explained here. Our solution is able to reconstruct the signal from two different channels without any knowledge about Mode-S uplink signals.

7.4.2. Mode-S Decoder

We have implemented the first open source decoder for Mode-S uplink using an initial and experimental implementation provided by [99]. The decoder has been implemented following the standard described in [76, 79]. Our decoder implements the following features:

- Preamble detection.
- Size packet detection: supports lengths of 56 or 112 bits.
- Sync Phase Reversal (SPR) detection which is located at 4.75 μs from the beginning of the packet.
- Sliding window implemented to analyze data continuously.
- Support input signals of 4 MHz and 8 MHz bandwidth.
- DBPSK demodulator for payload extraction.
- ICAO of aircraft: The last 24-bits of the payload are the parity bits xored with the ICAO address of the aircraft. The Cyclic Redundancy Check (CRC) is computed as specified in [76] and aircraft's address is retrieved.

- Symbol timing recovery implementation since transmitter and receiver are not synchronized.

Our decoder is the first open source implementation[1] and is able to decode properly the 92% of the Mode-S packets detected.

7.4.3. Testbed

The experimental set-up consists of 3 pairs of sensors with two RTL-SDRv3 "Silver" model attached to a Raspberry Pi-v3 with different Automatic Dependent Surveillance - Broadcast (ADS-B) antennas and internal oscillator configurations. The maximum bandwidth supported by the RTL-SDRv3 is 2.4 MHz, therefore every pair of sensors has the mission of reconstruct the Mode-S uplink packets sent at 1030 MHz with 4 MHz of signal bandwidth. In order to do that, both receivers have to tune at *1029.1* MHz and *1030.9* MHz respectively in such a way the overlap area is *0.6* MHz and the total bandwidth covered is 4.2 MHz. Figure 7.7 shows the different configurations tested on every pair of receivers depending on the resources shared:

- Local oscillator and antenna shared ("local oscillator + antenna"): This is the most favourable set-up where we use a clock-shared receiver based on two RTL-SDR[2] that shares the local oscillator (TCXO @ 28.8 MHz). They also share the same ADS-B antenna thought the 8-port active splitter.

- Antenna shared ("Antenna"): This configuration uses two vanilla RTL-SDRv3 receivers without sharing the internal clock and whose only shared resource is the ADS-B antenna thought the splitter.

- Non-shared resources ("None"): This configuration consists of two vanilla RTL-SDRv3 receivers without sharing the internal clock and attached to different antennas that are 1 meter away.

Our results are based on multiple traces collected in Madrid (Spain) where more than 30,000 Mode-S uplink messages were recorded. In order to compare the performance in every configuration we use a baseline simultaneously created by a high-end receiver, Universal Software Radio Peripheral (USRP) Ettus B210[3] that is set at 1030 MHz and covering the whole signal bandwidth (4 MHz). The USRP is also connected to the splitter to share the same ADS-B antenna that the RTL-SDR receivers use. Therefore, the baseline is defined as the number of Mode-S uplink packets correctly decoded with the USRP samples. Since we do not know whether the ICAO encoded in the packet belongs to

[1] http://github.com/openskynetwork/modes-uplink-decoder
[2] https://coherent-receiver.com/products/coherent-receivers/1-supervisory-and-4-coherent-channels
[3] https://www.ettus.com/product/details/UB210-KIT

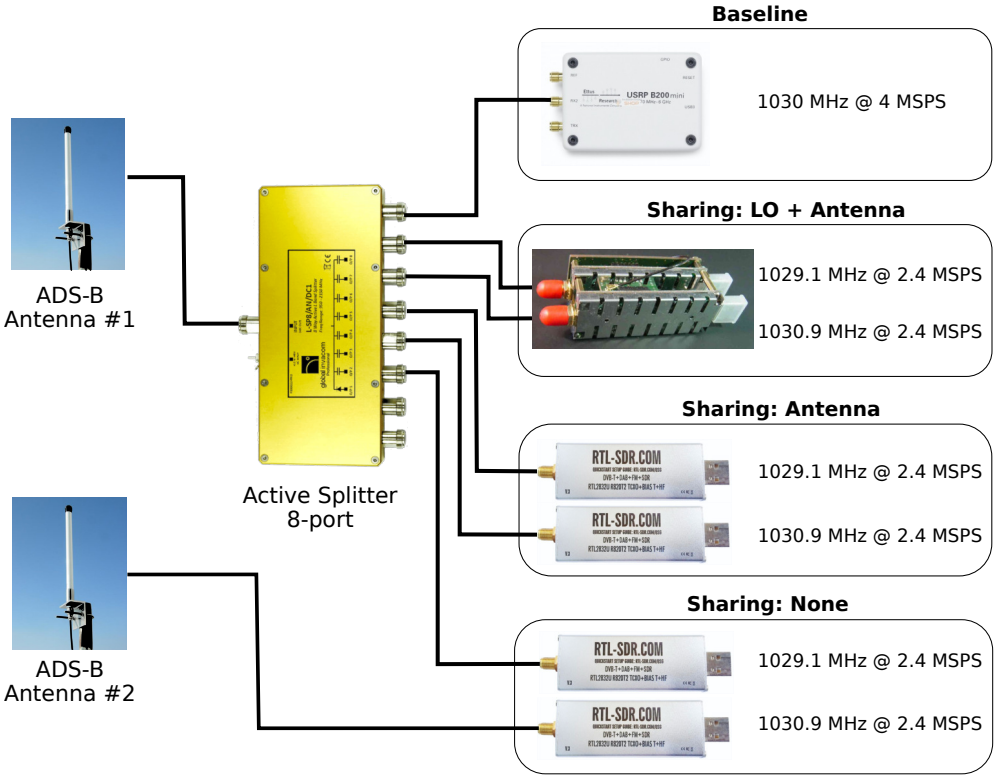


Figure 7.7: Testbed.

a real aircraft, we cross-verify that ICAO was also detected by the OpenSky [51] network in the Mode-S downlink channel at the time and place the trace is collected.

7.4.4. Evaluation metrics

In this section we evaluate the most important parameters of the system and how they can play an important role for the final signal reconstruction and decoding.

As we described in Section 7.3, one of the first synchronization steps is the time alignment between the signals. This step must be successfully performed to estimate the amplitude, frequency and phase offset accurately. All the synchronization steps over the signal are performed on the overlapping frequency area (which is common among the receivers). Therefore the width of the overlap area becomes a significant parameter of the system to be evaluated. The overlap area width optimization will also optimize the effective bandwidth available for every receiver, thus covering wider bandwidths with the same number of receivers. Figure 7.8 shows the estimated time difference between the two signals at 2.4 MHz for different widths of the overlapping area. Note that to illustrate with comparable magnitudes, the y-axis units are samples of the original signal. The smaller the overlap area is, the less signal data we have to perform the time synchronization, and as expected by the time-frequency relation, the coarser time resolution we obtain.

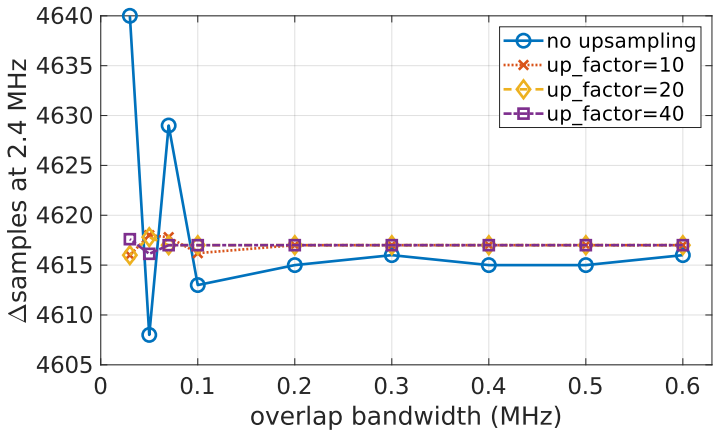


Figure 7.8: Sample synchronization vs. overlap area.

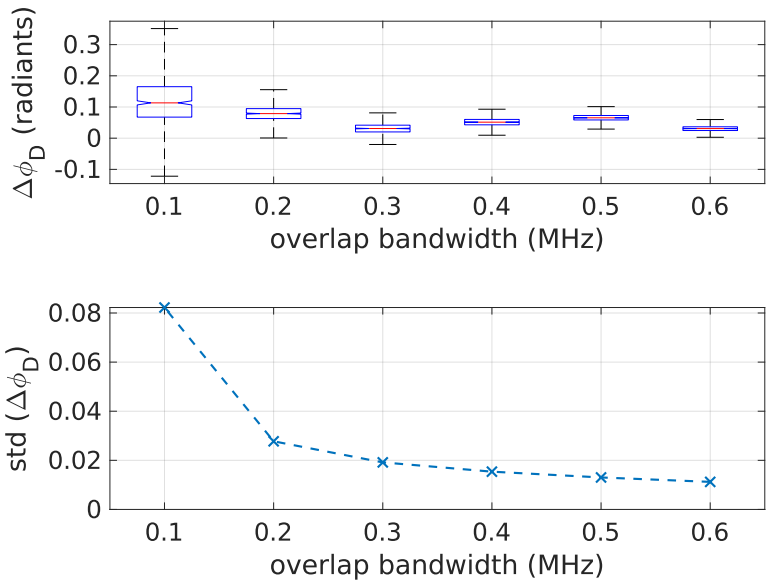


Figure 7.9: Phase delay evaluation vs. overlap area.

To mitigate this effect, and obtain a better time representation of the signal edges, we can upsample [66] the original signal in post-processing. Figure 7.8 shows how for the "no-upsampling" scenario the $\Delta samples$ estimation fluctuates depending on the overlap bandwidth size, and how this behaviour is much more contained in the "upsampling" scenarios. This is an indicator that we need more accuracy in the $\Delta\hat{t}$ since the existence of synchronization errors at sample level could hinder the final collaborative reconstruction of the signal. For higher upsampling factors we can see how the $\Delta samples$ remains stable for overlap bandwidths from 0.3 MHz onwards.

We have also performed an evaluation of the phase estimation of our system that becomes important for two reasons: 1) it gives the sub-sample synchronization among

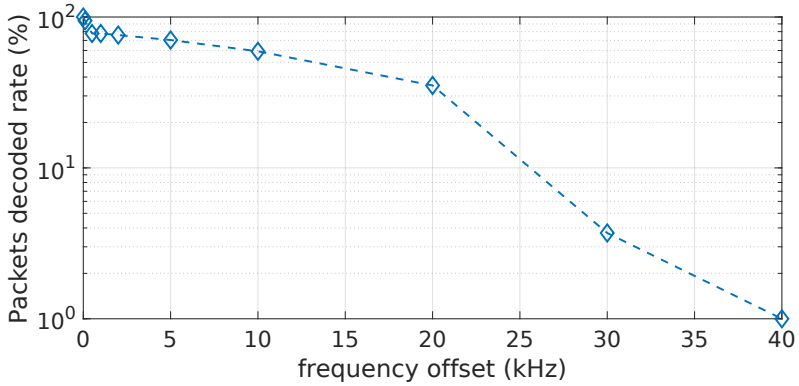


Figure 7.10: Frequency offset evaluation vs. packets decoded rate.

signals and 2) allows us to reconstruct and decode signals with phase coded information. Figure 7.9 (top) shows the phase delay estimation between bursts (potential packets) using different overlap area sizes. We assume that the phase delay estimation among packets will be similar for each pair of receivers once the signal has been corrected in time. For overlap bandwidths over 0.3 MHz we can observe that the phase delay values are contained in the same range with no important fluctuations. However for overlap bandwidths smaller than 0.3 MHz we can observe how the values are spread over a large range, meaning that the estimation is not performed accurately. The smaller the overlap area is, the bigger the standard deviation of $\Delta\phi_D$ appears to be (see Figure 7.9 bottom), due to the fewer data that the system has to properly perform the initial time offset estimation. It is possible this impact the subsequent phase delay estimation as the linear regression might have been computed with fewer data.

As we explained in Section 7.3, the final goal is to decode a signal that has been reconstructed collaboratively in the backend. Therefore, the packets decoded rate is a metric that must be evaluated in our system. As we mentioned before we use the Mode-S

Table 7.1: Different SDR receiver models and their frequency stability of LO.

Receiver	Local Oscillator stability	Frequency error at 1030 MHz
RTL-SDR-v3	± 2 ppm	± 2,1 kHz
Ettus B210 (USRP)	± 2 ppm	± 2,1 kHz
HackRF One	± 20 ppm	± 20,06 kHz
Adalm-Pluto	± 25 ppm	± 25,75 kHz
RTL-SDR (blue edition)	± 60 ppm	± 61,80 kHz
RTL-SDR (black edition)	± 120 ppm	± 123,60 kHz

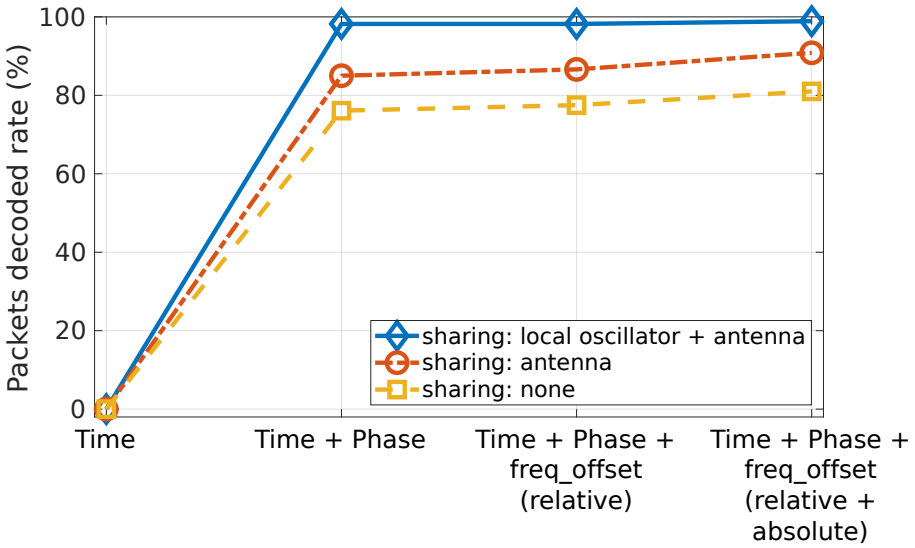


Figure 7.11: Mode-S uplink packets decoding rate in different scenarios (USRP used as baseline).

uplink decoder that we implemented (see Section 7.4.2) to evaluate the decoding of the signal. Figure 7.10 shows the performance of the message decoding depending on the frequency offset among signals. In essence, we manually introduce a frequency offset in one of the signals and we check how the decoder performs in these scenarios. We can observe how the packet decoded rate is close to 100% when the frequency offset is below 1 kHz. The higher the frequency offset, the smaller the decoding packet rate. This proves the importance of a proper estimate for the frequency offset among signals caused by the local oscillator of the different receivers (see Table 7.1).

Figure 7.11 shows the decoding packet rate for different experiment configurations (described before) and different signal synchronizations steps (explained in Section 7.3). The gentle case ("local oscillator + antenna") shows that is able to yields a 98% ratio of properly decoded packets once the time and phase are properly aligned. No improvements are visible even if the relative frequency offset is corrected since both receivers are sharing the local oscillator and therefore their relative frequency offset is negligible. For the shared antenna ("antenna") scenario using non-coherent receivers we may observe that it reaches 85% of decoded packets ratio, and in this case correcting the frequency offset helps to reach the 90%. For the most unfavourable and challenging case ("none"), in which antennas are not shared and non-coherent receivers are used, we are able to reach more than 80% of correct packet decoding ratio by applying all the signal alignment steps as Figure 7.11 shows.

7.4.5. Network Bandwidth

In order to perform the signal reconstruction in the backend, each IoT spectrum sensor needs to send I/Q data through the Internet connection. Our solution is based on a crowdsourcing approach where spectrum sensors are distributed among different locations and therefore they may use different Internet connections. Nevertheless the network bandwidth used for each IoT spectrum sensor is a concern due to the I/Q data transmission. Figure 7.12 shows how one single receiver sampling at 2.4 MSPS with 8-bit resolution would need to transmit $\sim$ 39 Mbit/s, or $\sim$ 15 Mbit/s if data is compressed with a generic DEFLATE [100] algorithm. Going further, packet detection is performed on the spectrum sensors, and then only small portions of I/Q data are sent where potential packets can be decoded. This alleviates the network bandwidth used by each spectrum sensor. Assuming a detection packet rate of 700 packets/second and knowing that a Mode S packet is 40μs long [79], the network bandwidth used for each IoT spectrum sensor decreases to 0.89 Mbit/s (0.35 Mbit/s compressed-data), which is affordable over most domestic Internet connections.

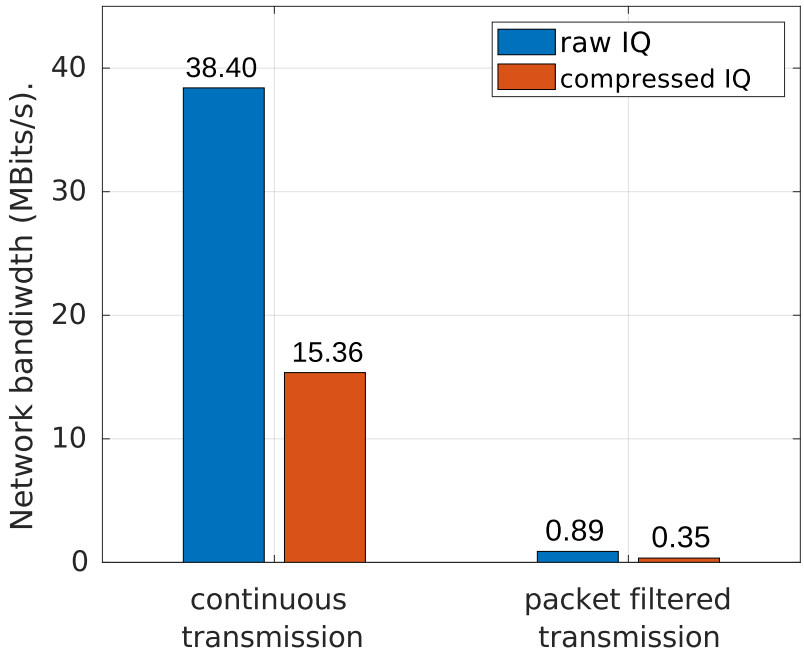


Figure 7.12: Network bandwidth used by one single sensor to deliver I/Q data to the backend.

7.5. Related work

Monitoring the electromagnetic spectrum at large scale is more challenging than traditional approaches. Several platforms use expensive and specialized hardware for sensing the spectrum at large scale such as DARPA's Spectrum Challenge, Google Spectrum Database [23] and Microsoft Spectrum Observatory [16]. Our solution relies on low-cost receivers for sensing the spectrum distributively. In [101], the authors used a collaborative spectrum sensing approach for detecting unauthorized transmissions using Power Spectral Density (PSD) data. In this work, we propose to work with I/Q data what allows us to reconstruct and decode signals. As I/Q data can imply a deluge of information sent to the backend, we expect that it can be applied either in presence of very high uplink bandwidth or, most likely, for a short period of time, with the objective of decoding specific messages.

The authors of [102] proposed a method to receive and decode weak signals, by applying maximum ratio combining in the cloud. Previous paper used also I/Q data, yet they tuned all receivers in the same frequency and assumed that the receiver can sample the whole signal of interest. Our problem space is complementary and requires dedicated solutions, as we deal with signals of higher bandwidth that the one single receiver can provide, a problem that emerges with the advent of low-cost spectrum sensors and has been so far largely ignored in the context of signal decoding.

Past work that considered wider bands focused on the problem of extracting information about the spectrum usage and which bands are more interesting to monitor (e.g., applying a multi-armed bandit game [85]). Some other work proposed instead to use the sparse Fourier transform to decode signals [103]. However, they did not look at the problem of decoding signals that are of larger bandwidth than the single receiver, and required access to dedicated USRP hardware rather than low-cost commercial off-the-shelf hardware as in this work.

Different works [104, 105] had the need of using coherent-receivers to achieve a precise signal synchronization over different channels, e.g., passive radars using low cost receivers. In [96], authors used coherent-receivers and a GPS Disciplined Oscillator (GPSDO) in order to synchronize the ADC clock in each channel. In contrast with these works, our solution uses non-coherent receivers and relies on the job done in the backend for the final synchronization and decoding of the signals.

7.6. Discussion

We have studied the problem of collaborative sampling the spectrum using low-cost and distributed IoT spectrum sensors to decode a wider signal bandwidth than the one of the single spectrum sensor. We have introduced the main concept of dividing the wideband signal in sub-frequency channels with a small overlap area, and addressed the main challenges to reconstruct the wideband signal in the backend. We have then presented a distributed architecture with the needed synchronization steps in order to align the sampled I/Q signals from independent receivers and decode the whole signal in the backend. We have tested and evaluated our solution with real signals (Mode S) sent by aircraft that are used for collision avoidance. We have implemented a Mode S uplink decoder that is released as open source[4]. Our results show that is feasible to use low-cost and distributed non-coherent spectrum sensors for decoding wideband signals, getting more than 80% of packets correctly decoded, and reaching nearly 100% of packets decoded when receivers share the local oscillator. Collaborative signal decoding using low cost IoT spectrum sensors has been successfully proved and can enable multiple applications for those existing systems that deploy low-cost receivers such as RTL-SDR at large scale.

[4]http://github.com/openskynetwork/modes-uplink-decoder

"Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning"

Winston Churchill (1874 – 1965)

8 Conclusions

In this thesis, we have presented ElectroSense, a collaborative and crowd-sourcing initiative for collecting and analyzing the Electromagnetic (EM) spectrum by using low-cost RF IoT sensors as well as more expensive RF spectrum devices to monitor the spectrum at large scale. We have identified the main research problems to enable smart and collaborative algorithms among low-cost spectrum sensors, which are challenging to solve due to the limitations of their components, both in the RF front-end and in the embedded board capabilities.

Chapters 2 and 3 of this thesis have described in detail the advantages and benefits of the ElectroSense framework, where we mainly focused the work on the sensor side by providing different capabilities and data pipelines (PSD, I/Q and decoding) that are used depending on the final application desired. Sensors are able to monitor the spectrum in real time and send the data to the backend by using mechanisms to reduce the uplink network bandwidth from every IoT RF sensor. In Chapter 4 we have introduced a precise and fast frequency offset estimator (LTESS-track) for SDR devices, which relies on the synchronization signals transmitted by LTE base stations as reference. One of the key aspects to enable the collaborative approach among sensors is to properly characterize the frequency offset of the Local Oscillator (LO) of the receiver. Our approach, LTESS-track, is 10 times faster than the best open source tools currently available, and is able to provide a new estimate every second.

In the Part III of the thesis we have investigated real scenarios where low-cost IoT RF sensors work collaboratively to reach a specific goal. In Chapter 5 we have proposed new algorithms for Time-of-Arrival (ToA) estimation for aircraft, glider or drone signals that can achieve nanosecond-level precision using real-world signals (ADS-B) captured with the cheapest SDR receiver available on the market. We have shown that the ADC resolution (dynamic range) of the RTL-SDR is the limiting factor for the achievable ToA precision. We have demonstrated that ToA estimations are worst when either the signal strength packet is clipped or drowned into quantization noise. Our best algorithm, *CorrPulse/S*, achieves sub-nanosecond prevision for ToA estimation for packets that are received with

signal strength well within the dynamic range of the receiver.

We have studied and evaluated the main research challenges for collaborative narrowband signal in Chapter 6. We have presented a distributed time-multiplexing mechanism to sample the spectrum in a coordinated fashion that exploits the similarity of the radio signal received by more than one RF receiver in the same radio coverage. We have addressed the strict time synchronization required among spectrum sensors to reconstruct the signal. We have also overcome errors in the timing information in the presence of noise sources and decode the data in the backend using low-cost RF spectrum sensors. Our approach is able to reconstruct the signal based on raw I/Q samples received by different low-cost receivers solving problems such as the sub-microsecond level time synchronization required between independent sensors and reducing the uplink network bandwidth of every sensor.

Finally, in Chapter 7 we have presented the collaborative wideband signal decoding performed in the backend using non-coherent receivers. To the best of our knowledge, we are the first proposing this using I/Q data received from very low-cost RF receivers and commodity hardware. We have proposed a methodology to enable the signal reconstruction in the backend by multiplexing in frequency a certain number of non-coherent spectrum sensors in order to cover a signal bandwidth that would not otherwise be possible using a single sensor. Our method does not use the knowledge of the modulation scheme to enable the signal recombination. We have demonstrated and evaluated our approach with two non-coherent receivers which collaboratively sample a real aviation signal (Mode-S uplink channel) of almost twice the RF bandwidth of each receiver.

In summary, we have proposed ElectroSense, a crowdsourcing framework for spectrum monitoring based on 3 main ideas: i) a sensing architecture that provides different spectrum data pipelines, ii) low-cost and software-defined IoT spectrum sensors to enable large-scale deployments, and iii) signal processing performed in the big data architecture. By using ElectroSense, we have successfully proved that complex tasks such as ToA estimation or collaborative signal decoding are affordable using very simple and low-cost RF receivers thanks to the signal synchronization mechanisms and algorithms proposed in this thesis.

Appendices

A Measuring Spectrum Similarity in Distributed Monitoring Systems

Measuring the similarity between data spectra collected by independent sensors at different locations and connected to a backend over the Internet could allow us to explore intelligent collaborative decisions. Sensors may collaborate to detect anomalies if they learn that they are in range of the same transmitter in that specific frequency band. Also, sensors may decide to monitor a bandwidth larger than the one of the single sensor (with each node sampling on a 2 MHz band only partially overlapped with other sensors in range) and then jointly reconstruct the spectrum more efficiently in the backend.

In this appendix we demonstrate and implement a system architecture able to determine the similarity of the radio signal received at nearby locations by different sensing node. In order to realize this vision, there are three fundamental challenges that need to be addressed with low-cost spectrum sensors: 1) time synchronization, 2) rapid decision making and 3) minimize network bandwidth.

A.1. Measuring Spectrum Similarity

The methodology proposed to measure the spectrum similarity is composed by three main concepts.

First, the sensors that are located in the range of the same transmitter have a coarse synchronization in time using Network Time Protocol (NTP). The sensors can then read spectrum data with synchronization with millisecond precision over the public Internet, and better that one millisecond precision in local area networks (as measured in ad-hoc experiments). In our architecture, each sensor synchronizes its internal clock with NTP server every 60 seconds. A finer synchronization could be achieved using the time reference provided by an additional GPSDO module, as explained in Chapter 6. Yet, this may not be available in all the spectrum sensors, as they are run by users of the community, with they own hardware.

Second, each spectrum sensor activates the sampling thread for a fixed period, and

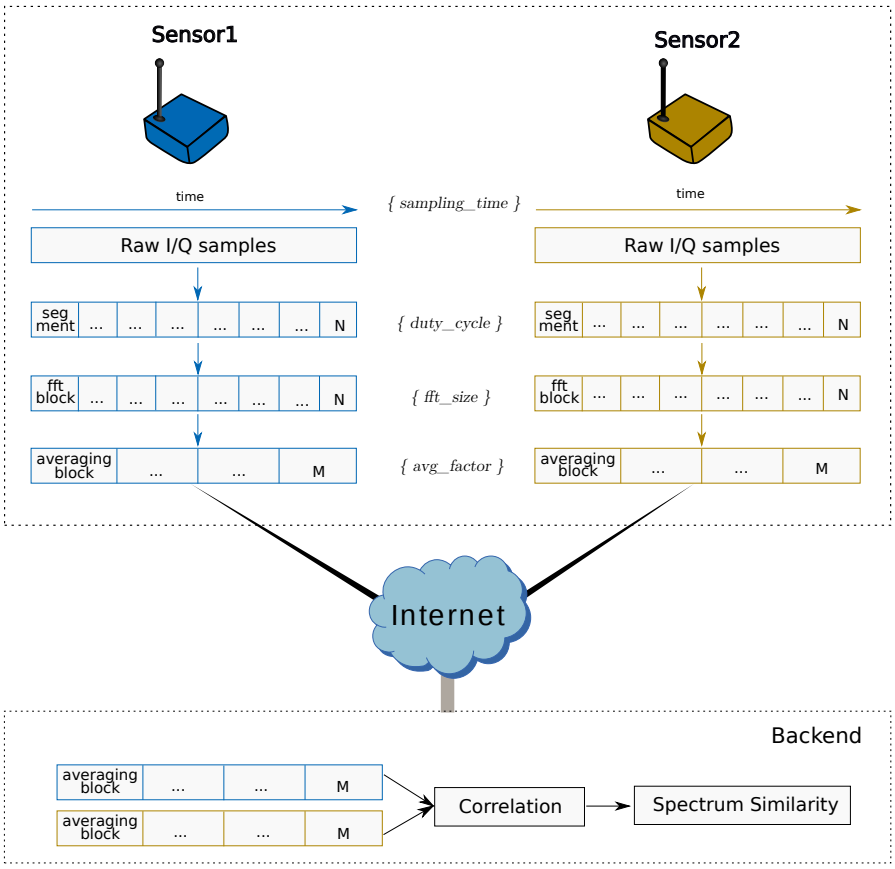


Figure A.1: Technique for computing spectrum similarity between two sensors.

then executes the other threads. This period is the same for each sensor. The total time to execute the entire set of operations should be such that the embedded machine can send spectrum data with the minimum delay (i.e. there are no other tasks pending).

Finally, in order to reduce the network bandwidth used, some operations need to be computed on the sensors. Sensors should decide the parameters such as averaging over larger set of Fast Fourier Transform (FFT) blocks, if the bandwidth should be reduced further. This however comes at the cost of additional delay in the decision making. In the following section we describe this problem in details.

A.1.1. Spectrum Correlation

We perform signal correlation analysis in the backend to understand the similarity in the spectrum data collected by different sensors. Figure A.1 shows the overview of the architecture that is responsible to compute the spectrum similarity (it is presented for two sensors, but it could be generalized to any number of sensors). First, the same center frequency and *sampling_time* is set in both sensors. During this sampling time the sensors continuously read the spectrum. After that, the raw data spectrum is split in N segments of *duty_cycle* long each one. For every segment, the FFT is computed

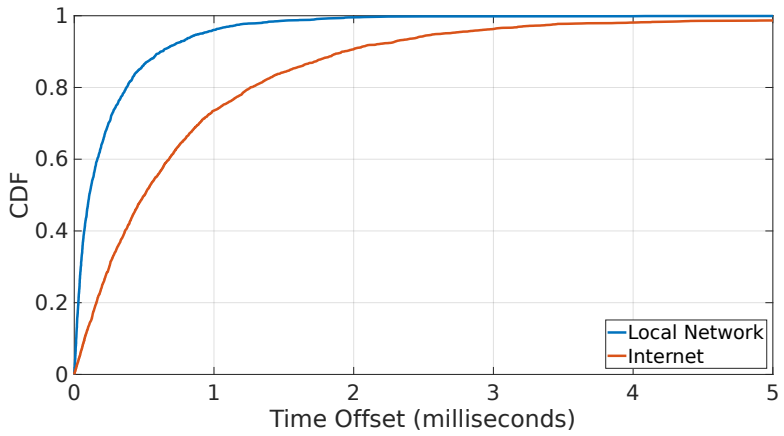


Figure A.2: Time offset between sensors in LAN and Internet.

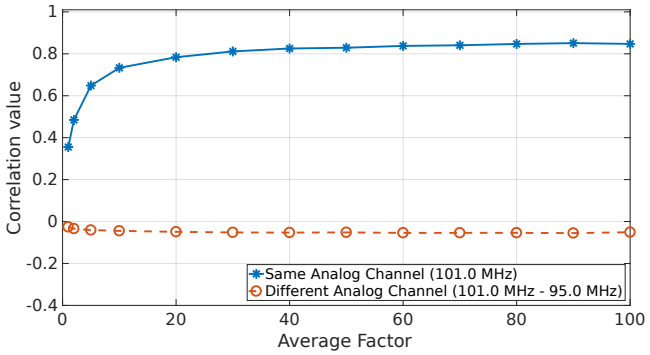
according to the *fft_size* (which defines the number of bins). Each *fft_block* has the spectrum information of 2.0 MHz bandwidth channel. Therefore, the FFT blocks contain frequency information that is averaged using *avg_factor* blocks to produce M averaging blocks. These averaging blocks are sent to the backend for every sensor involved in the similarity spectrum evaluation. Tuning the different values of the system (*sampling_time*, *fft_size* and *avg_factor*) the amount of data sent to the backend (bandwidth used) can be modulated depending of the needs.

The backend computes the Pearson correlation coefficient using the averaged FFT blocks received from every sensor. The correlation is computed for every pair of averaged FFT blocks coming for different sensors. Then, the average is computed using the correlation output for every averaging block. The system evaluates this value to determine if the sensors are reading similar spectrum and if it is possible to enable the collaborative approach between them.

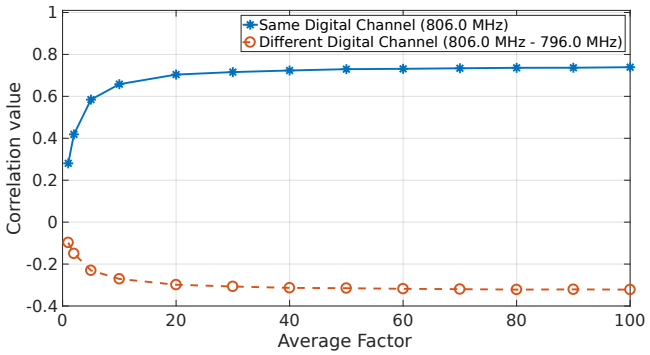
A.2. Evaluation

This section presents the experimental evaluation of our architecture. The aim is to determine the minimum network bandwidth and time needed to achieve a good signal correlation for analog and digital signals. For analog signals we choose Frequency Modulation (FM) and for digital signals we use LTE. The FM radio in Spain occupies the range from 87.5 MHz to 108 MHz. Each FM radio channel uses 200 kHz of bandwidth to transmit the signal. For LTE, we monitor three channels of 10 MHz bandwidth that are available in Spain at center frequencies equal to 796, 806 and 816 MHz.

Sampling Process' Synchronization. The synchronization between sensors is an important issue to achieve a good correlation between the signals. In our experiment, the sensors use Ethernet connection through Internet to synchronize their internal clock through NTP. This technology provides us an accuracy of 0.1 ms in a local network



(a) Analog Signal.

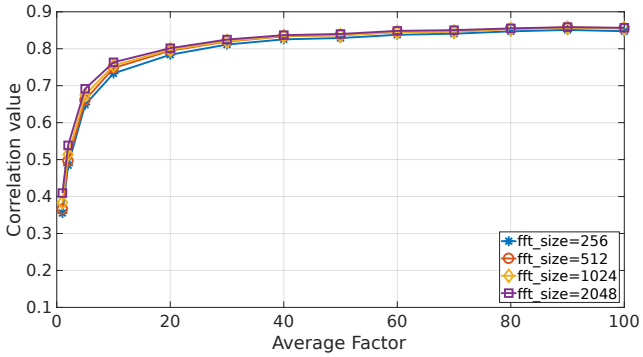


(b) Digital Signal.

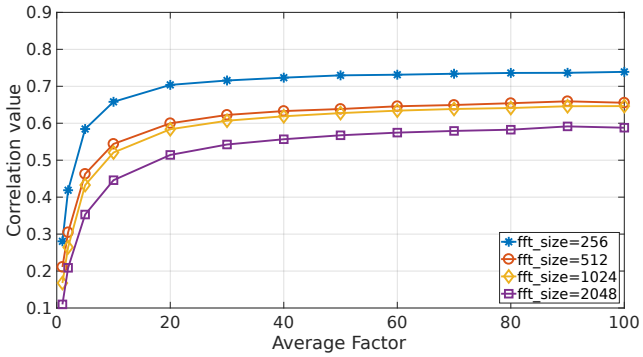
Figure A.3: Analog and digital signal correlation between sensors with different averaging (fft_size=256 and duty_cycle=2ms.)

scenario and around a millisecond over Internet. The backend is responsible to define the precise moment when the sensors start to sample at the same time. This operation provides a coarse synchronization due to the delays added by the network and the operative system. Figure A.2 shows the time offset between sensors when they are in the same local network or connected through Internet. The 90% of the time the sensors have a time offset lower than 2 milliseconds when they are connected through Internet. In our solution we are interested to enable the collaboration among sensors connected through Internet (no just in the same local network). This provides a lower bound for the amount of data to be transmitted from the sensors for spectrum similarity analysis.

Spectrum Correlation. The spectrum correlation value is the metric used to determinate if the sensors are receiving a similar signal in the same frequency at the same time. The data provided by each sensor covers 2.0 MHz of bandwidth. The spectrum resolution in this bandwidth is determined by the *fft_size* used. We run experiments with *fft_size=256*, which implies that each *fft_block* of 2.0 MHz contains 214 frequency bins. We also set duty_cycle equal to 2 ms. Figure A.3(a) shows the correlation between the analog signal acquired by two sensors that are 3 meters away. In the case that the



(a) Analog Signal.

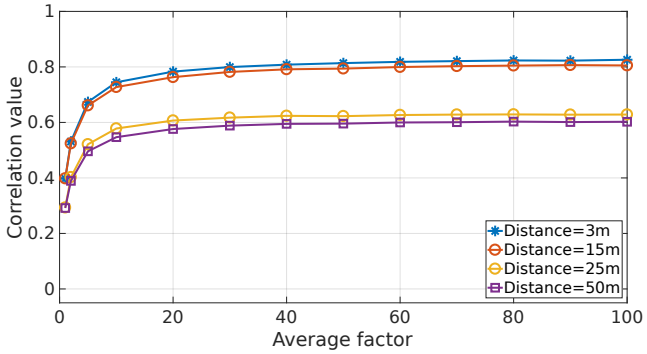


(b) Digital Signal.

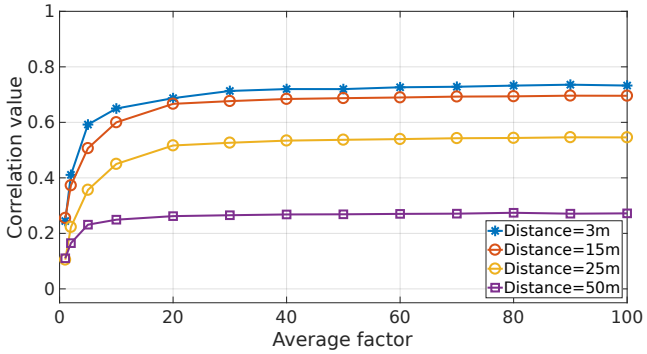
Figure A.4: Analog and digital signal correlation (same frequency band) between sensors with different *fft_size* (duty_cycle=2ms.)

sensors are set to the same frequency band (f_c=101 MHz) the correlation value reaches 0.7 for average factors higher than 10. Note that with a avg_factor of 10, the average is computed every 20 ms. The correlation keeps stable and close to 0 in the case that two sensors are set to a different frequency bands. Therefore, we can reliably detect that the sensors are not reading in the same frequency band. Figure A.3(b) shows how we reach a good correlation value between digital signals if the sensors are configured with averaging factor from 20 on-wards. Again the correlation over digital signals when two sensors are set to a different frequencies is stable and close to -0.3.

Impact of Spectrum Resolution. The FFT resolution is an important parameter to set in our system because it will decide, first, the computational cost on the sensor in order to compute the FFT and then the average of the FFT bins, and second the bandwidth used in order to transfer the FFT bins from the sensors to the backend. We would like to reach the maximum correlation value with the minimum data sent to the backend. Figure A.4 shows that there is not so much difference among using different FFT values (256 up to 2048) in order to reach a good correlation value. However, using digital signals, we can see how the lower is the fft_size the better is the correlation value.



(a) Analog Signal (101 MHz)



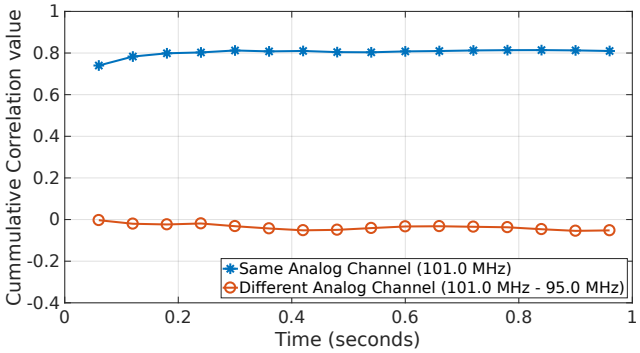
(b) Digital Signal (806 MHz)

Figure A.5: Signal correlation with different distances between sensors

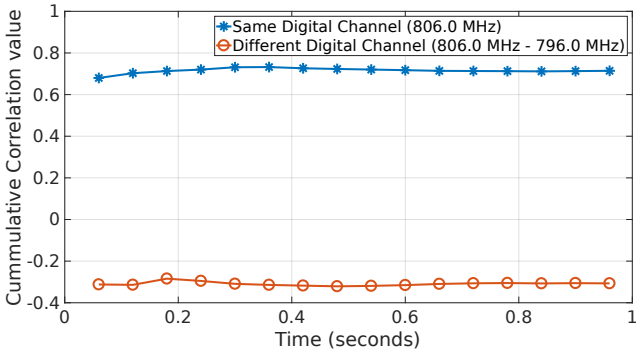
For digital signals that change faster that analog ones and contain more information seems to be a good option to use low values of fft_size to reach the maximum correlation between signals. In addition, larger fft_size tend to be affected by larger noise, which can negatively affect the correlation. Both analog and digital signals can be correlated using low values of fft_size (256), which also minimizes the bandwidth used by the sensors to perform the correlation on the backend.

Distance between the Sensors. The distance between the sensors is an important factor that can affect the spectrum correlation among sensors. If the sensors are located far from each other, the channel path loss can severely affect any collaborative strategy. This phenomenon is frequency dependent, since the path loss model decays with $20 \log_{10} f_c$, where f_c indicates the carrier frequency. Therefore higher density is expected at higher frequency to harness the spectrum correlation.

In this experiment we locate several sensors at indoor environment with different distances between each other. The distances between sensors are 3, 15, 25 and 50 meters respectively. We remark that the tests are conducted in indoor environment. Figure A.5(a) shows the signal correlation of the sensors located at different distances and scanning an analog signal. Up to 15 meters the correlation value is close to 0.8 and with



(a) Analog Signal.



(b) Digital Signal.

Figure A.6: Convergence of signal correlation using fft_size=256, avg_factor=30 and duty_cycle=2ms.

distances between 25 and 50 meters the correlation is around 0.6. Figure A.5(b) shows the same scenario but using digital signals. The correlation for distances up to 15 meters reports a value above 0.6 but for longer distances the correlation decrease dramatically. These results suggest that for analog signals the sensors could work collaboratively together for maximum distances of 50 meters and for digital signals the limit is around 15-20 meters.

Minimum bandwidth required. The goal of this experiment is to provide an answer to the following question: what is the minimum sampling time and the minimum data sent to the backend in order to measure the similarity in the spectrum ? In a distributed approach, like the one studied in this work, it is required to make decisions in the shortest possible time and with the minimum amount of data. Therefore, it is important to evaluate the convergence time to decide if the sensors are getting similar spectrum or not, and the minimum data sent over Internet to perform the correlation in the backend.

In order to detect when the system converges, we define the *cumulative correlation (CC) metric* as the average of the last K observations of correlation values.

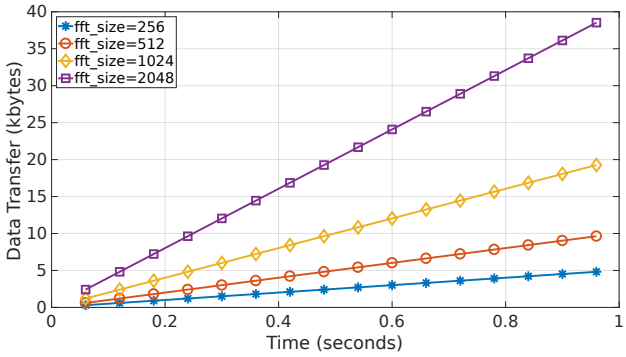


Figure A.7: Data transferred to the server for a duty_cycle=2ms and avg_factor=30.

Figure A.6 shows the CC metric over time for analog and digital signals. We can observe that after 0.2 seconds we reach a stable correlation value that allows us to distinguish if the sensors are reading a similar spectrum or not. Figure A.7 shows the data sent to the backend over time for different fft_size configurations. As we determined in the previous experiment, the best configuration of fft_size to perform the correlation in the backend is 256. In addition to that, we have already found that after 0.2 seconds the correlation gets stable. From Figure A.7, we can then conclude that every sensor needs to send only 1 Kbyte to the backend in order to get a reliable decision about the spectrum similarity.

A.3. Discussion

We have addressed the computation in the backend of the spectrum similarity of data collected by low-cost RF sensors in range of the same transmitter. We have proposed various methods to achieve reliable correlation in presence of noisy PSD data both for analog and digital signals. We have proved that our architecture provides good spectrum correlation for sensors in radio range. We have then studied problems such as how the spectrum correlation varies with the distance between sensors. Our system can reliably correlate the spectrum of analog and digital signals of different sensors using data collected during 200 ms and sending only 1 Kbyte of data per sensor to the backend.

Our experiments have shown that spectrum similarity using PSD data can be measured using analog and digital signals, and that a spectrum resolution of 256 bins provides the best performance in terms of similarity. The ElectroSense network operates by default using 256 bins, and it is then in the best configuration for measuring spectrum similarity using the PSD pipeline. This can allow to perform such a study for a preliminary verification of the potential to collaborate for two IoT spectrum sensors, for instance, in the scenarios presented in Chapter 6 and 7 of this thesis.

B Software Defined Radio (SDR)

According to Wikipedia a Software-defined Radio (SDR) is: *"a radio communication system where components that have been traditionally implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system.*

In the recent years, these SDR devices have caught the attention of practitioners, researchers and also the industry due to their great diversity and low cost price. The Figure B.1 shows the evolution over the last 18 years of the published scientific papers related to SDR devices (source: Web of Science[1]). The Table B.1 shows the SDR devices currently available in the market for a price less than 300 €.

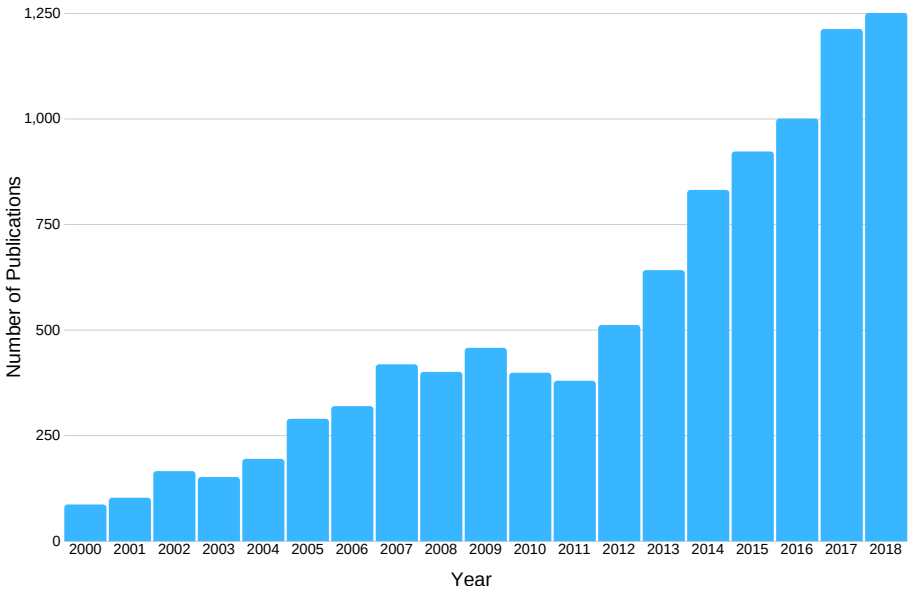
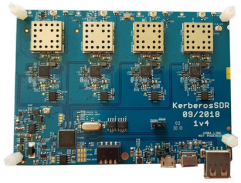


Figure B.1: Evolution of the publications related to SDRs (source: Web of Science).

[1] https://www.fecyt.es/es/recurso/web-science

Table B.1: SDRs available in the market (< 300€)

Image	Name	Transmitter	Frequency range	Sampling Rate (MSPS)	ADC (bits)	LO error (ppm)	Price (€)
	RTL-SDR v1	No	24 to 1766 MHz	2.4	8	± 60-100	15
	RTL-SDR v3	No	24 to 1766 MHz	2.4	8	± 1	15
	Kerberos 4-channels	No	24 to 1766 MHz	2.4	8	± 1	110
	Adalm Pluto	Yes	325 MHz to 3.8 GHz	20	12	± 5-20	121
	AirSpy	No	24 to 1700 MHz	10	12	± 0.5	220
	KiwiSDR + GPS	No	10 kHz to 30 MHz	30	14	± 5-10	250
	HackRF	Yes	1 MHz to 6 GHz	20	8	± 5-10	264

C Communication Activities

On the way of making the research work needed for this thesis, I had also the opportunity to communicate our progress along different small industrial conferences, events and workshops. The attendance at these events gave me the opportunity to explain ElectroSense from a different technical level perspective and collect useful feedback from the industry point of view.

C.1. Cyber Alp Retreat (2015-2019)

The Cyber Alp Retreat is an annual research event organized by armasuisse Science and Technology. It brings together researchers, partners, and collaborators that are contributing with their work to the Cyber and Information research program. The retreat offers an ideal platform for presenting ongoing project results, discussing new ideas, and coordinating among the different research activities. During the 4 years I attended this scientific retreat and I also participated on it by giving a talk every year about the advances and the ongoing research of ElectroSense.

C.2. T3chFest (February 2015)

On Thursday, February 11th 2016, UC3M opened the doors for the 4th edition of T3chFest, a yearly event that brings together students, researchers and start-ups as well as established companies and specialists in new technologies. I had the opportunity to give a talk[1] in front of students and people from the industry to introduce, at that time, what were the first ideas and implementations about ElectroSense.

[1] https://www.networks.imdea.org/whats-new/news/2016/imdea-networks-participates-t3chfest-uc3m

Figure C.1: ElectroSense talk at T3chFest.

C.3. Visiting Telefónica (July 2018)

On July 12th, we had an important visiting at IMDEA Networks Institute of the top executives managers of Telefónica. Telefónica's Global CTO and Telefónica Research CEO of Spain were visiting our labs and I had also the time to show them a ElectroSense demo, and how we could use it to provide useful information to mobile operators helping them in the new infrastructure deployments for IoT or 5G.

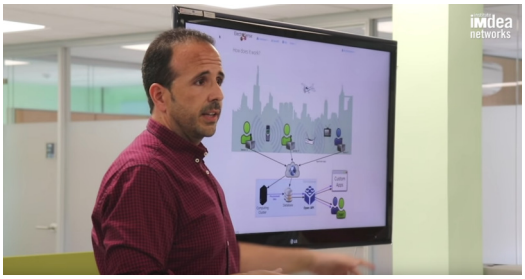


Figure C.2: ElectroSense presented to Telefónica.

C.4. Ericsson Innovation Day (November 2018)

On November 13th, Ericsson celebrated its Innovation Day at the company's R&D center in Madrid. It is one of the reference centers in Europe gathering more than 500 engineers working on the latest technologies. The event focused mainly on the innovation needed for the transformation that will involve the implementation of 5G technology. Along 2 days I was in the demonstration area showing the capabilities of ElectroSense in real-time to the attenders. They could see the EM spectrum usage in the room and we could detect and analyze several transmissions that were using 5G technology in the event.

Figure C.3: ElectroSense at Ericsson Innovation Day.

C.5. Madrid fair of Science and Technology (April 2019)

IMDEA Networks Institute joined the Madrid Fair for Science and Innovation, the leading national event for the dissemination of science and research, that was organized by the Madri+d Foundation for Knowledge. Together with other colleges and using ElectroSense we encouraged attendees to explore from the workings of FM radio with, which we are all familiar, to the IoT, at the cutting edge of technological innovation.

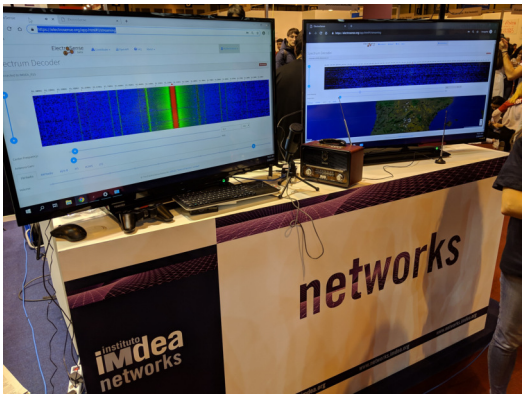


Figure C.4: ElectroSense used as learning tool to explain EM spectrum and radio FM to students.

C.6. 1st ElectroSense Workshop (April 2019)

The first ElectroSense workshop[2] was hold on KU Leuven University (Belgium), on 4-5 of April 2019 and was organized by the ElectroSense association. The two-days workshop was targeted at anyone interested in research and applications with spectrum data. The workshop served as a forum to discuss new ideas, the network itself, and research activities around ElectroSense. About 25 people attended the workshop where I personally had 3 different talks to explain the main features of the infrastructure on the sensor side and to draw the main research challenges for the collaborative signal decoding.

(a) (b)

Figure C.5: 1st ElectroSense workshop at KU Leuven University.

[2]http://workshop.electrosense.org

References

[1] R. Calvo-Palomino, D. Giustiniano, V. Lenders, and A. Fakhreddine, "Crowdsourcing spectrum data decoding," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, may 2017, pp. 1–9.

[2] R. Calvo-Palomino, D. Giustiniano, and V. Lenders, "Measuring Spectrum Similarity in Distributed Radio Monitoring Systems," in *International Tyrrhenian Workshop on Digital Communication*. Springer, Cham, 2017, pp. 215–229.

[3] R. Calvo-Palomino, F. Ricciato, D. Giustiniano, and V. Lenders, "LTESS-track: A Precise and Fast Frequency Offset Estimation for Low-cost SDR Platforms," in *Proceedings of the 11th Workshop on Wireless Network Testbeds, Experimental Evaluation and CHaracterization*, ser. WiNTECH '17. New York, NY, USA: ACM, 2017, pp. 51–58.

[4] S. Rajendran, R. Calvo-Palomino, M. Fuchs, B. V. den Bergh, H. Cordobés, D. Giustiniano, S. Pollin, V. Lenders, H. Cordobes, D. Giustiniano, S. Pollin, and V. Lenders, "ElectroSense: Open and big spectrum data," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 210–217, 2018.

[5] R. Calvo-Palomino, F. Ricciato, B. Repas, D. Giustiniano, and V. Lenders, "Nanosecond-precision time-of-arrival estimation for aircraft signals with low-cost SDR receivers," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IEEE Press, 2018, pp. 272–277.

[6] R. Calvo-Palomino, H. Córdobes, F. Ricciato, D. Giustiniano, and V. Lenders, "Collaborative Wideband Signal Decoding using Non-coherent Receivers ," in *Proceedings of the 18th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IEEE, may 2019, pp. 1–9.

[7] R. Calvo-Palomino, H. Cordobés, M. Engel, M. Fuchs, P. Jain, M. Liechti, S. Rajendran, M. Schäfer, B. V. den Bergh, S. Pollin, D. Giustiniano, and V. Lenders, "ElectroSense+: Empowering People to Decode the Radio Spectrum," *ArXiv*, vol. abs/1811.1, 2018. [Online]. Available: http://arxiv.org/abs/1811.12265

[8] R. Calvo-Palomino, "Radio-Signal Correlation for Collaborative Wideband Spectrum Monitoring System," Master's thesis, Universidad Carlos III de Madrid, Spain, 2015.

[9] R. Calvo-Palomino, D. Pfammatter, D. Giustiniano, and V. Lenders, "Demonstration Abstract: A Low-cost Sensor Platform for Large-scale Wideband Spectrum Monitoring," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, ser. IPSN '15. New York, NY, USA: ACM, 2015, pp. 396–397.

[10] B. den Bergh, D. Giustiniano, H. Cordobés, M. Fuchs, R. Calvo-Palomino, S. Pollin, S. Rajendran, and V. Lenders, "ElectroSense: Crowdsourcing spectrum monitoring," in *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2017, pp. 1–2.

[11] F. Minucci, S. Rajendran, B. V. den Bergh, S. Pollin, D. Giustiniano, H. Cordobés, M. Fuchs Roberto Calvo-Palomino, and V. Lenders, "Demo: ElectroSense - Spectrum Sensing with Increased Frequency Range," in *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, ser. EWSN 2018. USA: Junction Publishing, 2018, pp. 203–204.

[12] M. Hung, "Leading the iot, gartner insights on how to lead in a connected world," *Gartner Research*, pp. 1–29, 2017.

[13] "80 Mind-Blowing IoT Statistics," 2018. [Online]. Available: https://safeatlast.co/blog/iot-statistics/

[14] X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5G Ultra-Dense Cellular Networks," *IEEE Wireless Communications*, vol. 23, no. 1, pp. 72–79, 2016.

[15] "Frequency allocation." [Online]. Available: https://en.wikipedia.org/wiki/Frequency_allocation

[16] "Microsoft Spectrum Observatory," 2015. [Online]. Available: http://observatory.microsoftspectrum.com/

[17] A. Iyer, K. K. Chintalapudi, V. Navda, R. Ramjee, V. Padmanabhan, and C. Murthy, "SpecNet: Spectrum Sensing Sans Frontières," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2011.

[18] "RTL-SDR dongle v3," 2017. [Online]. Available: http://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/

[19] S. Rajasegarar, C. Leckie, and M. Palaniswami, "Anomaly detection in wireless sensor networks," *IEEE Wireless Communications*, vol. 15, no. 4, 2008.

[20] J. Unnikrishnan and V. V. Veeravalli, "Cooperative Sensing for Primary Detection in Cognitive Radio," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 1, pp. 18–27, 2008.

[21] A. A. Khan, M. H. Rehmani, and M. Reisslein, "Cognitive radio for smart grids: Survey of architectures, spectrum sensing mechanisms, and networking protocols," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 860–898, 2016.

[22] "Spectrum Collaboration Challenge - The world's first collaborative machine-intelligence competition to overcome spectrum scarcity." [Online]. Available: http://spectrumcollaborationchallenge.com/

[23] "Google Spectrum Database," 2016. [Online]. Available: https://www.google.com/get/spectrumdatabase/

[24] "IBM Horizon," 2016. [Online]. Available: https://bluehorizon.network/documentation/sdr-radio-spectrum-analysis

[25] "KiwiSDR - Wide-band SDR + GPS cape for the BeagleBone Black," 2013. [Online]. Available: http://kiwisdr.com/

[26] A. Chakraborty, M. S. Rahman, H. Gupta, and S. R. Das, "Specsense: Crowdsensing for efficient querying of spectrum occupancy," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[27] N. Kleber, A. Termos, G. Martinez, J. Merritt, B. Hochwald, J. Chisum, A. Striegel, and J. N. Laneman, "Radiohound: A pervasive sensing platform for sub-6 ghz dynamic spectrum monitoring," in *Dynamic Spectrum Access Networks (DySPAN), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–2.

[28] A. Nika, Z. Zhang, X. Zhou, B. Y. Zhao, and H. Zheng, "Towards Commoditized Real-time Spectrum Monitoring," in *Proceedings of the 1st ACM Workshop on Hot Topics in Wireless*, ser. HotWireless '14. New York, NY, USA: ACM, 2014, pp. 25–30.

[29] F. Ricciato, S. Sciancalepore, F. Gringoli, N. Facchi, and G. Boggia, "Position and velocity estimation of a non-cooperative source from asynchronous packet arrival time measurements," *IEEE Transactions on Mobile Computing*, vol. 17, no. 9, pp. 2166–2179, 2018.

[30] A. Nika, Z. Li, Y. Y. Zhu, Y. Y. Zhu, B. Y. Zhao, X. Zhou, and H. Zheng, "Empirical Validation of Commodity Spectrum Monitoring," in *Proceedings of 14th*

ACM Conference on Embedded Networked Sensor Systems (SenSys 2016). ACM, ser. ACM SenSys '16, 2016, pp. 96–108.

[31] "Raspberry Pi 3 Model B," 2016. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[32] "ElectroSense Open API," 2018. [Online]. Available: https://electrosense.org/app.html{#}!/api-spec

[33] "ElectroSense," 2018. [Online]. Available: https://electrosense.org/

[34] D. Pfammatter, D. Giustiniano, and V. Lenders, "A Software-defined Sensor Architecture for Large-scale Wideband Spectrum Monitoring," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, ser. IPSN '15. New York, NY, USA: ACM, 2015, pp. 71–82.

[35] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms," *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[36] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[37] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., 2015.

[38] J. Kreps, N. Narkhede, J. Rao, and Others, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[39] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[40] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, p. 2.

[41] M. Kornacker, A. Behm, V. Bittorf, E. al., M. Bittorf, T. Bobrovytsky, C. C. A. C. J. Erickson, M. G. D. Hecht, M. J. I. J. L. Kuff, D. K. A. Leblang, N. L. I. P. H. Robinson, D. R. S. Rus, J. R. D. T. S. Wanderman, and M. M. Yoder, "Impala: A modern, open-source SQL engine for Hadoop," in *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research*, 2015.

[42] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[43] T. Yucek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Surveys Tutorials*, vol. 11, no. 1, pp. 116–130, 2009.

[44] J. van de Beek, J. Riihijarvi, A. Achtzehn, and P. Mahonen, "TV White Space in Europe," *IEEE Transactions on Mobile Computing*, vol. 11, no. 2, pp. 178–188, feb 2012.

[45] N. E. West and T. O'Shea, "Deep architectures for modulation recognition," in *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2017, pp. 1–6.

[46] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Deep Learning Models for Wireless Signal Classification with Distributed Low-Cost Spectrum Sensors," *IEEE Transactions on Cognitive Communications and Networking*, 2018.

[47] A. Zanella, "Best Practice in RSS Measurements and Ranging," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2662–2686, 2016.

[48] "WebSDR," 2018. [Online]. Available: http://www.websdr.org/

[49] "LiveATC," 2018. [Online]. Available: https://www.liveatc.net/

[50] "Open Web Rx: Open Source SDR Web App for Everyone," 2018. [Online]. Available: https://sdr.hu/openwebrx

[51] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, M. Wilhelm, M. Schaefer, M. Strohmeier, V. Lenders, I. Martinovic, M. Wilhelm, M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm, "Bringing up OpenSky: A large-scale ADS-B sensor network for research," in *Proceedings of the 13th international symposium on Information processing in sensor networks*, ser. IPSN '14. Berlin, Germany: IEEE Press, 2014, pp. 83–94.

[52] "AirSpy," 2018. [Online]. Available: https://airspy.com/

[53] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S: A publish/subscribe protocol for Wireless Sensor Networks," in *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, 2008, pp. 791–798.

[54] "RTL SDR Silver," 2018. [Online]. Available: https://www.rtl-sdr.com/wp-content/uploads/2018/02/RTL-SDR-Blog-V3-Datasheet.pdf

[55] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[56] S. Rajendran, W. Meert, V. Lenders, and S. Pollin, "SAIFE: Unsupervised Wireless Spectrum Anomaly Detection with Interpretable Features," in *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2018, pp. 1–9.

[57] M. Strohmeier, M. Schäfer, M. Fuchs, V. Lenders, and I. Martinovic, "OpenSky: A Swiss Army Knife for Air Traffic Security Research," in *Proceedings of the 34th IEEE/AIAA Digital Avionics Systems Conference*, ser. DASC, 2015.

[58] M. Schäfer, P. Leu, V. Lenders, and J. Schmitt, "Secure Motion Verification using the Doppler Effect." in *Proc. of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2016.

[59] *rtl_test*, 2016. [Online]. Available: https://github.com/steve-m/librtlsdr

[60] "kalibrate-rtl," 2013. [Online]. Available: https://github.com/steve-m/kalibrate-rtl

[61] *LTE-Cell-Scanner*, 2012. [Online]. Available: https://github.com/Evrytania/LTE-Cell-Scanner

[62] "RTL-SDR Silver v3 specifications," may 2016. [Online]. Available: http://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/

[63] S. Sesia, *LTE - The UMTS Long Term Evolution: From Theory to Practice*, Wiley, Ed., 2011.

[64] E. U. T. R. Access, "Physical channels and modulation," *3GPP TS*, vol. 36.211, p. V8, 2016.

[65] ETSI, *Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception (3GPP TS 36.104)*, 2013.

[66] F. J. Harris, *Multirate Signal Processing for Communication Systems*, ser. Mobility '08. Prentice Hall, 2004.

[67] *Technical Provisions for Mode S Services and Extended Squitter*, International Civil Aviation Organization, 2008.

[68] *FlightAware - The world's largest flight tracking data company*, may 2018. [Online]. Available: http://www.flightaware.com/

[69] "FlightRadar24," may 2018. [Online]. Available: https://www.flightradar24.com/

[70] M. Strohmeier, I. Martinovic, and V. Lenders, "Ak-nn-based localization approach for crowdsourced air traffic communication networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 3, pp. 1519–1529, 2018.

[71] M. Schäfer, V. Lenders, and J. Schmitt, "Secure track verification," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 199–213.

[72] M. Strohmeier, V. Lenders, and I. Martinovic, "Lightweight location verification in air traffic surveillance networks," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. ACM, 2015, pp. 49–60.

[73] D. Moser, P. Leu, V. Lenders, A. Ranganathan, F. Ricciato, and S. Capkun, "Investigation of Multi-device Location Spoofing Attacks on Air Traffic Control and Possible Countermeasures," in *ACM Conference on Mobile Computing and Networking (Mobicom)*, 2016.

[74] K. Jansen, M. Schäfer, D. Moser, V. Lenders, C. Pöpper, and and J. Schmitt, "Crowd-GPS-Sec: Leveraging Crowdsourcing to Detect and Localize GPS Spoofing Attacks," in *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[75] M. Eichelberger, K. Luchsinger, S. Tanner, and R. Wattenhofer, "Indoor localization with aircraft signals." in *SenSys*, vol. 17, 2017, pp. 6–8.

[76] *Minimum Operational Performance Standards for Air Traffic Control Radar Beacon System / Mode Select (ATCRBS / Mode S) Airborne Equipment*, RTCA, 2011.

[77] S. M. Kay, *Fundamentals of Statistical Signal Processing, vol. 1, Estimation Theory*. Prentice-Hall, 1993.

[78] "dump1090 - a simple mode s decoder for rtlsdr devices," 2016. [Online]. Available: https://github.com/mutability/dump1090

[79] I. C. Aviation and ICAO, "Aeronautical Communications, Volume IV, Surveillance and Collision Avoidance Systems," 2007.

[80] G. Galati, M. Leonardi, I. A. Mantilla-Gaviria, and M. Tosti, "Lower bounds of accuracy for enhanced mode-s distributed sensor networks," *IET Radar, Sonar & Navigation*, vol. 6, no. 3, pp. 190–201, 2012.

[81] G. de Miguel Vela, J. B. Portas, and J. G. Herrero, "Correction of propagation errors in wide area multilateration systems," in *2009 European Radar Conference (EuRAD)*. IEEE, 2009, pp. 81–84.

[82] T. Strutz, *Data fitting and uncertainty: A practical introduction to weighted least squares and beyond*. Vieweg and Teubner, 2010.

[83] "GNU radio." [Online]. Available: http://gnuradio.org/

[84] A. Nayagam, J. M. Shea, and T. F. Wong, "Collaborative decoding in bandwidth-constrained environments," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 2, pp. 434–446, 2007.

[85] L. Shi, P. Bahl, and D. Katabi, "Beyond Sensing: Multi-GHz Realtime Spectrum Analytics," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*. Oakland, CA: {USENIX} Association, 2015, pp. 159–172.

[86] T. Zhang, N. Leng, and S. Banerjee, "A vehicle-based measurement framework for enhancing whitespace spectrum databases," in *Proceedings of the 20th annual international conference on Mobile computing and networking*. ACM, 2014, pp. 17–28.

[87] M. Souryal, M. Ranganathan, J. Mink, and N. El Ouni, "Real-time centralized spectrum monitoring: Feasibility, architecture, and latency," in *2015 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2015, pp. 106–112.

[88] M. Zheleva, R. Chandra, A. Chowdhery, A. Kapoor, and P. Garnett, "TxMiner: Identifying transmitters in real-world spectrum measurements," in *IEEE DySPAN*, sep 2015, pp. 94–105.

[89] M. Higginson-Rollins and A. E. E. Rogers, "Development of a Low Cost Spectrometer for Small Radio Telescope (SRT), Very Small Radio Telescope (VSRT), and Ozone Spectrometer," 2013.

[90] A. Almagbile, J. Wang, and W. Ding, "Evaluating the performances of adaptive Kalman filter methods in GPS/INS integration," *Journal of Global Positioning Systems*, 2010.

[91] S. M. Mishra, A. Sahai, and R. W. Brodersen, "Cooperative Sensing among Cognitive Radios," in *2006 IEEE International Conference on Communications*, vol. 4, 2006, pp. 1658–1663.

[92] A. Ghasemi and E. S. Sousa, "Collaborative spectrum sensing for opportunistic access in fading environments," in *2005 1st IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, DySPAN 2005.* IEEE, 2005, pp. 131–136.

[93] A. S. Cacciapuoti, I. F. Akyildiz, and L. Paura, "Correlation-aware user selection for cooperative spectrum sensing in cognitive radio ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 2, pp. 297–306, 2012.

[94] M. Sampietro, G. Accomoando, L. G. Fasoli, G. Ferrari, and E. C. Gatti, "High Sensitivity Noise Measurement with a Correlation Spectrum Analyzer," *IEEE Transactions on Instrumentation and Measurement*, vol. 49, no. 4, pp. 1–3, 2000.

[95] J. N. Laneman, D. N. C. Tse, and G. W. Wornell, "Cooperative Diversity in Wireless Networks: Efficient Protocols and Outage Behavior," *IEEE Transactions on Information Theory*, vol. 50, 2004.

[96] K. Jamil, M. Alam, M. A. Hadi, and Z. O. Alhekail, "A multi-band multi-beam software-defined passive radar. Part I: System design," 2012.

[97] S. Haykin, *Communication Systems*, 5th ed. Wiley Publishing, 2009.

[98] A. V. Oppenheim and R. W. Schafer, *Discrete-time Signal Processing.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[99] V. Dimitrakis, "Expanding the View on Avionics Communication The Mode-S Uplink," Master's thesis, ETH Zurich, 2017.

[100] P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," RFC 1951 (Informational), may 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1951.txt

[101] A. Chakraborty, A. Bhattacharya, S. Kamal, S. R. Das, H. Gupta, and P. M. Djuric, "Spectrum Patrolling with Crowdsourced Spectrum Sensors," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications.* IEEE, 2018, pp. 1682–1690.

[102] A. Dongare, R. Narayanan, A. Gadre, A. Luong, A. Balanuta, S. Kumar, B. Iannucci, and A. Rowe, "Charm: exploiting geographical diversity through coherent combining in low-power wide-area networks," in *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks.* IEEE Press, 2018, pp. 60–71.

[103] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi, "GHz-wide sensing and decoding using the sparse Fourier transform," in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 2256–2264.

[104] A. Capria, M. Conti, D. Petri, M. Martorella, F. Berizzi, E. Dalle Mese, R. Soleti, and V. Carulli, "Ship detection with dvb-t software defined passive radar," in *IEEE Gold Remote Sensing Conference*, 2010.

[105] G. Aloi, V. Loscri, A. Borgia, E. Natalizio, S. Costanzo, P. Pace, G. Di Massa, and F. Spadafora, "Software Defined Radar: Synchronization Issues and Practical Implementation," in *Proceedings of the 4th CogART '11.* New York, NY, USA: ACM, 2011, pp. 48:1—-48:5.