Department of Computer
& Communication Engineering
University of Thessaly
Volos Greece

Design and Development of an Integrated Framework for Online Evaluation of Sensing Delay and Energy Consumption Metrics During Spectrum Sensing

by

Virgilios Passas

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master
(Department of Electrical and Computer Engineering)
University of Thessaly
2014

Supervisors:

Athanasios Korakis
Leandros Tassiulas
Antonis Argyriou

To my grandparents and uncle,

Elenh, Vergos and Dimitris

ACKNOWLEDGEMENTS

First and foremost I wish to thank my advisor, Assistant Professor Thanasis Korakis for giving me the opportunity to work in such an inspiring research unit and more importantly for his guidance.

Also I would like to thank Professor Leandros Tassiulas and Lecturer Antonis Argyriou, who are in my advisor commitee, for their guidance in this thesis.

Especially, I would like to thank my friends and lab-mates in NITlab, for supporting me every day. First of all, I would like to thank Stratos Keranidis for supporting me from my first days in the lab and for our joint research and collaboration that we have all these years. Particular thanks go to Giannis Kazdaridis for designing all the hardware that was needed in order to implement and finish my Master thesis along with his advices during this thesis. I would really like to thank Kostas Chounos and Ilias Syrigos for helping me on the part of configuring the Atheros chipset and moreover the rest NITLab members.

Last but really important, I would like to thank my mother Anna and my father Grigoris for supporting me everyday in my whole live. I would also like to thank Fotini Vachlioti for being next to me all these years.

Part of this work presented in the 9th ACM International Workshop on Wireless Network Testbeds, WiNTECH '14 with title:
Online Assessment of Sensing Performance in Experimental Spectrum Sensing Platforms [14] and co-authors:
Stratos Keranidis, Kostas Chounos, Wei Liu, Thanasis Korakis, Iordanis Koutsopoulos, Ingrid Moerman and Leandros Tassiulas.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Cognitive radio systems have gathered a lot of research interest during the last decade. Accuracy of spectrum sensing and efficiency of free spectrum utilization are considered as the primary objectives in this emerging technology, which promises a boost in wireless network performance, through exploitation of underutilized licensed frequency bands.

As the focus of researchers is usually on these two major challenges, other aspects have been in part underestimated. In this master thesis, we consider two factors that are rather important for evaluation of cognitive platforms, namely sensing delay and energy efficiency. The first is related to the latency induced by the spectrum sensing process and its impact on sensing efficiency, which is tightly connected to both the QoS performance of secondary users and the protection of primary users. On the other hand, energy consumption is considered as a crucial issue in all types of wireless communications, due to restricted battery autonomy of mobile devices, as well as for moving towards "greener" solutions in telecommunications.

Therefore, it is important to extend existing testbed experimentation tools and develop new ones, in order to equip cognitive testbeds with such advanced monitoring capabilities. In order to demonstrate the applicability of our framework, we experimentally validate the performance of four different sensing platforms, as well as a real-time spectrum sensing engine that implements parallel processing on software-defined radios, in terms of the aforementioned metrics.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

«**Σχεδίαση και Ανάπτυξη ενός Ολοκληρωμένου Συστήματος Εκτίμησης της Καθυστέρησης της Ανίχνευσης και της Καταναλισκόμενης Ενέργειας κατά την Παρατήρηση Φάσματος σε Πραγματικό Χρόνο**»

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πανεπιστήμιο Θεσσαλίας

Βιργίλιος Πασσάς

Μεταπτυχιακή Διατριβή

Τα γνωστικά συστήματα ραδιοεπικοινωνιών έχουν συγκεντρώσει πολύ ερευνητικό ενδιαφέρον κατά τη διάρκεια της τελευταίας δεκαετίας. Η ακρίβεια στην ανίχνευση φάσματος και της αποδοτικής χρησιμοποίησης του ελεύθερου φάσματος θεωρούνται ως οι πρωταρχικοί στόχοι σε αυτή την αναδυόμενη τεχνολογία, η οποία υπόσχεται μια ώθηση στην ασύρματη απόδοση του δικτύου, μέσω της αξιοποίησης των ζωνών συχνοτήτων που χρειάζονται αδειοδότηση για να χρησιμοποιηθούν αλλά υποχρησιμοποιούντε.

Καθώς το επίκεντρο των ερευνητών είναι συνήθως σε αυτούς τους δύο στόχους, άλλες πτυχές έχουν εν μέρει υποτιμηθεί. Στην παρούσα μεταπτυχιακή εργασία, θεωρούμε δύο παράγοντες που πιστεύουμε ότι είναι εξίσου σημαντικοί για την αξιολόγηση των γνωστικών πλατφορμών, την καθυστέρηση κατά την διαδικασία της ανίχνευσης και την ενεργειακή απόδοση αυτών των συσκευών. Η πρώτη μετρική σχετίζεται με την καθυστέρηση που προκαλείται κατά την διαδικασία ανίχνευσης του φάσματος και τις επιπτώσεις της στην απόδοση ανίχνευσης, η οποία είναι στενά συνδεδεμένη και με το QoS των secondary χρηστών και την προστασία των primary χρηστών.

Από την άλλη πλευρά, η κατανάλωση ενέργειας θεωρείται ως ένα κρίσιμο ζήτημα σε όλους τους τύπους ασύρματων επικοινωνιών, λόγω της περιορισμένης αυτονομίας της μπαταρίας των κινητών συσκευών, καθώς και για τη στροφή προς τις "πράσινες" λύσεις στον τομέα των τηλεπικοινωνιών.

Ως εκ τούτου, είναι σημαντικό να επεκτείνουμε τα υπάρχοντα εργαλεία πειραματικών υποδομών και την ανάπτυξη νέων, προκειμένου να εξοπλίσουμε cognitive πειραματικές υποδομές με τέτοιες προηγμένες δυνατότητες παρακολούθησης. Προκειμένου να καταδειχθεί η εφαρμοσιμότητα της πλατφόρμας μας, επικυρώσαμε μέσω πειραμάτων την απόδοση τεσσάρων διαφορετικών πλατφορμών ανίχνευσης, καθώς και το οτι υπάρχει μηχανή ανίχνευσης φάσματος σε πραγματικό χρόνο που υλοποιεί παράλληλη επεξεργασία, ως προς τις προαναφερθείσες μετρικές.

CHAPTER I

Introduction

Cognitive Radio Networking is a rapidly evolving research thrust in wireless communications nowadays, aspiring to create a major paradigm shift in the wireless landscape through Dynamic Spectrum Access and Management.

Theoretical developments [9, 33] in the area have proposed various methods for identifying the presence of signal transmissions and evaluating spectrum occupancy. Parallel to theoretical developments in this area, realistic experimentation platforms [24, 30, 15, 21] have emerged, focusing on different aspects of spectrum access, like sensing and end-to-end information transmission. Sound experimental validation of proposed spectrum sensing solutions requires experimentation under real world network scale and settings. To this aim, experimental cognitive radio testbeds [28, 20, 8, 16, 29, 7] have been deployed, while tools for orchestration of complex scenarios [17] and collection [19] of experimental data have also been developed.

As dynamic spectrum access is now an increasingly explored area, there exists a vast variety of relevant works in this field that propose different cognitive solutions to facilitate context awareness in the available bands. In order to provide fair performance evaluation between competing cognitive solutions, the efficiency of most proposed schemes is evaluated based on network performance metrics. Such metrics mainly include achieved throughput performance, rate of packet delivery, end-to-end delay, spectrum utilization and other relevant metrics, estimating the performance of the network under test. Moreover, another category of widely used performance metrics exists, which is related to the accuracy of the detection procedure. Typically, accuracy of detection is measured in terms of probability of false alarm and probability of missed detection, considering a controllable interference source as reference.

Despite the recent advances in the field of Cognitive Radio, both theoretically and experimentally, there exist two prime factors for which experimental validation lags behind. These factors are:

- Spectrum Sensing Delay
- Energy Consumption during spectrum sensing

These factors and their interrelation decisively shape fundamental Cognitive Radio system performance in terms of spectral efficiency, access throughput and energy expenditure.

For instance, Spectrum Sensing Delay is related to the time duration required by each different cognitive solution, to determine the presence of active transmitters and also to quantify the spectrum utilization on a specific frequency. Under the term of Spectrum Sensing Delay, we group the time required by each different step of the sensing procedure, thus including time spent during the gathering of required number of samples, the subsequent measurement processing and handling, and the final decision making procedure. The total amount of time spent in processing and measurement handling translates to lost capture time opportunities and therefore to intervals during which the spectrum is not being monitored. As a result, constant monitoring of sensing delay provides for online estimation of achieved detection performance during the experiment execution. Due to the fact that the whole wireless environment is stochastic and uncertain in terms of spectrum occupancy and channel conditions, online performance monitoring is required instead of comparison with performance yielded in reference scenarios. For instance, uncontrolled factors, such as prevailing interference and traffic conditions in highly congested bands, in accordance with the effects induced by simultaneous operation of multiple collocated wireless nodes create an environment where performance can be highly unpredictable [12].

On the other hand, the way in which energy is consumed in cognitive networking raises enormous opportunities for prudent use of energy. It is therefore critical to view cognitive networking from an energy efficiency point of view. In order to estimate the energy consumption of cognitive networks accurately, it is important to understand, which are the different software and hardware configurations that impact energy expenditure. Having determined this, specific tools have to be developed that are adaptable and sufficiently accurate to monitor consumption during both low level physical (PHY) layer operations, as well as higher level configurations related with the overall consumption of the entire cognitive solution. More specifically, one needs to measure energy consumed during an individual measurement capture and thus the developed tools have to be able to cope with the high sampling rate of the sensing hardware. In addition, higher-level configurations may also affect energy consumption significantly. For example, the number of devices participating in the sensing procedure is such a configuration. As a result, the developed tools have to provide for proper energy consumption monitoring of the overall cognitive network setup, so that high level settings can be configured in an energy efficient way as well.

Having determined proper tools for gathering measurements related to the Sensing Delay and Energy Consumption factors, the next big step is to provide for efficient processing and storing of the corresponding measurements. The high sampling rate of sensing devices requires that monitoring of the above factors is also performed with a respectively high sampling rate. As a result, the problem of measurements handling becomes even more challenging due to the huge number of measurements required to be reliably collected and stored. Moreover, it is important to apply data processing techniques on the huge amount of captured data, so that processed representative data are finally stored in an efficient manner.

Consideration of the aforementioned metrics gets even more important due to the high level of heterogeneity that characterizes both the existing sensing techniques, as well as the type of available hardware. In more detail, available sensing techniques range from simple energy detection to more sophisticated methods, such as feature detection. In addition, the various available platforms range from low-end sensor nodes to generic mini PCs equipped with Wi-Fi 802.11 radios and high-quality reconfigurable software radios with high processing power. Consequently, the resulting performance in terms of the above factors significantly varies, based on the underlying software and hardware configurations that make up the different cognitive solutions.

Concluding, we understand the necessity of considering the two aforementioned metrics during experimental performance evaluation of cognitive experiments. Therefore, it is important to extend existing experimentation tools and develop new ones, in order to equip cognitive testbeds with such advanced functionalities.

CHAPTER II

Hardware Equipment

In order to design and develop this framework we exploit the advanced spectrum sensing platforms that are provided by the CREW federated testbed platform, which facilitates experimentally-driven research on advanced spectrum sensing, cognitive radio and cognitive networking strategies. We chose the w-iLab.t indoor testbed among the 5 testbeds that constitute CREW, as it offers all the required hardware and software components for the development of the proposed framework. More specifically, w-iLab.t features several sensing devices that span from commercial sensor and Wi-Fi nodes, to SDR platforms and device prototypes. Another basic characteristic that motivated the usage of the w-ilab.t testbed, was the adoption of OMF as its testbed control and management framework, which fact enabled us to build the proposed framework as a plug-in compatible with the well-adopted OMF. setup realistic topologies for testing the proposed framework, since its early development stages. In the rest of this section, we describe in detail the capabilities of the considered w-ilab.t sensing devices, while we also describe the hardware components that they were required in order to implement one of the design goals.

2.1 Characteristics of Sensing Devices

In our experiments, we consider the well-established commercial SDR USRP N210 Networked Series [27], which is a commercial SDR platform that utilizes the general purpose processor of a connected host machine for measurement processing and due to its applicability has gained widespread usage. In addition, we consider the embedded version USRP E110 [25], which allows standalone operation without requiring the use of a host machine, thus allowing us to evaluate how its limited internal processing capabilities impact sensing performance. The next device under consideration is the prototype imec Sensing Engine (SE) [21], which is a high-end prototype reconfigurable radio that is designed to process samples real-time. Finally, we also decided to experiment with the Wi-Fi compliant Atheros AR9380 [5] chipset, as this device is able to operate as a spectrum scanner as well. By including the AR9380 device in our experiments, we can investigate how well the hardware assisted FFT processing capabilities of this chipset perform, contrary to the rest research oriented sensing devices. More details about the capabilities of the devices under investigation follow below.

2.1.1 USRP N210

The Universal Software Radio Peripheral (USRP) is a commercial SDR platform (Figure 2.1) that utilizes a general purpose processor and has gained widespread usage. USRP consists of two parts, a fixed motherboard and a plug-in daughterboard. The motherboard contains ADC/DAC, an FPGA for digital down conversion with programmable decimation rate and a Gigabit Ethernet (GbE) interface for communication with the host PC. The USRP N210 [27] can be programmed through the USRP Hardware Driver (UHD) [26]. The XCVR 2450 daughterboard [32] provides basic RF front-end functionality in the 2.4 GHz and 5.9 GHz bands. Although the ADC is able to sample at the 100 MHz rate, the Gigabit Ethernet interface limits the

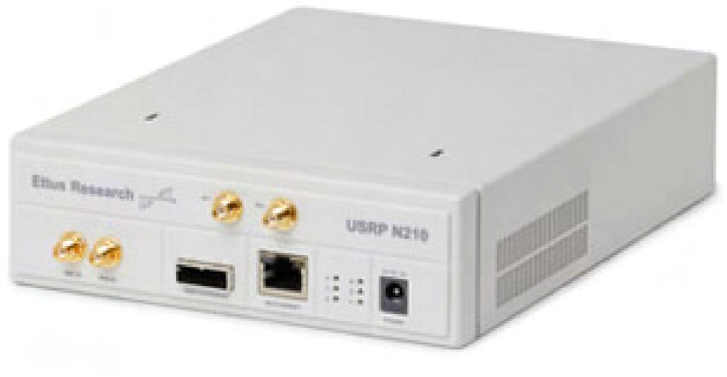


Figure 2.1: Ettus USRP N210

host bandwidth to 50 MSps, which results in the maximum streaming bandwidth of 25 MSps in each direction. In order to provide real-time sensing capability, the host machine should be able to achieve sufficient amount of parallel processing and thus this feature is host machine dependent.

2.1.2 USRP E110

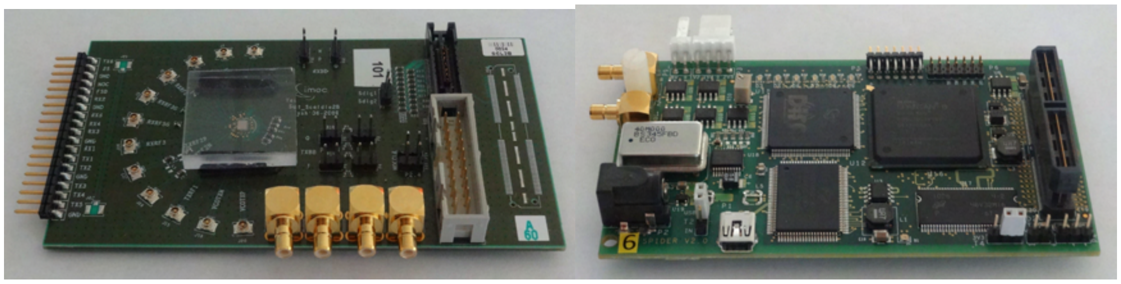
The USRP E110 [25] series combines a flexible RF frontend, FPGA and an OMAP 3, which includes an ARM Cortex-A8 and a C64 DSP (Figure 2.2). The extra features differentiate the E110 from the N210, as it does not require the use of a host pc for measurement processing and thus allows standalone operation for embedded applications, or applications that do not require the full processing power of a commodity CPU. The USRP E110 is also programmed through the USRP Hardware Driver (UHD) [26] and attached with the XCVR 2450 daughterboard [32] that is able to operate in the 2.4 GHz and 5.9 GHz bands. The attached ADC is able to sample at the 64 MHz rate, but the FPGA interface provides for a maximum total bandwidth of 10 MSps. However, this does not guarantee that the embedded processor will be able to process that many samples. Our own tests have shown that up to the 5 MSps bandwidth configuration the embedded processor is able to provide stable operation.

Figure 2.2: Ettus USRP E110

In the case that the results are processed at the embedded processor, real-time sensing can only be achieved at the low sampling rate of MSps, while in the case that processing takes place at an external host machine, we have to make sure that the host is able to achieve sufficient amount of parallel processing, in order to provide for real-time sensing. As a result the ability to provide for real-time sensing depends on the attached host machine and thus this feature is host machine dependent.

2.1.3 imec Sensing Engine

The prototype IMEC SE [21] is a reconfigurable radio and consists of two core components: an analogue RF front-end SCALDIO (SCAlable raDIO) (Figure 2.3(a)) and a DIgital front-end for Sensing (DIFFS) (Figure 2.3(b)). Both these ICs are low-power and flexible and targeted towards implementation of a cognitive radio as a mobile device. SCALDIO is a fully reconfigurable analog transceiver that features an FFT accelerator core, enabling the engine to perform spectral analysis. The receiver RF operating frequency is programmable from 0.1 to 6 GHz and the channel bandwidth is programmable between 1 and 40 MHz. Both the DIFFS processor and the ADC are running at 40 MHz. However, in our configuration the ADC is down-sampled to 20 MHz, resulting in the available channel bandwidth of 20 MHz.



(a) Scaldio 2b (b) Spider v2.0 (DIFFS)

Figure 2.3: imec SE components

Hardware-wise the IMEC SE was designed to be able to process samples real-time. As the measurement collection and processing procedures can run in parallel, we just have to make sure that processing finishes before new data is available, in order to provide for real-time sensing. The IMEC SE firmware that is used in our experiments is able to perform all required calculations, before the next vector of samples that needs to be processed is available and thus is capable of real-time sensing.

2.1.4 Atheros AR9380

The Atheros AR9380 [5] is a single-chip, dual-band (2.4/5 GHz), 802.11n compatible chipset that supports up to 3x3 MIMO transmissions, offering both high throughput performance, along with low power consumption in every operational state (Figure 2.4). The special characteristic of AR9380 is its ability to support spectrum sensing capabilities. The Ath9k [4] driver enables the user to configure the card, in order to sense the spectrum over a specified set of channels and derive the Power Spectral Density with an FFT resolution of 56 bins. The 56 bins correspond to the 52 usable plus the 4 pilot sub-carriers that are supported by the 802.11n protocol in the case that the 20 MHz bandwidth configuration is applied. The FFT operation is executed on the chipset and scanning results that correspond to 4μs long snapshots of the spectrum are reported per frequency at the fixed period of 55 ms. The channel switching delay was measured at 1 ms for in-band and 2 ms for out-band transitions.

Figure 2.4: Atheros AR9380 chipset

The channel switching overhead of the imec SE and AR9380 is obtained through the device specifications that are provided by imec and Atheros accordingly, while such values are not officially reported for the USRP devices and are derived through our own measurements. We experimentally verified that maximum configurable sampling rate equals 100 KSps. The driver provides an interface to configure both the actual sampling rate through adaptation of the decimation level at 255 levels, as well as the number of result sets that will be reported within the 55 ms interval. Although the AR9380 is able to sample at a sufficient sampling rate, this device is not able to provide for real-time sensing, as continuous sampling can only last up to 55 ms, upon which interval the device goes back to regular Wi-Fi operation.

The full set of specifications of the devices under investigation are represented in the following table:

Sensing device	Host requirement	Channel bandwidth (MSps)	ADC Rate (MHz)	Real-time sensing
USRP N210	Yes	25	100	Host-dependent
USRP E110	No	5	64	Host-dependent
imec SE	No	20	65	Yes
AR9380	Yes	20	40	No

Table 2.1: Hardware characteristics of sensing devices

2.2 Developed Hardware

This section describes the developed hardware components, which allowed us to monitor the power consumption of the sensing devices presented in section 2.1. Two hardware components have been developed: the Advanced Chassis Manager Card and the appropriate Power Adapters for the sensing devices under investigation.

2.2.1 Advanced Chassis Manager Card (ACM Card)

Towards the direction of implementing a small device capable of monitoring power consumption of Wireless Interfaces of each testbed node we have utilized the above components:

- Arduino Mega2560
- Arduino Ethernet Shield
- Texas Instruments INA139 current monitor
- Custom shield

Arduino Mega2560 [3] features the 8-bit on-board micro-controller Atmega2560 [6] running at 16MHz and integrates a 16 channel Analog to Digital Converter (ADC), which provides 10-bit resolution for each channel (i.e. 1024 different values).

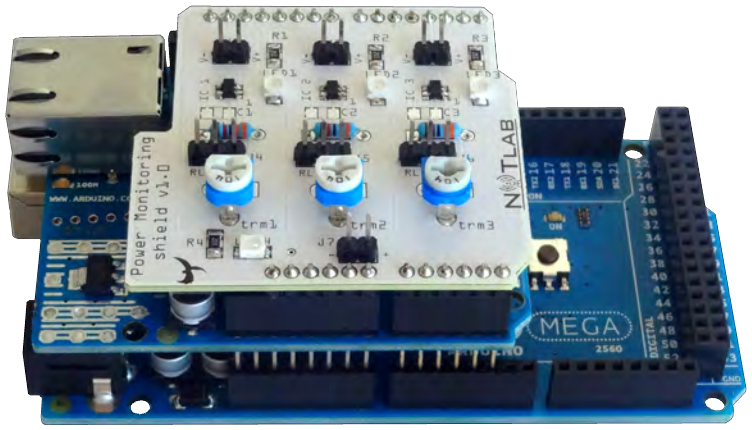


Figure 2.5: Developed ACM Card

Arduino Ethernet Shield [2] is aiming to enable network communication. It features Wiznet W5100 [31] Ethernet controller, which provides a network (IP) stack capable of both TCP and UDP communications. A key characteristic of this shield is that features a micro SD socket which we used to store the captured data.

Since the integrated ADC is not able to accurately digitize the attained voltage levels on the shunt resistor, in cases where the monitored voltage drop minimally varies (mV range), we equipped the developed card with the Texas Instruments INA139 [23] Integrated Circuit. INA139 is a high-side current-shunt monitor that converts a differential input voltage to an amplified value.

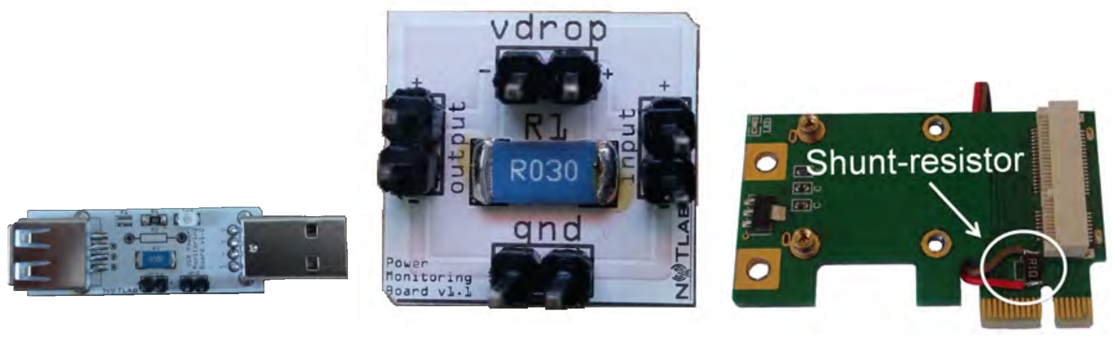
The developed platform that incorporates both Arduino-like boards and the fabricated PCB is illustrated in the Figure 2.5 and can achieve sampling rate of 63KHz with 10-bit resolution after modifications in the software, while only reducing the perceived accuracy by approximately 11%. More information for the NITOS ACM card you can find in our previous work [13].

2.2.2 Power Adapters

Due to the fact that each device has different power consumption, the appropriate measurements had to be taken in order to decide the proper value of the current-shunt

	USRP N210	USRP E110	imec SE	Atheros 9380
Rl	33 kOhm	33 kOhm	33 kOhm	22 kOhm
Rs	0.03 Ohm	0.03 Ohm	0.1 Ohm	0.1 Ohm
Max considered Power	12.872 W	12.872 W	2.97 W	3.1 W

Table 2.2: ACM card configurations per considered device



(a) imec SE adapter (b) USRPs adapter (c) AR9380 adapter

Figure 2.6: Power adapters

resistor for each one. Table 2.2 presents the power configurations per considered device.

We designed power adapters for the USRPs and imec SE and we modified a pci-e to mini pci-e adapter for the Atheros AR9380 by placing in series with the power supply pins a current-shunt resistor as shown in Figure 2.6(c). The values of the current-shunt resistor for every sensing device are dependent on the consumption of each device and are mentioned in Table 2.2.

CHAPTER III

Proposed framework

In this chapter we analyze the methodologies that were followed in order to gather the corresponding power consumption and sensing delay measurements. Considering the different evaluation metrics, we present extensive details about the developed tools that were used on the sensing devices under investigation.

3.1 Power Consumption Evaluation

In order to collect power consumption measurements, we need to measure the power needed for specific operations, such as the measurements collection and processing procedures per platform. Power consumption per specific operation can be determined by direct measurement of the input voltage and current drawn at the device under test. The current measurement can be obtained by inserting a high precision and low-impedance current-shunt resistor (R) in series with the measured device. Actual measurements can be taken using a digital oscilloscope as presented in Figure 3.1

By consistently measuring the voltage ($V_R(t)$) across the current-shunt resistor through proper voltage metering equipment, we are able to extract the instantaneous current draw of the device, based on Ohm's law. The instantaneous power consumption can be calculated as the product of the input voltage V_{IN} and the measured

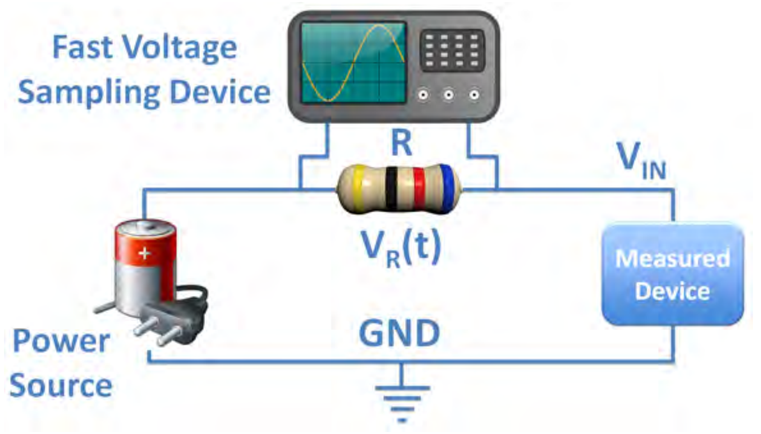


Figure 3.1: Representation of Power Measurement setup

current draw (Equation 3.1)

$$P(t) = V_{IN}\frac{V_R(t)}{R} \tag{3.1}$$

Estimation of the total energy consumption during a specific experiment, necessitates the accurate sampling of the instantaneous power consumption during the total experiment duration. Total energy consumption can be calculated as the integral of the power consumption over the specified duration ($Dt = t_1 - t_0$), as follows

$$E(Dt) = \frac{V_{IN}}{R}\int_{t_0}^{t_1} V_R(t)dt \tag{3.2}$$

However it should be made clear that through the voltage sampling equipment, only a finite number of samples of $V_R(\cdot)$ are acquired over $[t_0, t_1]$ at discrete time instances.

For this purpose, we use the developed NITOS ACM card (Section 2.2.1) as the fast voltage sampling device and measure voltage drop on resistors that are attached in series with the power supply of each device.

3.1.1 Arduino Software

In order to control the NITOS ACM card and send information such as the duration of sampling, the name of the produced file, we developed a tiny Web Server that is based on the Arduino Ethernet Library and operates on each individual card. In addition, all the cards are connected to a switch and then to a server. The web server is responsible to hadle the incoming requests and interpret them in terms of the developed functions on the NITOS ACM card. (More details on the **Appendix**) We choose the ethernet interface so as to be able to control it remotely and without using serial communication, which means running extra applications and slower exchange of informations between the server and the NITOS ACM cards. Moreover we developed an FTP client as a functionality of the card so that the collection of measurements can be centralized without further communication activities.

The flowgraph of the NITOS ACM cards is:

- Receive request for sampling for X seconds the pin Y and save the measurements to a file on the SD card with the name Z
- Sampling process
- Receive request to upload the file with name Z to the FTP server
- FTP upload process

3.1.2 Power Consumption Software

Towards providing for an automated procedure for the NITOS ACM cards, we developed a set of Python scripts that provide direct access to the collected results and precise power and energy consumption calculations. After the appropriate definitions by the user (*card ID, seconds, pin, name*), the Python script takes over to trigger the selected ACM cards to start sampling. When the acquisition in the ACM cards finishes, the script triggers their FTP clients to send the measurements to the FTP server. The Python script waits until all the ACM cards upload their files and then makes the appropriate calculations for power, average power and energy. Consecutively another Python script plots the aforementioned results separately for each ACM card. The same code for the calculations of power consumption is developed in Matlab. (See **Appendix** for the code)

3.2 Sensing Delay Evaluation

Spectrum Sensing Delay is related to the time duration required, by each different cognitive solution, to determine the presence of active transmitters and also to quantify the spectrum utilization on a specific frequency. Under the term of Spectrum Sensing Delay, we group the time required by each different step of the sensing procedure, thus including time spent during the gathering of required number of samples, the subsequent measurement processing and handling and the final decision making procedure. The total amount of time spent in processing and measurement handling translates to the lost capture time opportunities and therefore to intervals during which the spectrum is not being monitored. As a result, constant monitoring of sensing delay provides a real-time estimation of the detection performance during the experiment execution. Due to the fact that the whole wireless environment is stochastic and uncertain in terms of spectrum occupancy and channel conditions, online performance monitoring is required instead of comparison with performance yielded in reference scenarios.

Overall sensing delay is composed of various parts that result due to different components of each device. As a first step, all devices have to be configured properly based on the specifications of the sensing operation that will be executed. Second, the actual spectrum sensing operation also needs adequate time to be completed. Third, the time spent for transferring and storing of measurements is also not negligible and should be quantified. Next, as the measurements acquisition has been completed successfully, the next step is to process the collected measurements. Duration of this task is mainly dependent on the processing capabilities of each sensing solution. Finally, we also have to consider the dimension of multiple channels that need to be monitored sequentially, as the overall sensing delay is also affected by the channel switching delay.

Ideally, the exact amount of time that is spent in each subprocess should be calculated for each device. We managed to derive the Sensing Delay Distribution for the USRP devices, by incorporating high precision timer functions at appropriate parts of the UHD source code. However, such low level measurements cannot be reported for the imec SE and the AR9380 devices, where the different subprocesses run in hardware. For these devices, we are only able to measure the total induced sensing delay and compare it with the duration of the actual sampling phase.

3.2.1 Sensing Delay Software

The main aim of the Sensing Delay evaluation software, is to present the distribution of the Total Sensing Delay among the various subprocesses that consitute it and more specifically the configuration, sensing, processing and transferring of measurements and finally the channel switching where applicable. Through this approach, we would be able to better understand whether the delays are caused by the actual hardware or the software controlling the operation of each sensing engine. To this aim, we followed the simple and very effective solution of incorporating timestamps in the source code of the driver that controls the operation of each sensing device. (See **Appendix** for the code)

We discriminate the following ***sub processes*** under the Sensing Delay definition:

- **Initial configuration time**: This interval corresponds to the time required for each device to be properly configured, since it is powered up for the first time. It is calculated based on timestamps for every device except from imec SE in which occasion we refer to [21],[10] for deriving the exact values.

- **Duration of sensing one frequency**: The actual time spent for spectrum sensing. In our evaluation, we consider a common fixed interval for all the considered sensing devices. (See 4.3.1 [Experiment 3 - Phase 1])

- **Duration of sensing multiple frequencies**: Respectively, this interval corresponds to a multiplication of the fixed sensing duration by the number of required channel switchings that need to be performed in order to sense the requested bandwidth. (See 4.3.2 [Experiment 3 - Phase 2])

- **FFT processing time**: The time required by a given processing unit, so as to produce the FFT result, considering as input the spectrum samples collected from the corresponding devices. This time is dependent on the underlying hardware in which this process runs. For USRP N210 we use the host pc that it is attached on, while for the USRP E110 we use the embedded processor that it features. On the other hand Atheros AR9380 and imec SE perform these calculations on hardware.

- **Transferring time**: The time each device needs for transmitting the sampled data over a communication bus, plus the time required for storing the resulitng measurements. The USRPs N210 and E110, make use of the gigabit Ethernet and SD card respectively, while the imec SE uses the USB interface. Atheros AR9380 stores values in a specific file format (proc files [22]) that is stored in the RAM of the host PC. These times are reported with the aid of timestamps placed in the corresponding part of the source code. The imec SE does not spend any time for transferring of measurements, as they implemented that to be in parallel with the next sensing.

- **Channel switching time**: The time required by each device to configure the next frequency for sensing and it is reported by timers for all devices, except for the case of the imec SE where we rely on the data reported in [21],[10].

CHAPTER IV

Experimental Evaluation

In this section we present detailed results that have been collected through experimentation under various scenarios and different topologies. The following experiments have all been executed remotely in the NITOS and w-ilab.t testbeds and clearly demonstrate the extended experimentation capabilities of the framework. The full list of code and scripts used in our experiments are included in the **Appendix**.

The conducted experiments are presented in order of experimentation complexity. More specifically, we first start by presenting simple and then proceed with more sophisticated experiments. We start with a basic topology that consists of only Wi-Fi nodes, to illustrate how the energy consumption of a wireless network is related to the bit rate of the transmitting nodes. We proceed by presenting results collected from 2 spectrum sensing devices, while characterizing the PSD of a traffic flow generated between 2 Wi-Fi nodes that are transmitting in the 2.4 GHz band. In parallel with this operation we monitor the power consumption of the sensing devices.

Next, a more complex scenario is investigated, where the 4 sensing devices are used simultaneously to detect a Wi-Fi signal. Apart from characterizing their power consumption, we also focus on the distribution of the overall sensing delay among the various sensing sub-processes. Finally, based on the same experimental setup, we present a second phase of experiments, which compare performance while the

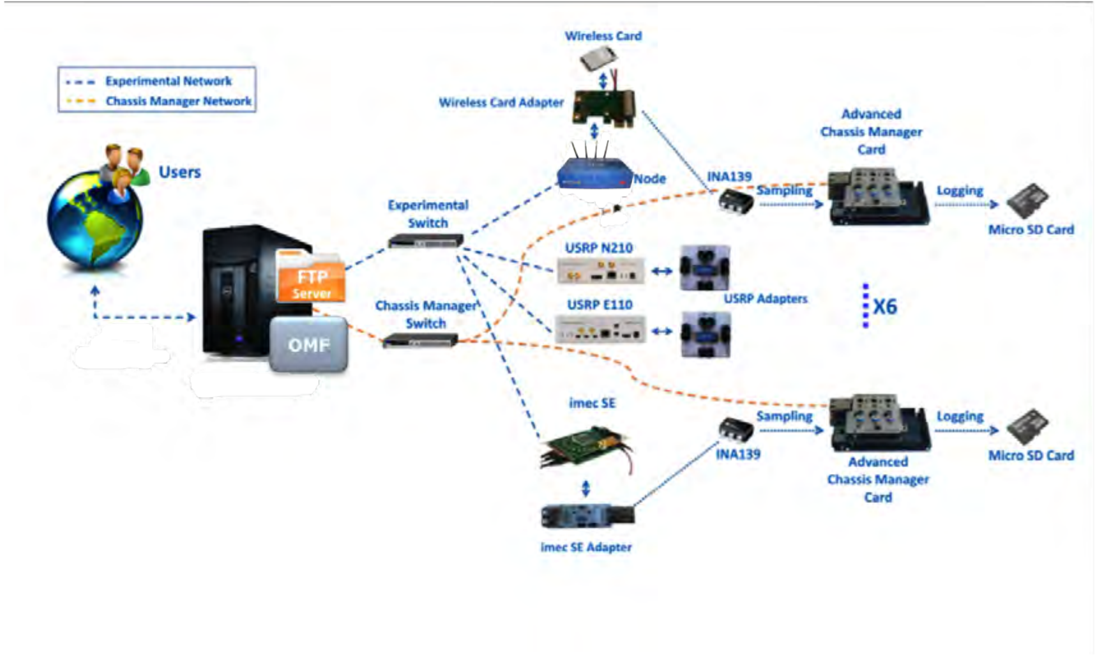


Figure 4.1: NITOS architecture

devices monitor the whole 2.4 GHz band that has total bandwidth of 80 MHz. As the bandwidth of the targeted band exceeds the maximum bandwidth of all considered sensing devices, in this scenario all devices have to perform channel switching to provide for proper monitoring and as a result the impact of the channel switching delay is highlighted.

Figures 4.1, 4.2 depict the architectures of NITOS and w-iLab.t testbeds after the deployment of NITOS ACM cards and the power adapters which used for the experimental evaluation of our framework.

4.1 Experiment 1

In this first experiment, we characterize the energy consumption of two wireless NICs, while network traffic is transferred under varying Application-layer Traffic rates through the use of the Iperf [11] traffic generator. More specifically, 2 w-ilab.t testbed nodes (ZotacM18, ZotacM20) are employed, in order to establish a wireless link in

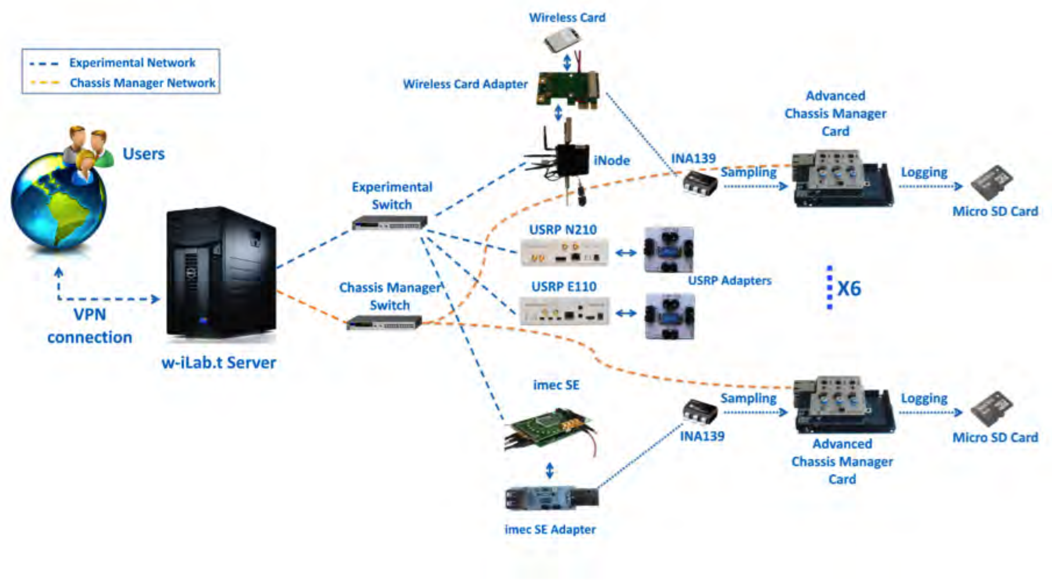


Figure 4.2: w-iLab.t architecture

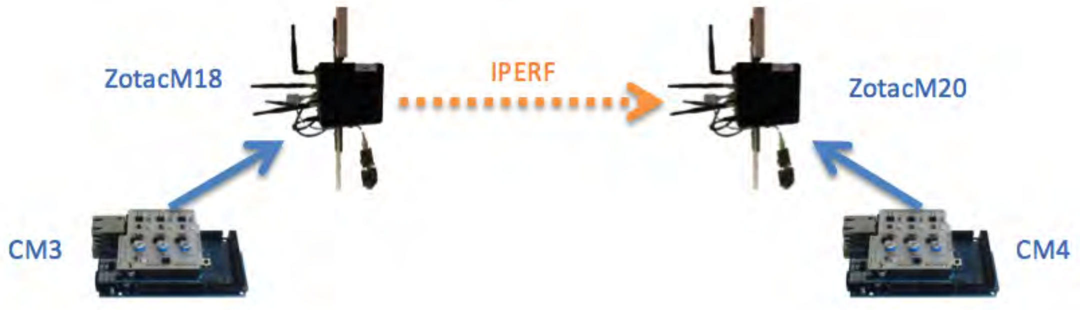


Figure 4.3: Experiment 1 - Experiment Setup

ad-hoc mode. In addition, we use the two NITOS ACM cards (CM3, CM4), which are attached in each one of the wireless NICs of the testbed nodes. In Figure 4.3, we depict the experimental setup in this experiment.

This experiment is executed in four phases. In the first phase, we configure the ZotacM18 to transmit to ZotacM20 at the Application layer Traffic Rate of 10 Mbps for 10 seconds. Next, during an interval of 5 seconds, no transmissions are taking place, while in the third phase we again activate the traffic flow for 10 seconds and

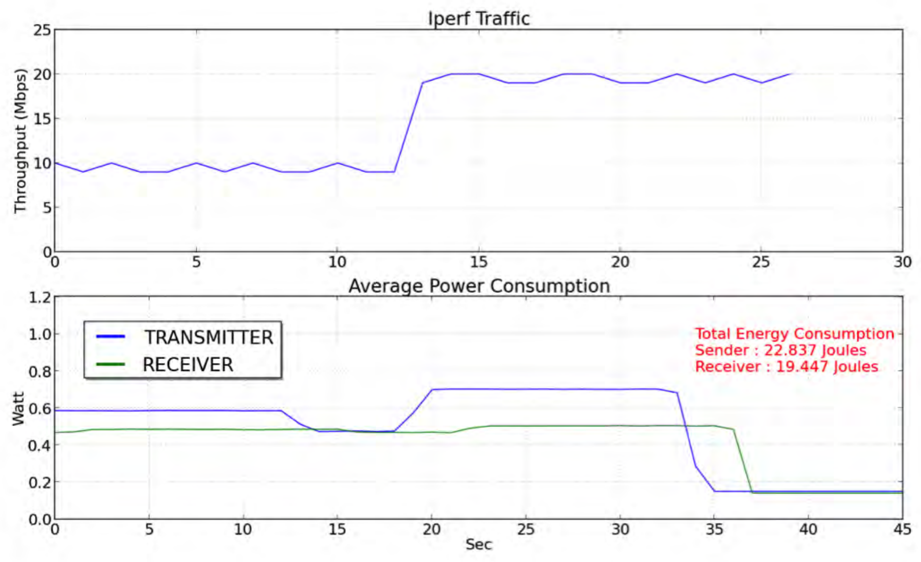


Figure 4.4: Experiment 1 - Power Consumption Results

set the Traffic Rate equal to 20 Mbps. In the final fourth phase, the wireless cards are deactivated and the experiment ends. Figure 4.4 plots the throughput performance and the resulting energy consumption.

We clearly observe that the average power consumption of the transmitter node (*Blue line*) is significantly higher (0.2 W), in comparison with the receiver node (*Green line*). Moreover, we notice that the operation of reception does not add much to the average power consumption, when compared with the consumption in idle mode.

4.2 Experiment 2

In this second experiment, we employ 2 spectrum sensing devices and more specifically the USRP N210 and the IEEE802.11n compliant Atheros 9380 chipset, in order to characterize the PSD of a traffic flow that is generated between 2 Wi-Fi nodes transmitting signals of 20 MHz bandwidth on channel 5 (frequency 2432 MHz) in the 2.4 GHz band. The Wi-Fi link is again established in ad-hoc mode, between the

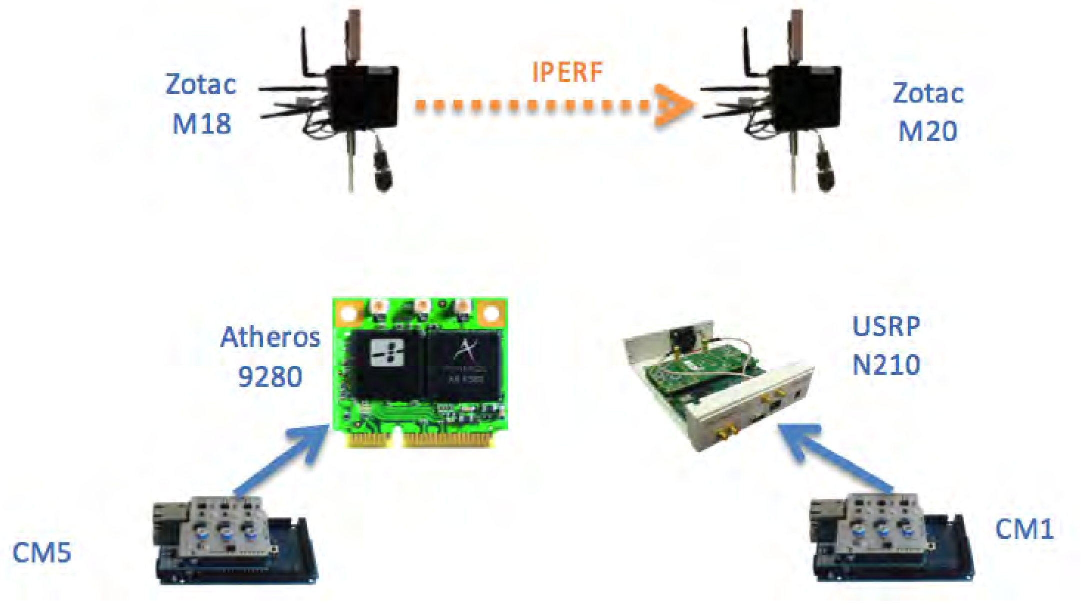


Figure 4.5: Experiment 2 - Experiment Setup

ZotacM18 and ZotacM20 nodes in w-iLab.t testbed.

In parallel with the PSD evaluation, we use the two NITOS ACM cards (CM5, CM1), which are attached in each one of the spectrum sensing devices, in order to measure the energy consumption in the spectrum sensing procedure. Figure 4.5 illustrates the experimental setup that was configured in w-ilab.t testbed.

In Figure 4.6, we plot the results related to the PSD and Power consumption evaluation that have been collected through this experiment. We observe that both the USRP N210 and the AR9380 devices are able to detect that 20 MHz of bandwidth are utilized by the Wi-Fi link, while the perceived frequency resolution varies between the 2 devices. On the other hand, regarding to the energy consumption performance, we clearly note the commercial Wi-Fi chipset offers a much more energy-efficient consumption profile, while performing spectrum sensing, in comparison with the research oriented USRP N210 device.

Next, we establish again the same experimental setup for the Wi-Fi link, while we configure the two sensing devices to monitor channel 7 (frequency 2442 MHz).

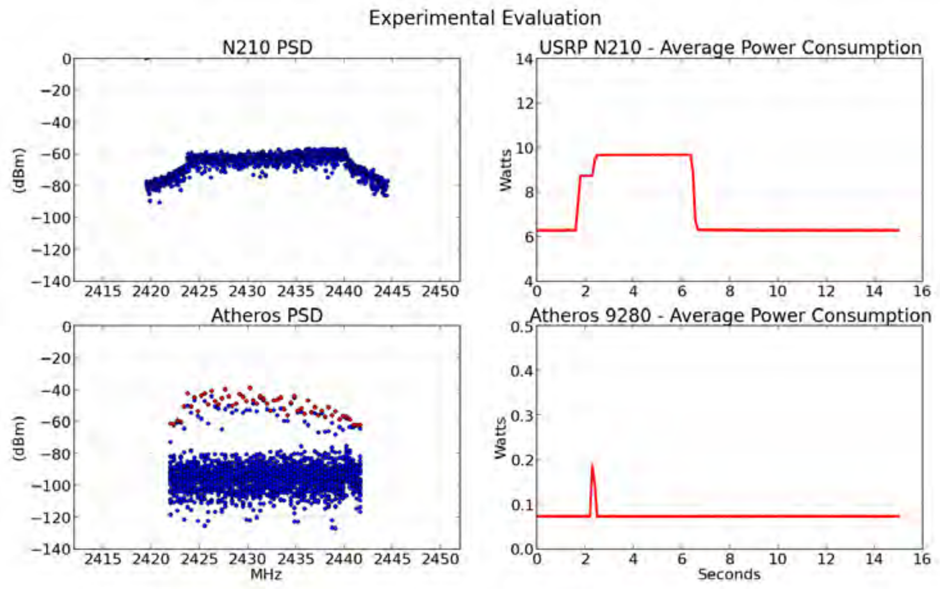


Figure 4.6: Experiment 2 - PSD and Power Consumption Evaluation

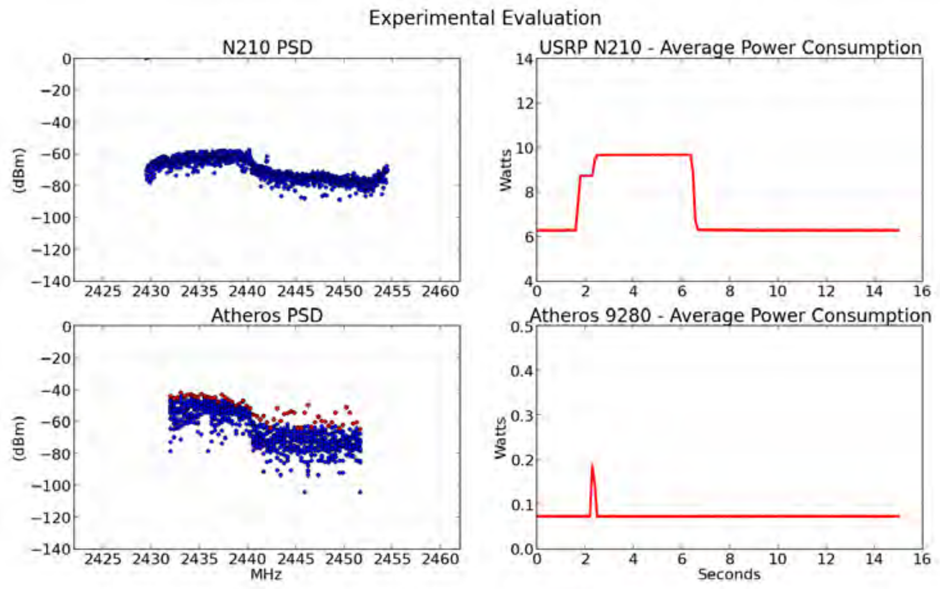


Figure 4.7: Experiment 2 - PSD and Power Consumption Evaluation Transmissions on overlapping frequencies

Through this experiment, we aim at evaluating the overlapping of channels in the 2.4 GHz band. In Figure 4.7, we observe that the PSD representation for the 2442

MHz frequency varies between -60 dBm and -80 dBm, clearly illustrating the level of overlapping between transmissions that occur in adjacent channels of the 2.4 GHz band.

4.3 Experiment 3

4.3.1 Phase 1 Monitoring a single channel of the 2.4 GHz band (20 MHz bandwidth)

In this experiment, we configure all the available spectrum sensing platforms to operate in parallel, in order to detect a signal generated by a pair of Wi-Fi nodes. The communicating pair of nodes consists of one transmitter and one receiver w-ilab.t testbed nodes that communicate over a single channel of the 2.4 GHz band. The bandwidth of the transmitted signal is 20 MHz corresponding the standard 802.11 channel bandwidth. Figure 4.8 illustrates the experimental setup in w-ilab.t.

In order to provide a fair comparison of the different sensing devices, we had to design an experimental scenario that would allow all devices to be tested under their operation limits. Having properly investigated the operation and capabilities of each different device, we reached the conclusion that the only hardware dependent specifications that could restrict the design of the experimental scenario would be related to the imec SE and the Atheros 9380 devices, as the USRP devices are fully configurable through software and thus generate no restrictions. In addition to the fact that the imec SE is the only device able to provide for real-time spectrum sensing, we decided to build a proper scenario that would allow for a fair comparison of the rest devices with the high-end imec SE.

Based on in-depth analysis of the device specifications that were analyzed in the introductory section, we concluded that the scenario would be designed on top of the imec SEs hardware restrictions and more specifically, on the 20 MHz sampling rate

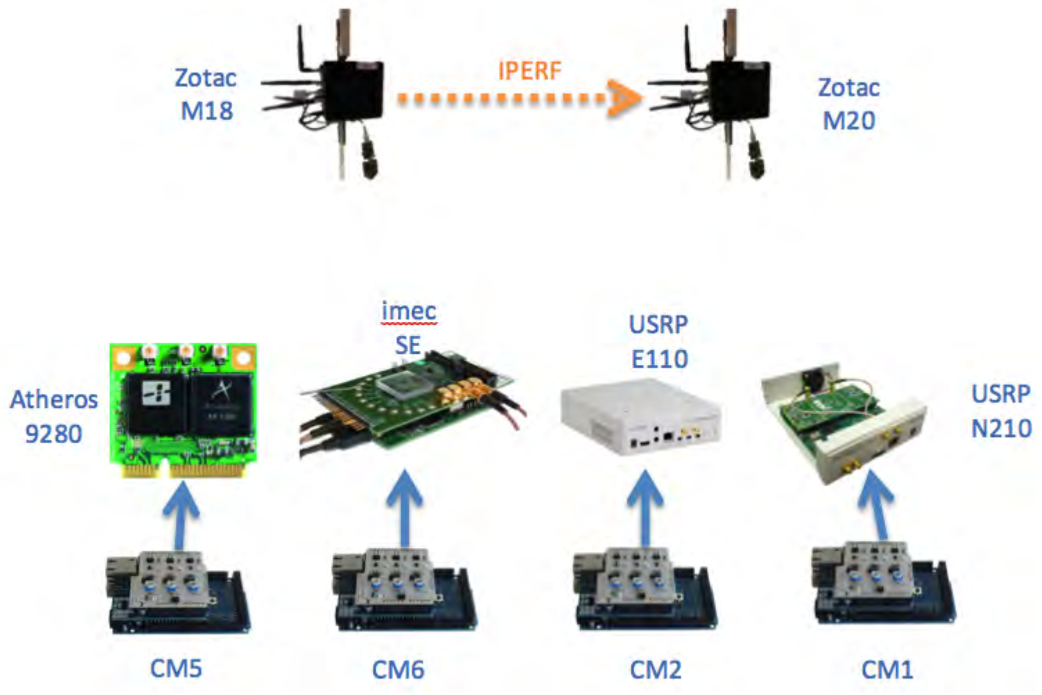


Figure 4.8: Experiment 3 - Experiment Setup

and the 128 bin size of FFT calculations. When the imec SE is sampling at this rate, it requires 6.4 s to gather 128 samples that are further fed to the on-chip FFT accelerator core for FFT processing, which finally provides the PSD results. The interval of 6.4 s is extremely short, in comparison with the interval of 27 us, which corresponds to the shortest transmission of 802.11n frames of 1500 bytes length at the maximum configurable Physical-layer rate of 450 Mbps. Based on this observation, we further decided to combine several FFT measurements before reporting a single sensing result and thus we use Max-Hold filtering over 10 consecutive FFT operations. This results in a total sensing time of 64s, within which interval we are able to detect the presence of 802.11n transmissions. In order to provide for a proper evaluation and comparison setup, we decided to configure all the devices to sense for the same interval of 64s. Having specified a common monitoring interval, the different devices will sense the medium for the same duration and collect a different number of samples

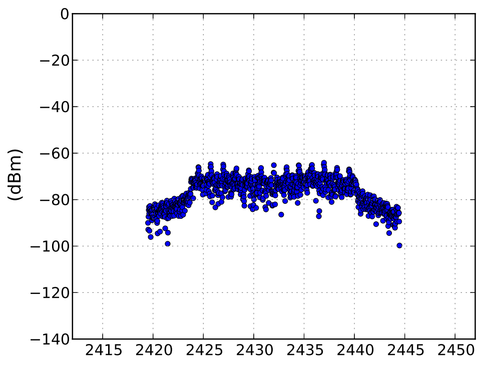
Sensing device	FFT resolution	Result sets	Collected samples
USRP N210	1024	1	1600
USRP E110	256	1	320
imec SE	128	10	1280
AR9380	56	6	336

Table 4.1: Sensing characteristics of sensing devices in the considered scenarios

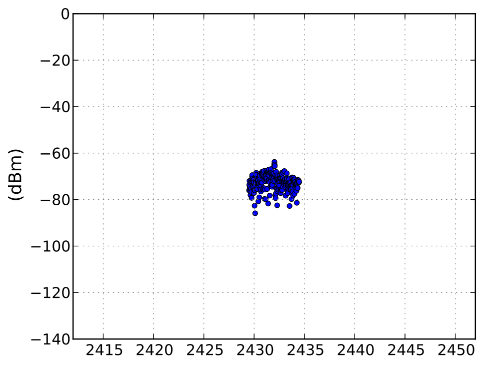
as specified by their sampling rate configurations. In Table 4.1 are listed the various characteristics as configured for each device in the considered experimental scenarios.

In Figure 4.9, we plot results related to the PSD evaluation that have been collected through the execution of the specified experimental scenario. We clearly observe how the maximum bandwidth and FFT-bin size specifications of each device affect the sensing efficiency. The imec SE along with the Atheros AR9380 device detect that the 20 MHz bandwidth are fully utilized, while the limited USRP E110 is able to monitor only 5 MHz of bandwidth. On the other hand, the USRP N210 is able to detect the drop is PSD when considering frequencies that exceed the central frequency by 10 MHz and thus is the only device able to characterize that the Wi-Fi transmission has the central frequency of 2432 MHz (channel 5).

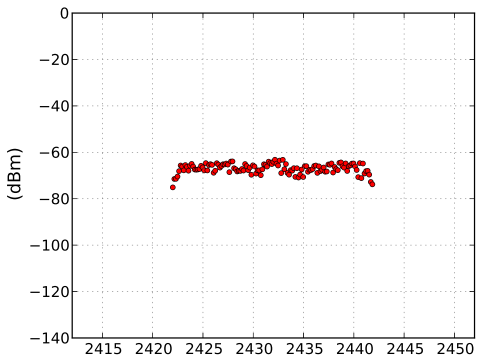
In parallel with the PSD evaluation, we use the four NITOS ACM cards (CM5, CM6, CM2, CM1), which are attached in each one of the spectrum sensing devices, in order to measure the energy consumption during the spectrum sensing procedure. As soon as the devices have been properly configured and the actual sensing procedure takes place, the power consumption of each device reaches a stable level, which equals 9.68 W for the USRP N210, 10.54 W for the USRP E110, 2.39W for the imec SE and 0.57 W for the AR9380 accordingly. As shown in Figure 4.10, in general, the commercial Atheros AR9380 Wi-Fi chipset present the best energy efficiency, while the embedded prototype imec SE is also characterized by a low power consumption profile, in comparison with the energy harvesting research oriented USRP devices. This



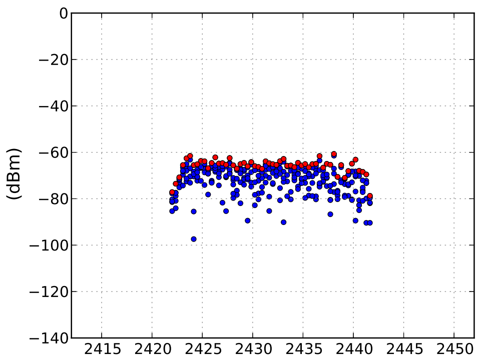
(a) USRP N210



(b) USRP E110



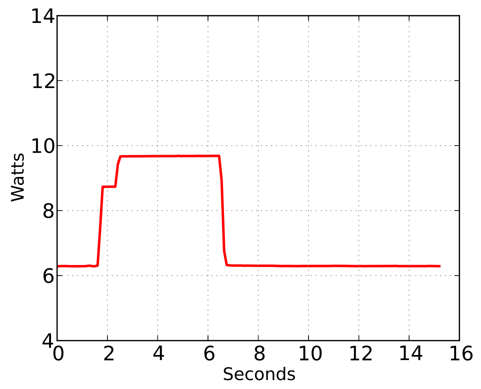
(c) imec SE



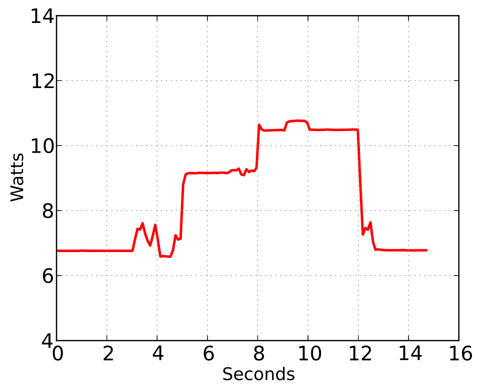
(d) Atheros AR9380

Figure 4.9: Experiment 3 - Power Spectral Density Evaluation

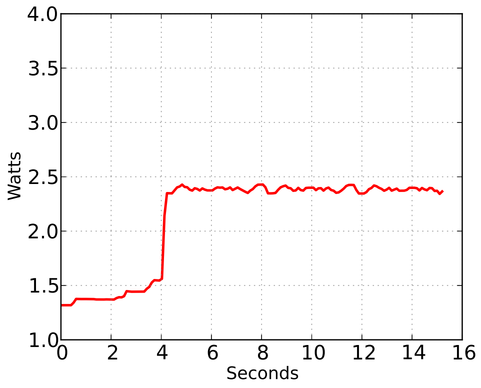
is not the total power consumption due to the fact that there is an extra consumption when the measurements are processed inside the sensing device or on another host machine. Both the imec SE and the AR9380 devices process measurements in hardware and as a result isolation of consumption due to processing cannot be obtained. Regarding the consumption of the USRP devices N210 and E110, we measure the consumption of the ATOM-based (Intel Atom D525, 1.8 GHz) host machine and the embedded processor, during measurements processing. Figures 4.11(a) and 4.11(b) depict the power consumption increase that the processing results in, which equals approximately 1.35 W for the ATOM and 0.4 W for the E110.



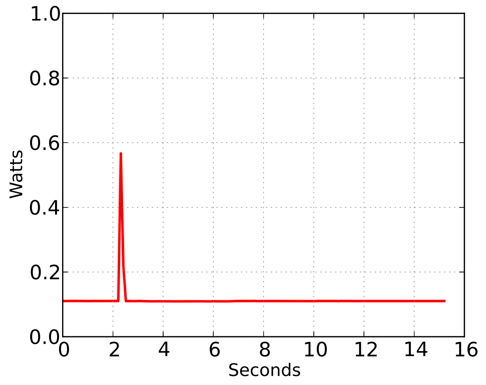
(a) USRP N210



(b) USRP E110

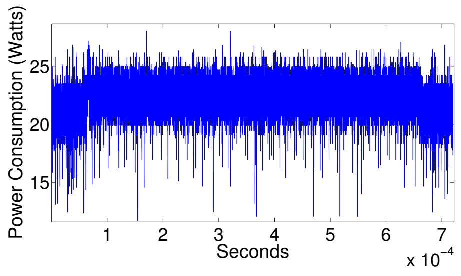


(c) imec SE

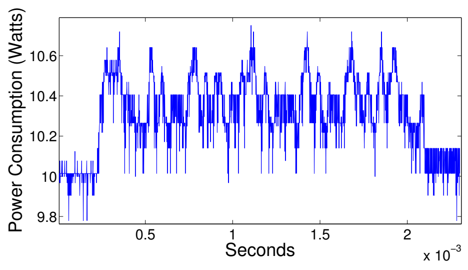


(d) Atheros AR9380

Figure 4.10: Experiment 3 - Power Consumption Evaluation



(a) USRP N210 - ATOM



(b) USRP E110

Figure 4.11: Power Consumption during processing of measurements collected over the 64μs interval

Next, we present the results collected by the Sensing Delay evaluation framework for all the considered devices under the aforementioned experimental scenario.

imec SE

Due to the fact that all the computations are made in the hardware, we are not able to distinguish between the different processes, so we rely on the data reported in [21],[10]:

- **Initial Configuration** is fixed to 50s
- **Common Sensing Time on a single frequency (20MHz)** is fixed to 64μs
- **Sensing on five different frequencies (80MHz) for 64s each** lasts 192μs
- **Transferring and Storing of collected measurements**: As a limited amount of spectrum measurements (128 for 20 MHz of bandwidth) can be stored in the internal memory of the imec SE, measurements transferring and storing does not induce any measurable overhead in the overall Sensing Delay. To this aim, we remark that it is able to operate in real-time sensing mode.
- **FFT processing** takes 34μs but it is not caused an overhead because it occurs in parallel with the configuration for the next frequency.
- **Channel switching** occurs in 50μs

USRP N210

In the USRP N210 we are able to identify all the different processes that constitute the total Sensing Delay, including source code of UHD driver and the code that we developed for the FFT computation. We measured the duration of all subprocesses, by including timestamp calculations in the source code. Our findings for the USRP N210 are:

- **Initial Configuration setup** takes 3.2 seconds
- **Common Sensing Time on a single frequency (25MHz)** is fixed to 64μs
- **Sensing on four different frequencies (80MHz) for 64s each** lasts 192μs
- **Transferring data (for one frequency) over gigabit Ethernet** lasts 79μs
- **FFT processing with 1024 FFT bins** lasts 77.91μs
- **Channel switching** occurs in 51.2ms

USRP E110

The differences with the USRP N210 are that the transferring of the data and the FFT processing occurs inside the E110 without the presence of a host PC. In addition, the configuration phase differs from the N210 due to the fact that extra parameters have to be configured, such as the FPGA clock. Our conclusions for the E110 follow:

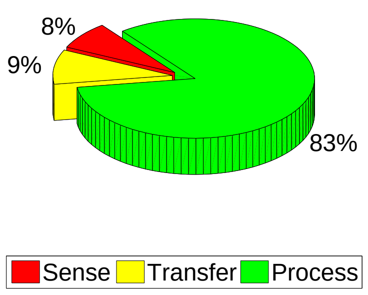
- **Initial Configuration setup** takes 5.5 seconds
- **Sensing one frequency (5MHz) for 64s** lasts 64μs
- **Sensing twenty frequencies (80MHz) for 64s each** lasts 960μs
- **Transferring data (for one frequency) to SD card** lasts 58μs
- **FFT processing with 256 FFT bins** lasts 734μs
- **Channel switching** occurs in 55.74ms

ATHEROS 9380

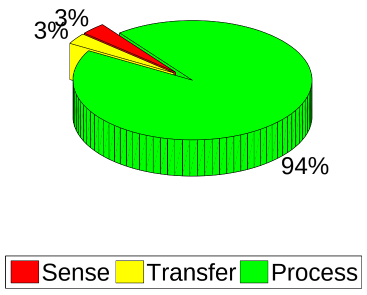
Some specific Atheros cards can perform the operation of spectrum sensing using the compat/backports [4] open source drivers (AR9380 is included). By specifying the files that contain functions associated with the spectrum sensing mechanism, we were able to isolate the different processes of the Sensing Delay and below we present the time spent by each one:

- **Initial Configuration setup** takes 20ms
- **Sensing one frequency (20MHz) for 64s** lasts 64μs
- **Sensing five frequencies (80MHz) for 64s each** lasts 192μs
- **Transferring data to proc files** is included to 55ms in which the AR9380 performs the spectrum sensing
- **FFT processing with 56 FFT bins** is included to 55ms in which the AR9380 performs the spectrum sensing
- **Channel switching** occurs in 1.723ms

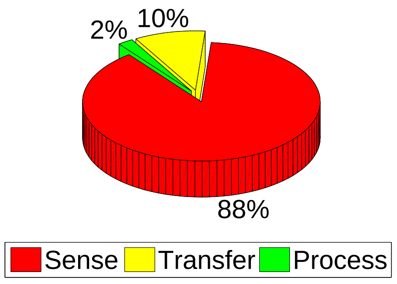
Figure 4.12 illustrates results related to the Sensing Delay evaluation of the 4 sensing devices in the specified experimental scenario. First, considering the configuration time, we remark that the USRP devices require significant amount of time prior to establishing an operational setup. Second, we observed that the USRP E110 spent most of the time during measurements transferring, due to the use of SD card for storage of samples instead of the fast Gigabit Ethernet interface of the USRP N210. Regarding to the AR9380 device, we observed that although the sampling procedure can be repeated very fast and more specifically during each symbol transmission (4 μs), the overall sensing procedure requires approximately 55 ms. This limitation comes due to the use of the ath9k driver, which performs spectral scanning only during the standard scanning operation for collection of Beacon frames that are transmitted by neighboring Wi-Fi APs. As the default Beacon transmission interval is 100ms, the ath9k driver configures the default time spent on each available channel



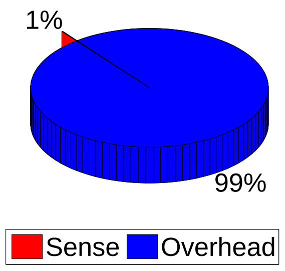
(a) USRP N210



(b) USRP E110



(c) imec SE



(d) Atheros AR9380

Figure 4.12: Experiment 3 - Sensing Delay Distribution during Sensing of a single 2.4 GHz Channel

approximately equal to 55 ms, thus controlling the duration of the spectral scanning procedure as well. Table 4.2 summarizes the times for each process of each sensing device under consideration.

Concluding, we remark that the only device able to approximate the real-time spectrum sensing capabilities of the imec SE, is the USRP N210 that nearly equally spends time between the sensing, transferring and processing sub-processes. Considering a multi-threaded USRP controlling software architecture, the overall sampling procedure could be run in parallel and we aim at investigating this option as part of our future work.

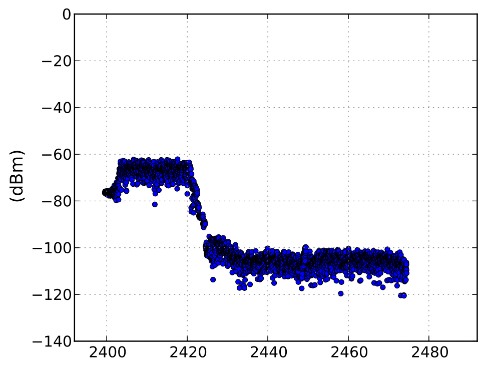
Duration/ Mode	USRP N210 (25MHz BW)	USRP E110 (5MHz BW)	imec SE (20MHz BW)	Atheros 9380 (20MHz BW)
Initial Configuration	3200 ms	5500 ms	50 μs	20 ms
Sensing of 1 Channel	64 μs	64 μs	64 μs	64 μs
Sensing of 80MHz	192 μs	960 μs	192 μs	192 μs
Transfer data of 1 Channel	79 μs	58 μs	-	-
FFT process of 1 Channel	240 μs	1800 μs	34 μs	-
Channel Switching Delay	50 ms	50 ms	50 μs	1.723 ms

Table 4.2: Sensing Delay evaluation results for the different devices

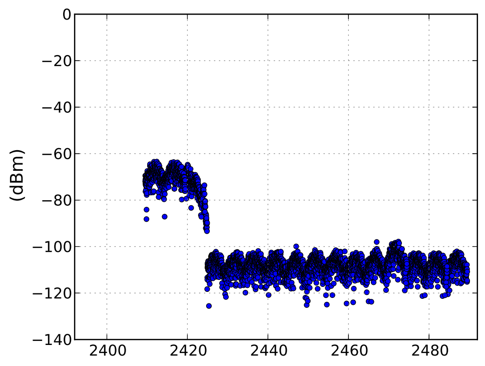
4.3.2 Phase 2 Monitoring the 2.4 GHz band (80 MHz bandwidth)

In this second phase, we consider a more generic scenario, where the targeted band is 80 MHz wide, so that all devices will have to perform channel switching to provide for proper monitoring of the whole band. Under this scenario, we configure the communicating pair of nodes to transmit on various channels within the 80 MHz wide band and evaluate the total sensing delay of all devices, which now further varies due to impact of the channel switching overhead. Based on the bandwidth specification of each device, the USRP N210 (25 MHz) needs to switch its operational frequency 3 times, the USRP E110 (5 MHz) 15 times, the imec SE (20 MHz) 3 times and the AR9380 (20 MHz) 3 times, to sense the whole 80 MHz wide 2.4 GHz band. In this scenario, the imec SE also needs to export the collected measurements, right before switching to the next frequency. As a result the tranferring time is no longer negligible. Considering the exact value of the channel switching overhead per device, we have to rely on specifications provided by the vendor that developed each platform, as the channel switching is a low level operation that cannot be easily isolated for each device. Such specifications are provided by imec and Atheros in the corresponding papers [21],[5].

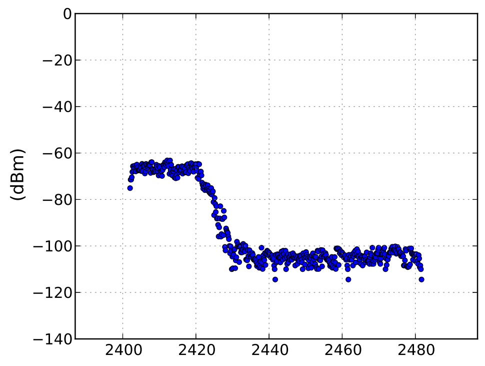
Figure 4.13 plots the PSD, as evaluated by each device. We observe that all devices detect channel 1 (freq. 2412 MHz) to be busy, while the rest part of the



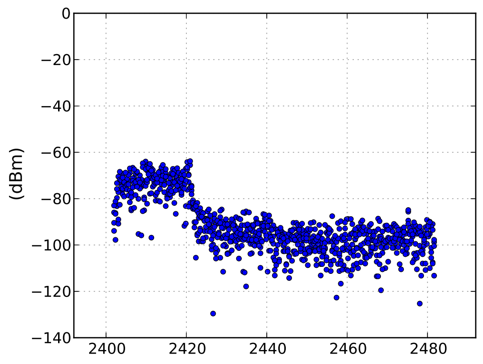
(a) USRP N210



(b) USRP E110



(c) imec SE



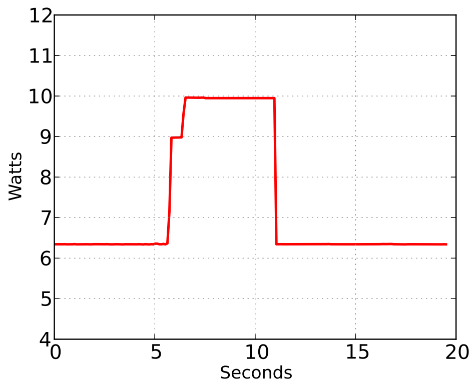
(d) Atheros AR9380

Figure 4.13: Power Spectral Density evaluation during Sweeping of the 2.4 GHz ISM Band

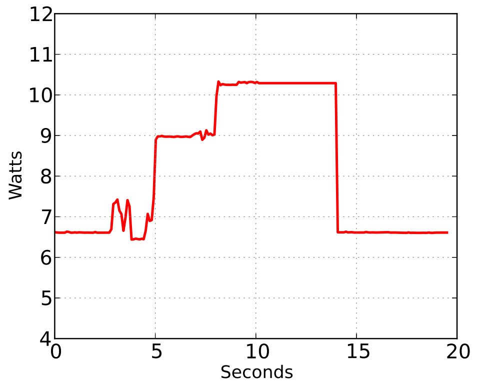
spectrum in the 2.4 GHz is characterized by Energy levels that are close to the Noise level (-90 dBm).

Considering the Power Consumption evaluation, we observe that as the interval of the total sensing procedure is prolonged due to channel sweeping, the resulting energy consumption is increased. Figure 4.14 plots how power consumption is progressively affected by the various steps of spectrum sensing, while also characterizing the total amount of energy spent during the execution of the Phase 2 experiment.

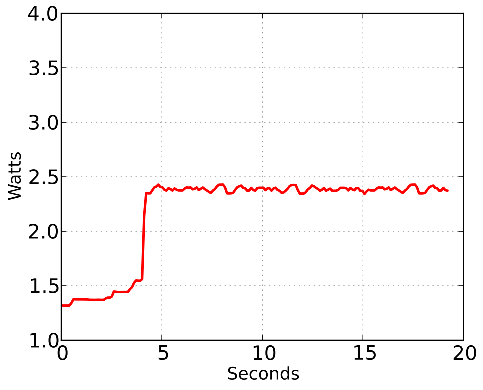
Regarding the impact of the Channel Switching (CS) overhead per device on the



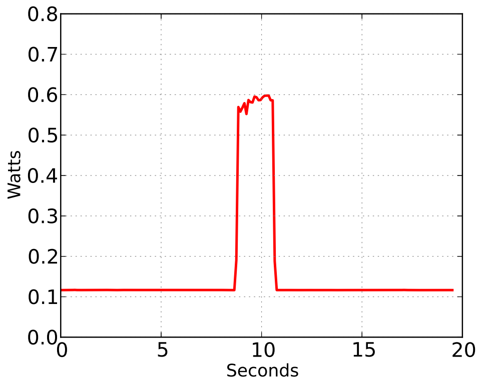
(a) USRP N210



(b) USRP E110



(c) imec SE



(d) Atheros AR9380

Figure 4.14: Power Consumption evaluation during Sweeping of the 2.4 GHz ISM Band

distribution of the total Sensing Delay, we observe that the most affected devices are the USRP ones. This is due to the fact that the CS overhead for the USRP devices is approximately 30 times higher than the overhead induced by the AR9380 (1.723 ms) and 3 orders of magnitude higher than the CS overhead of the imec SE (50 μs). In this case, we remark that the high CS overhead of the USRP N210 makes the device incapable of performing real-time Spectrum Sensing, even in the case that rest procedures are efficiently executed in multi-threaded mode Hence, the reason that consists our solution incapable of performing real-time sensing is the high processing

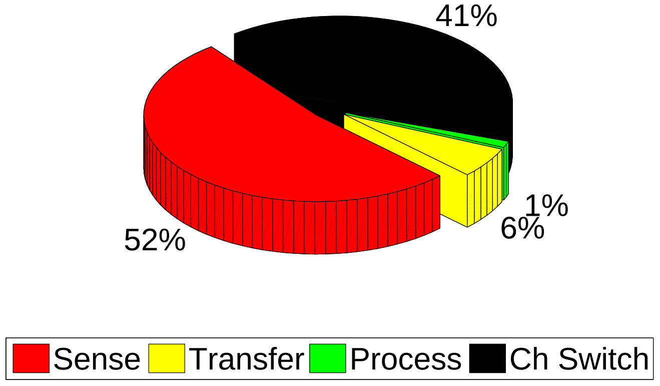


Figure 4.15: Sensing Delay Distribution of imec SE during Sweeping of the 2.4 GHz ISM Band

delay. So only the imec SE results in channel switching overhead values in the order of s that are comparable with the configured sensing interval of 64 μs *3 (for the 80 MHz) thus performing real-time sensing (Figure 4.15).

CHAPTER V

OMF/OML Extension

5.1 OMF

To enable ease of use of the developed framework, we integrated its functionalities into the OMF cOntrol and Management Framework [17]. Based on this integration, experimenters can fully configure the operation of the NITOS ACM cards and moreover collect and access the gathered measurements through the OMF Measurement Library [19].

OMF is a software framework used to manage and execute experiments. With OMF, we write an experiment script and run it using an experiment controller, which tells each of the testbed nodes in an experiment how to configure themselves and what applications to run.

Users on NITOS and w-iLab.t testbeds are able to use this framework during an OMF experiment by providing the necessary configuration for the python script 3.1.2. During the execution of an experiment, OMF instruments the python script along with the other resources the user has specified. More specifically, OMF will trigger the python script to activate the NITOS ACM cards in order to start sampling. After the execution of the user's defined experiment on the nodes, OMF will wait till the process of the data collection from the power consumption framework is finished, prior to terminate the OMF experiment. This way, the experimenter can be informed when the data is available for further analysis and processing.

5.2 OML

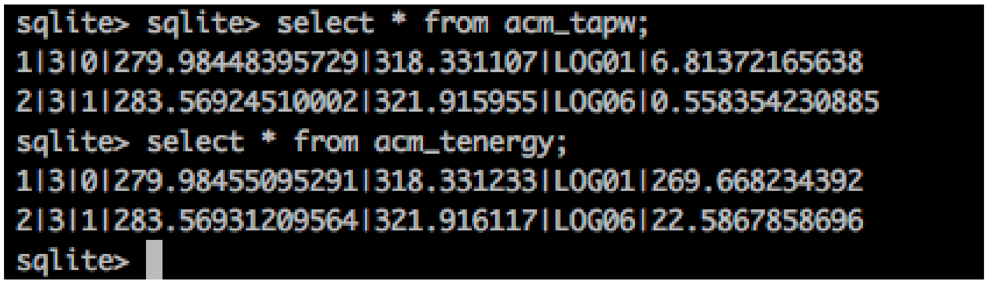
To enable access to the gathered measurements in a transparent way, we enhanced our python script by enabling storing measurements to OML Server through the python module oml4py [1].

OML is a software framework for measurement collection. It gives you a way to collect all the measurement data being recorded by a bunch of devices that are recording some sort of measurement data to a central location, via the network. OML is a generic framework that can be adapted to many different uses. Networking researchers who use testbed networks to run experiments would be particularly interested in OML as a way to collect data from their experiments. In fact, that's why the OML developed in the first place! However, any activity that involves measurement on many different computers or devices that are connected by a network could benefit from using OML. For instance, network monitoring, distributed simulations, or distributed sensor networks.

When the script finishes the calculations, it sends power and energy results for each NITOS ACM card to OML server for storing them into a database. From that database the results are accessible to the experimenter for further processing or illustrating purposes as shown in Figures 5.2 and 5.2. The code for OMF/OML can be found on the appendix.

5.3 OMF Web

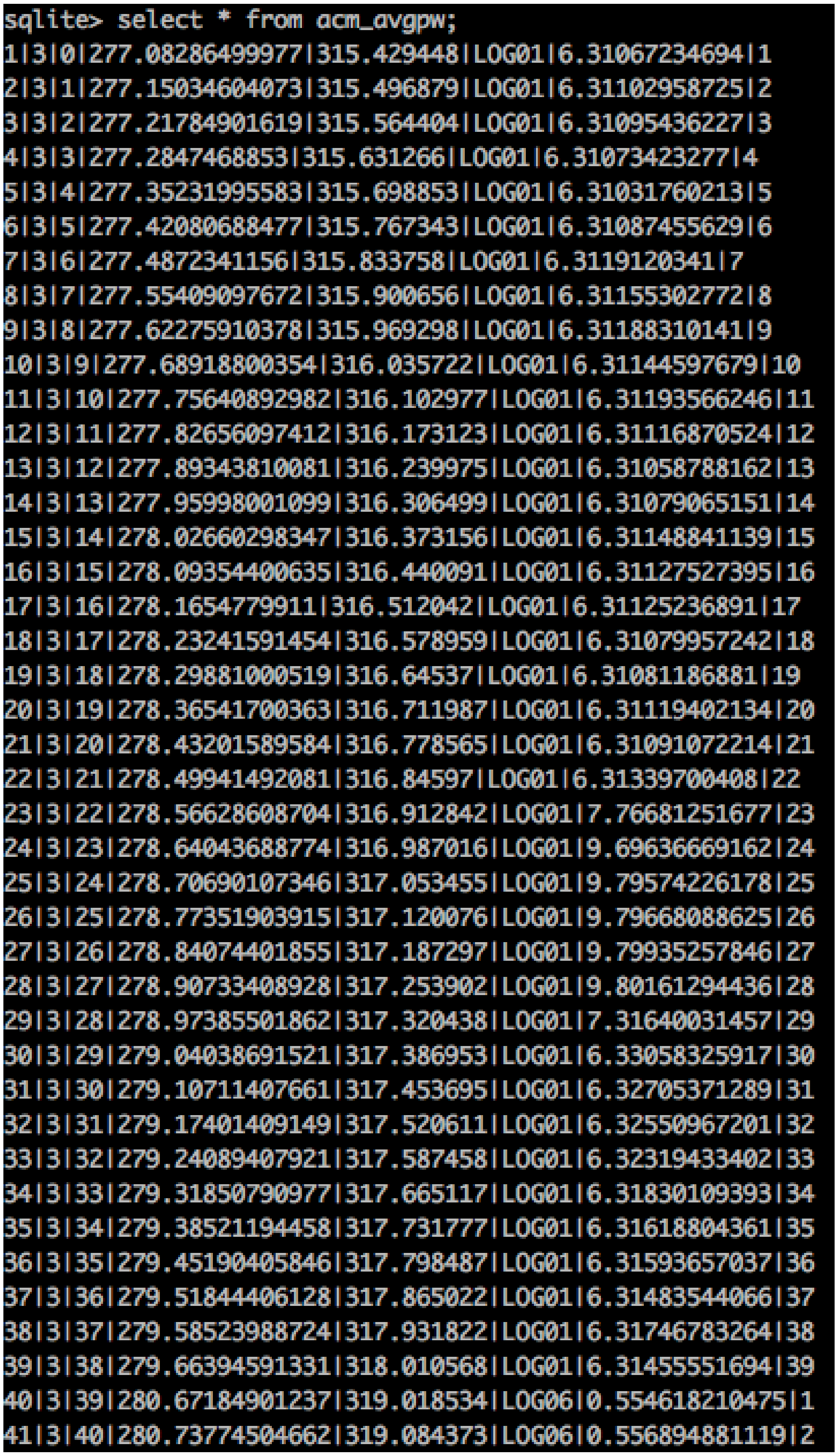
Another feature that we enable in our framework is the OMF Web [18]. OMF Web is a web-based data visualization service and the typical use case is to allow a user to investigate a data set stored in one or more databases as well as life data



```
sqlite> sqlite> select * from acm_tapw;
1|3|0|279.98448395729|318.331107|LOG01|6.81372165638
2|3|1|283.56924510002|321.915955|LOG06|0.558354230885
sqlite> select * from acm_tenergy;
1|3|0|279.98455095291|318.331233|LOG01|269.668234392
2|3|1|283.56931209564|321.916117|LOG06|22.5867858696
sqlite>
```

Figure 5.1: OML database - total energy & average power

streams. So in our case, having the results from the NITOS ACM cards stored into database alongside with the measurements from the experiment ran on the nodes we can visualize the results side by side as shown in Figure 5.3. This figure presents the power consumption of 4 Atheros 9380 network interface cards and the respective achieved throughputs where the setup was 2 pair of nodes with one transmitter and one receiver in each one of them.



```
sqlite> select * from acm_avgpw;
1|3|0|277.08286499977|315.429448|LOG01|6.31067234694|1
2|3|1|277.15034604073|315.496879|LOG01|6.31102958725|2
3|3|2|277.21784901619|315.564404|LOG01|6.31095436227|3
4|3|3|277.2847468853|315.631266|LOG01|6.31073423277|4
5|3|4|277.35231995583|315.698853|LOG01|6.31031760213|5
6|3|5|277.42080688477|315.767343|LOG01|6.31087455629|6
7|3|6|277.4872341156|315.833758|LOG01|6.3119120341|7
8|3|7|277.55409097672|315.900656|LOG01|6.31155302772|8
9|3|8|277.62275910378|315.969298|LOG01|6.31188310141|9
10|3|9|277.68918800354|316.035722|LOG01|6.31144597679|10
11|3|10|277.75640892982|316.102977|LOG01|6.31193566246|11
12|3|11|277.82656097412|316.173123|LOG01|6.31116870524|12
13|3|12|277.89343810081|316.239975|LOG01|6.31058788162|13
14|3|13|277.95998001099|316.306499|LOG01|6.31079065151|14
15|3|14|278.02660298347|316.373156|LOG01|6.31148841139|15
16|3|15|278.09354400635|316.440091|LOG01|6.31127527395|16
17|3|16|278.1654779911|316.512042|LOG01|6.31125236891|17
18|3|17|278.23241591454|316.578959|LOG01|6.31079957242|18
19|3|18|278.29881000519|316.64537|LOG01|6.31081186881|19
20|3|19|278.36541700363|316.711987|LOG01|6.31119402134|20
21|3|20|278.43201589584|316.778565|LOG01|6.31091072214|21
22|3|21|278.49941492081|316.84597|LOG01|6.31339700408|22
23|3|22|278.56628608704|316.912842|LOG01|7.76681251677|23
24|3|23|278.64043688774|316.987016|LOG01|9.69636669162|24
25|3|24|278.70690107346|317.053455|LOG01|9.79574226178|25
26|3|25|278.77351903915|317.120076|LOG01|9.79668088625|26
27|3|26|278.84074401855|317.187297|LOG01|9.79935257846|27
28|3|27|278.90733408928|317.253902|LOG01|9.80161294436|28
29|3|28|278.97385501862|317.320438|LOG01|7.31640031457|29
30|3|29|279.04038691521|317.386953|LOG01|6.33058325917|30
31|3|30|279.10711407661|317.453695|LOG01|6.32705371289|31
32|3|31|279.17401409149|317.520611|LOG01|6.32550967201|32
33|3|32|279.24089407921|317.587458|LOG01|6.32319433402|33
34|3|33|279.31850790977|317.665117|LOG01|6.31830109393|34
35|3|34|279.38521194458|317.731777|LOG01|6.31618804361|35
36|3|35|279.45190405846|317.798487|LOG01|6.31593657037|36
37|3|36|279.51844406128|317.865022|LOG01|6.31483544066|37
38|3|37|279.58523988724|317.931822|LOG01|6.31746783264|38
39|3|38|279.66394591331|318.010568|LOG01|6.31455551694|39
40|3|39|280.67184901237|319.018534|LOG06|0.554618210475|1
41|3|40|280.73774504662|319.084373|LOG06|0.556894881119|2
```

Figure 5.2: OML database - average power

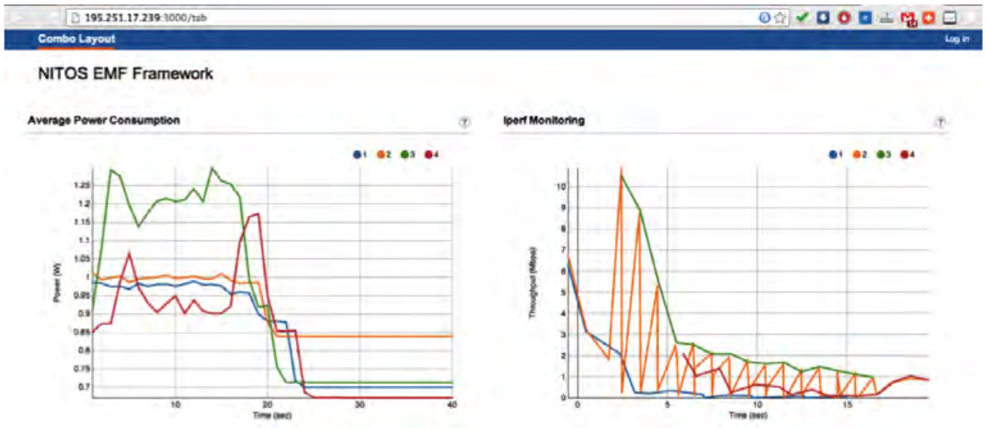


Figure 5.3: OMF visualization of measurements

CHAPTER VI

Conclusion

6.1 Conclusion

In this Master Thesis we demonstrate a framework that enables evaluation of cognitive devices, in terms of sensing delay and resulting energy consumption. In order to accomplish this we had to desing and develop both hardware and software components.

For measuring the sensing delay for each sensing device we had first to define the meaning of this metric. In this definition we included except from the time for sensing, the time that the device consumes in order to return the sensing result. Thus we were able to discriminate 4 process that consist the sensing delay and which are 1) sensing, 2) transferring, 3) processing and 4) channel switching. The experimental evaluation pointed out that only the imec SE is able to perform real-time sensing and that the USRP N210 can be turn on real-time sensing device with tha appropriate host machine and parallelizing the procedures of sensing and processing.

Towards the evaluation of the cognitive devices in term of power consumption, we used the NITOS ACM cards [13] and we design and develop the appropriate adaptors for each device in order to monitor the consumption. The outcome of this evaluation is that the USRPs are rather energy demanding in contrary with imec SE and especially Atheros AR9380 which consumes less than everyone of the devices under consideration.

Summing up, we present a monitoring procedure that has been directly integrated in the experimentation tools of cognitive testbeds and demonstrate how it aids in the online evaluation of 4 different cognitive platforms in terms of spectrum sensing delay and energy consumption during spectrum sensing.

APPENDIX

APPENDIX A

Framework's code

A.1 ACM Functions

The ACM card is accessible via terminal (with the curl application) or a web browser. The Figure A.1 below shows the respond of the ACM card to the request

```
http://192.168.1.2/commands
```

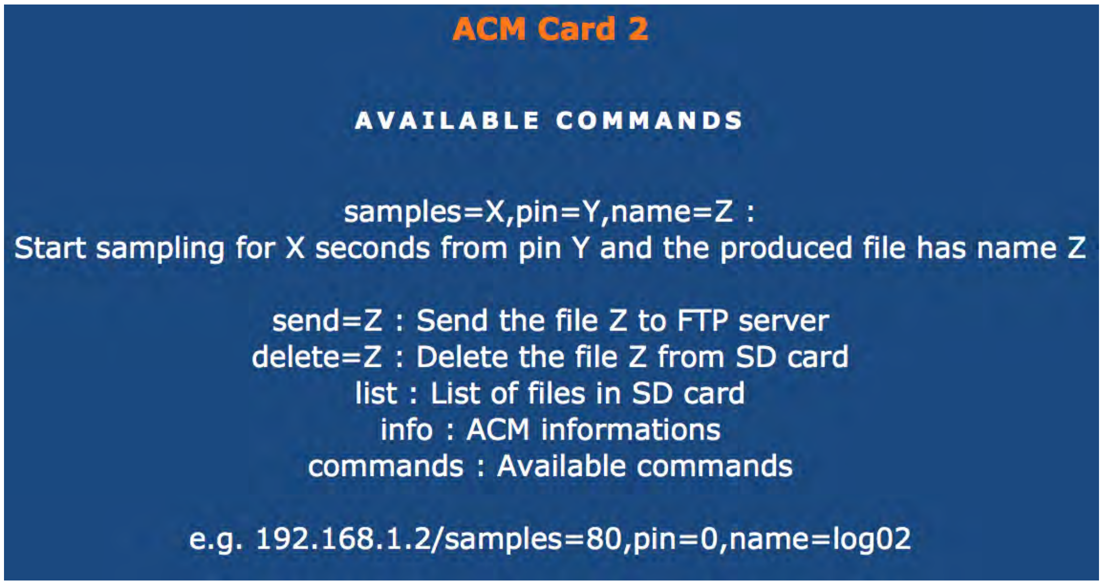
- Sample:

  ```
  http://192.168.1.2/samples=30,pin=0,name=log02
  ```

  With this command the ACM card will sample for 30 seconds the pin 0 and the measurements will be stored in a file with name LOG02.BIN. (The name of the file must not exceed the 8 characters XXXXXXXX.BIN)

- Upload file:

  ```
  http://192.168.1.2/send=log02
  ```



ACM Card 2

AVAILABLE COMMANDS

samples=X,pin=Y,name=Z :
Start sampling for X seconds from pin Y and the produced file has name Z

send=Z : Send the file Z to FTP server
delete=Z : Delete the file Z from SD card
list : List of files in SD card
info : ACM informations
commands : Available commands

e.g. 192.168.1.2/samples=80,pin=0,name=log02

Figure A.1: ACM Card list of available commands

With this command the ACM card will start uploading to the FTP server the file with name LOG02.BIN.

- Show files:

```
http://192.168.1.2/list
```

With this command the ACM card will show a list of all the files inside the SD card.

- Delete file:

```
http://192.168.1.2/samples=30,pin=0,name=log02
```

With this command the ACM card will delete from the SD card the file with name LOG02.BIN.

A.2 OMF Experiment Description - Experiment 1

```
# GPL
# Software Version: 1.0
# Virgilios Passas (virpassas@gmail.com)
# http://nitlab.inf.uth.gr/NITlab/

require 'open3'
require 'oml4r'

seconds = 0
@@stdout = nil

defProperty('theSender','zotaccm3.ping.cognitiveradio.wilab2.ilabt.iminds.be',"ID
    of sender3 node")
defProperty('theReceiver','zotaccm4.ping.cognitiveradio.wilab2.ilabt.iminds.be',"ID
    of receiver node")
defProperty('runtime',20,"Time in second for the experiment is to run")
defProperty('wifiMode',"adhoc","The mode of WIFI to use in this
    experiment")
defProperty('wifiType',"g","The type of WIFI to use in this experiment")
defProperty('channel','6',"The WIFI channel to use in this experiment")
defProperty('essid',"acm","The ESSID to use in this experiment")

defGroup('Sender', property.theSender) do |node|
   node.addApplication("iperf-5.4", :id => 'iperf10') {|app|
     app.setProperty('port', 5200)
     app.setProperty('bandwidth',"10000000")
     app.setProperty('udp',true)
     app.setProperty('time', 15)
     app.setProperty('client', "192.168.2.4")
     app.setProperty('reportstyle', 'o')
     app.setProperty('interval', '1')
     app.measure('transfer', :samples=>1)
     app.measure('losses', :samples=>1)
   }
   node.addApplication("iperf-5.4", :id => 'iperf20') {|app|
     app.setProperty('port', 5200)
     app.setProperty('bandwidth',"20000000")
     app.setProperty('udp',true)
     app.setProperty('time', 15)
     app.setProperty('client', "192.168.2.4")
     app.setProperty('reportstyle', 'o')
     app.setProperty('interval', '1')
     app.measure('transfer', :samples=>1)
     app.measure('losses', :samples=>1)

   }
   node.net.w0.mode = property.wifiMode
   node.net.w0.type = property.wifiType
   node.net.w0.channel = property.channel
   node.net.w0.essid = property.essid
   node.net.w0.ip = "192.168.2.3"
 end

 defGroup('Receiver', property.theReceiver) do |node|
   node.addApplication("iperf-5.4", :id => 'iperf10') {|app|
     app.setProperty('port', 5200)
     app.setProperty('server', true)
     app.setProperty('udp',true)

     app.setProperty('reportstyle', 'o')

     app.setProperty('interval', '1')

     app.measure('transfer', :samples=>1)

     app.measure('losses', :samples=>1)


   }

   node.addApplication("iperf-5.4", :id => 'iperf20') {|app|

     app.setProperty('port', 5200)

     app.setProperty('server', true)

     app.setProperty('udp',true)

     app.setProperty('reportstyle', 'o')

     app.setProperty('interval', '1')

     app.measure('transfer', :samples=>1)

     app.measure('losses', :samples=>1)


   }

   node.net.w0.mode = property.wifiMode

   node.net.w0.type = property.wifiType

   node.net.w0.channel = property.channel

   node.net.w0.essid = property.essid

   node.net.w0.ip = "192.168.2.4"

 end


 defEvent(:ACM_EVENT) do |event|

   tmp = 88

   if seconds > 0

    info "Wait until script finishes!"

         sl = @@stdout.readlines
         tmp2 = sl[sl.length - 1]
         puts tmp2
         tmp2 = tmp2.to_i
    puts tmp2
         if(tmp == tmp2)
                 event.fire
         end
   end
 end

 onEvent(:ACM_EVENT) do |event|
   info " "
   info "ALL PROCESSES ARE FINISHED!"
   info " "
   Experiment.done
 end

 onEvent(:ALL_UP_AND_INSTALLED) do |node|
   wait 30
   d = '--oml-domain'
   dom = "#{Experiment.ID}"
   d += ' ' + dom
   cmd = 'sudo python exp_ibbt.py -t 50 -c 3 3 --oml-id acm_server
       --oml-collect tcp:am.wilab2.ilabt.iminds.be:3003 -i acm'+' ' + d
   stdin, @@stdout, stderr = Open3.popen3("ssh
       efkerani@server.ping.cognitiveradio.wilab2.ilabt.iminds.be #{cmd}")

   wait 3
   group("Receiver").startApplication('iperf10')
   wait 2
   group("Sender").startApplication('iperf10')
   info "All my Applications are started now..."
   wait property.runtime / 2
   group("Sender").stopApplications
   wait 5
   group("Receiver").stopApplications
   group("Receiver").startApplication('iperf20')
   wait 5
   group("Sender").startApplication('iperf20')
   info "Sender's Application iperf20 is started now..."
   wait property.runtime / 2
   allGroups.stopApplications
   allGroups.exec("modprobe -r ath9k")
   info "All my Applications in nodes are stopped now."
   wait 330
   seconds = 1
 end
```

A.3 Sensing Delay code - Timestamps

```
//USRP N210 & USRP E110
#define timems_t uint64_t
#define ENDOFTIME UINT64_MAX
#define SECS(tv) (tv.tv_sec + tv.tv_usec / 1000000.0)


timems_t now(void);
timems_t timedelta(struct timeval *t1, struct timeval *t0);
timems_t usPassedSince(struct timeval t);


timems_t now(void) {
        struct timeval t;
        timems_t       m;

        (void) gettimeofday(&t, (struct timezone *) 0);
        m = t.tv_sec;
        m *= 1000000;
        m += t.tv_usec;
        return (m);
}
std::ofstream myfile;


#define freqstep 25000000 // can be changed
namespace po = boost::program_options;


static bool stop_signal_called = false;
void sig_int_handler(int){stop_signal_called = true;}


template<typename samp_type> void recv_to_file(
    uhd::usrp::multi_usrp::sptr usrp,
    const std::string &cpu_format,
    const std::string &wire_format,
    const std::string &file,
    size_t samps_per_buff,
    int num_requested_samples,
    size_t BlockPerSweep,
    double freq
```

```
){
    //timestamp
    timems_t chrono,chrono2;

    int num_total_samps = 0;
    //create a receive streamer
    uhd::stream_args_t stream_args(cpu_format,wire_format);
    uhd::rx_streamer::sptr rx_stream = usrp->get_rx_stream(stream_args);

    uhd::rx_metadata_t md;
    std::vector<samp_type> buff(samps_per_buff);
    std::ofstream outfile(file.c_str(), std::ofstream::binary);
    bool overflow_message = true;

    //setup streaming
    uhd::stream_cmd_t stream_cmd(
        //uhd::stream_cmd_t::STREAM_MODE_START_CONTINUOUS:
        uhd::stream_cmd_t::STREAM_MODE_NUM_SAMPS_AND_DONE
    );
    stream_cmd.num_samps = samps_per_buff;
    stream_cmd.stream_now = true;
    stream_cmd.time_spec = uhd::time_spec_t();
    //usrp->issue_stream_cmd(stream_cmd);
    size_t block_index = 0;
    double fc;

    //timestamp
    double t3,t2,t1,t4;
    struct timeval tim;
```

```
while(not stop_signal_called and (num_requested_samples !=
    num_total_samps or num_requested_samples == 0)){
chrono = now();
// calculate new frequency and set frequency
fc = freq + block_index*freqstep;
try { usrp->set_rx_freq(fc);}
catch(...){std::cerr << "ERROR: Exception caught when trying to set
    the frequency " << std::endl;}
block_index++ ;
if(block_index>=BlockPerSweep) block_index=0;
// issue stream command
usrp->issue_stream_cmd(stream_cmd);
myfile << now()-chrono << "\t" << fc <<"\n";

chrono = now();
size_t num_rx_samps = rx_stream->recv(&buff.front(), buff.size(), md,
    3.0);
chrono2=now()-chrono;
myfile << chrono2 << "\n";

    if (md.error_code == uhd::rx_metadata_t::ERROR_CODE_TIMEOUT) {
        std::cout << boost::format("Timeout while streaming") <<
            std::endl;
        break;
    }
    if (md.error_code == uhd::rx_metadata_t::ERROR_CODE_OVERFLOW){
        if (overflow_message){
            overflow_message = false;
            std::cerr << boost::format(
                    "Got an overflow indication. Please consider the
                        following:\n"
                    "  Your write medium must sustain a rate of %fMB/s.\n"
                    "  Dropped samples will not be written to the file.\n"
                    "  Please modify this example for your purposes.\n"
                    "  This message will not appear again.\n"
                ) % (usrp->get_rx_rate()*sizeof(samp_type)/1e6);
            }
            continue;
        }
        if (md.error_code != uhd::rx_metadata_t::ERROR_CODE_NONE){
            throw std::runtime_error(str(boost::format(
                "Unexpected error code 0x%x"
            ) % md.error_code));
        }

        num_total_samps += num_rx_samps;

        outfile.write((const char*)&buff.front(),
            num_rx_samps*sizeof(samp_type));
    }

  outfile.close();

}
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] *"An OML client module for Python"*, https://pypi.python.org/pypi/oml4py.

[2] *"Arduino Ethernet Shield"*, http://arduino.cc/en/Main/ArduinoEthernetShield.

[3] *"Arduino Mega 2560 Board"*, http://goo.gl/IFHwq.

[4] *"Ath9k wireless driver"*, http://wireless.kernel.org/en/users/Drivers/ath9k.

[5] *"Atheros AR9380 Chipset"*, http://goo.gl/51Mg7.

[6] *"Atmega2560 Micro-controller"*, http://www.atmel.com/Images/2549S.pdf.

[7] *"CORTEX-lab"*, http://www.cortexlab.fr/.

[8] *"Emulab Testbed"*, http://www.emulab.net.

[9] A. Ghasemi and E.S. Sousa. Spectrum sensing in cognitive radio networks: requirements, challenges and design trade-offs. *Communications Magazine, IEEE*, 46(4):32–39, April 2008.

[10] L. Hollevoet, S. Pollin, P. Van Wesemael, F. Naessens, Antoine Dejonghe, and Liesbet Van Der Perre. A 22 mw multi-standard reconfigurable spectrum sensing enabled digital frontend. In *Proceedings of the 2012 IEEE Workshop on Signal Processing Systems*, 2012.

[11] *"Iperf"*,http://iperf.fr/.

[12] Giannis Kazdaridis, Stratos Keranidis, Adamantios Fiamegkos, Thanasis Korakis, Iordanis Koutsopoulos, and Leandros Tassiulas. Novel metrics and experimentation insights for dynamic frequency selection in wireless lans. In *Proceedings of the 6th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, 2011.

[13] S. Keranidis, G. Kazdaridis, V. Passas, T. Korakis, I. Koutsopoulos, and L. Tassiulas. Online Energy Consumption Monitoring of Wireless Testbed Infrastructure Through the NITOS EMF Framework. In *Proceedings of ACM WiNTECH*, 2013.

[14] Stratos Keranidis, Virgilios Passas, Kostas Chounos, Wei Liu, Thanasis Korakis, Iordanis Koutsopoulos, Ingrid Moerman, and Leandros Tassiulas. Online assessment of sensing performance in experimental spectrum sensing platforms.

In *Proceedings of the 9th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, 2014.

[15] *"Microsoft Research Software Radio (SORA)"*, http://goo.gl/KoYkhQ.

[16] *"NITOS Wireless Testbed"*, http://nitlab.inf.uth.gr.

[17] *"OMF-cOntrol & Management Framework"*, http://omf.mytestbed.net/.

[18] *"OMF Web"*, http://goo.gl/4LlweC.

[19] *"OML Measurement Library"*, http://mytestbed.net/projects/oml/wiki/.

[20] *"ORBIT Tetbed"*, http://www.orbit-lab.org/.

[21] S. Pollin, L. Hollevoet, P. Van Wesemael, M. Desmet, A. Bourdoux, E. Lopez, F. Naessens, P. Raghavan, V. Derudder, S. Dupont, and A. Dejonghe. An integrated reconfigurable engine for multi-purpose sensing up to 6 GHz. In *Proceedings of DySPAN*, 2011.

[22] *"Proc Filesystem"*, http://goo.gl/73zRmu.

[23] *"Texas Instruments INA139"*, http://goo.gl/rPQLB.

[24] *"Universal Software Radio Peripheral (USRP)"*, http://www.ni.com/usrp/.

[25] *"USRP E110 Embedded Series"*, https://www.ettus.com/product/details/UE110-KIT.

[26] *"USRP Hardware Driver"*, http://goo.gl/PRrrFo.

[27] *"USRP N210 Networked Series"*, https://www.ettus.com/product/details/UN210-KIT.

[28] *"VT-CORNET"*, http://goo.gl/UTojEZ.

[29] *"w-ilab.t Testbed"*, http://www.crew-project.eu/wilabt.

[30] *"Wireless Open-Access Research Platform (WARP)"*, http://warp.rice.edu/.

[31] *"WIZnet W5100 Ethernet Controller"*, http://goo.gl/IWQMT5.

[32] *"XCVR2450 2.4-2.5 GHz, 4.9-5.9 GHz Tx/Rx"*, https://www.ettus.com/product/details/XCVR2450.

[33] T. Yucek and H. Arslan. A survey of spectrum sensing algorithms for cognitive radio applications. *Communications Surveys Tutorials, IEEE*, 11(1):116–130, First 2009.