Radio-Signal Correlation for Collaborative Wideband Spectrum Monitoring System

Roberto Calvo-Palomino
IMDEA Networks Institute &
Universidad Carlos III
Madrid, Spain
roberto.calvo@imdea.org

***Abstract*—Today's spectrum measurements are mainly performed by governmental agencies which drive around using expensive specialized hardware. The idea of collaborative spectrum monitoring has recently gained attention to capture the usage of the wireless spectrum at larger geographical scale. We present in this report the main challenges address in collaborative spectrum monitoring and we propose a methodology for making decisions based on signal correlation similarity between sensors. To reach this goal, we build a software-defined sensor architecture that enables distributed data collection in real-time over Internet. Our sensor design builds upon low-cost commercial hardware components with a total cost per sensor device below $100. Our results suggest that our architecture can be useful to coordinate a large number of sensors scanning at the same or different frequencies in order to reach the target collaboratively. Interesting application areas could be dynamic spectrum access in cognitive radios, re-construct a large-bandwidth signals or simply to gain an understanding of the spectrum usage at large scale to detect elevated electro-smog regions, anomalies in the signals or set policy enforcement in the electromagnetic space.**

I. Introduction

The electromagnetic (EM) spectrum is heavily used by diverse types of wireless communications, broadcasting services and radar. The governments are the responsible to manage and regulate the usage of the radio spectrum. Each country creates and maintains its own national frequency allocation plan which describes how the EM spectrum shall be used [1]. Despite, the EM spectrum is well-organized in terms of frequency allocation, 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 which using expensive and specialized hardware. This monitoring approach does not allow to create a well-scaled infrastructure to cover the pervasive deployment of wireless networks. Recent suggestions to enable wide-scale and real-time spectrum monitoring have been to build a networked and distributed infrastructure using remote (but yet expensive) spectrum analyzers [2]–[4] or leverage the masses to crowdsource the measurement stations by trading-off radio device sensitivity with commoditized spectrum monitoring hardware in order to build a low-cost solution [5], [6]. Our vision is aligned with these works of exploiting the crowd for scalable and distribute monitoring of the actual usage of the spectrum at different frequencies, locations and times. However, formed works consider the different sensing nodes as separate entities, each responsible to monitor the spectrum in a given area. We envision a collaborative environment where a large number of sensing nodes work together in a coordinated manner towards the common goal of monitoring the spectrum over the defined area of coverage.

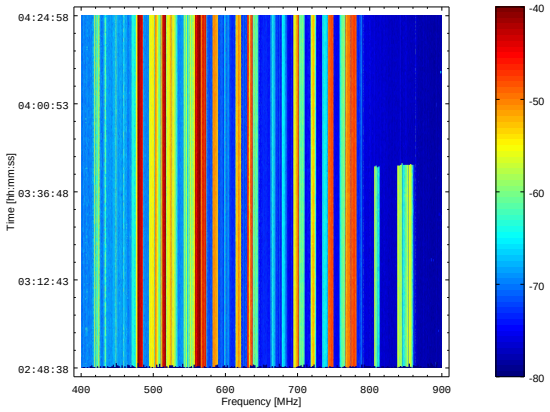

Fig. 1: Example: Spectrum analyzed in Madrid city at 2015/03/31 generated by our low-cost sensor platform [7] where we focus on the UHF TV spectrum. We could detect how a few DVB-T channels were switched off to make room for 4G networks (our sensing nodes can typically monitor the spectrum between 24 and 1766 MHz).

Our low-cost platform can detect changes in the spectrum [8] using a sensing single-node as Figure 1 shows. However, we believe in the distributed approach to spectrum scanning where the sensors collaborate together to reach a goal taking less time and getting better quality that if just one of the sensors makes the same action. Detect as soon as possible the available frequencies to cognitive radios users it could be a good example to use this system for optimizing their efforts to dynamically accessing the spectrum. Another example is to detect anomalies in the spectrum, attacks or policy violations. To achieve this goal and rely on the output of the system, the

sensors report their spectrum to a server where some fusion and correlation strategies should detect reliably these scenarios.

In this work, we suggest the strategies and architecture to correlate the signal that low-cost wideband spectrum sensing nodes receive at given locations and transmit to the server in order to making smart decisions about cognitive radio spaces, anomalies or attacks. We demonstrate and implement an actual architecture able to collect the sensor's spectrum data to determine the similarity of the radio signal received at nearby locations by different sensing nodes. In this paper, we present the main challenges to address in a collaborative and distributed spectrum system using low-cost software-defined sensing nodes and then present the strategies to correlate the signal between the them. Our work provides the following three contributions:

- We introduce different challenges to address in collaborative wideband spectrum monitoring. For each one we evaluate the environment and we study the best system configuration for correlating of the signals.
- We propose and evaluate a procedure to compute the signal correlation between sensors using FM radio bandwidth with spectrum resolution of *2.3 kHz* and time resolution of *0.1 sec*.
- We propose a closed-loop architecture to get a better spectrum and time resolution of correlation. This architecture is based on the feedback provided by the server to synchronize the sensing nodes in order to get the best signal correlation.

The remainder of this paper is organized as follows. We present related work in Section II. In Section III we present the ideas and benefits of frequency monitoring spectrum in a distributed and collaborative approach. Then, we present the hardware components of our sensor in Section IV and the scanning spectrum architecture in Section V. The challenges for collaborative spectrum monitoring are presented in Section VI. The correlation signal strategies are proposed and evaluated in Section VII. We finally draw the conclusion and future work in Section VIII.

II. Related Work

The spectrum utilization has been conventionally studied in cognitive radios. Spectrum sensing of a narrowband frequency spectrum is a part of the cognitive capabilities, where devices monitor the available band and detect spectrum holes [9], [10]. However, monitoring large spectra is more challenging and has been the focus of more recent investigations, where a wider spectrum is sensed to identify holes might then be accessed by cognitive radio users [11]. The current wideband spectrum monitoring techniques have clear limitations when trying to adopt them to low-cost commodity platforms. In [6], they proposed to use the RaspBerry Pi board to store wideband spectrum measurements performed by a portable and professional RF spectrum analyzer. In [5], the authors performed an initial feasibility study to verify the efficiency of using RTL-SDR USB dongles connected to laptop or smartphone devices to build a low-cost commoditized spectrum analysis system. According to their work, real-time spectrum monitoring will be possible in the very near future using commodity hardware.

Spectrum sensing strategies have been developed for cooperative and non-cooperative spectrum sensing solutions. Cooperative sensing has been introduced in [12], and it describes a scenario where a collection of sensing nodes distributively monitor a frequency spectrum. In [13], they proposed to use a cooperative environment to distinguish between an unused band and deep fade due to shadowing or fading effects. In cognitive radio is also proposed the collaborative approach to trying to detect licensed primary transmissions [14]. The works above did not complement the proposed cooperative environment with a system-level design and implementation.

On the other hand, SpecNet [4] is a platform that allows researches to conduct spectrum measurements studies remotely in real-time for opportunistic spectrum access. Google Spectrum Database [15] is 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 [2] is a platform that is currently operating in a few locations worldwide and it aims to collect usage of wideband spectrum from a number of sensing stations. Data is then sent to Microsoft Azure Cloud for storage and visualization. Currently, the cost of the sensing stations is more that $5000 whereas our sensor platform is less that $100.

A recent work [16] employs correlation techniques to achieve characteristics of mobile cognitive radio users in different environments with the objective to increase the performance of cooperative spectrum sensing. In [17], they proposed to use correlation as well to achieve low-noise measurements based on the processing of the input signal by two independent channels in parallel. Both studies use correlation techniques to achieve its goals, but they are evaluated in simulation environments. In this paper, we present a study done in a real environment using FM radio frequencies.

III. Distributed Spectrum Monitoring

In this section we show the requirements and benefits of a collaborative wideband spectrum monitoring system relying on a distributed and collaborative approach. We then identify the main components of the system.

A. Design Goals

We envision the deployment scenario where users continuously collect and share their sensed data in real-time and in this way build a distributed spectrum monitoring network.

Distributed. The system should be based on distributed sensing nodes, each can autonomously execute its tasks and send the data to a collecting unit.

Collaborative. The system should use the data from different sensors to detect anomalies or new free frequencies to use in cognitive radios. The communication between server and sensors must be continuously in both directions to send information in order to synchronize the sensors.

Wideband. The system shall be based on sensing nodes that allow for a wide range of frequencies being monitored.

Fig. 2: Sensing nodes collecting distributively the spectrum in Madrid city.

Software-Defined. The system shall allow an easy and flexible way to reconfigure the sensing nodes and extend its functionality.

Real-Time. The system should be capable of delivering data from the sensing devices to the collecting unit in real-time. If the connectivity between sensors and collecting unit gets temporarily interrupted, no data should be lost and the data will be sent when the connection is re-established.

Synchronization. Every sensing nodes should be synchronized in time at sufficient resolution in order to fuse the spectrum in a post-processing task in the server.

Low-Cost. The system shall be based on sensing nodes not exceeding a total cost of $100. The strict cost limitation of individual nodes is one of the key factors enabling the envisioned deployment scenario.

B. System Architecture

A high level representation of our system architecture is shown in Figure 2. The individual components communicate through the Internet. The two main system components are:

Sensor. The key component in the design of the system are its sensing nodes. The sensing nodes are small form-factor and they are synchronized in time to read the spectrum collaboratively, satisfying the system requirement defined in section III-A. The sensors also should receive useful information for getting the best well-synchronized spectrum.

Server. The server is responsible to store and process the data that sensors send. Also it takes care of large-scale processing tasks to fuse the data. In addition, the server should continuously compute some operations using the sensors spectrum data to send them the necessary information for synchronizing the process of spectrum scanning (sampling process).

IV. Sensor Hardware Platform

This section describes our hardware platform used for the evaluation of this work. Our sensor is built upon four major hardware components - a *single-board computer (SBC)* platform, and RTL-SDR USB dongle. The RTL-SDR USB dongle, together with the antenna connected to it, is used as RF receiving device. The SBC platform serves as the host computing device for RF receiver. The typical hardware configuration of our sensor is showing in Figure 3.

Fig. 3: Low-cost wideband spectrum monitoring hardware sensor platform.

A. RF Interface

To sample the spectrum, we rely on RTL-SDR USB dongles as RF interfaces. The intended purpose of these dongles is the reception of digital video broadcasting-terrestrial (DVB-T), digital audio broadcasting (DAB) and frequency modulation (FM) signals. However, it has been discovered that these dongles are based on Realtek's RTL2832U demodulator which can be turned into a cheap software-defines radio (SDR) since the chipset allows the transfer of the captures raw I/Q samples. We use the C library *librtlsdr* from the OsmocomSDR project[1] to adquire the digital samples. The theoretical limitation of the USB 2.0 interface is at 3.2MS/s. In practice only sampling rates below 2.4MS/s can typically be received without any losses.

Our architecture supports any dongle as long as it reports the raw I/Q samples. We rely on dongles with the Rafael Micro R820T tuner for the evaluation in this work, which supports a continuous tuning range between 24 MHz and 1766 MHz.

B. Single-Board Computer

SBCs are fully operational computing platforms, built on a single circuit board. Many different SBCs are available on the market. For our sensors, cost is one of the main factors we want to reduce. We decided to use the RaspBerryPi (RBPi), which stays in the lowest price category (i.e. below 50$), and provides a dedicated GPU. The GPU opens the possibility of having a second processing unit onto our sensing nodes. In this work, we use the RaspBerry Pi revision B which has a CPU clocked at 700 MHz and 512 MB of RAM. For network connectivity we rely on the on-board Ethernet and an USB WLAN stick attached to one of the available USB ports.

V. Architecture

The design of our sensor architecture is depicted in Figure 4. The spectrum monitor is the main component which implements all the features for spectrum analysis such as sampling, segmentation, windowing, DC removal, FFT, averaging, etc. The spectrum monitor directly accesses the RF front-end to retrieve I/Q samples from the RTL-SDR. *FFT*

[1] http://osmocom.org

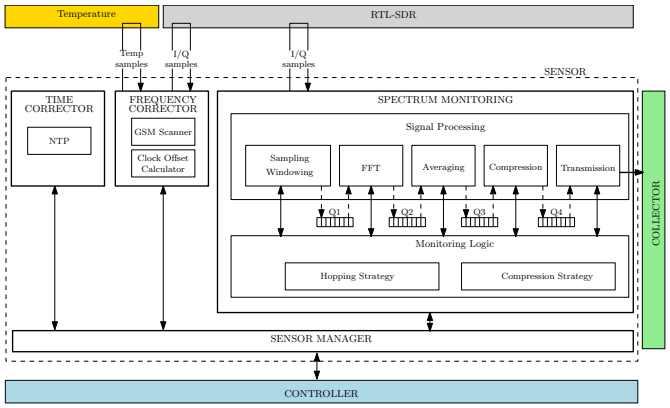

Fig. 4: Sensor architecture: The design of our sensing nodes consists of four main components - spectrum monitor, time and frequency corrector, as well as a sensor manager. Sensors interface the (external) controller and collector nodes, and (internally) access RF data through the RTL-SDR device.

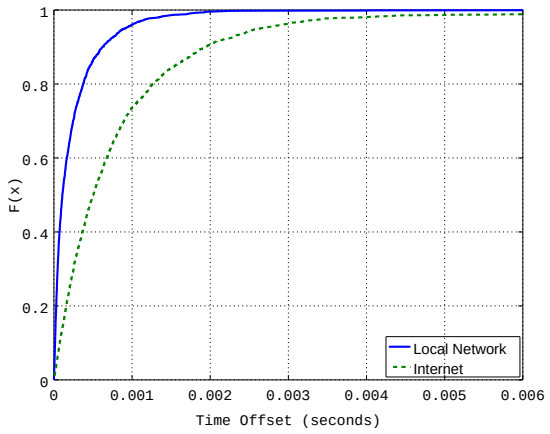

Fig. 5: ECDF about offset time between sensing nodes in local network (*0.1 ms)* and Internet (*0.5 ms)* scenarios

is then computed in the GPU that is faster than CPU (more details about this will be given in the following section). In addition, it compresses the data and transmits it over the network to a collector which is typically a remote server in the Internet. The spectrum monitor follows a frequency hopping and data compression strategy as specified by the sensor manager. The purpose of the hopping strategy is to optimize the frequency and time at which the spectrum monitor inspects certain frequency bands. The goal of the compression strategy is to adapt the data size that will be transferred over the network according to the bandwidth constraints of the network connection

The time corrector module is responsible to keep synchronized the internal clock of the sensor. The time-stamp is set every time samples are read from the RTL-SDR device. It is essential to reach a good time synchronization to compute the spectrum similarity between the sensors located in different places. For this goal we rely on Network Time Protocol (NTP) to synchronize the sensors over Internet.

Every signal processing blocks use queues to share the information about the spectrum analyzed. These finite size queues have an important role in the system because they are used to save the data read. A signal processing task could be blocked when it tries to write data if the queue is full.

VI. Challenges for Collaborative Spectrum Monitoring

In this section, we describe the challenges that we face to read the spectrum collaboratively using our proposed architecture. Typically the most critical part of a distributed system is the time synchronization between nodes. It is an important requirement to fuse the spectrum read in a post-processing task in the server. The sampling process predictability of each sensor is important to ensure that all the sensors read the spectrum in the same time. We start by describing the time synchronization setup.

A. Time Synchronization

Time synchronization between machines is always a feature of distributed systems in Internet. Typically each node reads information from their environment and send the data to the server where it will be processed. For that, it is essential to have information well-synchronized in terms of time. Network Time Protocol (NTP) is a networking protocol for clock synchronization between computers in Internet. NTP can usually maintain time synchronization in terms of milliseconds over the public Internet, and can achieve better that one millisecond accuracy in local area networks. Other alternatives to synchronize the time between nodes could be using GPS receivers, DVB-T information tables or the Precision-Time-Protocol (PTP). All of them have been discarded to include in our system because increase dramatically the cost of the sensor (in case of GPS), or the accuracy is around seconds (DVB-T), or it is needed an expensive equipment for hardware time stamping (PTP).

Using NTP system, two sensors were configured to synchronize the time each minute during 36 hours. Internet and local network environments were used to compare the different offsets depending on the latency with the *NTP* server. The figure 5 shows the offset distribution when the two sensing nodes are in the same local network and therefore have the same latency against the *NTP* server. In this scenario the median offset is 0.1 *ms*. On the other hand, if the sensing nodes are in different networks they have different latencies with the *NTP* server. In this case the median offset is 0.5 *ms*. That means that the data of both sensors in the server could have an offset of 0.5 *ms*. The higher the offset, more post-processing complex tasks will be executed in the server to compensate it.

In our system we use a RTL-SDR device that it can read 2.4 MSamples/second which it means that each sample is read in *0.41* µ*s*. Regarding with the offset time commented above, the RTL-SDR device can read 240 samples in 0.1 *ms* and 1200 samples in 0.5 *ms*. Therefore, using NTP to synchronize the nodes, the system should use batch of 240 samples (in local networks) and batch of 1200 samples (in Internet) to compute the signal correlation in order to use synchronized data.

B. Priority Scheduling

The sampling process in our system is implemented as a single thread that reads samples from the RTL-SDR dongle and sends the spectrum information to other threads using shared queues between them (figure 4). The other threads are responsible to compute tasks about signal processing and send the information to the server. All these tasks can interfere in the sampling rate of the system. Our sensing nodes have one core to compute the operations and our architecture uses six threads plus the system's threads. We have to ensure that sampling thread has the best priority in terms of execution. It is essential to read as many samples as possible and not lose information about the spectrum. The longer information we have on the spectrum, the easier it will be to apply correlation techniques in the server side.

The Completely Fair Scheduler (CFS) is a process scheduler which was merged into the 2.6.23 release of the Linux kernel. CFS scheduler implementation is not based on run queues[2]. The CFS scheduler defines one default policy to every tasks in the system. This scheduling policy is called *NORMAL*. It is the standard Linux time-sharing schedule that is intended for all threads that do not require the special priority mechanisms. The CFS scheduler also has some others scheduling policies to give high priority to the tasks. A task scheduled with *FIFO* policy has more priority that any other tasks scheduled with *NORMAL* policy.

The main signal processing tasks of our architecture (Sampling, FFT, Averaging, Compression and Transmission) have been scheduled with *FIFO* policy to ensure that these tasks have a high priority in the system in terms of execution. In order to simplify the naming in this report we refer to *FIFO* policy as *Priority* policy.

C. Sampling Rate

The RTL-SDR device can scan in the range of 24-1766 MHz in channels of 2.4 MHz bandwidth. For hopping between channels the RTL-SDR device takes 24*ms* to change to the new channel. Each time the sampling process reads from the RTL-SDR device with a certain spectrum resolution, the data is saved in a structure called *segment*.

The Figure 6 shows the segment intervals when the threads are scheduled with *Priority* and *non-Priority* policy. Using the *Priority* policy the system can read the segments faster in the both scanning scenarios. The first one is when the sensor scans in a single channel of 2.4MHz. The sensor can read about 5% more segments with *Priority* policy. In the second scenario the sensor scans the whole spectrum and the hops between channels are needed. In this case using *Priority* policy the sensor can read about 14% more segments in the same time. The Linux scheduler adds a bigger penalty between both scheduling policies (priority and non-priority) when the sampling process is blocked and it has to wait for the hopping to the new channel.

[2]CFS uses a red-black tree implements a timeline of future task execution. The main concept of CFS scheduler is *sleeper fairness*, which considers sleeping or waiting tasks equivalent to those on the run queue. In this way, tasks waiting for I/O receive a comparable share of the processor when the eventually need it. Also, the measure for processing time is nanoseconds (not time-slices).

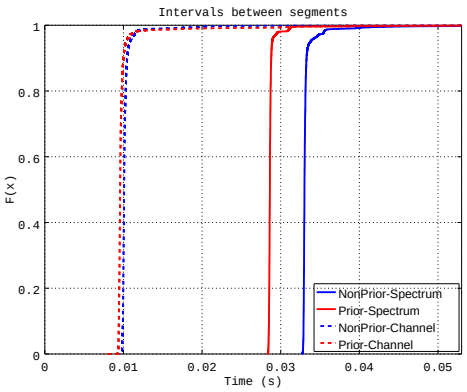

Fig. 6: ECDF of intervals with Priority and Non-Priority in the two scanning modes: whole spectrum (24-1766 MHz) and single channel (102.6-102.8 MHz).

The sampling rate is an important metric to know how many samples we can read from the RTL-SDR devices and how many samples can be processed in the sensor. The table I shows that the system can read and process more samples with a Priority policy. However the sensor processes one order of magnitude less samples in the whole spectrum scanning/processing mode, and two orders of magnitude less samples in the single-channel scanning/processing mode with respect to the sampling rate that the dongle USB can provide. It is important to remark that the sensor has a single-core unit to compute all the operations that the system needs to scanning and processing the signals.

Sampling Rate (MS/s)

Scanning Mode	Theoretical Read	Read&Processed (Priority)	Read&Processed (Non-Priority)
24MHz-1766MHz	0.024	0.0087	0.0075
Same channel of 2.4MHz	2.4	0.027	0.025

TABLE I: Sampling rate with different priority policies using GPU to compute the *FFT* with *fft_size=256* and *avg_factor=10*

D. Interference Task

The processing signal tasks are the core of the sensing nodes, as we commented in section V. These tasks are the responsible of sampling, *FFT*, averaging, compress and transmitting the data. All these tasks are implemented as threads in the operative system. We use queue structures to synchronize the threads and share the information between them. The memory and the queues size are finite, therefore the queues between the threads to share the data can be completely full. If the queue is full, the thread that writes in the queue must wait or discard data until the queue has some free space to allocated new data. We face on producer-consumer problem (also known as the bounded-buffer problem). There is no problem if the consumer processes faster the data that producer can write in the queue. But typically the opposite happens.

A possible scenario could be that other non-priority processes want to execute in the sensors while signal processing tasks are executing. These non-priority tasks could be used to

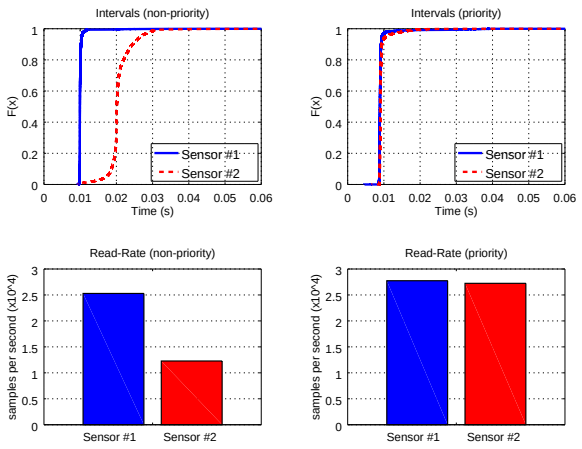

Fig. 7: Segment intervals and read rate of the sensors when an interference task executes (only in sensor 2) using non-priority (on the left) and priority (on the right) scheduling policies.

update part of the sensor's software or to obtain information about the sensors. The point is that the processes responsible for scanning spectrum could be blocked or they could execute with a minor execution rate expected. Another additional problem could be that these updating tasks execute just in one of the sensor, causing different sampling rate in both sensors. In order to get a good signal correlation between them we need to guarantee that sensors scan using the same sampling rate even an interference task executes in one of the sensors. The figure 7 shows the scenario when an interference task executes just in one of the sensors and the system is configured to use non-priority and priority scheduling policies. The results about the non-priority configurations when an interference task is executing are not good because one sensor reads at double rate than the other sensor. However, using priority scheduling policies the sampling process has more priority that the interference task. Therefore the sampling rate of both sensors is almost equal, which is very helpful to compute the signal correlation because the data of both sensors is synchronized in sampling time.

E. Improving processing time

Each segment read in the sampling thread is transformed to the frequency-domain using the *Fourier-Fast-Transform* in the *FFT* thread. The computation of the *FFT* is an expensive operation in our block chain. We have to be very carefully with the time spent for every thread in the chain. If the *FFT* thread takes long time to process its job, the read rate from the queue is slower than sampling thread can write in the queue. And the result will be that sampling process will be blocked and some samples are lost.

Our way to improve the processing of *FFT* is using *GPU* instead of *CPU* to compute the *FFT*. We rely on the library[3] which exploits the *BCM2835 SoC GPU* hardware to deliver ten times more data throughput than is possible on the

[3]http://www.aholme.co.uk/GPU_FFT/Main.htm

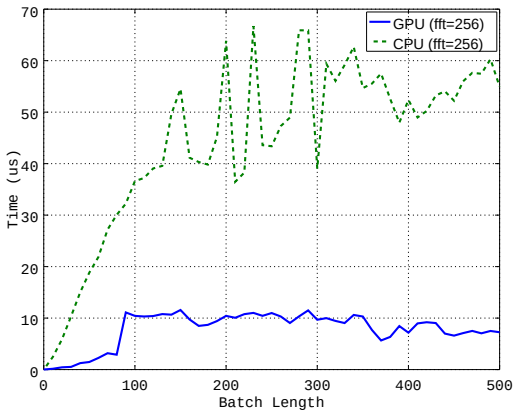

Fig. 8: FFT computation time using GPU and CPU of the Raspberry Pi board.

700 MHz ARM CPU. However, the communication between *CPU* and *GPU* has a latency of around 100 microseconds (which is much longer than the shortest transform requires). For this reason, we perform the execution of transforms in batch of segments. The figure 8 shows how *GPU* improves the computation time of the *FFT* in comparation to *CPU* regardless of the batch length used. The computation time is reduced around ten times in average. This improvement helps to process the segments faster and therefore avoid blocking the sampling thread.

In this section we have introduced different challenges to address in collaborative wideband spectrum monitoring. The results in summary are:

- We have achieved a time synchronization between the sensing nodes less than *1ms.* in local networks and Internet using NTP.
- We have proposed to use *Priority* scheduling policy to every signal processing tasks of our architecture.
- We can read and process 2.7×10^4 samples per second and in a single-sensor scanning in the same channel.
- We have demonstrated that sampling rate of the sensors is not affected by interference task thanks to *Priority* scheduling policy.
- We have optimized in ten times the *FFT* computation using GPU unit instead of *CPU*.

VII. Signal Correlation

The correlation between the data scanned from the sensors is important in order to make collaborative decisions. For instance, if a sensor finds out an anomaly in the signal, other sensors can help to confirm this anomaly using a correlation function with the data scanned in the same frequency. Another interesting collaborative scenario consists of a few sensors scanning together to cover a bandwidth larger that they could analyze it alone.

The correlation function depends on several factors as sensor noise, channel frequency, interference and data synchronization. In this report we focus in how the data is well-synchronized as it was explained in the previous section. In

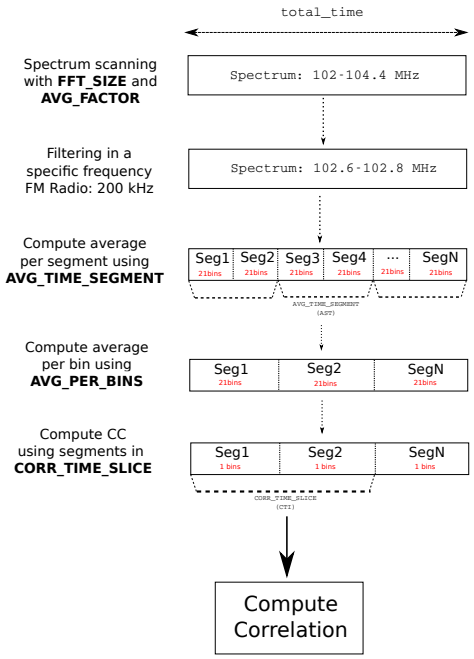

Fig. 9: Schematic process to determine the signal correlation of the sensors.

addition, the correlation depends on some scanning and post-processing parameters. The figure 9 shows the process to compute the correlation between two signals obtained from two different sensors. We consider the following relevant parameters to evaluate the signal correlation process:

fft_size: The spectrum resolution is given by the number of bins (fft coefficients) that contains one segment read in a 2.4 MHz bandwidth channel. Depending on the nature and bandwidth signal or the goal of the application it could be better to use a high or small number of bins.

avg_factor: This parameter is used in scanning time to reduce the influence of random noise. A configurable number of segments, denoted by this parameter, are then averaged to form a single *fft_size*-segment spectrum. Applying a high value of average could imply that two different signals (but similar in nature and shape) would have a high correlation.

avg_time_segment: This parameter is similar to the previous one but this one is used in post-processing time. It allows to get the raw data and compute average per segments in order to avoid synchronization problems and anomalies while we keep the sampling rate.

avg_per_bins: A segment contains *fft_size*-bins. In order to configure the spectrum resolution this parameter can be used to reduce the number of bins per segment computing the average between them.

corr_time_slice: This parameter specifies the time slice used to compute the signal correlation between both sensors. Depending on the accuracy required it is possible configure the process to use milliseconds or seconds slices.

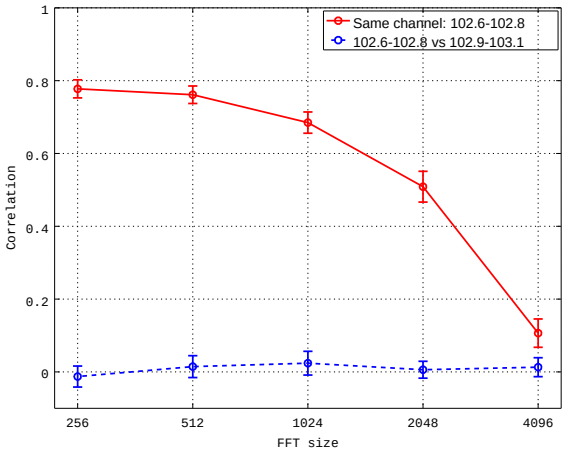

Fig. 10: Scenario 1: fft_size evaluation to correlate same and different FM radio channels using avg_factor(10), avg_time_segment(0.1), and corr_time_slice(10).

A. *FM Radio environment*

FM radio bandwidth has been chosen to evaluate our distributed spectrum monitoring architecture and the correlation procedure. 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. We configured our system with two sensors to scanning the frequency 102-104.4 MHz. Five different FM radio channels are located in that frequency. The sensors are located near each other in an indoor environment. Both sensors use Ethernet connection on the same local network to synchronize over time through *NTP*.

The first scenario that we present is shown in the figure 10. We use FM radio channels of 200 kHz bandwidth to determine the correlation between the two sensors. A FM radio channel has *fft_size* $\times$ 0.08 number of bins approximately because the RTL-SDR device always reads in a channel of 2.4 MHz bandwidth with a *fft_size* bins resolution. Both sensors were configured to use different values of *fft_size* but we keep the same value in the following parameters: *avg_factor=10 times*, *avg_time_segment=0.1 sec*, *avg_per_bins=1* and *corr_time_slice=10 sec*. Correlation between different signals is really close to 0 for any value of *fft_size*. On the other hand, the correlation between same signals is above 0.7 (threshold statistically accepted in Pearson correlation) just in cases *fft_size* is equal to *256, 512, 1024*. This configuration allows detect correlation between signals with two limitations: the minimum time resolution is *10 sec.* and the spectrum resolution is 200 kHz (FM radio channel bandwidth).

In the second scenario (shown in the figure 11) we skip some average process in the computation of the correlation between the signals in order to get better time and spectrum resolution. The figure 11 shows the correlation results using different values of *fft_size* in the scanning process. In this scenario we reduced the *avg_factor=1*, we keep the same *avg_time_segment*, we do not compute any average by bins and we set *corr_time_slice=1 sec*. The goal is to obtain the optimal

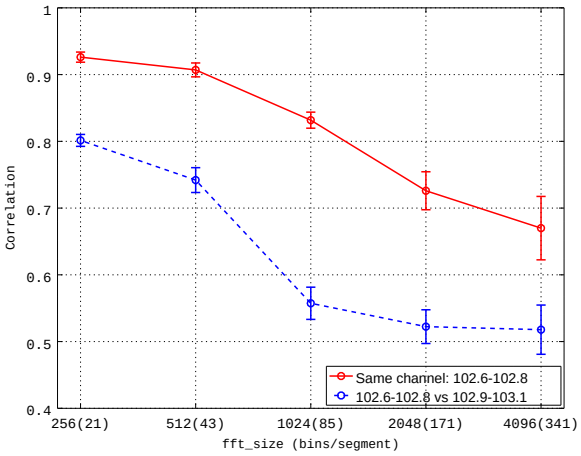

Fig. 11: Scenario 2: fft_size evaluation to correlate same and different FM radio channels using avg_factor(1), avg_time_segment(0.1), and corr_time_slice(1).

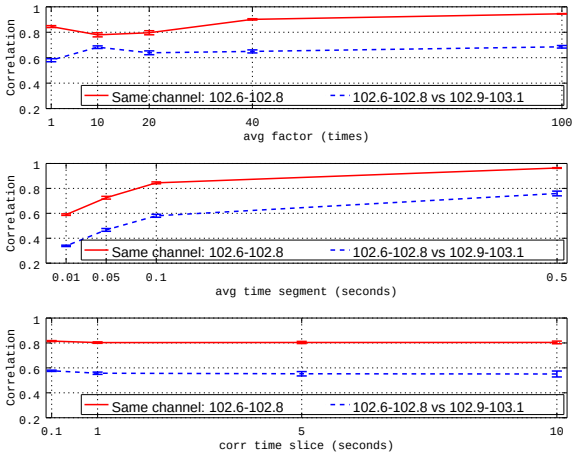

Fig. 12: Correlation of same and different signals evaluating the scanning and post-processing parameters in scenario-2 (*fft=1024*). On the top: *avg_factor* evaluation. On the middle: *avg_time_segment* evaluation. On the bottom: *corr_time_slice* evaluation

value of *fft_size* which give us a high correlation between same channels scanned in both sensors and low correlation between different channels. The correlation between same and different FM radio channels is high when *fft_size* is set to *256*. The spectrum resolution in this case (21 bins) is not enough to distinguish two different channels in terms of correlation (notice that we got a good results in scenario-1, figure 10, with the same *fft_size* because we compute more average-steps). However, if the *fft_value* is increased up to *1024* the correlation between same channels is high (0.85) and the correlation between different channels is not determinant (0.56). Therefore this scenario, using *fft_size=1024*, allows compare correctly different FM radio channels with a *0.1 sec.* time resolution and *2.3 kHz* spectrum resolution.

Using the scenario-2 we evaluate other important parameters in the correlation process. The *avg_factor* is another important parameter using in scanning time that determines how many segments are used to compute the average to get the final segment. It is really useful to reduce the noise and anomalies in the signal. The figure 12 (on top) shows the correlation between FM radio channels using different *avg_factor* values. No averaging in the scanning process (*avg_factor*=1) produces good results in terms of correlation between same and different signals. However, with *avg_factor=10* we can see how both correlations are close to each other. We think that this non-trend behaviour could be due to hardware-software artifact or due to fundamentals aspects of the FM radio transmission. For higher values of averaging (20 to 100) we can see how the correlation improves significantly. Depending on the nature of the signal it could not be a good idea use a large *avg_factor* value because the sensor takes around 1 second to get 100 segments (of 1024 bins) and compute the average to get the final segment. The *avg_time_segment* parameter, using to post-processing the data, specifies the time slice used to compute the average between segments. Computing average of segments per time allows us to reduce the noise and some synchronization issues between the sensors. The figure 12 (on the middle) shows how if we use a low value of average (0.01) the correlation in both cases (same and different signals) are not conclusive. On the other hand using a high value of average (0.5) the correlation in both cases is above 0.7, therefore we can not distinguish between two different FM radio signals. In order to get the best configuration to distinguish two different FM radio channels *avg_time_segment=0.1* should be used. The *corr_time_slice* is a post-processing parameter to define the time slice used to compute the correlation during a time series of data given. The figure 12 (on the bottom) shows how using a low value we get best results in terms of correlation. The most important point here is that using any value of *corr_time_slice* both scenarios (same and different channels) can be distinguished easily.

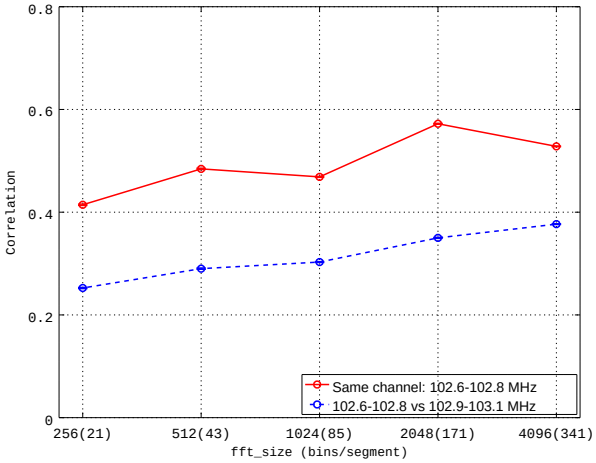

Fig. 13: Scenario 3: fft_size evaluation to correlate same and different FM radio channels using avg_factor(1) and no averaging strategies.

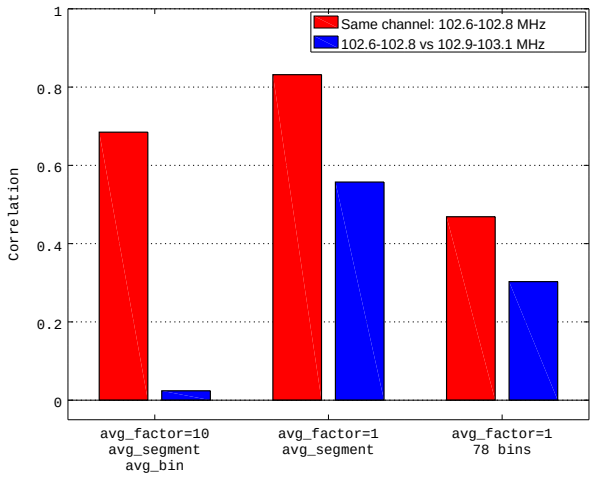

Fig. 14: Signal correlation using different averaging strategies (fft_size=1024)

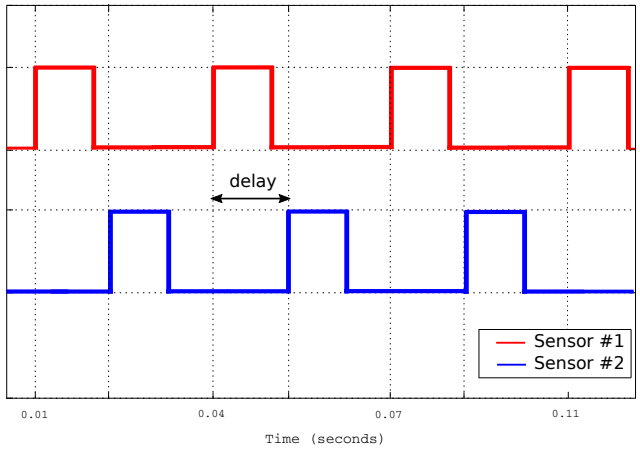

Fig. 15: Sampling rate representation of two sensors with no average strategies (*fft_size=1024*).

Lastly, we present the scenario where no averaging strategies are used to compute the signal correlation between the sensors. Analyze the data without averaging strategies give us the chance to get the best time and spectrum resolution that the hardware can provide. The figure 13 shows the signal correlation with different values of *fft_size* and no average. The results are no good to distinguish the cases in which the same or different signals are being comparing. This specific case is well-explained in the following subsections. We also explain how the non-average strategies affect dramatically to the signal correlation and how it can be solved.

B. Understanding the Average

The average is a technique highly accepted to find a central value of a discrete set of numbers. In our scenario the average is used to get the central tendency of bins, segments and seconds. Computing the average in terms of power allows us to reduce and filter noise and anomalies in the signal. On the other hand, computing the average means that we have less resolution in time and spectrum. If we compute the average in time each 0.1 seconds it means that we could not detect signals that take less than 0.1 seconds. Spectrum resolution is also affected by average. Taking 500 kHz to compute the average could not be enough to detect signals with a less bandwidth (for instance, FM radio channels use 200 kHz bandwidth).

The figure 14 shows the correlation summary with the different scenarios checked. In the first scenario the correlation using different sensors is good comparing the same (0.75) and different channels (close to 0). In the second scenario we skip the average process per bin and the *avg_factor* is set to 1. The results are good in terms of signal correlation but we can notice how the correlation is close to (0.7) when different signals are compared. In the first two scenarios averaging process were computed and therefore we lost time and spectrum resolution. At the end, if we completely reduce the averaging strategies we can see in the third scenario of the figure 14 how the correlation is worst.

At this point the question is why the correlation is getting worst when the raw data is analyzed without any average strategy. As we commented in the previous sections *NTP* keeps synchronized both sensors in time with a delay between them about *0.5ms*. On the other hand each segment is read about *3ms* in average. Therefore the synchronization time should not be a problem. The figure 15 shows a representation of the spectrum read from each sensor. In the default configuration the sensor reads 10 segments from the USB dongle, and then these segments are sent to the GPU to compute the *FFT*. Even the computation in the *GPU* is quite fast, the single-core of the sensor is blocked during 0.20 seconds roughly. At the end, both sensors are completely synchronized in time but the sampling threads are not executing at the same time in each *CPU*, therefore the data read for each sensor is not the same. We believe that this is the main reason because the signal correlation is not good when raw data (without average) is analyzed. Even reducing the sampling rate we have the same problem because the bottle-neck is the *FFT* thread. In the following section we propose a closed-loop architecture that would solve the problem.

C. Closed-loop architecture

Depending on the final application the scanning and post-processing parameters could change in order to get better results due to the nature and shape of the signals. The architecture proposed and implemented in this report can achieve good signal correlation using different averaging strategies but reducing time and spectrum resolution.

For applications requiring more time and spectrum resolution we need to avoid averaging strategies. For that, we need to compensate the execution delay that we have between the sensors (figure 15). We expect that this delay is stable in time. The idea is to compute and get this delay in the server side, where all the data is sent from the sensors. Once the delay is computed the server should send the information to the sensors in order to close the feedback loop. Then, one of the sensors should wait a certain time of milliseconds in order to get a better synchronization with another sensor. In this way the server is always computing the possible execution delay

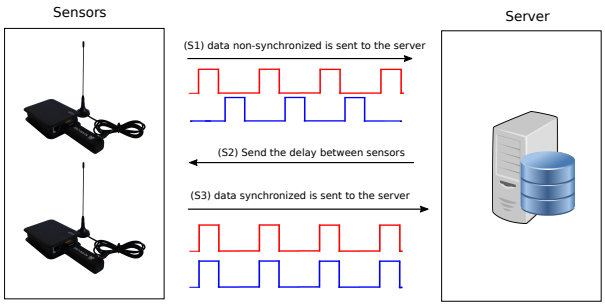

Fig. 16: Closed-loop scenario where the server computes the delay between the sensors and send back useful information to the sensors to mitigate the delay

between the sensors and send them the enough information to synchronize the sampling process as soon as possible. The figure 16 shows the *closed-loop* scenario where the delay between sensors is computed and compensated by the server.

We envision a system where the time synchronization between sensing nodes is achieved with NTP and the system could improve their accuracy using a fine synchronization provided by closed-loop architecture.

VIII. Conclusion and Future Work

We have presented the main challenges to collaborative wideband spectrum monitoring system. We have demonstrated how it is possible to reach a good time synchronization between the sensors in different system conditions while they are scanning the spectrum and sending the data to the server. We have proposed and evaluated the correlation process using FM radio channels of *200 kHz* of bandwidth. We have shown that our system can achieve a good correlation value between signals that are measured using two different sensors. Our experiments show a good time resolution (*0.1 sec.*) and spectrum resolution (*2.3 kHz)* to detect a good correlation in FM radio channels.

We have further proposed a distributed architecture for collaborative wideband spectrum monitoring system without averaging strategies in order to get better time and spectrum resolution. This closed-loop architecture allows to synchronize the sampling process of the sensors reducing the delay between them that it is due to fundamental limitations of hardware and software. Develop and evaluate this architecture is a future work to determinate if this proposed architecture will allow us to obtain an even better correlation strategy to make smart and collaborative decisions.

Acknowledgment

I would like to extend a warm thank you to my supervisor Dr. Domenico Giustiniano for his continued and precious help. I am also very thankful to Dr. Vincent Lenders and Damian Pfammatter for their collaboration and high quality contributions.

This report has been partially supported by the Madrid Regional Government thought the TIGRE5-CM program (S2013/ICE-2919)

Publication

An earlier version of the present work was accepted for publication [7] IPSN 2015 held in Seattle, USA.

References

[1] "National frequencies allocation in spain," https://es.wikipedia.org/wiki/Cuadro_Nacional_de_Atribucin_de_Frecuencias.

[2] "Microsoft spectrum observatory," http://observatory.microsoftspectrum.com/.

[3] J. Naganawa, H. Kim, S. Saruwatari, H. Onaga, and H. Morikawa, "Distributed spectrum sensing utilizing heterogeneous wireless devices and measurement equipment," in *New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 2011 IEEE Symposium on*, May 2011, pp. 173–184.

[4] 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, March 2011.

[5] 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.

[6] A. Arcia-Moret, E. Pietrosemoli, and M. Zennaro, "WhispPi: White space monitoring with Raspberry Pi," *Global Information Infrastructure Symposium, GIIS 2013*, 2013.

[7] 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 Symposium on Information Processing in Sensor Networks*, ser. IPSN '15, Seattle, WA, USA, April 2015, pp. 1–2.

[8] D. Pfammatter, D. Giustiniano, and V. Lenders, "A Software-defined Sensor Architecture for Large-scale Wideband Spectrum Monitoring," in *Proceedings of the 14th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '15, Seattle, WA, USA, April, pp. 71–82.

[9] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Comput. Netw.*, vol. 50, no. 13, pp. 2127–2159, Sep. 2006.

[10] E. Axell, G. Leus, E. Larsson, and H. Poor, "Spectrum sensing for cognitive radio : State-of-the-art and recent advances," *Signal Processing Magazine, IEEE*, vol. 29, no. 3, pp. 101–116, May 2012.

[11] Z. Q. Z. Quan, S. C. S. Cui, a.H. Sayed, and H. Poor, "Wideband Spectrum Sensing in Cognitive Radio Networks," *2008 IEEE International Conference on Communications*, 2008.

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

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

[14] J. Unnikrishnan and 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.

[15] "Google spectrum database," https://www.google.com/get/spectrumdatabase/.

[16] 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.

[17] 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.