A Software-defined Sensor Architecture for Large-scale Wideband Spectrum Monitoring

Damian Pfammatter
ETH Zurich
Zurich, Switzerland
pfammatterdamian@gmail.com

Domenico Giustiniano
IMDEA Networks Institute
Madrid, Spain
domenico.giustiniano@imdea.org

Vincent Lenders
armasuisse
Thun, Switzerland
vincent.lenders@armasuisse.ch

ABSTRACT

Today's spectrum measurements are mainly performed by governmental agencies which drive around using expensive specialized hardware. The idea of crowdsourcing spectrum monitoring has recently gained attention as an alternative way to capture the usage of wide portions of the wireless spectrum at larger geographical and time scales. To support this vision, we develop a flexible software-defined sensor architecture that enables distributed data collection in real-time over the Internet. Our sensor design builds upon low-cost commercial off-the-shelf (COTS) hardware components with a total cost per sensor device below $100. The low-cost nature of our sensor platform makes the sensing approach particularly suitable for large-scale deployments but imposes technical challenges regarding performance and quality. To circumvent the limits of our solution, we have implemented and evaluated different sensing strategies and noise reduction techniques. Our results suggest that our sensor architecture may be useful in application areas such as dynamic spectrum access in cognitive radios, detecting regions with elevated electro-smog, or simply to gain an understanding of the spectrum usage for advanced signal intelligence such as anomaly detection or policy enforcement.

Categories and Subject Descriptors

C2.3 [**Network Operations**]: Network monitoring

Keywords

Spectrum monitoring, Crowdsourcing, Wideband, Distributed

1. INTRODUCTION

Today, the *electromagnetic (EM) spectrum*, i.e. the range of possible frequencies for EM radiation, is heavily used by diverse types of wireless communications, radar and broadcasting services. To regulate the usage of the available radio spectrum, governmental bodies allocate frequencies to different radio services according to a well-defined process. Each country maintains its own national frequency allocation plan which dictates how the EM spectrum shall be used. However, despite this strict allocation scheme of the spectrum, the actual usage of the spectrum at different geographical places and times is often unknown.

This problem is due to the fact that today's spectrum measurements are primarily performed by governmental agencies which drive around using expensive and bulky specialized hardware [10, 17]. This monitoring approach does not scale well and is not able to cover the pervasive deployment of wireless networks as well as the increasing range of spectrum frequencies being used. Recent suggestions to enable wide-scale and real-time spectrum monitoring have therefore been to build a networked and distributed infrastructure using remote spectrum analyzers [1,10,16], or leverage the masses and to crowdsource the measurement stations by trading-off radio device sensitivity with cooperation [15,17]. Our vision in this work is aligned with these ideas of exploiting the crowd for scalable monitoring of the spectrum's actual usage at different frequencies, locations and times. However, in contrast to these previous works, we envision a monitoring system that builds upon low-cost hardware and we study the design challenges at system level such that a large number of nodes can be distributed to collaboratively enlarge the system's overall coverage. The collaboration would allow individual sensing nodes working together in a coordinated fashion towards a common goal of monitoring the spectrum over the defined area of coverage. For this, the individual nodes would report their spectrum sensing results to a fusion center that stores and assembles the collected data.

We believe that a fine-grained map of the actual EM spectrum usage would become a highly desirable service for a wide range of different applications. Cognitive radios could for example use the data as a baseline for optimizing their efforts to dynamically accessing the spectrum. The sensed data might also be used to locate regions with elevated electro smog, which may be an incentive for users to participate in this crowdsourcing system. Well another example might be to use the data for advanced signal intelligence tasks in order to detect anomalies, policy violations or attacks.

To support low-cost distributed wide-spectrum monitoring, we propose in this paper a flexible software-defined sensor architecture. Our solution is designed for large-scale deployments and has been tailored to low-cost sensor devices that can be operated over the Internet access of home users. Our target hardware platform for the sensors are therefore cheap commercial off-the-self (COTS) hardware components such as the RTL-SDR USB dongles [17] and single-board computers such as the Raspberry Pi. The total sensor cost of the hardware components (including antenna) is below $100. This makes the platform well suited for wide adoption and crowdsourcing. However, the challenges in building a distributed spectrum monitoring network are much more demanding, given that the sensing and processing limitations of low-cost hardware require efficient scanning, preprocessing, compression and noise

IPSN'15, April 14–16, 2015, Seattle, WA, USA

http://dx.doi.org/10.1145/2737095.2737119.

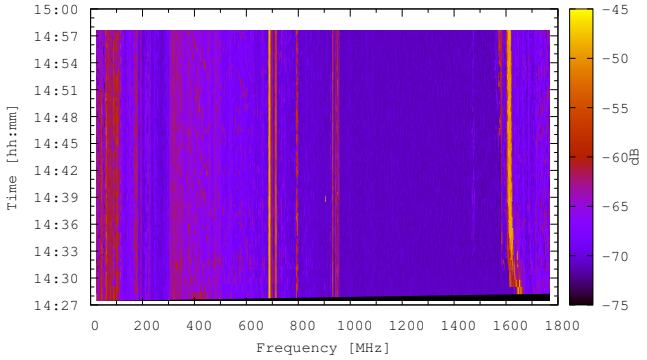

Figure 1: The figure shows a wideband waterfall frequency plot for the spectrum range between 24 and 1766 MHz that we generated with our low-cost sensor platform.

reduction mechanisms.

In this work, we suggest a conceptual design for the architecture of low-cost wideband spectrum sensing nodes. We demonstrate an actual sensor implementation for a spectrum range between 24 and 1766 MHz. As part of our investigations, we implement an actual system prototype to prove the feasibility and usefulness of a distributed spectrum monitoring system and assess the limits of the approach. Our sensor system is able to generate detailed real-time wideband spectrum maps such as the one depicted in Figure 1. Our main contributions are the following:

- We present a software-defined sensor architecture for distributed low-cost wideband spectrum sensing. Our sensor architecture is designed to be as flexible as possible in order to accommodate the requirements of various applications and collaboration models. The software code is made available as open source at `http://github.com/pdamian/rtl-spec`.
- We propose and evaluate different wideband scanning strategies to overcome the hardware limitations of low-cost radios. We show that it is possible to improve the overall spectrum sensing accuracy with strategies that prioritize frequency bands of bursty activities over bands which tend to remain constant over time.
- We propose an online calibration approach that is able to correct the large frequency error of low-cost sensors without human intervention. Our approach exploits information from GSM base stations in communication range of our sensors as well as onboard temperature measurements in order to reduce the error by a factor of at least 10 compared to an approach without correction.
- We demonstrate the feasibility of using low-cost single-board computers for performing in real-time all the necessary signal processing tasks for spectrum analysis and data compression. Our current implementation which exploits the onboard GPU of the Raspberry Pi can process FFT sizes up to 4096 (corresponding to a frequency resolution of 500 Hz) without data losses at the backend.

Our sensor design is unique in the field of spectrum monitoring solutions. Compared to specialized spectrum analyzers [21] or more advanced software-radio platforms like the Universal Software Radio Peripherals (USRPs), the hardware platform we rely on is a factor 50 to 500 times cheaper. In contrast to existing spectrum analysis software projects for RTL-SDR based USB dongles[1], our architecture is designed for distributed operation on single-board computers and incorporates various scanning strategies, spectrum analysis and error/noise correction techniques for flexible and configurable remote wide-spectrum monitoring.

The remainder of this paper is organized as follows. We present related work in Section 2. In Section 3, we present the ideas and benefits of monitoring the frequency spectrum in a distributed approach. We then introduce the hardware components of our sensor in Section 4. Afterwards, we present the system's composition from an architectural and design point in Section 5. The sensor design section is followed by an in-depth evaluation in Section 6. The conducted evaluation helps in understanding and optimizing our design decisions. We finally draw the conclusion in Section 7.

2. RELATED WORK

The spectrum utilization has been conventionally studied in cognitive radios. Spectrum sensing of a narrowband frequency spectrum is part of the cognitive capabilities, where devices monitor the available band and detect the spectrum holes [4, 6, 19].

While monitoring and sensing a narrowband frequency spectrum has been well studied [28], monitoring larger spectra is more challenging and has been the focus of more recent investigations, where a wider spectrum is sensed to identify holes which might then be accessed by cognitive radio users. One approach is to use high performance analog-to-digital converters (ADCs). Such advanced devices provide high sampling rates and can thus sense wideband spectra. However, high performance ADCs are very expensive, power consuming and require substantial computations for signal processing [11]. To monitor a 500 MHz band, a 1 GHz ADC is required, with a high cost of ≈$775 [3].

Another approach is to assume that all subchannels are independent. In this way, the wideband monitoring problem is translated to the one of monitoring narrow band channels one by one in a sequential order. A comparable method is for instance used, at a small scale, in Wi-Fi [28]. When the monitored spectrum is large and contains many narrow band channels, this approach gets slow and does not scale well. Furthermore, at any point in time, only a single narrow band channel is monitored. Hence it is likely to miss various short-lived signals. The huge complexity of the optimal detector for wideband spectra leads to the need for approximations or simplifications of the detection algorithm [6]. In this work, we approach this problem by introducing a hopping strategy that can adaptively scan the most appropriate set of bands.

Recent investigations proposed to adopt compressed sensing techniques to monitor wideband frequency spectra. Signals are typically sampled at sub-Nyquist rates and therefore acquired by relatively fewer measurements. This approach works when the occupation of the spectrum is sparse [18] [20]. However, all applications of compressed sensing [24] [7] [8] require access to dedicated hardware, and, to the best of our knowledge, there is currently no simple wideband monitoring technique based on compressed sensing that runs on unsophisticated low-cost *commercial off-the-shelf (COTS)* hardware.

Spectrum sensing strategies have been developed for cooperative and non-cooperative spectrum sensing solutions [12, 14, 26], some of which for wideband sensing [10, 13, 23]. Cooperative sensing has been introduced in [15], and it assumes that a collection of sensing nodes distributively monitor a frequency spectrum. Typically, each sensing node performs simple narrowband monitoring with a front-end which is less sensitive than the one used in non-cooperative solutions. One of the main problems in distributed and collaborative sensing is to reduce the load of the network. Thanks to spatial diversity gain, individual nodes can distributively apply compressed sensing techniques [23] or just send a few bits, which could suffice in certain applications [13]. The works above did not

[1] `http://sdr.osmocom.org/trac/wiki/rtl-sdr#KnownApps`

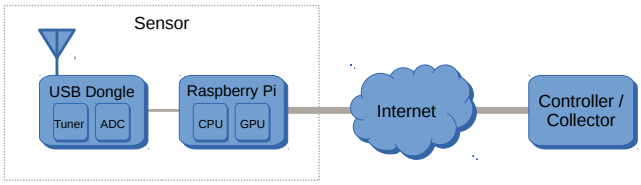

Figure 2: System architecture: The three system components – sensors, controllers and collectors – connect over a network, typically the public Internet.

complement the proposed theoretical analysis with a system-level design and implementation.

On the other hand, [10] introduced SpecNet, a platform that allows researchers to conduct spectrum measurement studies remotely in real time for opportunistic spectrum access. They implement and test distributed collaborative spectrum sensing applications. Microsoft Spectrum Observatory [1] is a system 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 one sensing station is more than $5000 whereas our target sensor platform is less than $100.

The current wideband spectrum monitoring techniques have clear limitations when trying to adopt them to low-cost commodity platforms. A recent work by Yoon et al. [27] employs analog bandwidth tunable filters and analog energy detectors, and provides a fast, low-cost and energy-efficient monitoring approach. In [5], they proposed to use the Raspberry Pi to store wideband spectrum measurements performed by a portable RF spectrum analyzer that is externally attached to the device. The work most related to the ideas presented in this paper comes from the preliminary investigations done by Nika, Zhang et al. in [17]. The authors performed an initial *feasibility study* to verify the efficiency of using RTL-SDR USB dongles connected to mobile devices (such as laptops or smart-phones) to build a low-cost commoditized spectrum analysis system. The conclusion of their work is the confirmation that commoditized real-time spectrum monitoring will be possible in the very near future.

3. DISTRIBUTED SPECTRUM MONITORING

In this section we show the requirements and benefits of a wideband spectrum monitoring system relying on a distributed approach. We then propose an architectural design, highlighting its components and their corresponding tasks towards the overall monitoring goal.

3.1 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. The system's expected fields of application, as well as its intended deployment scenario influence the definition of the following design goals:

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.

Small Form-Factor. The system shall consist of pocket-sized sensing nodes that allow for an easy distribution and sustain the envisioned deployment scenario.

Figure 3: Sensor hardware configuration: The sensing nodes consist of a single-board computer serving as the host computing device for a DVB-T USB dongle, which in combination with the attached antenna builds the front end for RF reception.

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

Distributed. The system shall consist of distributed sensing nodes, each of which can autonomously execute its tasks and send the data to a collecting unit.

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 sensors shall buffer their measurements in order to deliver them later when the connection is re-established.

COTS Hardware. The system's sensing nodes shall be built upon commercial off-the-shelf (COTS) hardware components.

Flexibility. The system shall remain flexible to cope with its different fields of application.

3.2 System Architecture

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

Sensor. The key components in the design of the system are its sensing nodes. To satisfy the system requirements defined in section 3.1, the individual sensors consist of small-sized low-cost embedded computing devices connected to a simplistic RF front end and a general purpose antenna. The sensors' architecture, their actual implementation (together with the involved signal processing steps) and an in-depth evaluation thereof, are the key considerations in this work.

Controller. The controller is the central server to command and control the associated sensing nodes. It is therefore the central access point to the distributed sensors. The controller's purpose is the assignment of suitable monitoring tasks to individual sensors, as well as the tracking of their states.

Collector. The collector nodes are the system's data collection units. Their responsibility is currently about large-scale preprocessing[2]. We also foresee that the collector will fuse spectrum data from distributed nodes. Approaches as the one proposed in [2] could also be integrated.

4. SENSOR HARDWARE PLATFORM

Our sensor is built upon four major hardware components – a *single-board computer (SBC)* platform, a *RTL-SDR USB dongle*, an *antenna*, and a *temperature sensor*. 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

[2]For instance the video filter in the collector interpolates the recorded data (i.e. fills gaps in the spectrum that have not been recorded at a certain point in time), and makes it ready to be displayed.

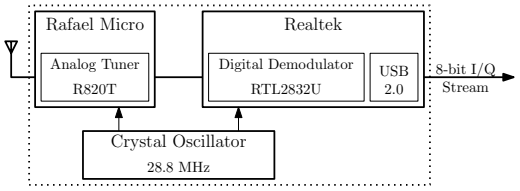

Figure 4: RTL-SDR block diagram: The USB dongles consist of two major components, an analog tuner and the RTL2832U demodulator. Both components are clocked with a shared onboard crystal oscillator.

both the RF receiver, as well as the connected temperature sensor. The typical hardware configuration of our sensors is shown in Figure 3. In what follows, we describe and motivate the usage of these hardware components in more detail.

4.1 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-defined radio (SDR) since the chipset allows the transfer of the captured raw I/Q samples. The RTL2832U embeds an 8-bit analog-to-digital converter (ADC) that samples at 28.8 MS/s. This however does not correspond to the actual rate of samples you can get out of the device mainly due to the limitations introduced by the USB 2.0 interface. The RTL2832U decimates the stream of samples to a user configurable rate. The theoretical limitation of the USB 2.0 interface is at 3.2 MS/s. However in practice only sampling rates below 2.4 MS/s can typically be received without any losses.

RTL-SDR USB dongles exist with different types of tuners. Our architecture supports any dongle as long as it reports the raw I/Q samples. For the evaluation in this work, we rely on dongles with the Rafael Micro R820T tuner, which supports a continuous tuning range between 24 and 1766 MHz. Figure 4 shows a block diagram of the dongle's overall architecture. While cheap, RTL-SDR dongles have limitations in terms of accuracy and sensibility. The dongles are clocked with a 28.8 MHz onboard crystal oscillator which is quite unstable with regard to temperature variations and frequency errors. The reported frequency instability is up to 50 ppm [9], way beyond the instability of professional spectrum analyzers. Further, the bandpass filters on the dongle for signal downconversion to intermediate frequency (IF) can only be tuned to bandwidths between 6 and 8 MHz while commercial spectrum analyzers support much smaller resolution bandwidths down to 1 Hz in order to decrease the noise floor and improve the level of structure detail in the scanned spectrum [10].

4.2 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 Raspberry Pi (RPi), 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 order to offload certain computational intensive calculations. In this work, we use the Raspberry Pi Revision B which has a CPU clocked at 700 MHz and 512 MB of RAM. For

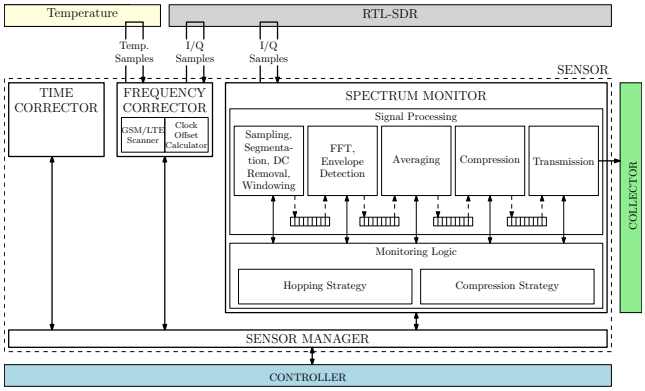

Figure 5: Sensor architecture: The design of our sensing nodes consists of four main components – spectrum monitor, time and frequency corrector, as well as a sensor manager. Sensors interface the (external) controller and collector nodes, and (internally) access RF data through the RTL-SDR device.

network connectivity we rely on the onboard Ethernet and an USB WLAN stick attached to one of the available USB ports.

4.3 Antenna

We currently equip our sensing nodes with the standard antenna shipped for the RTL-SDR dongles. It is a simplistic omni-directional antenna. Better antenna such as omni-directional discone antennas would certainly result in better reception quality for covering the wide range of frequencies our RF receiver supports. However, for our setup these antennas are typically too costly (often significantly more than our RTL-SDR dongle and SBC platform together) or soon too large to be distributed and therefore impractical considering our intended application scenario.

5. SENSOR DESIGN

This section describes our sensor architecture. We start by providing an overview and then detail out the system components for spectrum analysis, frequency hopping, error correction and control.

5.1 Architecture Overview

The design of our sensor architecture is depicted in Figure 5. The *spectrum monitor* is the main component which implements all 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. 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 transfered over the network according to the bandwidth constraints of the network connection.

The *sensor manager* communicates with the remote controller for the reception of monitoring requests. To cope with frequency and time errors of the used low-cost hardware, two separate modules constantly estimate frequency and time offsets and send them to the sensor manager. The frequency correction has also access to the RTL-SDR. The access to the RTL-SDR is mutually exclusive so that only one process can read I/Q samples at a time. The

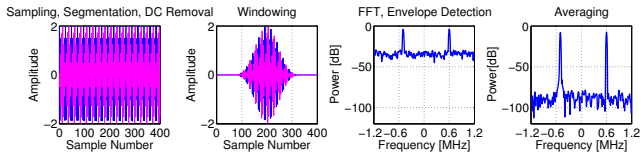

Figure 6: Signal processing steps on the sensor: The signal is first sampled, segmented and the DC offset is removed. Then, windowing is applied to reduce spectral leakage. After that, the signal is transformed into the frequency domain with the FFT and the envelope is determined. Finally, multiple FFTs are averaged to improve the signal-to-noise ratio.

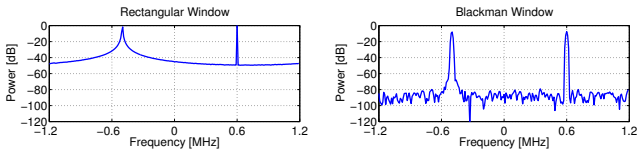

Figure 7: The spectra resulting from a test signal consisting of two sinusoidal components are compared for the application of rectangular and Blackman-Harris windows. The rectangular window is high in resolution but low in dynamic range, whereas the Blackman-Harris has lower resolution but higher dynamic range.

frequency corrector further interfaces with a temperature sensor attached to the SBC.

5.2 Signal Acquisition and Spectrum Analysis

The spectrum analysis techniques we have implemented in our sensor architecture are standard techniques as found on modern hardware-based digital spectrum analyzers [21]. The difference in our work is that we implement those techniques in software on a simplistic platform like a SBC and expose all parameters for remote configuration over the Internet through the sensor manager. In what follows, we describe the individual steps required for signal acquisition and spectrum analysis.

Acquisition and handling of digitized signals is implemented in a chain of different *signal processing blocks*. The individual blocks are connected through queues[3] building up a signal processing queuing system. The queuing system is controlled by the monitoring logic, and ends in the processed data being transmitted to the collector node. The introduction of queues between individual processing blocks has several advantages, of which the following two are considered most relevant here. First, the individual processing steps get decoupled from one another and can therefore be processed independently, i.e. interleaved or in parallel (dependent on the available processing architecture). And second, to increase efficiency in processing, blocks can be configured to process the queuing system's items at different granularities, i.e. one by one or in batches of several items. This is for instance helpful for the averaging block, which consumes a series of items and produces their average as single outcome. The involved signal processing steps are depicted in Figure 6 and described next.

Sampling. The sampling process interfaces with the RTL-SDR dongle and outputs a complex discrete time-domain signal. For our implementation, we make use of the C library *librtlsdr* from the OsmocomSDR project[4] to acquire the digital samples. The library allows specifying the dongle's tuning frequency, sampling rate, gain, etc. The resulting samples come as interleaved I/Q values in the form of unsigned 8-bit integers. The number of samples that are read from the RTL-SDR and all other parameters are configurable through the sensor manager. In addition, the sensor manager passes a frequency offset to the dongle to adjust the tuning frequency according to the actual error that is estimated by the frequency corrector block.

Segmentation. To reduce the noise level in our spectra, we need to record enough samples from the original analog signal depending on the desired spectral resolution. We do so by recording a time-domain signal of N samples, where $N = avg\_factor \times fft\_size$, and then divide it to a sequence of avg_factor segments, each having a number of fft_size samples. As the names suggest, fft_size belongs to the FFT length we intend to apply later and avg_factor to the number of segments that will be averaged.

DC Removal. The next step is to remove the DC offset. The digitized output of the RTL-SDR dongle typically contains some *DC bias*, i.e. the average of the digitized time samples is not zero [25]. In the frequency representation, the DC bias shows up as energy at the center frequency we tuned to. Very often the DC component does not come from the original analog signal but from imperfections within the process of analog-to-digital conversion. For large-sized FFTs[5], the DC bias becomes inconveniently large and needs therefore to be removed. This removal is the purpose of this processing step. We remove the DC bias by calculating the average value of the samples, and then subtracting that average from the original samples within that segment. The result is a time sequence with negligibly small DC component.

Windowing. Before using the FFT to transform the individual segments to the frequency domain, the segments are typically multiplied by a smoothing curve, also referred to as windowing function. The concept is illustrated in Figure 7, where we compare the spectra resulting from the same test signal, once using a rectangular windowing function (which corresponds to the use of no window), and once with a 4-term Blackman-Harris window. The used test signal consists of two sinusoidal waveforms, one having a frequency (+0.6 MHz) matching a basis function of the FFT, the other with a frequency (−0.5 MHz) lying between two of the basis functions. White Gaussian noise is added to the test waveform. The resulting signal is then sampled at 2.4 MS/s. The first sine component matching a basis function can be represented by a single point in the spectrum, whereas the second one cannot and becomes a peak with tails extending a significant distant away. This effect is known as *spectral leakage* and occurs when the FFT is calculated on a sample sequence that does not represent a complete period of the underlying time-domain signal. Since in practice this periodicity is not assured, the window smooths the time sequences near both ends (see Figure 6) and thus simulates periodic characteristics to mitigate the effects of leakage. Different windowing functions provide a tradeoff between resolution (i.e. the width of the peaks) and spectral leakage (i.e. the amplitude of the tails) [22]. The selection of an appropriate windowing function is therefore dependent on the intended field of application, because these functions trade off errors in the spectrum's amplitudes, frequencies or the spectrum's overall shape. The sensor manager controls which window is applied[6] in the signal processing chain.

[3] I.e. the output of a block is stored to a queue which then serves as the input for the next block in the processing chain.

[4] `http://osmocom.org`.

[5] The amplitude value of the DC component typically behaves proportional to the chosen FFT length [25].

[6] Our current prototype implements rectangular, Hanning and Blackman-Harris windows and easily allows the integration of arbitrary windowing functions.

Fast Fourier Transform. Each windowed segment is next transformed to the frequency-domain using the FFT (of length defined by fft_size). The resulting signals represent the frequency composition of the original time-domain signal. The computation of the FFT is the most expensive operation in our block chain. We therefore exploit the on-board GPU to calculate the FFTs. By doing so, we additionally improve the level of parallelism within our sensor's computation process. As the communication between CPU and GPU introduces a latency of around 100 microseconds (which is much longer than the shortest transform requires), we perform the execution of transforms in batches. For this, we rely on the *GPU_FFT* library which delivers throughput improvements up to ten times compared to the 700 MHz ARM CPU [7].

Envelope Detection. As we are not interested in reconstructing the acquired signals but only in the spectrum, we remove the phase information of the complex frequency-domain values. To do so, we determine the envelope of the FFT by computing the power values of our frequency-domain complex signals.

Averaging. Finally, averaging intends to reduce the influence of random noise, filtering out weak features of our signals. A configurable number of segments, denoted by avg_factor, are then averaged to form a single fft_size-point spectrum. The drawback of increasing avg_factor is that short-lived signals could be missed when averaging over longer periods. It is worth mentioning here that instead of segmenting our original N-point[8] time-domain signal and averaging the FFTs over multiple segments, we could also make use of a longer (i.e. N-point) FFT. Longer FFTs provide better frequency resolution, but the greater number of samples in the spectrum dilutes the information by the same factor [22], i.e. the contained level of noise is not reduced such as when we average over multiple FFTs.

5.3 Compression and Data Transfer

Compression is key to the design of our sensor as we want to minimize the amount of data that is transfered over the network in order to scale the system to deployments where sensors are placed at people's home. Standard spectrum analyzers do not have this issue since the output of the analysis is displayed on the device itself. The data generated for spectrum analysis can be huge: the raw I/Q data that the dongle produces when its sampling rate is set to the maximum value is around 20 Mb/s. Clearly, transferring all this data over the network is not an option. The data rate after all the spectrum analysis steps is already smaller but still can be quite significant when the configured frequency and time resolution is high. For example, when sampling a 2.4 MHz bandwidth using an FFT size of 4096 and without averaging, the data rate is still in the order of 2 Mb/s.

The spectrum monitor running on the sensor itself controls the level of compression to adapt dynamically to the actual bandwidth and resource availability. We first apply a lossy compression step as follows. The outcome of the FFT is first converted to a decibel scale. We do so by first scaling the FFT's output and then computing the dB values according to $10 \times log_{10}(\mathrm{Re}^2 + \mathrm{Im}^2)$, where Re and Im correspond to the real and imaginary part of the complex values. The dB values are then rounded to a fixed precision of one decimal point. In addition, we smooth the recorded data by enforcing a lower bound. This bound is chosen to cut off values lying outside the sensitivity range of the RTL-SDR dongles and reduces the added noise in the compression step.

In addition to lossy compression, we then apply a loss-less compression step. For this type of compression, we currently make use of the *zlib*[9] data compression library. The library implements the *DEFLATE* compression algorithm, a variation of *LZ77* (Lempel-Ziv 1977). The library allows the specification of a compression level that trades off between compression ratio and speed. In addition, zlib provides facilities to control memory and processor usage. We compress each FFT individually. The compression ratio is dependent on the chosen configuration of the sensor (e.g. the chosen FFT size and averaging factor). We will evaluate the compression factor for different sensing configurations in Section 6.

The compressed data is then handed over to the transmission block for data transfer over the network. The transmission block adds minimal meta-data to the spectrum data for the receiving collector such as for example timestamps and location information. The packed data is then ready for transmission to the collector node. The transmission block is further responsible to conform with any bandwidth restrictions requested by the monitoring logic. The *bandwidth restriction* is implemented using the following concept: First we select a time window over which we will measure our sending rate. Based on our target bandwidth this gives us a maximal number of bytes we are then allowed to transmit within this time window. After each chunk of data we sent, we monitor whether or not we have already exceeded this limit. If so, we wait until the end of the time window before the next chunk of data is transmitted. By doing so, we enforce the bandwidth limitation requested by the monitoring logic.

Another benefit of the aforementioned signal processing queuing system appears in case the recorded data exceeds the enforced network restrictions. The queuing system can cope with temporary network congestion by delaying transmission of recorded data while the rest of the decoupled signal processing keeps operating unaffectedly. Long-term oversaturation inevitably fills up the entire queuing chain and leads to a slowdown in the overall monitoring process. The slower acquisition degrades the achieved time resolution such that no recorded samples are lost.

5.4 Frequency Hopping Strategies

The hardware limitations of our RF interface limit the signal bandwidth that we can sample to 2.4 MHz. To monitor wider ranges it is therefore necessary to hop over different bands. A common strategy for this is to sweep sequentially over the band of interest. In fact, even high-end hardware-based spectrum analyzers are generally not able to monitor the entire spectrum simultaneously and rely on sweeping for monitoring large frequency bands [21]. The sequential sweeping approach may be well suited when the sweeping time is low but with a USB dongle as RF interface, a full sequential sweep may take more than 30 seconds to complete. When the RF interface is tuned to a particular band, it will miss RF emissions on other bands. To mitigate this problem, we propose in this work two more advanced hopping techniques which aim at increasing the quality of the monitored spectrum.

Random Hopping. This strategy is based on a random inspection of frequency bands. This may be implemented by dividing the entire spectrum in bands of width as supported by the dongle and selecting a band for each scan at random. A disadvantage of this implementation is that band inspection is always performed with identically tuned center frequencies. The center frequency as well as low and high frequencies which are subject to more distortion due to DC removal and other type of filtering will therefore always experience the maximum error. An alternative approach which mitigates this problem is to hop by tuning to random center

[7] `http://www.raspberrypi.org/accelerating-fourier-transforms-using-the-gpu`

[8] Where N is equal to $avg\_factor \times fft\_size$.

[9] `http://www.zlib.net`

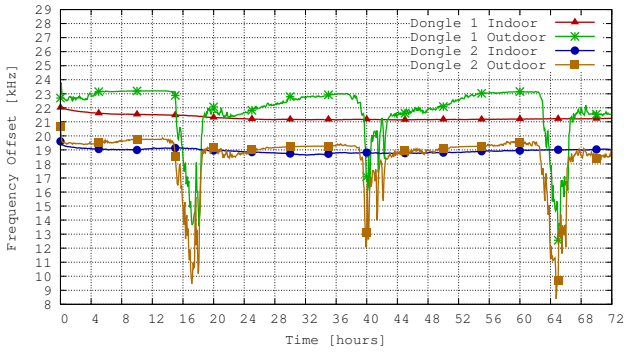

Figure 8: Frequency error over time of two sensors placed next to each other in indoor and outdoor experiments during 72 consecutive hours.

frequencies. For this reason, we rely on the later scheme in our implementation.

Similarity-Based Hopping. Another strategy we propose is to hop randomly but to visit bands of particular interest more frequently than others. In reality, the spectrum is generally sparse [4] and not all frequency bands will be used in the same way. The similarity-based strategy therefore builds upon history, with the idea to inspect more often bands whose spectrum have higher variability. To determine the variability of two different spectra, we rely on the correlation of the signals in the frequency domain, i.e. after the FFT. We therefore define a similarity metric which reflects the similarity of two spectra. Let $X[f]$ and $Y[f]$ be two power vectors measured in dB at two different times for the same frequencies f. Further, let $R_{X,Y}[f]$ denote the cross-correlation of both vectors. We then compute the correlation shift Δ_f and peak r as for maximal correlation shift M by

$$\Delta_f = \underset{m\in\{-M,\ldots,M\}}{\arg\max} R_{X,Y}[m] \tag{1}$$

$$r = \max_{m\in\{-M,\ldots,M\}} R_{X,Y}[m] \tag{2}$$

The similarity of the spectrum s is then defined as

$$s = w \cdot r + (1-w)\Delta_f, \quad \text{where } 0 \le w \le 1, \tag{3}$$

where w is a factor that allows to put more weight on the power magnitude or frequency shift of the correlation. Note that in order to avoid having low similarity values for empty bands, we have to get rid of the thermal noise of the receiver since this noise is uncorrelated over time and will hence exhibit low values for s. This can be achieved by substituting all power levels in $X[f]$ and $Y[f]$ that are below a threshold value to a constant noise floor value or by applying a low-pass filter to $X[f]$ and $Y[f]$ before computing the correlation. In this work, we do the latter and apply a low-pass filter to get rid of the thermal noise in the spectra that we cross-correlate.

The goal of the similarity-based hoping strategy is then to specify the probability of inspection of a band according to the similarity s of the last two spectra. If the similarity is high, the probability may be reduced. When the similarity is low, the probability of inspection should be set to higher values to increase the temporal resolution in this band.

5.5 Frequency and Time Correction

Frequency Correction. The crystal oscillator used for the clock reference in the USB dongles may have inaccuracies in the order of 50 ppm [9]. This inaccuracy is then reflected in significant frequency errors. In addition, the oscillator is not stable with regard to temperature variations. These problems are highlighted in Figure 8. The figure shows the frequency error tracked over time for two different dongles which were placed next to each other in one indoor and one outdoor experiment for 72 consecutive hours. As can be seen, the error can be as high as 23 kHz. In addition, the error is dongle-dependent and may vary significantly over time. In the outdoor experiment, where larger temperature variations are happening, the frequency offset varies by more than 14 kHz. Using more expensive clock sources would alleviate this problem but when using low-cost hardware, the problem is inevitable. In our system architecture, we thus include a frequency corrector module which estimates the error of the dongle and compensates for it in the spectrum monitor when setting a center frequency for tuning. Since the frequency error has the potential of drifting over time (e.g. when the temperature changes), the correction occurs continuously.

The frequency correction module relies on two different techniques to correct the offset at runtime without human intervention. The first technique we rely on is based on GSM signals of nearby cellular base stations. Since our dongle can be tuned to GSM frequencies, we can make use of it to synchronize to GSM signals imitating how cell phones operate to correct their clock drifts. Cellular networks provide an excellent availability and coverage (outdoor and indoor) which is beneficial considering the intended deployments. The GSM correction process consists of two steps: First, we scan and locate nearby base stations and then use them in a second step to calculate our local oscillator offset. Currently, we rely on the open source project Kalibrate-RTL[10] for this purpose. In GSM, the local oscillator of cell phones is adjusted using the frequency correction bursts broadcasted by GSM base stations on their frequency correction channel (FCCH). The initial frequency error must however not be too large since otherwise the FCCH peak might fall outside the 200 kHz GSM channel bandwidth. Therefore, we first perform a rough initial error correction for all the dongles.

The error correction with GSM base stations is very accurate because GSM base stations have themselves internal clocks that may not have errors larger than 0.05 ppm. However, this process is time consuming. The initial GSM scanning process may take up to 6 minutes to complete with the used dongle and follow-up clock offset corrections require roughly 10 seconds. During this time, the usage of the dongle is blocked and we cannot use the dongle to scan the spectrum. In order to avoid repeating this correction process frequently, we propose to use a temperature sensor to track variations in temperature. Temperature sensors are extremely cheap and thus fulfill the low-cost requirement of our design goals introduced in Section 3.1. As we will show in our evaluation, the temperature is a good indicator of the clock drift and can as well be used to estimate the frequency offset. The frequency corrector module predicts the offset by using a linear model of the temperature. With this approach, the error correction with GSM base stations is needed only once for the initial scanning, and changes can be tracked by the temperature sensor.

Time Correction. The time corrector module aims at synchronizing system time across all our sensing nodes. This is particularly important when measurements from different sensors get aligned to compare the spectrum at different locations but identical times. The system time used to timestamp the recorded samples is derived from the clock of the SBC. We synchronize time of the sensors over the Internet with the Network Time Protocol (NTP). Each sensor therefore runs an NTP client on the SBC. Note that other

[10] `https://github.com/steve-m/kalibrate-rtl`

time synchronization protocols such as IEEE 1588 Precision Time Protocol (PTP) may in principle provide better synchronization accuracy than NTP. However, PTP requires native networks with QoS support to prioritize the PTP packets which is not a feasible option in a crowdsourcing scenario where we operate the sensors over the public Internet. Further, since we make timestamps on the SBC, the precision of NTP is already beyond our measured confidence of the time-of-arrival of the received RF signals, given the jitter of data transfer of the samples over USB and the jitter of the operating system.

5.6 Sensor Management

The *sensor manager* is the main interface within the sensor. It is responsible to interface externally with the controller node, and internally with all other relevant components within the sensor. On its outside interface, the sensor manager listens for incoming *monitoring requests* from the controller. The monitoring request contains at least a configurable frequency span to monitor. However, the controller has the full potential of configuring the sensor's monitoring behavior. It can do so by specifying a set of optional parameters, such as for instance the hopping strategy to apply or the time for which the sensor should monitor the corresponding spectrum. The parameters not specified by the controller are determined by the sensor manager itself. Based on the received monitoring request, the sensor manager is then responsible to pass appropriate configuration commands to the other components within the sensor. Once configured, the sensor manager requests the sensor's components to start, and then keeps track of their state. If necessary, it issues on-the-fly reconfiguration of the components. An example of reconfiguration is when the frequency corrector recomputes the frequency error, which requires reconfiguring the spectrum monitor to adjust its tuning center frequency.

6. SENSOR EVALUATION

This section summarizes the experiments conducted for the evaluation of our system. Our aim is to determine the sensing limitations of our low-cost sensor architecture as well as to quantify the benefits of our suggested improvements for wideband scanning and frequency error corrections.

6.1 Primitives' Response Times

We first measure the response times of primitives interfacing with the RTL-SDR dongle. The goal of this evaluation is to learn how costly individual operations are with respect to the response times. The measured response times are summarized in Table 1. For each method, we show the mean response time $\bar{x}$ with the corresponding standard deviation s in milliseconds for $n = 726$ observations. This number of observations corresponds to the number of hops for a full sweep from 24 to 1766 MHz with a hopping bandwidth of 2.4 MHz.

set_gain. From Table 1, we first observe that setting the gain of the RF receiver requires on average 52 ms. Typically the receiver has low sensitivity and it is then desired to set the gain just once to a high value. Assuming this is the case for the monitoring strategy, the response time of setting the device's gain can be neglected.

set_freq_correction. The time to update the crystal oscillator and therefore correct the dongle's frequency error is below the measured precision of 1 ms. The measured result corresponds to the time after which the called function returns. However, it does not consider the time needed to compute the correction factor, which is calculated by the frequency correction thread as explained in Section 5.5.

set_samp_rate. Changing the device's sampling rate on average requires 44 ms. Our hopping approach allows the sampling rate to be changed for individual hops. However, considering the measured result it is advisable not to change the sampling rate on a per-hop basis.

Response Times

Primitive	$\bar{x}$ [ms]	s [ms]	Primitive	$\bar{x}$ [ms]	s [ms]
set_gain	52	2	read(4096)	2	< 1
set_freq_correction	< 1	< 1	read(8192)	3	1
set_sample_rate	44	2	read(16384)	7	< 1
retune	55	1	read(32768)	11	5
read(256)	< 1	< 1	read(65536)	27	< 1
read(512)	< 1	< 1	read(131072)	55	< 1
read(1024)	< 1	< 1	read(262144)	109	< 1
read(2048)	1	< 1	read(524288)	219	1

Table 1: Analysis of the mean response times $\bar{x}$ and corresponding standard deviation s in milliseconds of primitives interfacing with the RTL-SDR receiver.

retune. Retuning the dongle to a certain center frequency requires 55 ms on expectation. Doing the calculations for a full sweep provides a lower bound on the sweeping time: 726×55 ms = 39.93 s.

read. This primitive reads a sequence of interleaved I/Q samples. The response time of the read procedure is strongly dependent on the actual number of samples requested from the RTL-SDR dongle. The results therefore include measurements for different sample lengths (shown inside parentheses). As expected, the response time for acquiring samples increases with the number of samples read. Doubling the requested number of samples cause approximately twice the response time. Sample lengths below 2048 have response time below 1 ms, and do therefore not significantly increase the monitoring time. In order to evaluate the impact of averaging, we may for example choose an FFT size of $fft\_size = 512$ together with an averaging number of $avg\_factor = 64$. This leads to a sample length of $512 \times 64 = 32'768$ being read at each individual hop, for a total time of 11 ms. A total of 726 hops therefore results in a lower bound for the sweeping time of $726 \times (55\text{ ms} + 11\text{ ms}) = 47.916$ s[11].

6.2 Sweeping Times

This experiment measures the time required to sequentially sweep over the full wideband of 1742 MHz[12]. The sweeping time is defined as the time required for the data of a single sweep to arrive at the backend collecting node. Differently from the previous section, the sweeping time includes the full chain of signal processing (including steps such as Fourier transform and compression) as well as the time spent to transmit the recorded data over the network. In the experiments, sensor and collector are placed on distant locations and connected over the public Internet.

We study the time required to sweep over certain ranges of frequencies and summarize the results in Figure 9. As a result of the dependency of sample length on the response time reported in Section 6.1, we here measure the sweeping time for different sample lengths. For the analysis, sample length refers to the number of samples $N = fft\_size \times avg\_factor$ that are read (per hop) from the RF receiver. The figure shows that the sweeping time increases linearly with the scanned frequency span[13] and that the slope of the linear curve increases when enlarging the number of samples read at each hopping step. We further observe the following phenomena:

[11] $\#HopsPerSweep \times (RetuningTime + ReadingTime)$

[12] $1766MHz - 24MHz = 1742MHz$

[13] Note that no network congestion was observed during the tests.

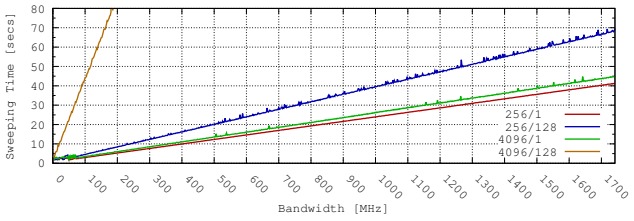

Figure 9: Time required to sweep over a certain range of frequencies. The different curves show how the time increases with a growing per-hop sample length as defined by fft_size and avg_factor.

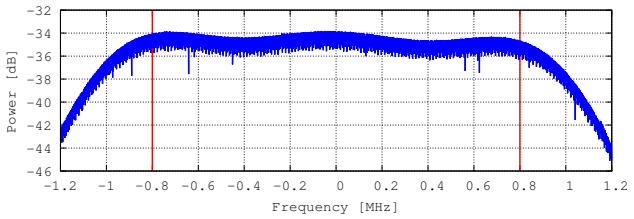

Figure 10: Frequency response of RTL-SDR receiver: The frequency response of the RTL-SDR is measured by sweeping a constant-amplitude sinusoidal waveform through a band of 2.4 MHz when the RTL-SDR receiver is tuned to 500 MHz.

- For small number of samples read at individual hopping steps, the sampling process is the main factor influencing the sweeping time. Indeed, in the previous section, we estimated the sweeping time for the full wide band and a sample length of 256 to be 39.93 s. The actual time measured in the experiment is 41.29 s.
- Increasing the number of samples leads to a higher volume of data being compressed and transmitted over the network. The influence of FFT, compression and network transmission increases, and the difference between theoretical estimations and actual measured results gets larger. This can be seen on the following example: before, we estimated the sweeping time (24 – 1766 MHz) for a sample length of 32′768 to be 47.916 s. The actual time measured is 69.627 s.

6.3 Frequency Response

We determine the RTL-SDR's frequency response, using its default finite impulse response (FIR) low-pass filter which uses 32 taps (note that more taps can not be added due to hardware limitations). For this evaluation, we use an USRP as source for generating a constant sinusoidal waveform of constant amplitude, and sweep it over the full 2.4 MHz band the RTL-SDR can receive. In the shown evaluation, the RTL-SDR stayed tuned to a center frequency of 500 MHz, and the test signal was swept with a frequency resolution of 1 kHz.

The result of the test is shown in Figure 10. It shows that the frequency response is approximately flat between −0.8 and 0.8 MHz around the center frequency, and it then reduces its sensitivity by up to 10 dB. This indicates that it may be advisable to remove the tails of the band below -0.8 and above 0.8 MHz. The drawback of this operation is that a higher number of hops are needed to cover the entire band, increasing the lower bound of the sweeping time by 50%. An alternative is to implement a filter that is matched to this frequency response, which works as long as the data is received above the sensitivity of the RTL-SDR receiver.

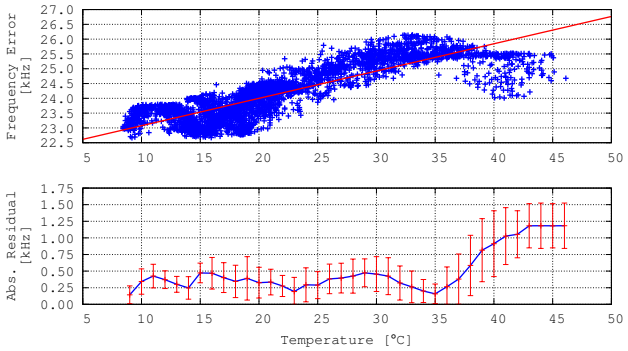

Figure 11: On the top: correlation between the frequency error and the temperature using an on-board cheap temperature sensor. On the bottom: absolute residual of the frequency error using the linear model of the temperature.

6.4 Frequency Correction

In this experiment, we place one sensor in outdoor environment over a full time period of 3 days (72 hours) and we record both the frequency error as measured by tuning to GSM base stations (see Section 5.5) as well as the data of ambient temperature gathered by the sensor. The top of Figure 11 plots the frequency error versus temperature, and models their relationship using linear regression. The Pearson cross-correlation coefficient reported at lag zero between frequency error and temperature is 0.947. This indicates that frequency errors and temperatures are strongly correlated, and that a simple temperature sensor connected to the SBC suffices to estimate the error. Using a linear model for frequency error as a function of temperature, we can correct the error of the RTL-SDR without the time-consuming operation of stopping to sense the wideband spectrum in order to tune to the GSM base station for the frequency correction. With this technique, we incur the absolute residuals as shown in the bottom of Figure 11. We observe that the error is approximately 30 – 40 times smaller than the original offset for temperatures below 35 degrees. The model is less effective for temperatures above 35 degrees, but it still guarantees a reduction of the error by at least a factor of 10.

6.5 CPU Utilization

Our software-defined approach as well the simplistic hardware used in this work introduce severe challenges in making the task of spectrum monitoring possible. Since most of the involved signal processing steps in software are highly demanding with respect to computational requirements, we expect the sole single-core processor on the sensor to be confronted with a permanently high load[14]. In the following, we evaluate the sensor's CPU utilization for different monitoring configurations. The goal is to identify possible bottlenecks in the processing steps according to the chosen configuration. We consider four realistic configurations:

- *256/1:* This configuration refers to FFT sizes of 256 without any signal averaging.
- *256/128:* FFT size is set to 256 and averaging factor to 128. This means that we average 128 FFTs on the sensor before transferring the result to the collector.
- *4096/1:* The FFT size is 4096 and no signal averaging is performed.

[14] As mentioned before, one way we already reduce the load on the CPU is by moving some of the tasks to the GPU. We currently do this for the FFT.

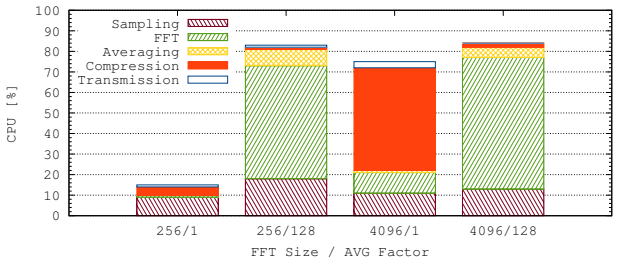

Figure 12: CPU utilization of involved signal processing steps in four different configurations. In all configurations, the sensor is able to deliver the resulting data without any loss due to overload.

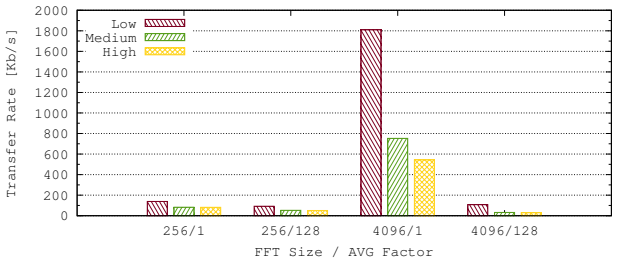

Figure 13: Data upload transfer rates for different configurations and compression levels.

- *4096/128:* The FFT size is 4096 and the averaging factor is 128.

In each configuration, the sensor tunes to a different band after it has acquired $N = avg\_factor \times fft\_size$ samples. The compression level of the DEFLATE algorithm is set to 6, corresponding to the default value.

The evaluation results are shown in Figure 12. As sensing results in quite constant load, we omit to show the standard deviation or confidence intervals. We differentiate the CPU utilization of the different threads of our software-defined architecture to perform *sampling*, *FFT*, *averaging*, *compression*, and data *transmission*. As one can see, the total CPU load for *256/1* is only 16%. The signal processing load is almost negligible while the load is dominated by the signal sampling task. The picture greatly changes when increasing the averaging factor to 128. For *256/128*, the overall load rises to 83%. This time, the main bottleneck are the FFT calculations with 58% CPU load compared to only 7% for performing the actual averaging. It may seem counterintuitive that increasing the averaging factor also increases the CPU load for the FFT computations. The reason is that the CPU receives much more data per time unit since when averaging is performed, the dongle switches band less frequently in order to acquire enough samples for averaging 128 FFTs. The time to switch bands is quite significant as we have shown earlier in this section. The CPU therefore is mostly waiting for data when no averaging is performed while the *256/128* configuration is able to feed the CPU with enough samples to remain busy almost all the time. It is worth noticing that despite the fact that the Fourier transforms are computed on the GPU, the time waiting for and transferring calculated FFTs dictates the load of the CPU in this particular configuration.

The *4096/1* configuration has a much higher overall load of 77% compared to *256/1*. The most demanding task this time is the compression task with a load of 51%. In contrast, the load for compression with the *4096/128* configuration is negligible. It shows that averaging prior compressing significantly improves the performance of the compression algorithm. On the other hand, the increased number of FFTs that must be computed per band increases significantly the overhead of the FFT computation task such that the overall load of the *4096/128* configuration is highest. Nevertheless, the performance of our system is such that the overall load is still manageable by the platform and the sensor can deliver results without any losses at all four configurations.

6.6 Evaluation of Transfer Rate

A key requirement for crowdsourcing spectrum monitoring is to have limited upload data rates of the measurements in order not to overload the network connections of the participants. We now evaluate the data transfer rates of our sensor in different configurations.

We consider the same configurations with regard to FFT sizes and averaging factors as before. In addition, we explore the data rates using different compression levels:

- *Low:* In this configuration, the compression of the DEFLATE algorithm is disabled. Compression is only achieved by the quantization of the power levels and removal of the values below the noise floor of the device.
- *Medium:* In this setup, the DEFLATE algorithm additionally compresses each FFT with the default compression level 6.
- *High:* The compression level of the DEFLATE algorithm is increased to the maximum level 9.

The results are shown in Figure 13. The bars in the graph represent the average data upload transfer rates in Kb/s. As one can see, medium compression roughly halves the transfer data rates compared to low, while high compression only slightly improves over medium level compression. Except for the configuration *4096/1*, all three other configurations result in transfer rates below 100 Kb/s for medium level compression (which we view as an acceptable threshold for wide-scale deployments). With *4096/1* however, the data transfer rate remains above 500 Kb/s, even when the highest compression level is chosen. To overcome this problem, the averaging factor should be increased. As we can see, configurations with averaging factors of 128 produce smaller data rates as their counterparts with averaging factor 1. Interestingly, *4096/128* with compression level medium has a lower transfer rate (32 Kb/s) compared to *256/128* (52 Kb/s). These results demonstrate that in our system it is possible to increase the spectral resolution of measurements while not increasing the transfer rate requirements when averaging is applied. This insight is a nice property that can be exploited in deployments where the operator is willing to trade-off sweeping *time* versus *frequency* resolution of the spectrum. By using larger FFT sizes in combination with larger averaging factors, the frequency resolution and SNR are increased at the same time without imposing additional data transfer overhead over configurations with smaller FFT sizes and equal amount of averaging.

6.7 Sensitivity Compared to USRP

Next, we compare the spectrum sensed with our sensor to the spectrum that is sensed with a USRP N210. The USRP is expected to provide higher sensitivity since the hardware is roughly 50 times more expensive and includes better hardware components such as a higher resolution ADC, faster clock and better RF filters. For this experiment, we record the spectrum of a real GSM900 downlink channels. The GSM channel bandwidth is 200 kHz. For both setups (our sensor and USRP), we apply the following configuration: the time-domain signals are sampled at a rate $F_s = 2.4$ MS/s and a FFT size of $N = 512$ is used. The segments are windowed using a 512-point Hamming window. Both platforms use the same type of omnidirectional antenna and are placed close to each other to

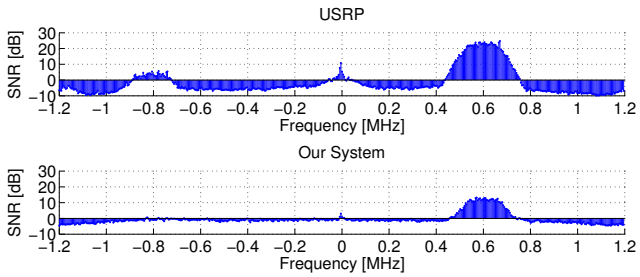

Figure 14: Spectrum sensed with USRP compared to spectrum sensed with our sensor using the RTL-SDR dongle. Both setups are able to detect the downlink of a base station at 0.6 MHz while our system misses the signals from the GSM base station at −0.8 MHz because the SNR of our system is approximately 10 dB lower than the USRP.

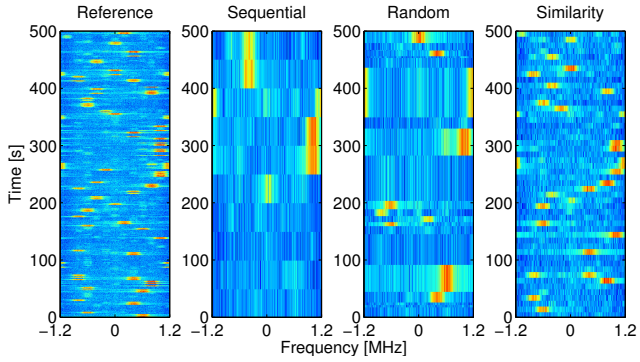

Figure 15: Comparison of sequential, random and similarity-based hopping strategies for bursty signals.

experience the same signal reception quality.

The result of this experiment is shown in Figure 14. The top figure shows the spectrum of the USRP, the bottom plot the spectrum with our system. Both spectra are plotted on the vertical axis relative their recorded hardware's average noise floor. As we can see, the USRP is more sensitive than our system, but we are still able to detect the GSM downlink at 0.6 MHz. The USRP's sensitivity is around 10 dB larger than the sensitivity of our sensor. This is the reason why the USRP detects a signal at −0.8 MHz whereas our sensor does not. The signal at −0.8 MHz is also from a GSM base station but this one is located further away from our location of measurement. This downlink signal is beyond the sensitivity of our sensor and therefore missed. Clearly, our sensor is more limited than platforms such as the USRP. Nevertheless, the cost benefits of our proposed setup would allow to deploy more sensors such that additional sensors closer to the base station would be able to capture the signals that are missed by the sensors which are located too far away from the base station to hear those signals.

6.8 Hopping Strategies

Finally, we evaluate the effect of different frequency hopping strategies for scanning larger bands than the RTL-SDR supports. For this evaluation, we consider two types of signals: *bursty* data signals generated by the GSM base stations to mobile phones, and *constant* signals from a downlink channel of a GSM base station. We compare the proposed random hopping and similarity-based hopping strategies to a hopping strategy that performs sequential

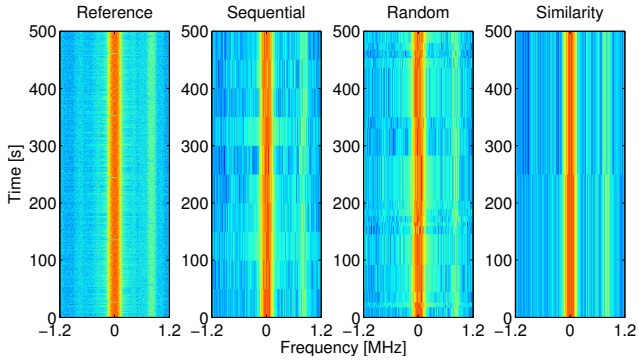

Figure 16: Comparison of sequential, random and similarity-based hopping strategies for constant signals.

hopping. To evaluate the sensed spectrum quality of these strategies in a consistent and reproducible manner, we first record continuously during 500 seconds the signals on a band of 2.4 MHz with a single sensor platform. Then, we emulate the operation of the different hopping strategies by performing the hopping offline after the signals have been acquired by the sensor platform. This allows us to compare only the effect of the hopping strategies without any other dongle or channel-related temporal noise effects. The hopping strategies are emulated by acquiring only parts of the signal depending on the strategy. The sequential strategy acquires samples every 50 seconds, corresponding roughly to the time it takes to sequentially sweep over the entire band of the RTL-SDR dongles. The random strategy acquires the samples at random times with a probability that is related to the number of acquisitions of the sequential sweeping strategy. The similarity-based metric acquires the samples five times more often (i.e., every 10 seconds) on the bursty channel and five time less frequently (i.e., every 250 seconds) on the constant channel.

The results of these strategies are visualized in Figures 15 (bursty channel) and Figure 16 (constant channel). In addition, we include the reference spectrum as acquired through time-continuous sampling of the signal. We see for the bursty channel that the similarity-based strategy is best at capturing the behavior of the channel. The similarity as expressed by our similarity metric s between this strategy and the reference averaged over the 500 seconds is 60%. Random hopping may be more accurate when the band's samples are acquired more often but then has a smaller time resolution when the band is inspected less often. On average the similarities to the references averaged over the 500 seconds for the the random strategy is 55.5% and 54.6% for the sequential hopping. This results in similar average similarities. Nevertheless, the random strategy has the advantage that the burstiness of the link is detected better than with the sequential sweeping.

The constant GSM downlink channel in Figure 16 is on the other hand quite similar for all strategies. The similarities compared to the reference spectrum averaged over the whole duration of the experiment are 87.7% for the sequential, 87.9% for the random, and 87.9% for the similarity-based hopping strategy. Since the channel is fairly constant, the inspection time and re-occurrence of the band is not that relevant. This result highlights the benefits of the random, and in particular the similarity-based strategy which focuses on bands with higher variability, such as bursty links. The similarity-based metric is able to capture the bursty nature of the links at the cost of marginal quality losses for the bands with constant frequency spectrum.

7. CONCLUSION

We have proposed a distributed spectrum monitoring system whose low-cost sensing nodes record and transmit the wideband spectrum in real-time to a remote data collector. We have demonstrated that our software-defined sensor architecture runs on limited hardware such as RTL-SDR based USB dongles attached to embedded credit card-sized computers such as the Raspberry Pi. We have shown that our platform is capable of performing in real-time the required signal processing steps of modern hardware-based spectrum analyzer such as sampling, segmentation, DC removal, windowing, FFT and averaging as well as data compression for the maximum supported signal sampling rate that the USB dongles supports, i.e. 2.4 MS/s. Our system is able to sense, analyze and compress a band of 2.4 MHz without any losses with a spectral resolution below 10 kHz, uploading compressed data stream of a wideband spectrum of 1742 MHz to a remote collector at a rate of only 52 Kb/s and within less than 70 secs.

We have further proposed methods to circumvent the physical limitations of low-cost hardware for wideband scanning. For example, we have presented a spectral similarity-based frequency hopping strategy that prioritizes the inspection of bands with large variance in order to improve the detection of bursty signals. We have also developed a frequency corrector module that is able to reduce the large frequency error caused by the crystal oscillator offset of low-cost hardware by a factor of more than 10 without affecting the sweeping time.

While our sensor architecture runs on limited hardware with less accurate measurement capabilities than higher-end spectrum analyzers, we believe such a setup to be well suited to commoditize and crowdsource the task of wideband spectrum monitoring in the future. The lower sensitivity and resolution of the COTS hardware may be overcome by deploying more sensors than otherwise possible with dedicated and expensive hardware. The achievable spectrum sensing quality might therefore still be adequate when enough sensors participate in the system. Our work is only the first step for a large-scale coverage at great level of density. Research must yet introduce collaborative strategies in regions where multiple sensors are monitoring simultaneously as well as fusion algorithms that take advantage of current evolutions in the Internet architecture such as cloud computing and software-defined networking.

Acknowledgement

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

8. REFERENCES

[1] Microsoft spectrum observatory. `http://observatory.microsoftspectrum.com/`.

[2] Microsoft spectrum observatory - source code repository. `https://spectrumobservatory.codeplex.com/SourceControl/latest`.

[3] 12-bit, 1-gsps analog-to-digital converter. *Texas Instruments*, October 2009.

[4] 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.*, 50(13):2127–2159, Sept. 2006.

[5] A. Arcia-Moret, M. Zennaro, and E. Pietrosemoli. WhispPi: White Space Monitoring with Raspberry Pi. In *Proceedings of the Global Information Infrastructure and Networking Symposium*, 2013.

[6] 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*, 29(3):101–116, May 2012.

[7] W. Bajwa, J. Haupt, G. M. Raz, S. Wright, and R. Nowak. Toeplitz-structured compressed sensing matrices. In *Statistical Signal Processing, 2007. SSP '07. IEEE/SP 14th Workshop on*, pages 294–298, Aug 2007.

[8] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi. Ghz-wide sensing and decoding using the sparse fourier transform. In *INFOCOM, 2014 Proceedings IEEE*, pages 2256–2264, April 2014.

[9] M. Higginson-Rollins and A. E. Rogers. Development of a low cost spectrometer for small radio telescope (srt), very small radio telescope (vsrt), and ozone spectrometer. 2013.

[10] 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.

[11] B. Le, T. Rondeau, J. Reed, and C. Bostian. Analog-to-digital converters. *Signal Processing Magazine, IEEE*, 22(6):69–77, Nov 2005.

[12] W.-Y. Lee and I. Akyildiz. Optimal spectrum sensing framework for cognitive radio networks. *Wireless Communications, IEEE Transactions on*, 7(10):3845–3857, October 2008.

[13] O. Mehanna and N. Sidiropoulos. Frugal sensing: Wideband power spectrum sensing from few bits. *Signal Processing, IEEE Transactions on*, 61(10):2693–2703, May 2013.

[14] A. W. Min and K. G. Shin. An optimal sensing framework based on spatial rss-profile in cognitive radio networks. In *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, SECON'09, pages 207–215, Piscataway, NJ, USA, 2009. IEEE Press.

[15] S. Mishra, A. Sahai, and R. Brodersen. Cooperative sensing among cognitive radios. In *Communications, 2006. ICC '06. IEEE International Conference on*, volume 4, pages 1658–1663, June 2006.

[16] 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*, pages 173–184, May 2011.

[17] 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*, HotWireless '14, pages 25–30, New York, NY, USA, 2014. ACM.

[18] P. Paysarvi-Hoseini and N. Beaulieu. Optimal wideband spectrum sensing framework for cognitive radio systems. *Signal Processing, IEEE Transactions on*, 59(3):1170–1182, March 2011.

[19] H. Rahul, N. Kushman, D. Katabi, C. Sodini, and F. Edalat. Learning to share: Narrowband-friendly wideband networks. *SIGCOMM Comput. Commun. Rev.*, 38(4):147–158, Aug. 2008.

[20] M. Rashidi, K. Haghighi, A. Panahi, and M. Viberg. A nlls based sub-nyquist rate spectrum sensing for wideband cognitive radio. In *New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 2011 IEEE Symposium on*, pages 545–551, May 2011.

[21] Rohde and Schwarz. Spectrum analyzer fundamentals - theory and operation of modern spectrum analyzers. February 2013.

[22] S. W. Smith. *Digital signal processing: a practical guide for engineers and scientists*. Newnes, 2003.

[23] Z. Tian. Compressed wideband sensing in cooperative cognitive radio networks. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–5, Nov 2008.

[24] J. A. Tropp, M. B. Wakin, M. F. Duarte, D. Baron, and R. G. Baraniuk. Random filters for compressive sampling and reconstruction. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, pages III–III. IEEE, 2006.

[25] H. Tsurumi and Y. Suzuki. Broadband rf stage architecture for software-defined radio in handheld terminal applications. *Communications Magazine, IEEE*, 37(2):90–95, Feb 1999.

[26] D. Xue, E. Ekici, and M. Vuran. Cooperative spectrum sensing in cognitive radio networks using multidimensional correlations. *Wireless Communications, IEEE Transactions on*, 13(4):1832–1843, April 2014.

[27] S. Yoon, L. E. Li, S. C. Liew, R. R. Choudhury, I. Rhee, and K. Tan. Quick sense: Fast and energy-efficient channel sensing for dynamic spectrum access networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2247–2255. IEEE, 2013.

[28] 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.