Benchmarking Resource Usage for Spectrum Sensing on Commodity Mobile Devices

Ayon Chakraborty, Udit Gupta and Samir R. Das
Department of Computer Science, Stony Brook University, Stony Brook NY 11747, U.S.A.
{aychakrabort, ugupta, samir}@cs.stonybrook.edu

ABSTRACT

Effective management of various white space spectra may require spectrum sensing at finer spatial granularity than is feasible with expensive laboratory-grade spectrum sensors. To enable this, we envision a future where commodity mobile devices would be capable of spectrum sensing as needed, possibly via crowd-sourcing. However, since mobile devices are resource limited, understanding their resource usage in this set up is important, specifically in terms of overall latency and energy usage. In this work, we carry out a comprehensive performance benchmarking study using 4 different USB-powered software radios and 2 common smartphone/ embedded computers as mobile spectrum sensing platforms. The study evaluates latency and energy usage using a suite of commonly used sensing algorithms specifically targeting TV white space spectrum. The study shows that latency due to sensing and computation and related energy usage are both modest.

CCS Concepts

•**General and reference** → **Measurement;** •**Hardware** → *Wireless devices;*

1. INTRODUCTION

As the demand for wireless spectrum grows exponentially so is the need for large-scale spectrum sensing. Large-scale sensing serves various spectrum management needs depending on the spectrum. For example, we anticipate that such large-scale sensing would be a key enabler towards building a shared spectrum access infrastructure by finding underutilized spectrum whitespaces. It is widely understood that spectrum databases [6, 2] that solely rely on wireless propagation models for finding such whitespaces are largely error-prone [22] and needs to be 'measurement-augmented' [13]. Many similar needs arise in spectrum monitoring applications where the spectrum owner must monitor spectrum usage to evaluate spatio-temporal usage patterns [22] or to detect unauthorized use [24].

Depending on specific applications, there has been various proposals about how spectrum sensors should be deployed. For example, existing work deploys lab-grade spectrum sensors or high-end software radio platforms where convenient [15], including public vehicles [22]. Many shared spectrum designs for exploiting white spaces also deploy similar sensors on the access points (AP) [18, 11]. Regardless, it is always too expensive or impractical to deploy laboratory-grade devices at any significant spatial granularity covering arbitrary areas. Recently, there have been propositions to scale sensing at finer spatial granularities using crowd-sourcing mechanisms [19]. Initial prototypes have shown an early promise using commodity mobile devices [16, 23, 17] as spectrum sensors.

However there is a concern that such mobile spectrum sensors using commodity hardware may be too resource limited so be able to gather useful data in a crowd-sourced scenario. For example, the sensors may have slower sampling rate and higher retuning latency. This increases the 'scan time' to sense a given amount of spectrum. Further, spectrum sensing typically uses computationally involved algorithms. Limited computational power on the mobile platform makes signal detection slower or the accuracy poorer. Finally, the energy budget on a mobile platform is limited and both sensing and computation could be energy intensive.

Contributions: Our goal in this work is to perform systematic performance evaluations to address these concerns. We use a testbed comprising of 4 different USB-powered, small form-factor, software-radio platforms and 2 different smartphone/ embedded computer platforms. These platforms are all off-the shelf and provide representative capabilities of a mobile spectrum sensor that could be built using today's commodity technology. We evaluate resource usage in terms of latency and energy consumptions for the chosen platforms for 3 commonly used detection algorithms for TV signals. Our general conclusion is that the resource usage is quite modest. In particular, energy consumptions could be less than 20% of energy cost of popular smart phone applications. Overall the latency for sensing (including the necessary computation time) is also modest – within about 100s of ms that could enable a range of sensing applications.

2. MOBILE SPECTRUM SENSING

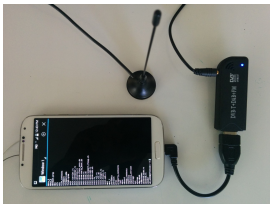
In this section we describe the prototype mobile spectrum sensors we considered in our benchmarking study along with three general categories of algorithms that are highly used for determining spectrum occupancy.

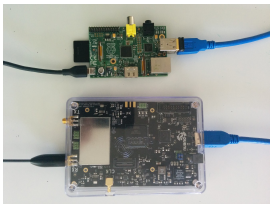
2.1 Mobile Spectrum Sensor Prototypes

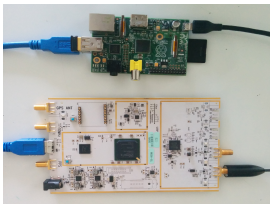
The mobile spectrum sensor consists of two units – 1) the sensing unit that primarily obtains the signal samples, and 2) the compute unit that runs the signal detection algorithms. Since the emphasis of this work is on "commodity" mobile platforms, we focus on small-factor, low power devices easily available in the market. Unfortunately, there is no combination of commodity sensing-plus-

HotWireless'16, October 03-07, 2016, New York City, NY, USA

DOI: http://dx.doi.org/10.1145/2980115.2980129


(a) RTL Dongle with phone


(b) BladeRF with RPi


(c) USRP B200 with RPi

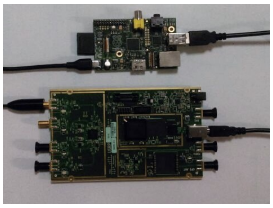

(d) USRP B210 with RPi

Figure 1: Some examples of platform configurations considered in this work. RPi stands for Raspberry-Pi.

	RTL-SDR	BladeRF	USRP B200	USRP B210
Radio Spectrum	24MHz - 1.7 GHz	300MHz - 3.8 GHz	50MHz - 6GHz	50MHz - 6GHz
Sample Size	8-bit	12-bit	12-bit	12-bit
Sample Rate	2.5 Msps	40 Msps	61 Msps	61 Msps
Frequency Correction	$\pm 50ppm$	$\pm 2ppm$	$\pm 2ppm$	$\pm 2ppm$
FPGA	N/A	Altera Cyclone 4E	Spartan 6 XC6SLX75	Spartan 6 XC6SLX150
Interface	USB 2	USB 2/3	USB 2/3	USB 2/3
Form factor	USB flash drive	5" x 3.5"	5.9" x 3.8"	5.9" x 3.8"
Cost	$\approx$ 20$	$\approx$ 420$	$\approx$ 670$	$\approx$ 1100$

Table 1: Capabilities and cost of the sensing units considered

compute unit that has a form factor of a mobile phone, but approximations are possible.

Sensing unit: There are several sensing units currently available that are USB-powered though they indeed vary significantly cost, capability and power consumption. We choose 4 of these devices. They are basically USB-powered SDRs: 1) RTL-SDR [5], 2) BladeRF [1], 3) USRP B200 [7], 4) USRP B210 [8]. The latter 3 are FPGA-based devices. Each sample (I/Q) from the RTL-SDR is a byte while it is 12-bits for the other three sensors. More bits means less quantization noise. RTL-SDR has a maximum sample rate of 2.5Msps, BladeRF has 40Msps and the USRPs got 61Msps each. Sample rate is simply how many samples can be collected per unit time. This also determines the signal bandwidth that can be sensed at a time. See Table 1 for a general comparison of features among the sensors.

Compute unit: The choice of the compute unit is relatively straightforward. One of the chosen platform is a late-model mobile phone with USB On-the-Go support (Samsung Galaxy S4 with a quad-core 1.6 GHz CPU, 400 MHz GPU and 2 GB memory) so that it can act as a USB host. The other one is a popular low cost embedded platform that can easily be battery driven (Raspberry Pi with 700 MHz CPU and 256 MB memory). Figure 1 shows pictures of our chosen platforms.

2.2 Sensing Algorithms

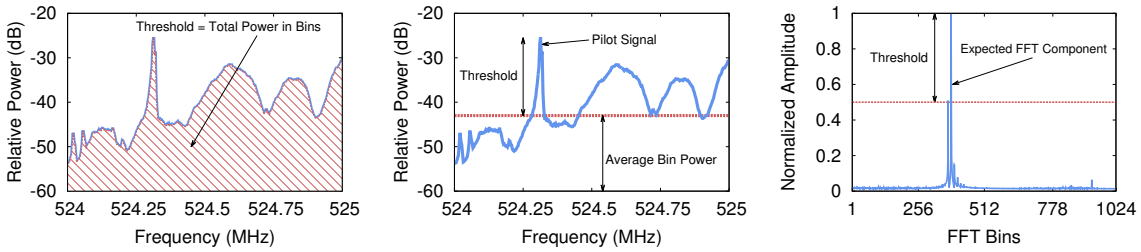
Spectrum sensing is a signal detection task, performed by a suitably designed algorithm. The algorithm could be generic or specific to the type of signal. Regardless, *these algorithms may incur significant computational overhead on the mobile platform, specifically when a high detection performance is desired.* We assume that the detection must be performed locally on the mobile platform itself, as transferring the signal samples to a remote server will consume a very significant cost, both in terms of latency and energy. Performance here is characterized by probability of detection (P_D) and probability of false alarm (P_{FA}). Given limited resources, understanding the performance vis-a-vis resource 'cost' in terms of latency and battery energy is important. We will evaluate these aspects in our chosen testbed with respect to three representative algorithms of differing characteristics for primary signal detection in the TV white space (TVWS). We use TVWS just as a case study given a significant literature in this space, and given the spectrum range limitations of the chosen platforms. In future, we would like to study other bands and signals as well. In the following subsections we briefly describe these algorithms. Each algorithm operates directly on a set of I/Q (in-phase and quadrature phase) samples collected by the sensing unit. Two important parameters are the sampling rate ($samp\_rate$) of the ADC and the sampling time (T). They yields the number of samples $N = T \times samp\_rate$. The sampling rate also determines the bandwidth of the signal that can be sensed.

Energy Based Detection: This is the simplest and the most intuitive of the algorithms and can be used for any signal even when the signal properties are unknown. The idea here is to determine the total power contained within the channel (or a part thereof) of interest. We determine this by computing the sum of the squared magnitude of each bin of an M-bin FFT on the I/Q samples. This estimates the total channel power and forms the test statistic. If this quantity is more than a predetermined decision threshold (e.g., noise floor specific to the device), we determine that the channel is occupied (signal present).

While the basic idea is straightforward, there are several issues. It is hard to select the decision threshold perfectly due to the uncertainties in noise floor estimation. Some parts of noise (e.g., quantization noise and thermal noise) can indeed be estimated by known methods [10]. More bits in the ADC and also more bins in FFT reduce the noise and improve detection performance. However, other sources of noise such as due to channel fading, mismatch between the I and Q signal pathways, non-ideal behavior of the oscillator, etc are hard to estimate [21]. Some form of measurement-based calibration needs to be adopted in practical settings. This can be done in a device specific fashion by measuring the channel power in a channel known to be empty.

It is also well-understood that more samples improve the detection performance.[1] However, there is a limit to this in practical settings at very low SNRs (so-called 'SNR wall' [21]), where even infinite number of samples will not provide any acceptable detec-

[1]In our work every M-th sample is averaged and fed to the M-bin FFT. This is equivalent to repeated FFTs and averaging the powers in the bins later.



(a) Energy based detector (b) Feature based detector (c) Autocorrelation based detector

Figure 2: Visual clues for the three sensing algorithms, interpretation of the threshold.

tion performance. Given a number of samples available, performance can be improved by increasing the number of bins in the FFT. But given the $O(M \log M)$ complexity, this increases computation time.

Feature Based Detection: This techniques takes advantage of some known features within the signal. For example, in the ATSC TV signal there is a pilot at 309 KHz offset from the lower edge of the channel and the pilot signal's power is less than the channel's aggregate power by approximately 11.3 dB [9]. This also means that the bin corresponding to the pilot tone has a much higher signal strength compared to its neighboring bins in the FFT, if we apply the previous energy-based detection. We can indeed apply a very similar method but now just focused on detecting the presence of pilot instead of evaluating the entire channel power. Since the pilot occupies a significantly smaller bandwidth (basically a tone) this helps to reduce the noise compared to the case when the entire channel is considered. Definitely, increasing the number of bins in the FFT and prolonging the sensing time (more samples) provide advantages in this case too. In particular, larger number of bins can discriminate the pilot better even when the signal is very weak. A very similar technique is used in [13, 22] to track the pilot in presence of a sufficiently high noise floor.

With the same number of bins, this techniques could be only slightly slower than the energy-based method as a few modest additional computation steps are needed.

Autocorrelation Based detection: The above two techniques are fundamentally based on computing the power within the whole or part of a channel. Now, instead of calculating any power directly, we exploit the fact that the pilot tone is a sinusoidal signal and autocorrelating the signal samples obtained from a bandwidth containing the pilot must contain a corresponding sinusoidal signal in the autocorrelation function (ACF). The autocorrelation-based detector [14] is based on this intuition.

Assume we autocorrelate N samples and look at the ACF at each lag from 0 to N. If the signal is periodic (with a period p) and has multiple cycles within the interval $(0, N)$, the ACF will also be periodic and show maximum correlation (peak amplitude of the ACF) at lags that are multiples of p. We tune the sensing unit at a center frequency f_c and obtain I/Q samples at a certain sample rate. Suppose the pilot tone is expected at f_{pilot}. The ACF function now contains a sinusoid with a frequency $|f_c - f_{pilot}|$. It is advisable to keep the sample rate small but enough to accommodate the interval (f_c, f_{pilot}) within the Nyquist bandwidth. The presence of the sinusoid can be detected again by taking an FFT of the ACF function itself and detecting the presence of a peak at the desired frequency ($|f_c - f_{pilot}|$).

The autocorrelation-based technique is generally understood to be able to detect presence of very weak signals. This is because it is not based on direct energy comparisons that could be impacted by noise. Noise is generally uncorrelated and and thus even when the signal is very weak, the above correlations could be discovered at specific frequencies. However, while we expect a better detection performance with this technique, it is computationally intensive. Computing autocorrelation itself requires an FFT and inverse FFT. Another additional FFT computation is needed to detect the peak.

2.3 Signal Detection Performance

The three algorithms have different detection performances in terms of P_D or P_{FA}. This performance also depends on 1) the received signal power or SNR, and 2) the number of samples (proportional to the sensing time at a given sample rate) fed to the algorithm [20, 21]. Thus, depending on the context and expected performance a given algorithm can be preferred over the other to optimize resource costs. For brevity, in this short paper we do not report the signal detection performance; we limit our analysis only to latency and power/energy measurements. We use two cases with 10K and 100K samples (10ms and 100ms sensing time for the studied sample rate of 1 Msps) as representatives. Here we want so assure the reader that these many samples are enough for reasonable detection performance even for fairly weak TV signals. For example, our analysis (not reported here) shows that even in the smaller, 10K samples case and for a very weak TV signal (-90 dBm), P_D could be higher than 85% with P_{FA} less than 15% for all algorithms studied here independent of the platform. 100K samples further increases the P_D to more than 92% with P_{FA} slightly below 10% for the same TV signal for all the three algorithms.

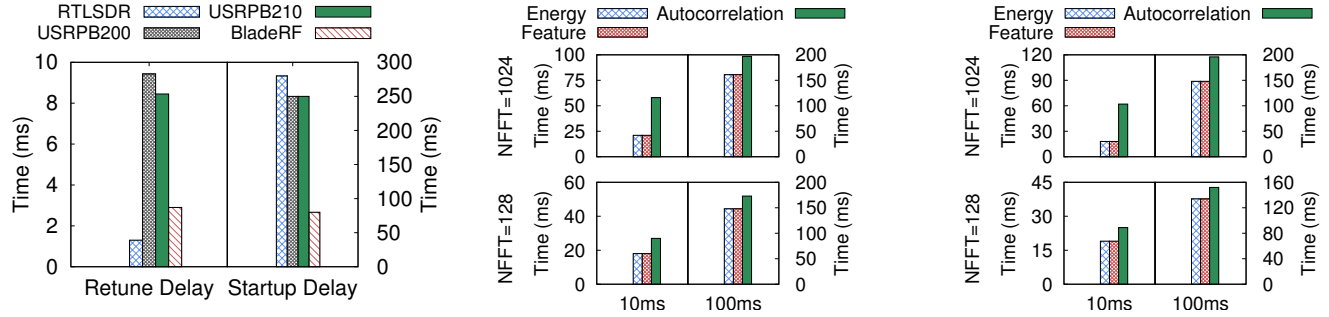
3. BENCHMARKING RESOURCE USAGE

We essentially evaluate two important metrics for the signal detection task: 1) latency and 2) energy usage. We perform these evaluations separately for the sensing and compute units. These benchmarking efforts highlight that both latency and energy usage are reasonable showcasing the feasibility of mobile spectrum sensing.

3.1 Latency

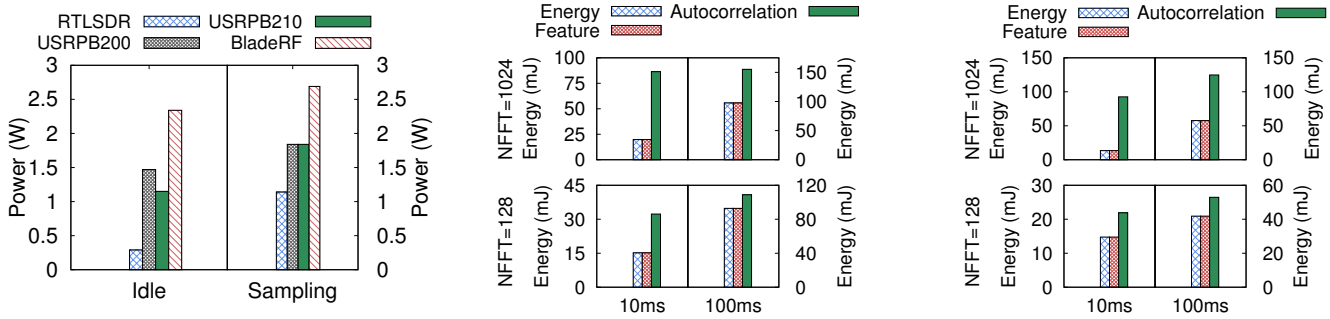
Latency measures the responsiveness of the mobile spectrum sensor platform. A series of events take place from the time when the mobile spectrum sensor is instructed to start sensing till the time it reports the spectrum occupancy decision. This involves turning on or retuning the sensing unit, obtaining and processing the I/Q samples in the compute unit and determining spectrum occupancy via a sensing algorithm presented in the previous section. Three latency components are relevant:

1. *Startup latency (sensing unit):* This is the time elapsed after instructing the device to power up until the first sample is ob-



(a) Sensing unit latency (b) Compute unit latency (Raspberry Pi) (c) Compute unit latency (phone)

Figure 3: Latency measurement results.



(a) Sensing unit power (b) Compute unit energy (Raspberry Pi) (c) Compute unit energy (phone)

Figure 4: Power/energy measurement results.

tained.

2. *Retune latency (sensing unit):* This is the time elapsed after issuing a sensing instruction involving a frequency change or change in sample rate until the first sample is obtained, assuming that the device is already powered up.
3. *Compute latency (compute unit):* This is the computation time of the sensing algorithm from the time the first sample is received by the compute unit. The algorithms are implemented in C and ported to the Android and Raspbian platforms. Pipelining is used to improve latency by overlapping sensing and computation as serializing them makes the overall latency very high.[2]

The latency results are presented in Figure 3. Note in subfigure 3(a) that the retune latency is more than an order of magnitude smaller than startup latency ($<$ 10 ms vs. 100s of ms). This indicates that samples can be fetched relatively quickly if the device is on. However, we will later see that the idle mode power consumption is relatively high and thus keep the device powered on while samples are not needed may not be a recommended practice. Subfigures 3(b) and (c) present the compute latency for the two chosen platforms. Two sensing times are used 10ms and 100ms. These translate to 10K and 100K samples, respectively, at 1Msps sample rate. These samples are averaged before feeding to the actual algorithm (involving FFT computations etc.) as explained in Section 2. We use two FFT sizes in the algorithm, 128 and 1024. As expected, the autocorrelation detector is somewhat slower than other two detectors. The difference is more significant for larger FFT sizes and also for smaller number of samples. The latter is due to the fact that the averaging times dominate for larger number of samples.

[2]This actually requires a tricky implementation. The sensing unit delivers the samples to the compute unit in terms of blocks of adjustable size (e.g., 1024 bytes). The compute unit processes a block while the sensing unit fetches the next. Obviously, block size impacts performance. We experimented with various block sizes. For the parameters of our experiments, 1024 bytes block works well. This is the size used in all reported results.

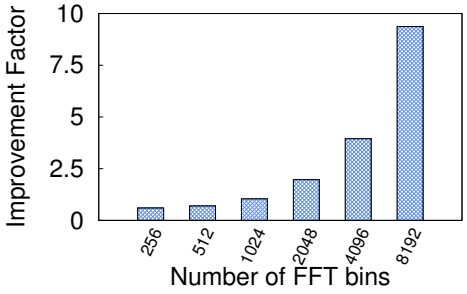


Figure 5: Improvement in computation latency for performing FFT in CPU vs. GPU in Samsung Galaxy S6 smartphone for different bin sizes.

Improvement using GPU: We also consider using the GPU (Samsung Galaxy S4 only, having a Qualcomm Adreno-320 GPU) for faster computation. We use openCL framework [4] that gives an excellent choice for executing parallel code on mobile GPUs. However, use of GPU provides only modest improvements for the FFT sizes considered in this work (upto 1024). Significant improvements were noted with larger sizes, e.g., a factor of $\approx$9 improvement with size 8192 (see Figure 5 for details). Thus, we deem GPU to be useful only when sensing over a wider band.

3.2 Energy Consumption

Just like latency, we evaluate energy consumption in both the sensing unit and compute unit. For the sensing unit, there are two states considered: 1) *Idle state:* the device is powered up, but otherwise idle, 2) *Steady state:* the device is sensing and sending the I/Q samples to the compute unit. An external power meter (specif-

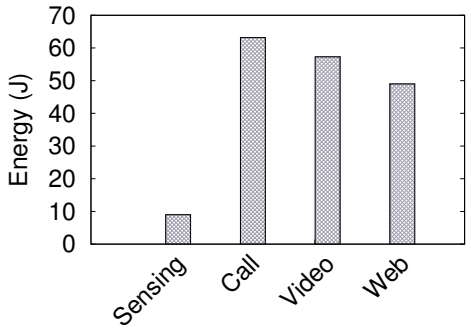


Figure 6: Comparison of spectrum sensing energy consumption with respect to typical applications in a smartphone. One minute duration is assumed.

ically, Monsoon power monitor [3] along with a separate logging laptop[3]) is used to log real time power usage in mW or W. When appropriate, power is integrated over time to determine energy (in mJ or J).

The power/energy measurement results are presented in Figure 4. The experimental parameters for subfigures (b) and (c) are exactly the same as those in Figure 3, e.g., sampling for 10ms or 100ms at 1Msps sampling rate and two FFT sizes – 128 and 1024. Figure 4(a) shows that the idle mode power consumption is relatively high except in RTL-SDR. This is due to the lack of FPGA in RTL-SDR. The plots in Figure 4(b) and (c) generally track the corresponding latency numbers, except for autocorrelation-based detector. For the latter, the energy expenses appear a little higher than what latency figures would normally indicate. This is perhaps due to significant CPU hungry computations.

3.3 Comparison with Typical Applications

To give the reader a sense how the energy values compare with a typical use cases on a smartphone platform, we compare spectrum sensing with three other applications, viz., making a call, video playback, web browsing. For the latter use cases we borrow the experimental results from [12]. For video playback and web browsing (on WiFi) the phone consumes approximately 455mW and 353mW. Additionally, screen backlight consumes about 500mW. Making a phone call (GSM) it consumes about 1054mW. Compare these with an RTL-SDR monitoring the lower 1MHz edge of a TV channel (contains the pilot) and using a sample rate of 1 Msps. Assuming the sensing budget to be 10ms, the sensor in sampling state consumes about 10mJ, the computation takes approx. 90mJ for autocorrelation detector using 1024-bin FFT, hence 100mJ per scan. Each scan spans about 70ms time including sensing and computation. Considering a 1-minute time slot, assume sensing is scheduled every 2 seconds, there would be 30 scans. The 30 scans in a minute will consume 3000mJ that take about $30 \times 70 = 2100$ms. Adding some retuning latencies etc., the sensor will be practically idle for 57 seconds that consumes about 5700mJ. Thus it consumes about 8.7J in a minute. Figure 6 compares this with the corresponding energy consumptions for the other applications. Spectrum sensing costs only about $15 - 20\%$ of any of these other applications and still ensures the phone is actually idle most of the time.

4. CONCLUSIONS

[3]For measurements, the Monsoon meter needs to power the device via a USB port directly. For measurements on the sensing unit, custom USB cables were created so that the Monsoon powers the unit, while the compute unit is still connected via USB to provide instructions and receive measurement samples.

Ever-surging demand for wireless spectrum, dynamic spectrum sharing and possibility of wireless technologies to co-exist in the same spectrum make it necessary to enable real-time spectrum monitoring at a high degree of spatial granularity. In this paper we have made a case that imparting spectrum sensing capabilities on commodity mobile devices is the only reasonable mechanism to achieve such scale. We have established the feasibility of our proposition by performing a set of benchmarking experiments using spectrum occupancy detection in TVWS as a test case. We have used a suite of USB-powered, small form-factor SDR platforms as sensing units supplemented by commodity mobile phone/embedded computers as compute units. We have applied several common spectrum sensing algorithms to study their sensing performance vis-a-vis the total latency and energy cost. We have shown that energy usage is modest relative to common smartphone applications.

5. ACKNOWLEDGEMENTS

This work was partially supported by NSF grant AST-1443951.

6. REFERENCES

[1] Blade-RF. http://nuand.com/.
[2] Google Spectrum Database. http://www.google.com/get/spectrumdatabase/.
[3] Monsoon Power Monitor. https://www.msoon.com/LabEquipment/PowerMonitor/.
[4] openCL. https://www.khronos.org/opencl/.
[5] RTL-SDR. http://sdr.osmocom.org/trac/wiki/rtl-sdr.
[6] Spectrum Bridge website. http://spectrumbridge.com.
[7] USRP B-200. http://www.ettus.com/product/details/UB200-KIT.
[8] USRP B-210. http://www.ettus.com/product/details/UB210-KIT.
[9] Second report and order and memorandum opinion and order in the matter of unlicensed operation in the TV broadcast bands. FCC ET Docket 08-260, Nov. 2008.
[10] W. R. Bennett. Spectra of quantized signals. *Bell System Technical Journal*, 27:446–471, 1948.
[11] M. Buddhikot, C. Kim, and J. Ryoo. Design and implementation of an end-to-end architecture for 3.5 GHz shared spectrum. In *Proc. IEEE DySPAN*, 2015.
[12] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proc. USENIX ATC*, 2010.
[13] A. Chakraborty and S. R. Das. Measurement-augmented spectrum databases for white space spectrum. In *Proc. ACM CoNEXT*, 2014.
[14] S. Chaudhari, V. Koivunen, and H. V. Poor. Autocorrelation-based decentralized sequential detection of OFDM signals in cognitive radios. *Signal Processing, IEEE Transactions on*, 57(7):2690–2700, 2009.
[15] A. Iyer, K. K. Chintalapudi, V. Navda, R. Ramjee, V. Padmanabhan, and C. Murthy. SpecNet: Spectrum sensing sans frontieres. In *Proc. NSDI*, 2011.
[16] A. Nika, Z. Zhang, X. Zhou, B. Y. Zhao, and H. Zheng. Towards commoditized real-time spectrum monitoring. In *Proc. ACM HotWireless*, 2014.
[17] D. Pfammatter, D. Giustiniano, and V. Lenders. A software-defined sensor architecture for large-scale wideband spectrum monitoring. In *Proc. IEEE IPSN*, 2015.
[18] S. Sen, T. Zhang, M. M. Buddhikot, S. Banerjee, D. Samardzija, and S. Walker. A dual technology femto cell architecture for robust communication using whitespaces. In *Dynamic Spectrum Access Networks (DYSPAN), 2012 IEEE International Symposium on*, pages 242–253. IEEE, 2012.
[19] J. Shi, Z. Guan, C. Qiao, T. Melodia, D. Koutsonikolas, and G. Challen. Crowdsourcing access network spectrum allocation using smartphones. In *Proc. ACM HotNets*, 2014.
[20] R. Tandra and A. Sahai. Fundamental limits on detection in low SNR under noise uncertainty. In *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, volume 1, pages 464–469. IEEE, 2005.
[21] R. Tandra and A. Sahai. SNR walls for signal detection. *Selected Topics in Signal Processing, IEEE Journal of*, 2(1):4–17, 2008.
[22] T. Zhang and S. Banerjee. A Vehicle-based Measurement Framework for Enhancing Whitespace Spectrum Databases. In *Proc. ACM MobiCom*, 2014.
[23] T. Zhang, A. Patro, N. Leng, and S. Banerjee. A wireless spectrum analyzer in your pocket. In *Proc ACM HotMobile*, 2015.
[24] M. Zheleva, R. Chandra, A. Chowdhery, A. Kapoor, and P. Garnett. Txminer: Identifying transmitters in real-world spectrum measurements. In *Proc. IEEE DySPAN*, 2015.