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 whitespace. It is widely understood that spectrum databases [6, 2] that solely rely on wireless propagation models for finding such white spaces 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-
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 1.6GHz 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, specifically when a high detection performance is desired. We assume 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. 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 detection performance.

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.

<table>
<thead>
<tr>
<th>Sensing unit</th>
<th>RTL-SDR</th>
<th>BladeRF</th>
<th>USRP B200</th>
<th>USRP B210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio Spectrum</td>
<td>24MHz - 1.7 GHz</td>
<td>300MHz - 3.8 GHz</td>
<td>50MHz - 6GHz</td>
<td>50MHz - 6GHz</td>
</tr>
<tr>
<td>Sample Size</td>
<td>8-bit</td>
<td>12-bit</td>
<td>12-bit</td>
<td>12-bit</td>
</tr>
<tr>
<td>Sample Rate</td>
<td>2.5 Msps</td>
<td>40 Msps</td>
<td>61 Msps</td>
<td>61 Msps</td>
</tr>
<tr>
<td>Frequency Correction</td>
<td>±50 ppm</td>
<td>±2 ppm</td>
<td>±2 ppm</td>
<td>±2 ppm</td>
</tr>
<tr>
<td>FPGA</td>
<td>N/A</td>
<td>Altera Cyclone 4E</td>
<td>Spartan 6 XC6SLX75</td>
<td>Spartan 6 XC6SLX150</td>
</tr>
<tr>
<td>Interface</td>
<td>USB 2</td>
<td>USB 2/3</td>
<td>USB 2/3</td>
<td>USB 2/3</td>
</tr>
<tr>
<td>Form factor</td>
<td>USB flash drive</td>
<td>5” x 3.5”</td>
<td>5.9” x 3.8”</td>
<td>5.9” x 3.8”</td>
</tr>
<tr>
<td>Cost</td>
<td>≈ 20$</td>
<td>≈ 420$</td>
<td>≈ 670$</td>
<td>≈ 1100$</td>
</tr>
</tbody>
</table>

Table 1: Capabilities and cost of the sensing units considered

![Figure 1: Some examples of platform configurations considered in this work. RPi stands for Raspberry-Pi.](image)

![Figure 1: Some examples of platform configurations considered in this work. RPi stands for Raspberry-Pi.](image)
Autocorrelation based detection: 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 technique 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 technique 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 $p$. The autocorrelation function will contain a peak at the desired frequency $f_c = f_{\text{pilot}}$. To find the peak, we can just perform another FFT on the ACF and look at the corresponding bin for the pilot 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.

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

**Signal Detection Performance**

The three algorithms have different detection performances in terms of $P_D$ and $P_{FA}$. 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-
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 1Mbps 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.

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 (specific-
Every 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


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

3For 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.