1-2014

The Challenge of Continuous Mobile Context Sensing

Rajesh Krishna BALAN
Singapore Management University, rajesh@smu.edu.sg

Youngki LEE
Singapore Management University, YOUNGKILEE@smu.edu.sg

Kiat Wee TAN
Singapore Management University, williamtan@smu.edu.sg

Archan MISRA
Singapore Management University, archanm@smu.edu.sg

The Challenge of Continuous Mobile Context Sensing

Rajesh Krishna Balan, Youngki Lee, Tan Kiat Wee, and Archan Misra
School of Information Systems, Singapore Management University
{rajesh,youngkilee,williamtan,archanm}@smu.edu.sg

***Abstract*—In this paper, we highlight the challenge of continuously sensing context data from mobile phones. In particular, we show that the energy cost of this type of continuous sensing is extremely high if a) accuracy is desired, and b) power optimisations do not work well if multiple tasks are sensing concurrently. Our results are derived from our experience in building the *LiveLabs* context sensing platform. We present results for different types of sensing tasks; ranging from simple sensing using just one sensor all the way to multi-sensor sensing performed by concurrent high-level tasks. We end with a discussion of the challenges of supporting multi-task sensing across heterogeneous devices and operating systems.**

I. Introduction

Today, it has become exceedingly common to use smartphone-embedded sensors for various context-aware applications and services. For example, Jigsaw [12] is developed for real-time monitoring of location, activity, and sound-driven events, while CoMon [9] enables sharing of phone sensors among nearby users to extend sensing capability and to lower the overall phone energy consumption. ACE [14] proposed techniques to further reduce energy consumption by leveraging patterns of contexts occurring in users' real life. Interesting systems are also emerging that monitor complex contexts such as a human's stress and emotional state [17] and conversational patterns [10].

To achieve good accuracy with low energy consumption, prior work in this area has used many techniques, such as duty cycling, multimodal fidelities, and location-based triggers, etc. However, these types of optimisations are only possible when the task that requires the sensing support is well understood and defined.

Our research team has been building, for the past two years, the *LiveLabs* platform which allows real-time mobile experiments to be run in-situ on the real phones of opt-in participants located in real environments. Currently, *LiveLabs* is deployed at the Singapore Management University campus and has (as off Nov 2013) more than 380 active participants. *LiveLabs* is slated to be deployed at a public airport, a mall, and a resort island from early 2014 onwards.

For *LiveLabs* to be successful, we need to collect sensor data from our participants' cellphones in real time and use that sensor data to provide the context triggers for various experiments that hope to run on the *LiveLabs* infrastructure. For example, an experiment could be specified as *"Send a coupon for $20 to all participants who are interested in coffee, are no engaged in any important task, and walk by the coffee store on level two, between the hours of 8 a.m. to 9 a.m., and do not stop to buy coffee.* To support this experiment, the system might need to use a number of sensors such as Wi-Fi, accelerometer, social data, etc. to figure out the location, activity, and interests of the user in order to figure out the subset of participants who might satisfy the experiments' context triggers. If this was the only experiment that needed to be executed, optimising the context collection mechanisms for this particular experiment is quite possible and should result in a fairly optimal tradeoff between power consumption and accuracy of sensing.

However, our experience with designing the *LiveLabs* context sensing systems to support multiple concurrent experiments has convinced us that when the number of experiments that are running concurrently is more than a small handful, it becomes much harder to determine an optimal tradeoff between accuracy and energy consumption that will satisfy all the experiments. In particular, we found that it becomes necessary to either optimise for power consumption (at the expense of accuracy) or sensing accuracy (at the expense of power consumption) as other alternatives result in unfair experiment partitions where some experiments get high accuracy sensing while other experiments get low accuracy sensing. These sub-optimal partitions still consume about as much power as just providing all experiments with high fidelity sensing. Note: in this paper, we do not consider the latency of the sensing results beyond quantifying the costs of storing the sensing data to internal memory and the cost of transferring the data over a Wi-Fi interface. In practice, this is a third dimension, that we discuss further at the end of the paper, that also needs to be factored into any optimisation decision.

In the rest of this paper, we will briefly describe how *LiveLabs* works, and then describe, with more details, the challenge of providing real-time high accuracy low power continuous sensing in an environment with multiple experiments. We then provide measurements that show the power consumption of various phone sensors when used individually and when used together for some common tasks. Finally, we end with some discussion of other issues and provide our current thoughts on how to resolve this conundrum.

II. Background & Related Work

In this section, we describe the *LiveLabs* experimentation environment and the key related work.

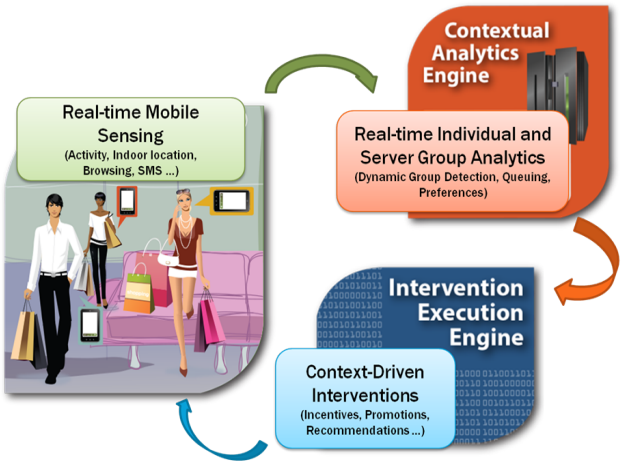


Fig. 1. The *LiveLabs* Sensing Loop

A. What is LiveLabs?

The LiveLabs Urban Lifestyle Innovation Platform (or *LiveLabs* in short) went live in September 2013 at the Singapore Management University (SMU) and has, as of November 2013, more than 380 registered participants. The goals of *LiveLabs* are to provide a testbed where mobile experiments (of all forms) can be tested in-real time on actual devices carried by real participants in real environments.

Figure 1 shows the three main components of *LiveLabs*. It starts with real-time mobile sensing on the participants' cellphones. Note: all participants opt-in to *LiveLabs* and we have an explicit and comprehensive data collection and privacy policy. The data collected is then fed into our contextual analytics engine where we identify the real-time contextual triggers needed by the intervention execution engine. This engine collects the experiments specified by various experimenters (using our experiment specification system) and determines the contextual triggers needed for each experiment. When the triggers are satisfied (as determined by the contextual analytics engine using data collected by the mobile sensing components), the engine sends the experiment to the appropriates participant's phone where their reaction to the experiment is monitored and the results of the experiment are then returned to the experimenter for their analysis and use. This cycle of sensing, analytics, and interventions is repeated many times over as new participants, analytics, and interventions/experiments enter *LiveLabs*.

Currently, *LiveLabs* is deployed only at SMU. However, in the near future (by the end of 2014), we aim to deploy *LiveLabs* at a public airport, a public mall, and a public resort island where members of the public will become part of the *LiveLabs* participant pool. These environments will also allow us to test many more types of experiments (such as logistics, purchasing, leisure, and travel related experiments) that cannot currently be tested on our university testbed comprising of mostly undergraduate student participants.

B. Related Work

Researchers have long recognised that the energy-overheads of continuous mobile sensing pose a major challenge to the realisation of many forms of context-aware pervasive computing. Jigsaw [12] attempts to reduce such energy overheads for multiple sensing applications simultaneously. It employed (a) an adaptive processing pipeline, where subsequent stages of the context extraction progress would be short-circuited or aborted if specific features of interest (e.g., usage artefacts or background ambient noise) were detected in the earlier pipeline, and (b) correlation across sensing pipelines, effectively using cheaper sensors to trigger the activation of more energy-intensive sensors. In contrast, the Orchestrator framework [7] focuses on resource-sharing among multiple context-aware applications running concurrently, by looking for opportunities where a common set of sensors can be used to compute multiple different contexts. Researchers have also investigated the accuracy vs. energy tradeoffs for specific applications associated with specific sensors–e.g., the A3R framework [20] dynamically adjusts the accelerometer sampling frequency and processing logic to preserve the accuracy of locomotive activity detection.

More recently, various forms of cloud-based processing have also been explored to reduce the energy overheads of either the actual data sensing or processing phases. The SociableSense framework [18] has investigated the use of reinforcement learning mechanisms to adapt both the sensor sampling frequency (specifically for accelerometer, microphone and Bluetooth sensors) and the processing *location* (i.e., on the device or in the cloud), for a specific application focused on capturing interactions among colleagues in the workplace. Similarly, the Co-GPS approach utilises the cloud to significantly reduce the power of GPS-based location sensing by effectively modifying the functions of the embedded GPS receiver, such that the actual computation of the position is no longer instantaneous but deferred to the cloud, thereby allowing the GPS radio to duty-cycle extremely aggressively. This approach also illustrates a trade off between the energy overhead and the *latency* of context determination–in general, even if the sensor data is sensed continuously, energy can be saved if the resulting context is not needed instantaneously but can instead be computed in a deferred fashion. However, a general purpose framework for multi-context optimisation of cloud-assisted mobile sensing is still an open problem.

An alternative approach towards reducing the energy overheads involves exploiting the statistical correlation across different forms of context, arising from patterns of daily living, as a means for *short-circuiting* the processing of As embodiments of this approach, ACE [14] applies rule mining techniques to discover individual-specific exclusionary correlations among different contexts of an individual (e.g., a user detected to be "driving" cannot be at home), whereas ACQUA [11] utilises probabilistic query processing to evaluate contextual predicates that have a higher probability of short-circuiting a more complex set of query predicates. These approaches are usually quite effective in reducing energy for context that is repetitive, but is not applicable for detecting more *spontaneous* context (e.g., is user A moving about in the mall alone or in a group?).

Finally, another approach towards lower-energy continuous context sensing involves cooperative *load-sharing* across a

group of devices, usually located near one another. Embodiments of this approach include CoMon [9] where proximate devices negotiate sensing tasks so as to amortise their overall energy consumption, and ErdOS [19] where devices look to effectively perform round-robin activation of commonly available sensors between nearby smartphones. This approach is applicable only to *ambient* context–i.e., context that is effectively common to multiple neighbouring phones (e.g., noise levels, or movement speeds).

As opposed to these software-based approaches, hardware-based approaches promise significantly larger drops in the sensing energy overhead–these approaches much of the power spent in background mobile sensing is consumed not by the sensor itself, but by the general purpose computing hardware (CPU, memory) that remain in a power-hungry *active* mode at all times. The LittleRock approach [16] demonstrated how a separate low-energy co-processor could achieve a two order-of-magnitude decrease in the computational energy of mobile sensing, by effectively letting the main processor be awakened only when specific higher-layer context events were detected. Apple's recent iPhone 5S adopts this principle, using a separate M7 co-processor for processing motion events (computed from the accelerometer, gyroscope and compass sensors). Similarly, the latest Moto-X smartphone uses a separate audio co-processor to look for audio context efficiently, while the main processor remains in low-power mode. While such approaches are likely to make extraction of certain popular varieties of context significantly more energy-efficient, we believe that additional system overheads (e.g., storage and wireless communication) will continue to present challenges for the design and operation of a general-purpose context collection engine on commodity mobile devices.

It is important to realise that innovative approaches for such continuous mobile sensing seek to trade off energy against two distinct metrics: *accuracy*, indicating the fidelity of the inferred context and *latency*, indicating the delay with which the context is recognised after its actual occurrence. Very recently, researchers have proposed the LAB programming abstraction [8] to allow programmers to explicitly indicate such tradeoffs. However, quantitatively expressing such tradeoffs at application design time in the context of LiveLabs is not trivial, as the requirements and trade off sensitivity itself will vary with the sensed context (e.g., an individual's fine-grained locomotive activity may only be needed if he is in a place with a sufficiently long queue).

III. Why Is Continuous Sensing Hard?

The key takeaway of this paper is that supporting multiple sensing applications while providing high accuracy and low power consumption is not easy as the accuracy requirements of one application will override the power gains possible by optimising another application. As a result, the overall power consumption is still high even with all the effort spent in optimising specific applications.

To provide more insight into this phenomena, the rest of this section will do the following: 1) provide quantitative numbers for the energy consumption of various phone sensors, 2) show that the energy consumption is not a simple linear function when multiple sensors are used together, and 3) show that the energy consumption when running multiple sensing tasks is driven by the most demanding (accuracy-wise) sensing task. We then end with our current proposed solution.

A. Testing Methodology

For all the tests in this section, we used a Samsung Galaxy S IV smartphone. We wrote separate test applications to determine the power drain of various sensors. These applications were all run as foreground applications with no additional processing or storage (unless otherwise stated) so that the true power consumption of the sensors could be determined. This is important as Android background applications need to use Wakelocks which are not very energy efficient (Wakelocks appear to cause a large amount of CPU and memory power consumption). Hence, to avoid having the power consumption of the sensors be just a small almost negligible portion of the total energy consumption (which would be true for a background application), we measured all the sensors using a foreground application. We ran the data collection and energy measurement software for each sensor result for 20 minutes per sensor result except for Figures 2 and 3 where each sensor result was generated from a 10 minute experiment per sensor. We report the average additional power consumption value (over the base system), over the 10 or 20 minute time period, for each sensor result in this paper. To accurately compute the power drain caused by our various applications, we used a Monsoon hardware power monitor and measured the power drain with a clean phone system (phone on with no applications running, screen off, sensors off, LTE enabled with data enabled) and then with just the sensing application running.

B. Energy Consumed By Individual Sensors

We start by showing the power consumption of individual sensors as well as the cost of storing and transmitting the sensor results. The results in this portion should be seen as complementary to prior per-sensor energy measurements such as the work done by Carroll and Heiser [2].

Inertial Sensors: Figure 2 shows the power consumption of the inertial sensors, i.e., accelerometer, gyroscope and compass. We observe that, similar to other measurement studies, that the gyroscope and compass consume more energy than the accelerometer and that the power consumption and accuracy of the inertial sensors is greatly affected by the sampling modes used (the actual sensing rate per mode is listed in Table I). The main energy costs are not only the cost of the sensor itself (sensing cost), but also the energy cost of the CPU to process the data and the energy cost of storing the data in internal flash memory. In particular, as we show later, the storing of the data into the phone's internal flash memory is the dominant energy cost. In the *LiveLabs* infrastructure, this storage cost is inevitable as our context collectors collect data regularly with the collected data only periodically sent back to the *LiveLabs* data server (sent at periodic intervals or whenever the phone is connected to Wi-Fi; whichever comes first).

Pressure and Light Sensors: Figure 3 shows the power consumption for the pressure sensor (Barometer) and light sensor. We observe that the energy consumption of the pressure sensor shows a similar pattern to the inertial sensors. However, the absolute consumption values of the pressure sensor are much higher. Thus, given that the sampling rates for the pressure sensor are similar to the inertial sensors, we conclude that the pressure sensing is an expensive sensor to use — energy wise.

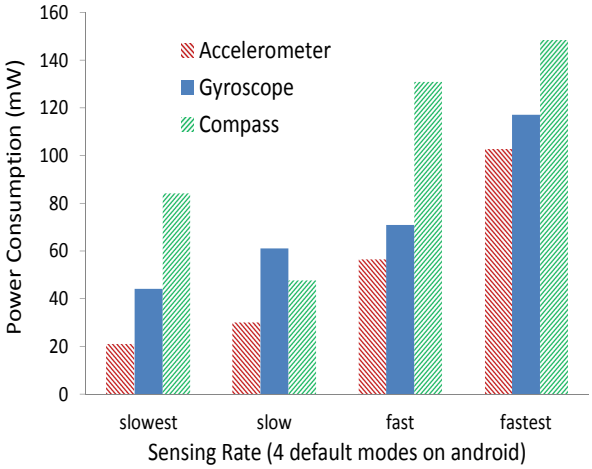
On the other hand, the light sensor is very energy efficient and its power consumption does not increase even as we change the sensing mode. However, it must be noted that the sampling rate of the light sensor, unlike other sensors, does not change when using different modes; it is always fixed at 5.5 Hz.

GPS Sensor: Figure 4 shows the power consumption of the GPS sensor. The sensing rate of the GPS sensor is very low compared to other sensors. Hence, most of its power drain is actually from the sensing task itself (i.e., getting signals from satellites and triangulating the location) and not from repetition. We observe that the power consumption saturates at around a 13 minute duty cycle.

Cost of Storing Sensor Data to the Internal Flash Memory: Figure 5 shows the power consumption caused by storing high-rate sensor data to the internal flash memory of the phone. Note: this is the built-in additional storage memory and not an external user-replaceable SD card. In particular, it shows the costs for using the accelerometer and light sensors — both with and without storing of data to the internal flash memory. We observe that the energy cost of storing data to the internal flash memory dominates the cost of using the sensor.

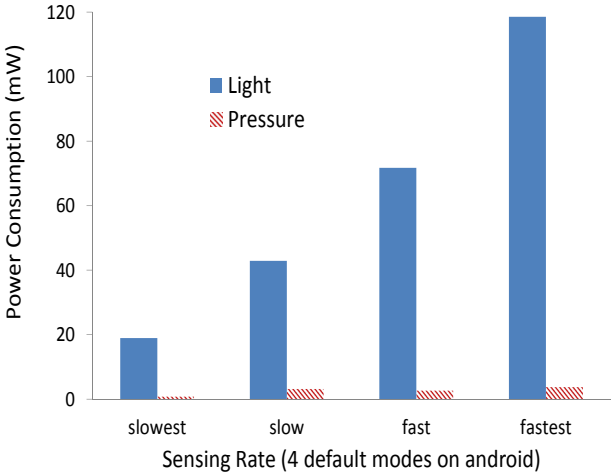
This is because, as shown by Carroll and Heiser [1], storing to the internal flash memory requires powering up the CPU and memory sub-systems and this dominates the power consumption. Our results are using a foreground sensing application with very little CPU usage (and the CPU is in a low power state when it is used). Thus, saving to the internal flash memory is extremely power hungry as the CPU transitions to a more power-hungry state and the memory subsystem gets activated as well. Hence, we need to be frugal at storing data and either store as little as possible or combine the stores from multiple sensors to amortise the energy overheads. Note: if the sensing application was running as an Android background application, the cost of saving to internal flash memory will be much lower ($\approx$ 20 to 40 mW extra) as a background process, on Android at least, has to use Wakelocks which already wake up the CPU and memory components. However, in this case, the background sensing application will consume much more power than the foreground application ($\approx$ 3 times more) even if it does not save to internal flash memory (as the CPU and memory components are in full power mode and active).

Wi-Fi Transmission Cost: Finally, Figure 6 shows the power consumption caused by using the Wi-Fi interface to send sensor data to a back-end server. In this case, accelerometer data collected using the *slowest* sensing mode (5.5 Hz) is being sent to a local server over a 802.11g link. We observe that if the Wi-Fi interface is already on and the reporting period (i.e., when the stored accelerometer data is sent back to the server) is greater than 3 minutes, storing the sensor data in memory (at the cost of higher memory usage) and then sending it, over Wi-Fi, to the server consumes less power than storing the data into internal flash memory and sending it to the server later (the cost of storing into internal flash memory is shown in Figure 5).



The power consumption rises as the sensing rate goes up with the compass at *slow* rate being the only anomaly. We have verified this result across multiple runs; however we are not sure why this results occurs.

Fig. 2. Power Consumption of the Inertial Sensors



The power consumption of the light sensor increases almost linearly as the sensing rate goes up while the consumption of the pressure sensor is almost constant (within experiment error) as it only has one sensing rate.

Fig. 3. Power Consumption of the Light and Pressure Sensors

C. Energy Consumed by a Group of Sensors

Next, we measure the cost of using a combination of sensors to understand if the power consumption scales linearly with the number of sensors used. Figure 7 shows the power consumption when multiple sensors are used together. power. In particular, when using the *slow* and *slowest* modes, the power consumption is similar to a compass-only cost. However, when using the *fast* and *fastest* modes, the power consumption of the inertial sensors is more than any single sensor but still

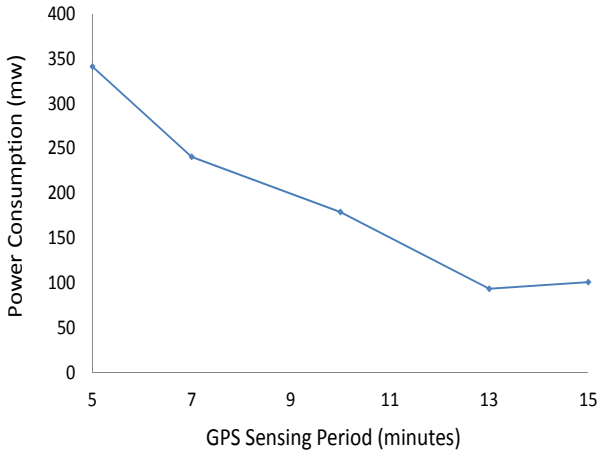


Fig. 4. Power Consumption of GPS Sensing

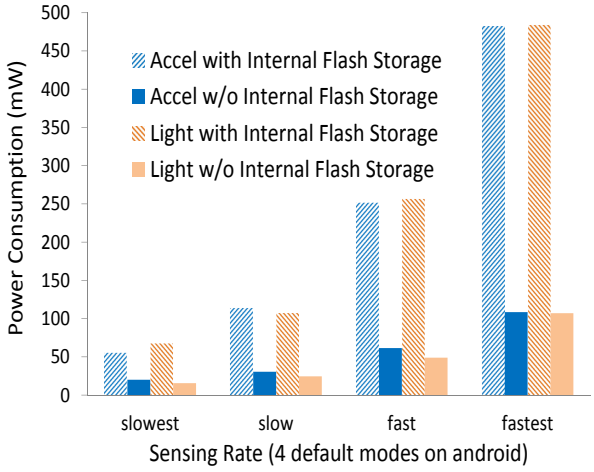
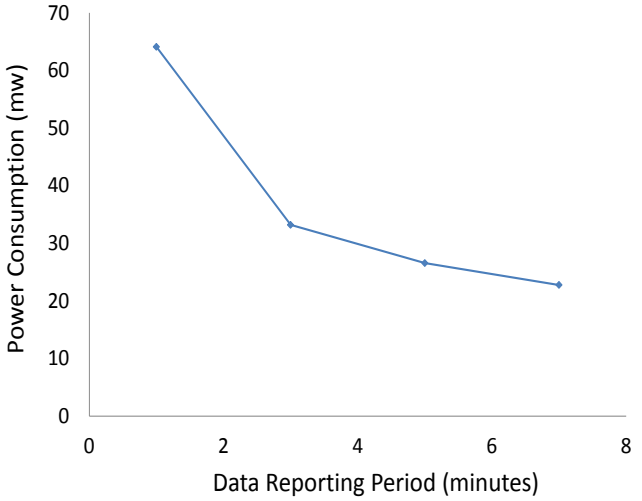


Fig. 5. Power Consumption of the Internal Flash Storage

much less than a linear sum of the power costs of each individual sensor used. We speculate that this non-linear power behaviour is due to the phone-specific SoC design of the sensing components. However, we will need more analysis to determine the true reasons for these power measurements. However, when we turn on the pressure and light sensors, we see a sharp rise in the power consumption (almost equal to the cost of the pressure sensor). We speculate that this is because, on our test phone, the pressure and light sensors use separate internal circuitry from the inertial sensors. Hence, while the power cost of some sensors can be amortised together, this does not apply across all sensors. Hence, if power is a key constraint, we need to be careful when choosing which sensors to enable together on specific devices.

D. Energy Consumption Across Sensing Tasks

Next, we analyse the power consumption when various sensors are used to achieve a higher level task. Figure 8 shows the power consumption to store diverse phone events that can be used to infer the user's context. We observe that the power consumption for obtaining and storing these events is almost zero. This is because the operating system is already tracking these events and thus the cost of "sensing" them is the small



This test used the accelerator's *slowest* sensing mode without storing the data in internal flash memory

Fig. 6. Power Consumption of the Wi-Fi Interface

Sensor type	Sensing mode			
	Slowest	Slow	Fast	Fastest
Accelerometer	5.5	16.7	50	100
Gyroscope	5.5	16.7	50	100
Compass	5.5	16.7	50	100
Pressure	5.5	5.5	5.5	5.5
Light	5.5	16.7	50	100

TABLE I
ACTUAL ANDROID SENSING RATES OF DIFFERENT SENSORS (IN HZ)

cost to read the events from the OS and then to store it somewhere else. In addition, the events is generally low and thus, this cost could increase for more active phone users who might generate more events that need to be stored.

Figure 9 shows the power consumption to perform activity recognition using data from the inertial sensors followed by processing of the data to infer the user's activity. We observe that activity recognition is a lot cheaper than sensing and storing raw accelerator data as the CPU cost for activity recognition is low and, in particular, is much lower than the storage cost of saving the entire raw accelerometer data to the internal flash memory. We can reduce the cost of activity recognition even more by duty cycling the recognition engine as, generally, the activity performed by the user does not change that frequently. Hence, we can use a 10 second or more duty cycle which will reduce the energy cost even further.

E. Tension Between Power and Accuracy

We next analyse the tradeoff between accuracy and power consumption when performing a sensing task. Figures 10 and 11 show the accuracy and power consumption for activity recognition, respectively. We observe that when we use a higher sensing rate with more features, the recognition accuracy goes up; but not very significantly. However, the power consumption increases pretty significantly due to both the increased sensing rate and the extra CPU cycles needed to calculate the extra domain features. Hence, we observe

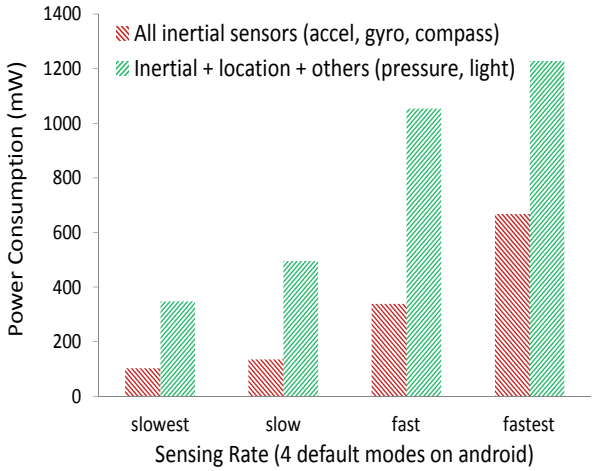
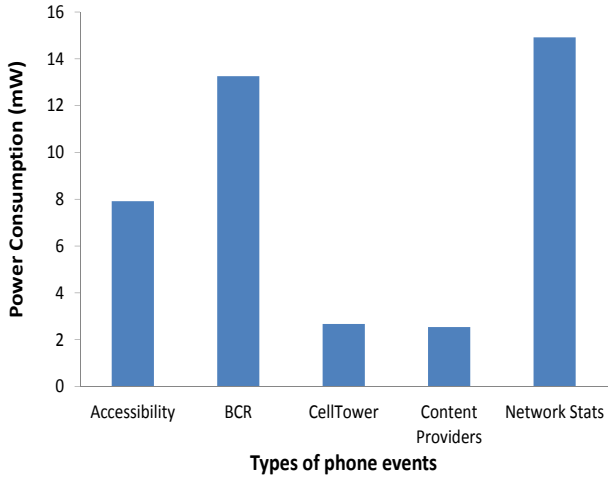


Fig. 7. Power Consumption of Sensor Groups



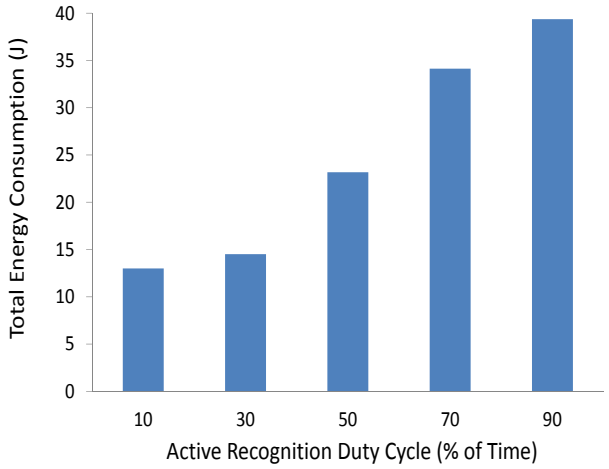
The Android events we captured were Accessibility – all events pushed by applications and services to the accessibility framework, BCR – system events (battery draw, incoming SMS, incoming call etc.) sent to the broadcast receiver subsystem, CellTower – all cell tower ID changes, Content Providers – activity seen by a few key applications (contacts, phone log, SMS, and calendar), Network Stats – a count of the network packets sent and received (in terms of bytes), for each polled time period (every 60 minutes), for each application that used the network in the polled period along with the total system count.

Fig. 8. Power Consumed to Sense and Store Diverse Phone Events

that high accuracy results are possible, but at a significantly increased power costs — at least for activity recognition. More details about the activity recognition system used for this test can be obtained from Yan et. al [20].

F. Solution? No Free Lunch

In this section, we observed that the costs of sensing does not scale linearly with the number of sensors used. In particular, the costs of adding additional sensors does not scale linearly for most sensors (although the absolute cost still increases). However, as shown in Figure 7, the cost of sensing increases dramatically when higher accuracy sensing modes and computation is used. Thus the cost of just one high accuracy sensing task (that uses either the *fast* or *fastest* sensing rates) can dwarf the costs of other low accuracy



The activity recognition task used accelerometer data at 5.5 Hz (*slowest* rate) to detect walking, standing, and sitting activities in a ten minute interval using different duty cycling intervals.

Fig. 9. Power Consumption of an Activity Recognition Task

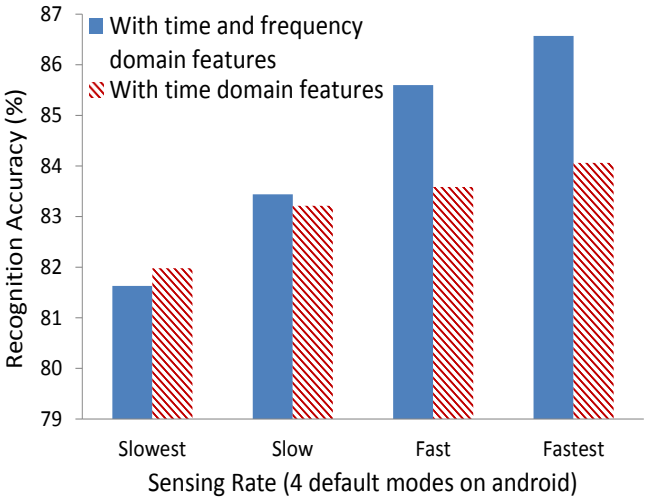


Fig. 10. Activity Recognition Accuracy Versus Sensing Rate Tradeoff

sensing tasks (that use the *slowest* or *slow* rates). Thus, a practical sensing solution will need to be aware of the sensing modes used by each task to ensure that the overall power budget is maintained.

We thus use the *slowest* sampling rate for all sensors in the *LiveLabs* context sensing software, coupled with storing the sensor data in the internal flash memory with infrequent uploads (at most every 10 minutes) using Wi-Fi to the back-end server on both iOS and Android. This allows our sensing software to use only a little bit of power (on both platforms, we use at most 5% more power than not having our software installed) while achieving some minimum level of sensing that is still adequate for some applications (coarse grained indoor location, detecting certain types of activities, etc.). In addition, we can improve the power efficiency further by optimising the way we write sensor data to the internal flash memory.

However, this exceptional energy performance does result in a lack of high fidelity sensing data. To overcome this limitation, our software can be triggered, by the *LiveLabs*

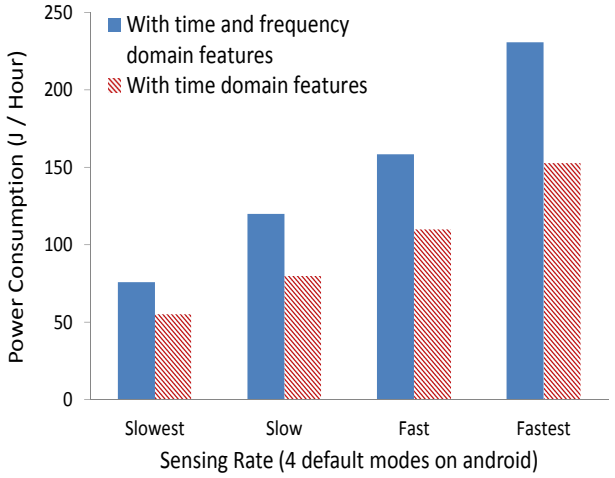


Fig. 11. Activity Recognition Power Versus Sensing Rate Tradeoff

server, to increase the sensing rate for short periods of time in order to collect task-specific higher accuracy sensing data.

G. Limitations of Evaluation

While the results in this section do give a very good overview of the challenges of continuous mobile sensing, they do have their limitations. First, all the results were measured on a single device, a Samsung Galaxy S IV. It is quite likely that the absolute power consumption of various sensors will be different on other phones due to differences in internal circuitry and chip usage. Second, all the results were generated using a foreground Android application. As explained in Section III-B, a background Android application will have a very different power profile. Even with these limitations, however, we believe that the results in this paper will help other researchers understand the kinds of issues that they need to address when they develop sensing applications on other devices.

IV. Discussion

A. The Problem of Heterogeneity

In this section, we discuss a very real and important practical challenge we faced when building the sensing components of *LiveLabs* — namely, the heterogeneity inherent in the smartphone market. In particular, there are two dominant OS manufacturers (Apple's iOS and Google's Android) along with other plausible alternatives (Microsoft's Phone 8, Blackberry's OS 10, etc.). Even if we consider just iOS and Android, the differences in their sensing capabilities are quite staggering.

First, since iOS is only available on iPhones, the possible sensors are fixed to those available on the few iPhone devices still commonly used (iPhone 3gs, 4, 4s, 5, 5s, and 5c in Singapore), However, even within this small set of devices, there are already key differences in the available sensors and the accuracy of specific sensors. For example, a magnetometer was added from the iPhone 3gs onwards, while a gyroscope was introduced from the iPhone 4 onwards. Finally, the iPhone 5s uses a new hardware chip for the accelerometer. In particular, the iPhone 5s uses a Bosch accelerometer chip [4] while all previous iPhones (such as the iPhone 5 [6]) use STMicroelectronics developed accelerometer chips. Even this small difference has already caused sensing issues with the iPhone 5s [15].

The situation gets worse when you factor in the OS itself. For example, in iOS 6, it was possible to obtain Wi-Fi MAC address data programatically using software APIs. However, in iOS 7, this capability has been removed. Thus sensing applications that use specific APIs need to change every time a new OS version is released. Fortunately, these OS releases occur only once a year and thus the impact is limited to just a few times a year.

The situation, unfortunately, is much worse for Android. Here, the sensors available on each phone are determined by the phone manufacturer itself. Hence, phones from Samsung will have different sensors from phones from HTC etc. For example, the accelerometer on a Samsung S III is made by STMicroelectronics [5] while the accelerometer on a comparable HTC One phone is made by Bosch [3]. These differences, as stated above and demonstrated by prior researchers [13], can lead to very different sensing results.

To make matters even worse, even phones from the same manufacturer can have very different sensing capabilities. For example, Samsung's top-of-the-line S III, S IV, and Note 2, and Note 3 devices have a barometer in them. However, this sensor is absent in the other Samsung devices such as the Ace 3, Light, Round, and Express 2. Finally, updates to the underlying Android OS can and do require changes to the sensing software to overcome new restrictions that have been added to the OS. This is made worse by the custom changes made by each device manufacturer to the underlying OS. For example, on Samsung S IV devices, a Wi-Fi scan only returns new results every 4.5 seconds (more frequent scans will return cached data). This behaviour is not found on other smart phones and appears to a specific patch added by Samsung to the underlying OS (most probably as a power saving technique).

All these heterogeneity factors make it very hard to build a common sensing platform that can work across different devices and OSs. As we are finding out in the process of building *LiveLabs*, a key challenge is monitoring the release of new OSs and devices and adjusting our sensing software to match; otherwise our first realisation that something is not right will be when either the participants complain or we find that our data feed has stopped or become corrupted — both situations should be avoided as much as possible.

B. Optimising For Latency

As mentioned at the end of the introduction, a third optimisation criteria for sensing systems is latency. In particular, this is the latency between when the sensing result was obtained and when it is consumed. For example, in a low latency system, the sensing results are consumed as soon as they are obtained while in a high latency system, the sensing results are stored for some period of time and then transmitted to the results processing server for consumption.

Many sensing applications save the sensed data to internal memory and only transmit it periodically to a backend server.

However, this is not suitable for real-time or near real-time applications, no matter how accurate the sensed data is, as the sensed data will be stale when used. For example, collecting accelerometer data at the *fastest* rate for 5 minutes but only processing it 1 hour later will result in a very accurate, yet probably not very useful to a real-time application, activity record of the user's activities 65 minutes ago! For applications that focus on data collection with offline data use (e.g., collecting test and training sets for research algorithms etc.), this latency is not a factor. However, for applications that want to provide real-time or even near real-time user interventions, this latency can be a significant factor.

Thus, it is crucial that the latency at which the sensing data is sent to the server matches the application requirements as much as possible. In the *LiveLabs* environment, we plan to run our data collection in low fidelity mode with periodic uploads to the server throughout the data. This will allow us to generate user profiles that can be used to generate interesting experiments and test scenarios. In addition, we have the ability to dynamically configure the sensing collection software to both increase its sensing fidelity (at the expense of increased power consumption) and to upload the sensing data immediately to the back-end server. We plan to use these dynamic features to test various real-time context-sensing and context-aware applications and services.

V. Conclusion & Future Work

In this paper, we highlighted some of the issues with continuous context sensing — in particular that the energy required to sense continuously is high enough that the user notices. This problem becomes particularly difficult when multiple sensing tasks, such as activity recognition and indoor location, need to run concurrently. In this situation, it becomes hard to optimise the energy consumption of the sensing as the benefits gained in one task could be overridden by the higher rate sensing (for accuracy reasons) of another task. Our results suggests that a user-agreeable solution will involve using low fidelity low energy consuming sensing for most of the time with short infrequent bursts of high accuracy high energy sensing. However, we are currently testing various duty cycling approaches to understand these accuracy and energy tradeoffs when performing context sensing across multiple tasks.

VI. Acknowledgements

This work is supported in part by the National Research Foundation Singapore under its IDM Futures Funding Initiative, and administered by the Interactive & Digital Media Program Office, Media Development Authority. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the granting agency, or Singapore Management University.

The authors would also like to thank the entire *LiveLabs* team for their help in making *LiveLabs* a reality! Specifically, we would like to thank Koh Quee Boon and Vignesh Subbaraju for building the iOS context sensing software and the activity detection subsystem respectively.

References

[1] Carroll, A. and Heiser, G. An analysis of power consumption in a smartphone. *Proc. of the USENIX Annual Technical Conference (USENIX ATC)*, Boston, Massachusetts, June 2010.

[2] Carroll, A. and Heiser, G. The systems hackers guide to the galaxy: Energy usage in a modern smartphone. *Proc. of the Asia-Pacific Workshop on Systems (APSYS)*, Singapore, July 2013.

[3] Chipworks Inc. *Inside the HTC One*. http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-htc-one/.

[4] ChipWorks Inc. *Inside The iPhone 5s*. http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-iphone-5s/.

[5] Chipworks Inc. *Inside the Samsung Galaxy SIII*. http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-samsung-galaxy-siii/.

[6] iFixIt. *iPhone 5 Teardown*. http://www.ifixit.com/Teardown/iPhone+5+Teardown/10525.

[7] Kang, S., Lee, Y., Min, C., Ju, Y., Park, T., Lee, J., Rhee, Y., and Song, J. Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. *Proc. of the International Conference on Pervasive Computing and Communications (PerCom)*, Mannheim, Germany, Mar. 2010.

[8] Kansal, A., Saponas, S., Brush, A. B., McKinley, K. S., Mytkowicz, T., and Ziola, R. The latency, accuracy, and battery (lab) abstraction: Programmer productivity and energy efficiency for continuous mobile context sensing. *Proc. of the International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, Indianapolis, Indiana, Oct. 2013.

[9] Lee, Y., Ju, Y., Min, C., Kang, S., Hwang, I., and Song, J. Comon: Cooperative ambience monitoring platform with continuity and benefit awareness. *Proc. of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Lake District, United Kingdom, June 2012.

[10] Lee, Y., Min, C., Hwang, C., Lee, J., Hwang, I., Ju, Y., Yoo, C., Moon, M., Lee, U., and Song, J. Sociophone: Everyday face-to-face interaction monitoring platform using multi-phone sensor fusion. *Proc. of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Taipei, Taiwan, June 2013.

[11] Lim, L., Misra, A., and Mo, T. Adaptive data acquisition strategies for energy-efficient, smartphone-based, continuous processing of sensor streams. *Distributed Parallel Databases*, 31(2), June 2013.

[12] Lu, H., Yang, J., Lane, N. D., Choudhury, T., and Campbell, A. T. The jigsaw continuous sensing engine for mobile phone applications. *Proc. of the Conference on Embedded Networked Sensor Systems (Sensys)*, Zurich, Switzerland, Nov. 2010.

[13] MEI research Ltd. *Performance of Smartphone On-Board Accelerometers For Recording Activity*. http://www.actipal.com/MEI-TOS2011-Onboard_Accelerometer.pdf.

[14] Nath, S. Ace: Exploiting correlation for energy-efficient and continuous context sensing. *Proc. of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Lake District, United Kingdom, June 2012.

[15] O'Grady, J. D. *iPhone 5s accelerometer bug linked to new chip*. http://www.zdnet.com/iphone-5s-accelerometer-bug-linked-to-new-chip-7000022062/.

[16] Priyantha, B., Lymberopoulos, D., and Liu, J. Littlerock: Enabling energy-efficient continuous sensing on mobile phones. *IEEE Pervasive Computing*, 10(2), Apr. 2011.

[17] Rachuri, A. K., Musolesi, M., Mascolo, C., Rentfrow, P. J., Longworth, C., and Aucinas, A. Emotionsense: A mobile phones based adaptive platform for experimental social psychology research. *Proc. of the International Conference on Ubiquitous Computing (UbiComp)*, Copenhagen, Denmark, Sept. 2010.

[18] Rachuri, K. K., Mascolo, C., Musolesi, M., and Rentfrow, P. J. Sociablesense: Exploring the trade-offs of adaptive sampling and computation offloading for social sensing. *Proc. of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, Las Vegas, Nevada, Sept. 2011.

[19] Vallina-Rodriguez, N. and Crowcroft, J. ErdOS: Achieving energy savings in mobile OS. *Proc. of the International Workshop on Mobility in the Evolving Internet Architecture (MobiArch)*, Bethesda, Maryland, June 2011.

[20] Yan, Z., Subbaraju, V., Chakraborty, D., Misra, A., and Aberer, K. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. *Proc. of the International Symposium on Wearable Computers (ISWC)*, Newcastle, United Kingdom, June 2012.