Piggyback CrowdSensing (PCS): Energy Efficient Crowdsourcing of Mobile Sensor Data by Exploiting Smartphone App Opportunities

Nicholas D. Lane[†], Yohan Chon[‡], Lin Zhou[*], Yongzhe Zhang[⊓], Fan Li[Ш]
Dongwon Kim[‡], Guanzhong Ding[⋈], Feng Zhao[†], Hojung Cha[‡]

[†]Microsoft Research Asia, [‡]Yonsei University, [*]UC Santa Barbara
[⊓]Shanghai Jiao Tong University, [Ш]Pearson, [⋈]National Tsing Hua University

ABSTRACT

Fueled by the widespread adoption of sensor-enabled smartphones, mobile crowdsourcing is an area of rapid innovation. Many crowd-powered sensor systems are now part of our daily life – for example, providing highway congestion information. However, participation in these systems can easily expose users to a significant drain on already limited mobile battery resources. For instance, the energy burden of sampling certain sensors (such as WiFi or GPS) can quickly accumulate to levels users are unwilling to bear. Crowd system designers must minimize the negative energy side-effects of participation if they are to acquire and maintain large-scale user populations.

To address this challenge, we propose *Piggyback CrowdSensing* (PCS), a system for collecting mobile sensor data from smartphones that lowers the energy overhead of user participation. Our approach is to collect sensor data by exploiting *Smartphone App Opportunities* – that is, those times when smartphone users place phone calls or use applications. In these situations, the energy needed to sense is lowered because the phone need no longer be woken from an idle sleep state just to collect data. Similar savings are also possible when the phone either performs local sensor computation or uploads the data to the cloud. To efficiently use these sporadic opportunities, PCS builds a lightweight, user-specific prediction model of smartphone app usage. PCS uses this model to drive a decision engine that lets the smartphone locally decide which app opportunities to exploit based on expected energy/quality trade-offs.

We evaluate PCS by analyzing a large-scale dataset (containing 1,320 smartphone users) and building an end-to-end crowdsourcing application that constructs an indoor WiFi localization database. Our findings show that PCS can effectively collect large-scale mobile sensor datasets (e.g., accelerometer, GPS, audio, image) from users while using less energy (up to 90% depending on the scenario) compared to a representative collection of existing approaches.

SenSys'13, November 11–15, 2013, Rome, Italy

Categories and Subject Descriptors

C.2.4 [**Distributed System**]: Distributed applications

General Terms

Algorithms, Human Factors, Performance

Keywords

Crowdsourcing, Smartphone Sensing

1. INTRODUCTION

Mobile crowdsourcing systems are becoming increasingly prevalent in society. Commuters monitor highway congestion using GPS readings from other drivers stuck in traffic [9, 38]. Commercial location services rely on WiFi maps built from data collected from millions of smartphone users [6, 3]. City planners similarly leverage smartphone microphones to track noise pollution levels [34]. Now, even shoppers are receiving incentives to provide images of store interiors, allowing for the assessment of product displays [7]. This growing class of crowdsourcing systems depends on collecting mobile sensor data from large numbers of smartphone users. Keeping the burden placed on participating users as low as possible is critical to these systems' success. Yet, surprisingly, many existing systems employ relatively simplistic and inefficient data collection strategies.

Common data collection strategies include relying on (1) highly engaged users to manually capture high-quality data that tightly matches application requirements; (2) periodic or random data sampling; or (3) context-driven sampling, typically based on location. Accompanying each strategy are significant negative side-effects, including, poor data quality, excessive energy consumption, and high user engagement. For mobile crowdsourcing to reach its considerable potential, new approaches are required for mobile sensor data collection that would both enable energy efficiency and reduce the level of engagement required from users.

Toward meeting these challenges, in this paper, we propose *Piggyback CrowdSensing* (PCS) – a smartphone- and cloud-based system for energy efficient crowdsourcing of mobile sensor data. PCS is designed to intelligently leverage the opportunities for collecting sensor data that frequently occur during everyday smartphone user operations, such as placing calls or using applications. We refer to these situations as *Smartphone App Opportunities*. At these times,

the energy cost of sensing can be significantly reduced because required smartphone components (for example, CPU or even the sensor itself) are already activated from an idle state. Under PCS sensor data collection, computation and uploading is performed as a background process, without user involvement, and at times when the these actions' energy consumption can be minimized.

Efficiently using sporadic user-driven sensing opportunities to satisfy crowdsourcing sensing requirements is challenging. For example, simply applying a greedy strategy and exploiting every smartphone app opportunity to sense would quickly consume too much energy and neglect potentially better, later opportunities in favor of earlier ones. Instead, PCS's operation is guided by predictive models that capture the smartphone app usage patterns that are specific to each user. By predicting upcoming sensing opportunities, PCS can compare current opportunities to sample against to future ones, which might occur, for instance, at a highly valued location. The PCS app prediction model drives a decision engine that can balance current and future opportunities against pending tasks to either sense, upload, or apply computation to the data.

This paper makes the following contributions:

- We propose to systematically exploit Smartphone App Opportunities for mobile crowdsourcing. Under this approach, predictable smartphone usage patterns – such as making calls or browsing the web – are leveraged to decrease mobile crowdsourcing's energy consumption.

- We develop an architecture and algorithms designed to maximize the benefit possible from unpredictable Smartphone App Opportunities. Specifically, our algorithms (1) accurately predict smartphone app usage, which in turn enables (2) local intelligent decisions by smartphones as to when to sample, compute, and upload sensor data.

- We evaluate our PCS prototype using (1) a large-scale trace of 1,320 smartphone users, (2) a separate field trial (21 days, 11 users), and (3) one end-to-end application case study using the PCS system. Our results show that PCS can collect more sensor data (between 0.7x and 3x) with the same energy budget when compared to a representative set of crowdsourcing benchmarks, while still performing the necessary computation and uploading tasks.

The remainder of the paper is organized as follows. §2 highlights the benefits of piggyback crowdsensing. In §3, we begin to describe PCS and present an overview of the PCS architecture. §4 continues by providing the specifics of the key algorithms in our design. Finally, in §5, we detail the implementation specifics of the PCS prototype. §6 presents experiments that evaluate both individual PCS components and compare overall PCS performance with alternative crowdsourcing approaches. §7 discusses the end-to-end performance of representative prototype crowdsourcing applications built with PCS. In §8, we acknowledge our study's limitations and outline our plans for future work. Related work is discussed in §9, and §10 concludes this paper.

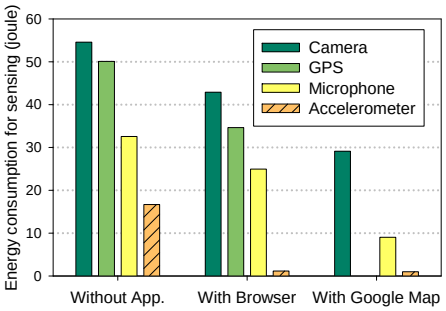

Figure 1: Energy consumption for sensing falls if performed when various smartphone applications are already being used by the user (i.e., piggybacked).

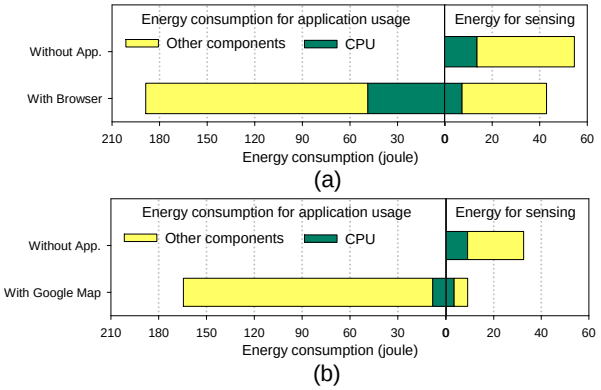

Figure 2: Energy consumption of (a) camera and (b) microphone sensing with app usage. The additional energy consumption required for sensing falls when smartphone apps are already being used by the user.

2. SMARTPHONE APP OPPORTUNITIES

In this section, we discuss the collection of mobile sensor data by *piggyback crowdsensing*. In particular, we highlight potential energy-savings by exploiting Smartphone App Opportunities – the opportunities presented by smartphone usage, such as when users place phone calls or browse the web.

Smartphone Sensing with Lower Energy. We begin with an experiment that measures the energy consumption of sampling various sensors (viz. camera, microphone, GPS, and accelerometer) opportunistically – at the same time smartphone applications are used in the foreground. In this experiment, and for the remaining experiments in this section, we report results using the Motorola Razr XT910 smartphone with energy measured using AppScope [42].

Figure 1 compares the energy cost of sampling each sensor under three scenarios. While the user is either, (1) browsing the web (WiFi connection), (2) using Google Maps (3G connection), or finally (3) when the phone is idle, and must be woken from a sleep state. Our results show that if the camera and microphone are sampled when the phone is in an idle state, rather than exploiting the opportunity when applications are used, sensing can cost as much as an additional 98% and 495%, respectively. In the case of the GPS, all the energy necessary to sample can be trivially saved if the application itself uses the GPS (as in the Google Maps

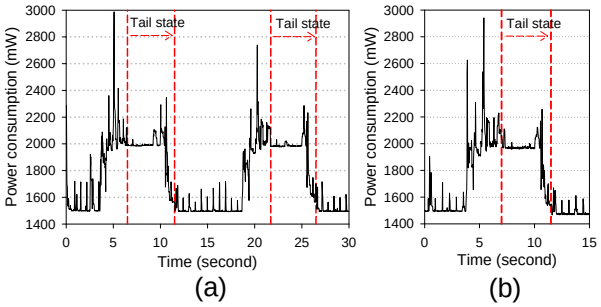

Figure 3: Energy consumption of sending (a) two 16KB data segments with a pause in-between and (b) a single 32KB segment (i.e., without any pause). Note the clear waste in energy in (a) compared to (b) due to the additional tail state energy.

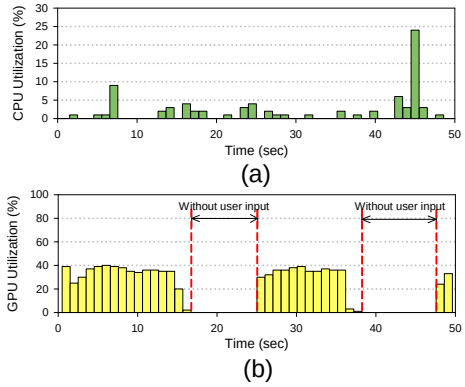

Figure 4: Utilization of (a) CPU and (b) GPU with use of Google Map.

case), but even when the user is simply web browsing a saving of around 34% can still be achieved.

Waking the phone from an idle state to collect sensor data requires a number of system components to be activated. The full CPU and related subsystems are needed for most smartphones to be able to sample a sensor – by themselves these components can consume between 200 mW and 600 mW. One key reason why we find piggybacking sensing with other apps can save energy is because much of this CPU overhead can be saved. Figure 2 shows the energy consumption of microphone and GPS sensing with and without app usage at the same time. Performing sensing while an app is in use requires 43% less CPU-related energy during the sampling operation. The processing required to capture sensor data is often fairly small and so can be performed by the CPU being awake a little longer or by time-slicing the computation with the primary user initiated workload. In the case of low-power consuming sensors (e.g., the accelerometer) piggybacking can have an even larger benefit. The cost of waking and using the CPU for sensing is largely the same across sensing modalities, even for low-cost sensors. As a result, often the actual sensor related cost of collecting sensor data can be relatively tiny, as shown in Figure 2(b). Because piggybacking largely lowers the CPU component of the energy needed to sense it is able to have a large overall larger impact on these relatively "cheap" sensors.

Piggyback Uploading with Lower Energy. In our next experiment, we demonstrate how data collection (i.e., uploading) can benefit from app piggybacking. Figure 3 presents a time-series of power consumption when uploading data independently or when batched with the uploading needs of another app by piggybacking. This figure shows the transmission of two 16KB data files separately (Figure 3(a)) as well as single transmission of 32KB (Figure 3(b)). The single transmission is caused by two apps transmitting data without a pause in-between. In this particular experiment, although the same amount of data is transmitted in both cases (by sending the data as two separate files) the energy cost is twice as high compared to the case of the piggyback case. The reason for this is that (1) the bandwidth of the wireless connection could cope with the faster data rate without increasing the upload duration; and, (2) the tail state during communication can be amortized across the two piggybacking transmissions. Thus, the piggyback uploading potentially may save significant energy if sensor data uploading occurs at a similar time of the uploading of other applications.

Piggyback Computation with Application Usage. Finally, we investigate the relationship between computation and app usage. Figure 4 presents a time-series of CPU/GPU utilization while Google Map is used. Our key observation is that during user initiated (e.g., GUI-based) apps the CPU is idle while the user pauses before next interacting. These results indicate: (1) piggybacking crowd-required computation with app usage may not degrade the user experience – if the app already has low CPU utilization; and, (2) piggyback sensing may potentially utilize slack time of the CPU/GPU if it is not often used while the user is not interacting with the screen.

3. PCS OVERVIEW

In this section, we introduce *Piggyback CrowdSensing* (PCS) by briefly describing its core components and overall architecture. PCS has been designed to effectively exploit opportunities to collect, compute on and upload sensor data presented by *Smartphone App Opportunities* – common smartphone user operations (e.g., placing calls, using applications). This approach enables PCS to lower the energy burden placed on users when participating in crowdsourcing applications.

Figure 5 shows the overall architecture of PCS . The majority of functionality resides on the smartphone, allowing it to independently make decisions as to when is the most energy efficient time to perform pending tasks required by crowd applications. We begin our description with the cloud-side components.

PCS Cloud Infrastructure. The primary function of the cloud is to offer building-block services for external crowdsourcing applications, in addition to storing data uploaded by PCS-enabled smartphones.

Crowdsourcing Application Support. Crowdsourcing applications use PCS via a set of external facing APIs, which offer sensor data collection and data processing operations. All applications must define a set of *crowdsourcing tasks*. Minimally, these tasks must specify the sensor, sampling duration, maximum upload latency (after which the sampled data is no long needed) and the phone-side data processing to be applied. In addition, optional specifications can be set that describe the desired spatial and temporal preferences (e.g., sample within an specified area or within a range of

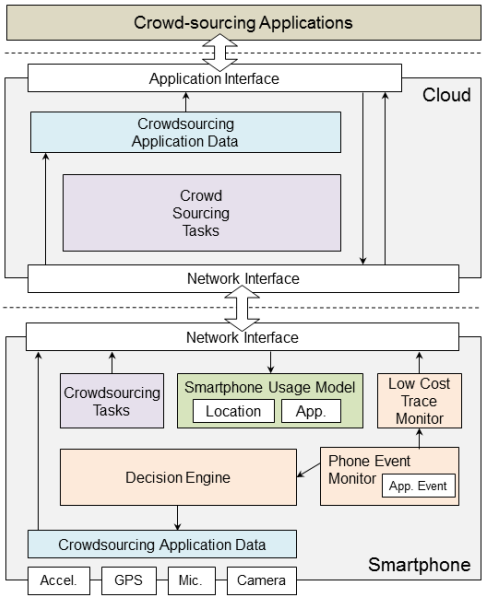

Figure 5: PCS Prototype Implementation

hours during the day) as well as the preferred upload latency. PCS meets these preferences on a best-effort basis depending on the availability of Smartphone App Opportunities and the battery budget determined by each user.

PCS Smartphone Software. Phone-side components are responsible for making intelligent sensor sampling, computation and uploading decisions that balance, for example, expected sample quality with the resulting energy cost.

Sensing Decision Engine. Each time an application is launched the Sensing Decision Engine (SDE) is invoked. The role of the SDE is to compare the opportunity to perform any pending crowdsensing tasks (viz. sensing, computation, uploading), presented by the (user) invoked application, against future opportunities represented by predicted patterns of smartphone app usage. Opportunities are evaluated differently depending on the type of crowdsensing task to be performed, with each type of task (e.g., sensing or computation) applying a different notion of utility and cost. For example, an opportunity to sense based on a particular app running on the foreground is evaluated with utility depending on an expected level of sample quality (app category specific, see §4.3 for more) and the sampling context (e.g., time and location) – and with cost based on the expected marginal energy required for the sample (again app category specific). In contrast, an opportunity to compute measures utility based on the expected negative impact on user experience – we use the expected amount of idle CPU time while this process runs as a proxy – the cost is again based on the marginal energy for the task. The goal of the SDE is to select those opportunities that will maximize utility – while still respecting a user-specified battery budget constraint (i.e., cost). If the current opportunity (i.e., application usage) is selected, then the operation begins immediately – otherwise it is delayed until the next time an app is invoked.

Smartphone App Usage Model. User-specific app usage is predicted using an online model (i.e., that learns incrementally) operating locally on the phone. We adopt predictive features previously demonstrated to be effective, for example: the prior app that is used, the time of day, day of week, location. The output of our app usage model is the probability of when each app will be used at different times of day. Some predictions are trivial when they are related to smartphone OS related processes which are deterministically scheduled. Location in our app usage model is based on a mobility model (described next) as directly estimating the location of the user when each app is used would consume too much energy.

Mobility Model. Mobility patterns for each user as modeled because (1) smartphone app usage often have location dependencies, and (2) crowdsourcing applications typically have spatial and/or temporal sensor sampling preferences. The mobility model is best-effort and relies on the location requests (GPS or WiFi) made by user initiated apps (or those initiated by the smartphone OS). These opportunistic location estimates are supplemented by location estimates requested by the mobility model itself, these are made when required from a small daily energy budget allocated to this component. As in the case of the app usage model, the mobility model is personal to each user with the training and execution of this model performed exclusively on the phone.

4. PCS ALGORITHMS

In what follows, we detail the core algorithms that collectively comprise the PCS architecture.

4.1 Smartphone App Usage Model

The goal of this model is to predict when certain apps will be invoked by each user. To achieve this we adopt an online formulation that becomes personalized for each user based on their particular usage patterns. Training occurs incrementally, each time apps are used.

Predictive App Usage Features. We adopt a series of features proposed in prior studies (e.g., [37, 19]) that predict future app usage. However, we exclude those features that are energy expensive to compute since app prediction must occur frequently.

Specifically, we compute features based on: location, time, phone state and the previous app used. The time of day when an app is used is quantized into four time intervals, each lasting six hours[1]. Similarly, the day of week is categorized into either weekend, or weekday. Location is based on a coarse estimated position of the user (one of a series of regions the user moves within) using the mobility model (described in §4.2). The previous app used is represented as an app identifier, only capturing the immediately previous app. Finally, a variety of phone states are used: vibrate mode, screen on, airplane mode – each of which are represented as binary indicator variables.

Online Boosting App Prediction Model. We learn patterns of smartphone app usage with an Online Boosted Naive Bayes Model [32]. This classifier design allows the phone to perform incremental training and inference with little overhead (see §6.2). Previously, batch versions of a naive bayes model using a superset of the features we used have been successful in modeling application usage [37]. In §6.2 we present results showing our approach has similar levels of prediction accuracy to [37]. However, PCS is agnostic to the precise model used to predict app usage behavior.

[1]$\{6am-12pm, 12pm-6pm, 6pm-12am, 12am-6am\}$

We use the specific formulation of an online boosting model provided in [32] that is designed to act as a online equivalent of AdaBoost.M1 [23]. A naive bayes model is used as the weak learner during the boosting process, with each bayes model trained being applied to the features described above. Each bayes model will assign a conditional probability to each possible smartphone app depending on the value of the single feature (e.g., time of day) it is trained on. Unlike many classification problems labeled training data is plentiful since features are labeled (i.e., annotated with the ground truth app) by system event logs. This enables our app prediction model to be constantly revised without manual intervention.

The output of the boosting process is a series of weak learners, h_1, h_2, .., h_m (single feature bayes models), that are based on weighted training examples (i.e., features computed against observations of apps being used.) Boosting incrementally trains these learners one after the other in response to errors in the training data. Like AdaBoost.M1, our online boosting algorithm will increase the weight of examples provided to the new weak learner (h_m) that were misclassified by the previous weak learner (h_{m-1}). However, because training is a streaming process the entire dataset can not be re-weighted and the new weak learner (h_m) trained. Instead the same effect is produced by maintaining weights ($\lambda_m^{correct}$ and $\lambda_m^{incorrect}$) for each weak learner that influence how the learner is revised (i.e., shifting the classifier decision boundary for a respective feature) each time it either incorrectly, or correctly classifies a new training example – in addition to updating ϵ_m, the error for the model.

Any time an app prediction is required the boosted model can be used, although accuracy will increase as it is exposed to greater amounts of data. Prediction is performed by applying,

$$h(x) = \text{argmax}_{y \in Y} \Sigma_{m=1}^{M} log \frac{1-\epsilon_m}{\epsilon_m} I(h_m(x) = y) \qquad (1)$$

where, y is any app in the set of all apps used (Y), ϵ_m is the error term for each weak learner (h_m) and x is the vector of computed features. When the SDE is invoked app predictions are made for the remainder of the day by applying this model. These predictions are used by the SDE to guide its decision process of matching crowdsensing tasks to Smartphone App Opportunities.

4.2 Mobility Model

We employ a fairly rudimentary model to predict user mobility. We acknowledge that this model may not provide enough accuracy for some scenarios; but the over-riding concern is to keep the energy overhead of this technique low. Importantly, just as is the case of our use of an online boosting learning algorithm to predicting app usage patterns, PCS is also agnostic to the specific mobility model used and could adopt more sophisticated methods to increase system performance.

Mobility is modeled under PCS as follows. The physical region in which PCS users move is tessellated into a grid of square tiles, the size of which is application-specific (e.g., an indoor deployment will use smaller tiles than an outdoor one). Similarly, time is also divided into discrete blocks, for example, we use a set of 8 blocks – four used by days during the week and four used by days in the weekend. Each day is partitioned along the same boundaries used to quantize application start times, as detailed in the previous subsection. Let each time interval block correspond to a completely separate grid of tessellation tiles. Further, let each user be represented by a separate data structure – containing time interval blocks and tessellation tiles. Populate this data structure based on the mobility pattern of each users by counting each time a user visits a specific tessellation tile, with respect to a specific block of time. Finally, to predict the location of a user for a particular time interval (i.e., within the set of 8 discrete blocks) simply select the tessellation tile most frequently visited for that block.

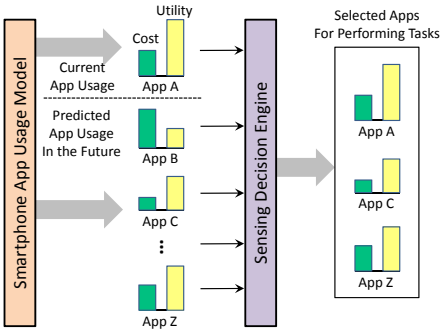

Figure 6: Sensing Decision Engine operation

4.3 Sensing Decision Engine

The role of the Sensing Decision Engine (SDE) is to evaluate current and future opportunities for the phone to perform tasks (viz. sense, compute, upload) requested by crowdsourcing applications using PCS . Figure 6 illustrates the decision process performed by the SDE each time an app is started. The current opportunity to perform a task (o_i), presented by initiated app (ap_i), is compared to upcoming opportunities ($\{o_j, ..., o_k\}$), based on the predicted app $\{ap_j, ..., ap_k\}$ determined by the Smartphone App Usage Model. Selection of an opportunity o_i relative to alternatives $\{o_j, ..., o_k\}$ is done by balancing expected utility (e.g., quality of the sensor sample) and the cost (e.g., energy consumed), which can vary, for example, depending on which app is being used or the task to be performed. Any decision made by the SDE can only ever be approximate since actual app usage in the future almost certainly will differ from the prediction. Whenever the currently executing app is included in this plan Sensing Decision Engine will then initiate the task by sampling, uploading or applying computation.

We describe in detail: (1) how utility and cost is determined for each type of crowdsensing task; and, (2) how the optimization is performed that determines how the decision engine behaves.

Utility. We begin by describing proof-of-concept utility functions, different definitions for provided for each of the three varieties of crowdsensing tasks. These can be easily replaced to meet the needs of crowdsourcing applications, or to improve PCS performance, without impacting any other components in the system.

We now describe each utility function definition in turn.

Sensing. PCS defines the utility of sensing not with respect to individual apps but instead categories of apps. Applications are assigned into 15 different categories (described in §6.1). This is due to the shear number of applications we perform this process using the Mechanical Turk [1].

Because utility functions for sensing are so application specific, PCS provides a utility function framework which later a crowdsourcing app can customize (or even completely replace). This function design assess the utility (u_i) of an opportunity (o_i) based on two criteria. First, the expected quality of the data sampled (u_q). Second, how closely contextual conditions for sampling match the requirements of the crowdsourcing app (u_l). These two factors are combined to determine the final utility for an opportunity. The default is a linear combination of u_q and u_l with weight assignments to each term that are configurable by the crowdsourcing application developer (i.e., $u_i = w_q u_{q_i} + w_l u_{l_i}$).

Our strawman definition for sensor data quality (u_q), that is used for the later evaluation, is for u_q to be based on accuracy achieved (e.g., $f1$-score) when performing a computational task (e.g., object-recognition, speech recognition – called p_j.) For example, the accuracy when performing object recognition using the smartphone camera while a user is talking on the phone. Formally,

$$u_q(p_j, c_k) = 2 \cdot \frac{precision(p_j, c_k) \cdot recall(p_j, c_k)}{precision(p_j, c_k) + recall(p_j, c_k)} \quad (2)$$

where $precision(p_j, c_k)$ and $recall(p_j, c_k)$ are the mean precision [14] and recall [14], found experimentally, when performing p_j while c_k is the foreground smartphone application. Note, our use of recall and precision in Equation 2 is simply the standard formation of $f1$-score, which combines both recall and precision metrics.

Our strawman definition of utility based on the contextual conditions for sampling (u_l) is highly app-specific and assumes uniform geo-graphic sampling is required. To do this we use the same data structure used to predict user mobility described previously in §4.1. We determine based on this data structure the current tessellation tile (tt_i) and time block (tb_i) for a user. The utility u_l for o_i then is simply the sum of all user visits to tt_i at tb_i divided by the sum of all visits across all tiles and blocks.

Computation. Computation attempts to quantify the utility (u_i) of a computation opportunity experimentally based on the amount of idle CPU time while the app is used. The intuition is to coarsely capture the availability of the CPU during a specific use of an app. Although this measure is not perfect it is very easy to acquire data for, such as, simply by PCS gathering smartphone OS statistics using app usage.

Uploading. Similar to the utility of computation, utility (u_i) for an uploading opportunity (o_i) attempts to capture how much negative user experience may occur if an uploading task is performed at the same time as the app in question. To capture this, we base our utility on the amount of data that is transferred on average during the app's operation. Apps that heavily use the network might be negatively effected by PCS also using available wireless bandwidth. In contrast, apps that do not use the network will have a high utility value, but this is balanced by their high cost since there is no opportunity for PCS to piggyback on any app generated traffic. Determining the utility value for each app is simple, PCS simply tracks how much data is transferred each time the app is used.

Cost. We define the cost (c_i) of a any crowdsensing task (o_i) in terms of the energy consumed. Just as we did with utility, cost is empirically determined; measurements are performed on an app *category* basis rather than individual categories to reduce the number of measurements required. Also similar to utility, three key parameters are used in the estimation of c_i: (1) the sensor (if needed); (2) the radio interface (if needed); and (3) the current app (if any). We experimentally build functions for c_i, just as we did for u_q, by performing a careful series of experiment that quantify the energy required to perform sensing, data transfer and data processing assuming different app categories are in use. As the energy consumed by an app and during sensing can vary in time, for simplicity, we use only an estimate of energy usage based on the mean power drawn and operation duration – as measured over multiple experiment runs.

Optimization. The Sensing Decision Engine formulates the planing process - i.e., the selection of Smartphone App-based Opportunities to either sample, compute or upload – as a stochastic knapsack problem with deadlines [33, 36] (SKP). Under SKP, items are selected from a larger collection to be placed inside a knapsack that is subject to a weight limit. Each item is associated with a weight and value. Certain items have a deadline associated before which they must be performed. The objective is to find the subset of items that will maximize the value of the knapsack, without exceeding the weight constraint or violating deadline constraints. However, item weights and values can be stochastic and so are uncertain at the time items are selected. We formulate the PCS planning problem under SKP as follows.

The item collection is determined by all pending crowdsensing tasks. This set of pending tasks contains: all sensing requests made by the external crowdsourcing applications, along with an upload and when necessary computation task that are generated each time a sensing task is complete. Uploading tasks may or may not have a deadline associated, depending on the specification of the crowdsourcing application (see §3). All computation tasks have a deadline set to be prior to the uploading deadline. During the SKP optimization an item is a pairing of a pending task and predicted app usage. In other words, an item is generated for each combination of task and predicted app during which the task could be performed (as predicted by the Smartphone App Usage Model). Item values are set by u_i, and item weights are set by c_i by applying the utility and cost definitions described above; consequently, both are parameterized based on the type of crowdsensing task considered (e.g., sensing or uploading), the sensor modality (if the task is to perform sensing) and the app category. Uncertainty of u_i and c_i are decided by the confidence the Smartphone App Usage Model has in its prediction of the app usage. For our boosting model, confidence is simply based on the probability assigned to each app based on the computed features. Knapsack capacity is set to the battery budget (b) allocated for crowdsourcing tasks by the user (available as a configuration option). Solutions to SKP typically attempt to maximize the expected value of the selected objects subject to a corresponding expected penalty for exceeding knapsack capacity. More formally, we must solve:

$$max \sum \beta^s r_i \; \mu_i \; \lambda_i \; x_i \; - \; d \cdot E[> b | x_i] \quad (3)$$
$$s.t. \; x_i \in \{0, 1\}$$

where, x_i indicates if a potential task/app pair i is selected or not, d is the penalty factor per unit of weight exceeding capacity, r_i is the reward per unit of weight for the task/app

pair i, μ_i is the expected weight of task/app pair i, λ_i is the expected value of the task/app pair i, all rewards for the task/app pair are subject to β^s a decay function that looses value the closer the time s approaches a specified deadline; and finally, $E[> b|x_i]$ is the expected weight above b.

A dynamic programming solution to this particular SKP is intractable. We apply a knap sack heuristic developed to cope with the same formulation. This heuristic is commonly applied to perishable items (e.g., fruit) which have rewards the decay over time while the item is waiting to be purchased [25]. This *Index-Knapsack* heuristic relies on a series of index equations that are generated by first representing the problem as a specific form of markov decision process (a weakly-coupled MDP). Index equations are then found by decomposing the MDP. [24, 25] shows that by computing this index across the set of items at each time step the index can proxy for the item values. An approximate solution can then be found through a conventional knap sack formulation, solvable with a text book dynamic programming solution.

The Sensing Decision Engine solves this optimization each time an app is initiated by the user. The index equations are determined off-line but once computed they can be stored locally on the smartphone allowing the optimization to be done locally. If the initiating app is included in the resulting selected opportunities then the chosen crowdsensing task is immediately scheduled. Otherwise all pending tasks are delayed in preference towards a future opportunity.

5. PCS IMPLEMENTATION

We conclude our description of PCS by a describing our prototype implementation. Figure 5 shows the core prototype components and highlights where in the architecture they reside (e.g., cloud or smartphone).

Smartphone Software. Our prototype is implemented for the Android smartphone platform, and spans three system services and one user application. Each system service performs one of the following functions (1) sensor sampling, (2) managing cloud communication, and (3) collecting application usage and mobility data – with minimal energy overhead. The single user application offers various privacy settings.

We now describe in additional detail each of these four components, in turn.

Sensor Sampling. We embed the SDE algorithms that governed sensor sampling within an event-activated service. By using the notifications from standard Android system events this PCS component can primarily sleep; consuming virtually no energy until an opportunity to sense presents itself. Once activated, if sensing occurs then the resulting data is stored for later delivery to the cloud.

Privacy Control. Given the sensitivity of the sensor data collected by PCS providing the user with control over their own data is paramount. All data is forced to reside on the smartphone for at least 24 hours, during which time users are able to delete any data they are uncomfortable being collected for processing. For this purpose our client incorporates a simple interface which allows users to view all images and play all audio clips collected, which they can then manually choose to delete. To further simplify this process users, with the press of a single button, can decide to purge all collected sensor data for the previous 1, 6 or 24 hours. Finally, as a preventative measure users can also select to pause data collection for an upcoming time interval (again 1, 6 or 24 hours) if they anticipate sensitive events occurring – or alternatively, users are able to inform the client to never collect data at a certain place (e.g., home, office).

Low-energy Smartphone Monitor. Traces of application usage and user mobility are required to build the Smartphone App Usage Model. Logging smartphone usage is simple and requires little energy, requiring only the recording of a variety of system events, including not only which application is invoked but events, such as phone calls being placed (which are also valuable opportunities to collect data). However, tracking user mobility requires GPS estimates that if not carefully controlled can consume large amounts of energy. PCS is conservative in building/maintaining its user mobility model, preferring instead to operate with a weak model that lowers crowdsensing performance (e.g., poor geographic coverage) rather than overly burden the phone battery. GPS estimates used for the mobility trace are sourced in two ways: (1) exploiting the GPS samples required, for example, by other user applications such as for maps and navigation; and, (2) a small amount of direct calls to the GPS. Direct GPS calls are made during a day up until a fixed daily budget of energy is met. PCS allocates this daily budget simply by attempting to accumulatively (over multiple days) sample location uniformly throughout the day, with a lower weight placed on regular sleeping hours (i.e., 12am - 7am). Later, in §6.4 we investigate the use of this approach, finding a surprising amount of GPS estimates can be gained from user-initiated applications, and that PCS can perform well even if this component uses minimal amounts of energy.

Cloud Communication. Most crowdsourcing applications targeted by PCS comfortably tolerate short delays in data collection (e.g., [9, 38, 34, 7, 6, 3, 26]). PCS exploits this flexibility to minimize the energy consumption (and cellular data plan cost) of cloud communication. The default policy for all communication is to delay until the smartphone is both line-powered and has a WiFi network available, which typically occur while the smartphone is recharging overnight. This simple, but effective heuristic, allows cloud networking to occur with little penalty to the phone battery. If the crowdsourcing application specifies a latency deadline for the delivery of data, PCS attempts to meet this requirement. However, cellular data is only used if this option is enabled by the user. Otherwise only WiFi is utilized. Regardless of network interface the battery budget provided the user is always respected.

During communication the cloud provides: (1) the assignment of apps to app categories and any app category sensor utility functions and (2) new/revised crowdsourcing tasks. In turn, the smartphone provides the cloud all data collected, and computation performed since the last communication.

Cloud Infrastructure. We build the PCS Cloud Infrastructure using the storage, computation and distributed system services offered by Windows Azure [10]. The primary function of this infrastructure is to facilitate the interaction between PCS and external crowdsourcing applications.

External Crowdsourcing Application Support. Through a series of standard RESTful APIs external applications can submit to PCS crowdsourcing tasks (briefly defined in §3) that specify sensor data sampling requirements (e.g., modal-

ity, sampling rate, etc.). Further, external applications can specify additional post processing to be performed to the data. Our prototype currently supporting various types of sensor data classification and interpretation (e.g., object recognition, sound volume) – we detail these supported sensor algorithms in §6.1. Once a crowdsourcing task is submitted external applications are able to use the PCS APIs to track the accumulation of data and retrieve it when required.

6. EVALUATION

In this section, we evaluate PCS and study its ability to perform energy-efficient crowd sourcing of mobile sensor data. Our key finding is that under a wide variety of crowdsourcing scenarios – each with their own particular combination of sensor data sampling, computation and collection requirements – PCS is able to significantly lower the energy consumed by the user's smartphone compared to a number of commonly implemented benchmark strategies.

6.1 Methodology

We briefly describe the key components and definitions in our experiment methodology.

Benchmarks. We compare PCS against three baseline strategies, which we now describe:

Periodic Sampling. In the absence of prediction `periodic` simply samples at a fixed sampling frequency. This frequency is set based on the fixed battery budget set for each experiment.

Context-driven. Many crowdsensing applications have a strong temporal and spatial requirement. The `context` baseline attempts to maintain uniform sampling across time and space, using the same mobility model employed by PCS (specified in §6.3).

Application-driven. The most simplistic approach to leveraging smartphone app opportunities is to greedily use each one whenever it occurs. We compare this approach, performed by `app-driven`, against PCS to quantify the benefit of the app prediction and decision optimization we propose.

Datasets. We use two datasets. First, we are given access to `AppJoy` – a dataset of smartphone usage patterns from the authors of [40]. This dataset is comprised of 1320 people and contains the app used, along with start time, coarse location and duration. The data was collected worldwide as part of a public Android application release. We pair this large-scale trace with `CrowdTest` – a dataset we collect ourselves as part of a 11 person, 21 day deployment to examine data quality. All users are provided with Android phones running PCS and a simple sampling app that implements `periodic` and `context`. All phone sensors are used (GPS, WiFI, accelerometer, microphone). To assess data quality we employ students from Yonsei university to manually inspect the data and assign ground-truth categories depending on the content. The PCS utility function during all experiments (unless otherwise stated) is set with the objective of uniform spatial coverage; utility ignores potential other factors, such as, time of day or the piggy-back app type.

Smartphone App Categories. Due to the diversity of smartphone applications it is impractical to profile the resources (e.g., energy) used by them all. Instead, we group app into categories which we list in Table 1, along with representative app examples that fit in each category.

Categories	Applications
Phone	GrooVe IP, AIVC, Google Voice, Voxer Walkie-Talkie PTT
SMS & Chat Clients	IM+, Go SMS, tablet talk, Handcent SMS, Kakao Talk
Audio & Music	Poweramp, PlayerPro Music, WavPlayer,Winamp
Browser	Dolphin browser, Firefox, Opera Mobile, Chrome
Video	YouTube, VPlayer, MX Player,DicePlayer, Plex, GTV Box
Email	Enhanced email, Gmail, Yahoo! Mail, Hotmail
Reading / Books	Kindle, Bible, Google Play Books, iQuran, Blio eBooks
Photo viewing Apps	Picasa Mobile, InstaPics, FX Photo Editor, Adobe Photoshop
Games	Angry birds, fruit ninja, doodle jump, minecraft
System	Task Killer, JuiceDefender, Tasker, Easy Battery Saver
Map & Navigation	Google Maps, Navigon, GPS Phone Tracker, Street View
Social Networking	Facebook, Twitter, FourSquare, Google+, LinkedIn
Video + Microphone	Skype, Viber, ooVoo, T-Mobile Video Chat,
Camera	Camera Zoom, Paper Camera, Pano, Vignette, HDR Camera
Others	Root Explorer, Flashlight, Speedtest, Go Launcher EX

Table 1: Categories of Smartphone Apps used in PCS evaluation.

To categorize all of the apps used in our 1320 person application trace we use the Mechanical Turk [1]. Our task requests users to categorize the application and provide a link to a webpage that assists their decision. We have each app categorized by three people, and then determine the final category based on majority voting.

We empirically profile the energy consumption of five applications from each category (see §6.2) to determine the variance of energy consumption when PCS exploits apps within categories. We find the average within category range of energy consumption is ±13%. This profiling also is used for our actual cost category functions used by PCS in our experiments.

Sensor Data Quality Categories. To provide a measure of sensor data quality under PCS and alternative strategies, we categorize collected audio clips into three categories: (1) human voice, (2) background sounds, and (3) silence. Categories are assigned to audio clips based on the volume and the output of the speech recognizer, a subset of sounds are manually checked after categorization to verify the assignment is valid. In cases where an audio clip can be assigned to multiple categories the "dominant" sound of the clip is attempted to be assessed and used for classification. Human-voice audio clips contain one or more people speaking. We consider an audio clip as silence if the file includes virtually no words and little noise (i.e., less than -70dB in volume). Finally, audio clips are categorized as a background sound type if they contain few words but are loud enough not to be in the silent category.

6.2 Micro-benchmarks

We begin by performing two brief micro-benchmarks that investigate the quality of sampling the microphone under PCS and the `periodic` and `context` baselines. In addition, we also profile the impact of PCS on smartphone resources to measure any negative impact to the user experience (e.g., when the user runs other apps while PCS is operating).

Sensor Data Quality. We perform a preliminary investigation using the `CrowdTest` dataset to see if exploiting Smartphone App Opportunities causes negative changes in sensor quality. Examining data quality is difficult without a specific scenario, as a consequence, in §7 we show the performance of important single scenario – a complete system that constructs an indoor WiFi fingerprint database. As a result, we differ evaluation of accelerometer and WiFi quality and instead focus on audio.

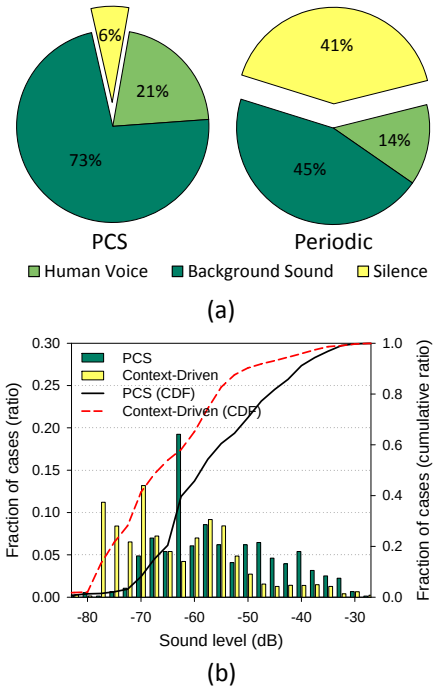

Figure 7: Manual examination of audio clips captured under PCS in comparison to `context` and `periodic` baselines.

Benchmark Tools	Benchmark Features	Camera Sensing	Mic. Sensing	GPS Sensing
NenaMark2	FPS*	7.9%	6.3%	0.0%
NeoCore	FPS	3.5%	4.5%	0.3%
AnTuTu	RAM	12.8%	5.4%	-0.3%
	CPU	13.9%	4.8%	-0.8%
	2D Rendering	50.5%	9.0%	0.3%
	3D Rendering	15.5%	4.9%	-2.0%
Average		17.4%	5.8%	-0.4%

*FPS: frame per second

Table 2: Impact on smartphone app performance due to PCS operating in the background.

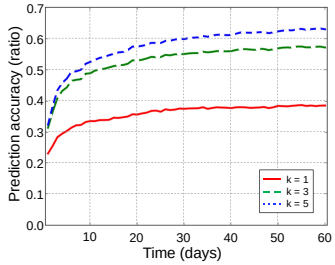

Figure 8: Average accuracy increases for the PCS app prediction model as time passes.

Figure 7(a) presents a breakdown of collected audio clips under PCS and `periodic` sampling. From this figure it appears that sampling using app opportunities does not harm sensor data quality. In fact, it appears that audio clip quality improves at least for the categories we adopted. A large fraction of audio clips collected during `periodic` are silence. In contrast, the fraction of human voice and background sounds captured by application-driven sampling (PCS) is much higher. However, we do make strong claims about this comparison other that we do not find any evidence PCS lowers the quality of captured audio data.

Figure 7(b) provides a coarse characterization of the audio clips captured under PCS and `context` sampling. By comparing the CDFs of each strategy we can observe audio clips when the apps are active and louder. Again PCS does not appear to hurt sensor data quality. Audio clips captured by `context` (which is location and time driven) are often taken while the smartphone remains in pockets and bags – limiting what can be overhead. Even those clips captured by periodic sampling that are relatively loud should not be assumed to be necessarily of higher quality. We find (by listening) that such audio clips are often dominated by the sound of the microphone rubbing against clothing material.

PCS Overhead. For PCS to be practical it should not overtly impact the usability of the smartphone. To investigate this issue, we execute a series of smartphone application benchmarks while PCS samples various sensors in the background as well as other typical PCS operations (e.g., revising app prediction models). For this experiment we use the Motorola Razr XT910 smartphone and three well known Android benchmarks: *NenaMark2* [4], *NeoCore* [5] and *AnTuTu* [2]. NenaMark2 measures graphics performance and stresses in particular the GPU, performance is measured in frames-per-second. NeoCore is another popular graphics benchmark, also measuring performance by frames-per-second. AnTuTu performs a range of benchmarks, including memory, cpu, storage, and both 2D and 3D graphics; the unit of measurement varies between each benchmark, with each a different composite index.

From Table 2 we see PCS while sampling the GPS has almost no effect, this is because sensing is almost entirely performed by the GPS unit without meaningful assistance from other components. Similarly, while PCS samples the microphone there is very little effect with the average impact on benchmark results being at most 9%, and an overall average of 5%. Finally, we test PCS while accessing the camera. Clearly this has the largest impact, on average a 17% slowdown occurs across the board. The most effected index is 2D rendering, which is lowered by 50%.

6.3 Smartphone App Usage Model

Our next experiments examines our Smartphone App Usage Model, focusing on the performance of predicting upcoming app usage. These experiments are based on the `AppJoy` dataset. Some features described in §4.1 are unavailable in this dataset (e.g., phone context) and so are not used (although they are used in our experiments in §7). We start by examining overall accuracy throughout the user population (1320 people) before concluding by considering the relationship between accuracy and the deployment time (given we use an online learner).

Accuracy within User Population. We examine the prediction accuracy across the entire user population. Figure 9 shows a CDF of per-user prediction accuracy. This figure is drawn assuming the model has been trained for 30 days. We report three accuracy metrics based on the correct app being within: the first prediction made (i.e., $k = 1$); first two model predictions made (i.e., $k = 2$); and, first three

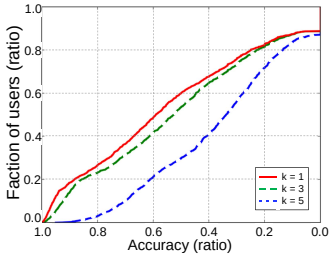

Figure 9: CDF of PCS app prediction accuracy across the user population.

model predictions made (i.e., $k = 3$). This is a standard metric for app prediction (see [37]). We find that our reported accuracy levels for these three k values are comparable with recent work that uses a batch prediction model (compared to our streaming prediction model) and a *superset* of the features we select (we exclude commonly used features that drain significant energy). For example, [37] reports 63% for $k = 3$ within its dataset of 111 users (compared to our 1320).

Time to Train Online Model. Figure 8 shows the average prediction accuracy for our Smartphone App Usage Model as time elapses and the online model as an opportunity to become personalized to the user's app behavior. We see from this figure after 20 days the model is approaching its maximum accuracy levels. By 30 days the model has approximately reached an accuracy plateau.

6.4 PCS Performance

In the following experiments we investigate the performance benefits of PCS which we measure largely by increased energy efficiency. Our findings show for the same energy budget PCS is able to collect a larger amount of data, compared to our suite of benchmarks.

More Data for Less Energy. Figure 10 presents six crowdsourcing application scenarios that we evaluate under PCS and compare to three different baselines (baselines described in §6.1). All of these experiments use the `AppJoy` dataset to perform a trace based analysis of how much additional sensor samples would be collected under each scenario. To make the analysis realistic each event in the scenario (e.g., data computation, sensor sampling, uploading) is performed with five representative apps from each of the 15 app categories – as well as five times without any app activity. During these benchmarking experiments energy is estimated using AppScope [42]. We implement all the computation required locally on the phone with the exception of the speech recognition module that is implemented as service external to PCS (although feature extraction is performed on the phone.) In the speech recognition case we setup a WiFi connection and measure the network overhead of using a cloud solution.

Each figure shows the raw count of *out performance* – in terms of additional sensor samples collected beyond each baseline – during the replay of the `AppJoy` dataset. These experiments use a Motorola Razr XT910 smartphone. We assume that either 1% or 2% of the standard battery is allocated by the user to crowdsourcing. All figures report results assuming these two battery budgets.

GPS/Accel Pothole Detection. In this first scenario we consider pothole detection, similar to the system described in [21]. Smartphones are assumed to collect GPS and accelerometer data. Furthermore, we implement similar pothole detection techniques as described in the paper, implemented directly on the phone. These computations are scheduled by PCS at times when other apps are being used to save energy. The results of these feature extraction routines are sent to the PCS cloud (assuming WiFi). From Figure 10(a) we find that PCS dominates the simple `periodic` baseline (2.5x to 3x) and provides significant gains over the `context` and `app-driven` (0.7x to 1.0x).

GPS/WiFi/Accel Train Mobility Classifier. Our next scenario targets the crowdsourcing of training data for a transportation mode classifier. We define a crowdsourcing application to PCS that will collect WiFi, GPS and accelerometer data and perform local extraction of the transportation mode features described in [35]. These features are then transmitted to the PCS cloud (again via WiFi). Figure 10(b) shows PCS causes gains of between 1x to 1.5x over the nearest two baselines (`context` and `app-driven`). The reason why this out performance is marginally better than the prior scenario appears to be the additional sensor data being collected that then provides more opportunities for PCS to exploit its app opportunity based advantage over the baselines.

Accel/Train Activity Classifier. The following scenario is very similar to the previous one. Here we assume the same scenario but consider the activity recognition domain. We collect as part of the crowdsourcing scenario accelerometer data and assume the smartphone will extract the same domain features detailed in [31]. All other details from the prior scenarios apply here. In Figure 10(c), we observe PCS performance is strong against both `periodic` and `context`.

Microphone/Speech Recognition. In this scenario, we assume audio data is collected and speech recognition is applied. Recently in [18] these two operations were used as a building block to perform location analysis. Figure 10(d) demonstrates that PCS can collect between 3x to 2x more samples than both baselines. This scenario uses the open source CMU Sphinx recognizer [8] for this task. Mel-frequency cepstral coefficients [22] (MFCCs) are extracted on the phone and then sent to a remote module over WiFi for processing.

Camera/No Computation. In this scenario, presented in Figure 10(e), we consider just the camera being triggered using piggybacking. We examine this scenario without computation. Opportunistic image capture has been proposed in work such as [18] and others. Unlike the prior scenario the gains are much lower since camera piggybacking itself does not have as large gains as the microphone (see §2) and this scenario lacks computation to further demonstrate the power of piggybacking. Although PCS is capable of performing this action we believe it is too privacy invasive to be practical.

GPS/No Computation. Our final scenario again does not consider computation but simply attempts to sample the GPS. This is a building block activity for many transportation and traffic related crowdsourcing scenarios. For example, the commercial crowdsourcing company Waze [9] requests this from its users so that it may learn traffic patterns and driving routes. From Figure 10(f), we find that PCS has modest gains over `app-driven` and context but significant gains over `periodic`.

Because AppScope is able to produce a detailed energy breakdown we are able to examine how PCS uses its energy

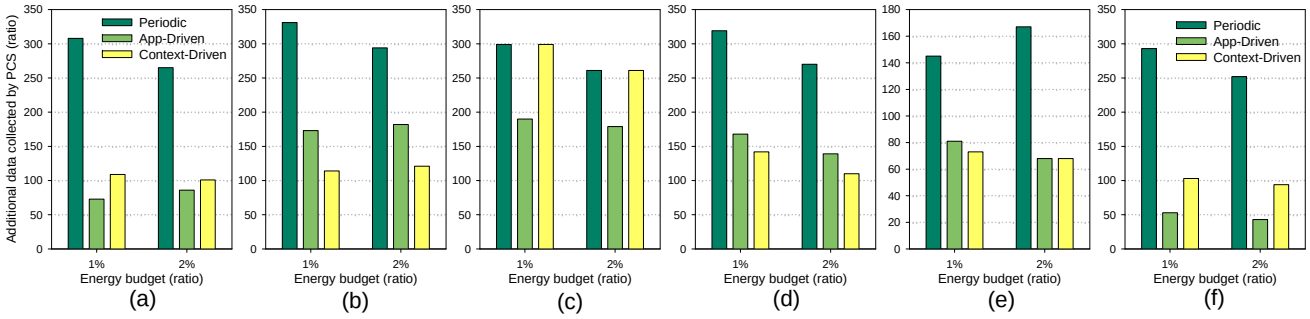

Figure 10: Marginal increase in the quantity of collected sensor data when comparing PCS to three baseline strategies. Six crowdsourcing scenarios are shown: (a) Pothole Detection; (b) Training a Transportation Mode Classifier; (c) Training an Activity Classifier; (d) Word Recognition from background audio; (e) Camera sampling; and finally, (f) GPS sampling.

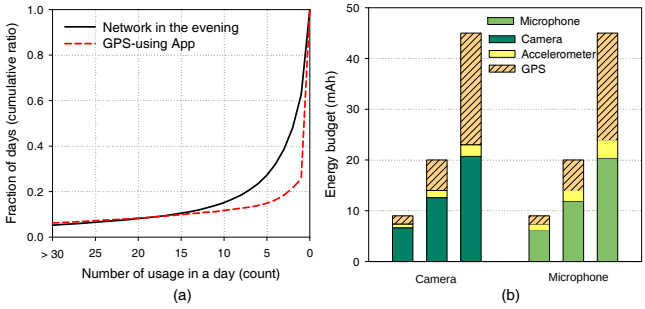

Figure 11: (a) presents daily opportunities to upload data in the evening and gather extremely low-cost GPS reading based on application that already use the GPS sensor. (b) breaks down how PCS uses its energy budget during opportunistic operations.

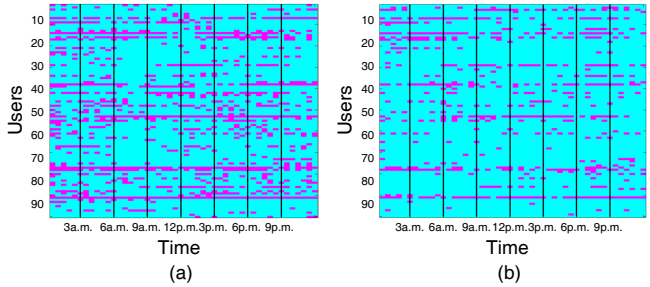

Figure 12: Temporal patterns of sampling for representative users. Two visualizations are associated with collecting audio, one during the (a) weekend, the other during (b) weekday.

budget. Figure11(b) breaks down the average daily energy budget allocation for the camera and microphone (which also uses speech recognition) scenarios above. You can see that the majority of energy is consumed by the primary sensor. Most of the remaining budget each day is used to perform additional GPS estimates to support building an adequate mobility model.

Personalized Sampling Patterns. Understanding why PCS can produce results such as the scenarios described above is just as important as the out performance itself. To gain a greater understanding, we visualize the sampling decisions made by the PCS smartphone software for 100 people in the `AppJoy` dataset. Results in Figure 12 show all users have a unique sampling strategy dynamically determined based on predicted opportunities to sample. These figures also present interesting aggregate insights. For example, Figure12(b) illustrates data collection occurring on the weekend, which is noticeably sparser than the weekdays (see Figure12(a)). The reason for this is due to the frequency of app opportunities, we find fewer apps are used in the weekend than the weekdays.

Uploading and Localization Opportunities. Although we enable crowdsourcing applications to request data to be delivered with respect to a deadline, the majority of existing crowdsourcing apps can tolerate latency of a day or more. For this reason we set the default strategy for PCS to upload at night and exploit the times when the phone is charging and has WiFi available. Figure11(a) looks at the `AppJoy` dataset and presents the fraction of days when users have WiFi available during the evening while recharging. We are able to extract this from the system apps that we find in our trace. This figure shows for around 60% of the days, users have 2 or more times per evening when their phone is recharging and WiFi is available.

Similarly, we attempt to keep the overhead of our mobility model low by exploiting location samples requested by user driven smartphone apps. Figure11(a) also presents the frequency that users invoke an app that makes a location request to the smartphone OS. In our data trace we find that for around 80% of the days in our trace users use around 3 GPS requiring applications. These "free" GPS samples accumulate overtime to help build our mobility model.

7. CASE STUDY: INDOOR LOCALIZATION WIFI FINGERPRINT DATABASE

In the following section, we perform an end-to-end application case study in which PCS is used to construct a WiFi-based indoor localization fingerprint database. Importantly, this case study allows us to examine the performance of PCS within a well understood workload (i.e., tasks including WiFi scanning and computation, such as, step-counting) that has clear performance metrics (primarily, the accuracy of localization). During a 25-day 15-user experiment, we find PCS is able to construct a fingerprint database with signifi-

cantly lower amounts of energy consumed on the end-user's device relative to a conventional crowdsourcing approach. This gain in energy efficiency is achieved with only a modest reduction in localization accuracy.

Implementation. A WiFi fingerprint database contains information collected from WiFi scans performed at a variety of locations within the deployment area. Each database record includes both the scan information and the ground-truth location where the scan was performed. This database can be used to estimate the location of a user by comparing the similarity of WiFi scan information collected at an unknown user location against database records. For this case study, we adopt the WiFi fingerprinting techniques of RADAR [12].

Because PCS operates opportunistically (i.e., without the user being in the loop) the key challenge in using PCS to construct an fingerprint database is that the ground-truth location of WiFi scans are unknown. Typically, users are requested to provides this information manually by indicating their approximate position on a floor-plan. Instead, we estimate ground-truth using inertial sensor data and rely on the availability of the building floor-plan. This requires the use of previously developed techniques including: turn detection, floor-plan turn association and step distance estimation; we implement these techniques based on the description provided in [29]. Turn detection uses the accelerometer and magnetometer to recognize when a user is turning while walking inside the building, such as when they approach a sharp corner of a corridor. Turn association enables detected turns to be tied to specific corridor corners within the building floor-plan – this is done by considering the sequences of user turns and the distances (as estimated by the step counter) between turns. The ground-truth position of users when WiFi scans are performed can be estimated based on the location of specific corners they encounter (provided by the building map). By using step counting for a limited number of steps before and after a recognized corner turn, additional ground-truth positions can be recognized.

To construct the WiFi fingerprint database, PCS is configured and setup as follows. Accelerometer, magnetometer and WiFi sensor data are collected locally on smartphones based on app opportunities. This data is delivered to the cloud infrastructure via the APIs described in §5. The tolerable delay for data collection is high, so data is only delivered from smartphones to the cloud using the default policy of only transmitting when the phone is line-powered and WiFi is available. Ground-truth of collected WiFi scans is performed post-facto in the cloud using the location specific techniques detailed above (e.g., step counting etc.). Records for the fingerprint database are formed based on the estimated ground-truth location and the WiFi scans performed at that position.

Methodology. We conduct a 25-day 15-user experiment within a single floor of a typical office building. Figure 13 shows the floor-plan of the deployment region, and denotes the position of WiFi access points. The majority of the working space of this floor is open plan and contains only a few offices.

Each participant carries an Android phone during the experiment with PCS installed. Participants are requested to use the phone as their primary smartphone device so that their app usage behavior during the experiment remains relatively natural. The Smartphone App Usage Model is initially trained using the `AppJoy` dataset but the model of each participant will become personalized during the experiment based on their app usage. To estimate the energy used by PCS we again use AppScope [42].

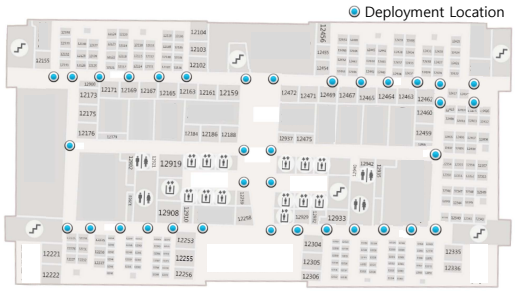

Figure 13: Floor-plan of the case study deployment area.

At the conclusion of the experiment, a second baseline WiFi fingerprint database is constructed using a conventional surveying approach that could also be crowdsourced. A small group of users collect WiFi scans using their smartphones at a list of specified locations (detailed on a provided floor-plan) that cover the floor using a grid layout. Users indicate which of the specified locations they have selected using a smartphone app and a WiFi scan occurs. The energy consumed when performing this surveying using the smartphone (excluding GUI related energy) is estimated again using AppScope. However, we do not quantify the amount of additional user burden that occurs due to the user being in the loop – for example, when they manually provide their position. The size (i.e., number of records) of both the fingerprint databases is the same – although the positions where PCS collects data is not as systematic as the surveying approach and is impacted by user mobility and the limited locations where ground-truth can be estimated (e.g., certain corridor corners).

For the purposes of comparing the location accuracy of each fingerprint database, a test dataset of WiFi scans is collected that systematically covers the entire floor plan. For these measurements, precise location ground-truth is determined carefully using a laser range-finger that provides the distance to nearby walls. We use the RADAR localization algorithms with each fingerprint database to compare location accuracy for this test set.

Results. Figure 14 presents the trade-off of energy and localization accuracy under PCS and compared to the benchmark surveying approach. We examine this trade-off by replaying PCS logs collected during the experiment through an off-line version of the PCS decision engine (SDE). To produce this figure we assume PCS is provided with a variety of energy budgets. From this figure, we find under different budgets different fingerprint database records are collected and as a result location error is often impacted. Each data point shown on this figure indicates a possible operating point of PCS at a certain energy budget. However, in practice PCS will only operate at one of the optimal accuracy/energy operating points along the two pareto frontiers (the two solid curves) shown in the figure.

Two location estimate error metrics are shown in Figure 14; one reports average localization error and the other reports the error for 95% of all locations in the test dataset.

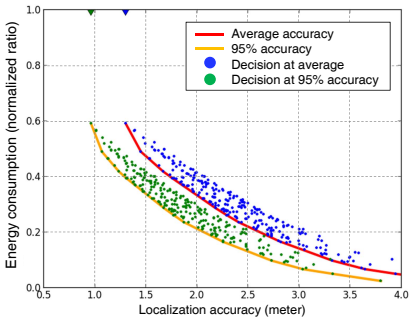

Figure 14: On average, PCS is less accurate than the conventional approach to constructing a WiFi fingerprint database. However, PCS also uses noticeably less energy.

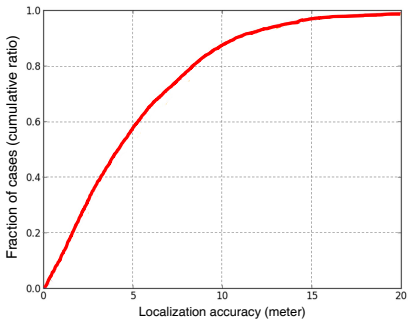

Figure 15: CDF of location accuracy under PCS.

Energy (shown on the y-axis of the figure) is normalized with respect to the total energy consumed by the benchmark scheme. In other words, 0.5 on this scale represents only half of the energy we use when constructing the benchmark fingerprint database. The two data points sitting at the top of the y-axis (at the 1.0 level) represent the performance of the benchmark surveying approach under both location error metrics. In Figure 14, we can observe many operating points where the energy consumed is less than a third of the benchmark. At this specific level of energy savings (0.3) the cost is approximately a 0.5 meter increase in location error (average) relative to the benchmark performance.

Figure 15 shows a CDF across all test location positions we collect – but for one single PCS operating point. Specifically, this figure assumes an energy budget of 50% of the energy of the benchmark technique (i.e., 0.5 on the y-axis in Figure 14). We find that, for example, error in localization is 5 meters or less for 60% of test locations.

8. DISCUSSION

Our results show PCS to be a promising direction towards energy-efficient mobile crowdsourcing. However, these same results also highlight limitations of PCS that we plan to address in future work.

We have specified strawman utility and cost functions. Given the application-specific nature of these functions our expectation is that developers of crowdsourcing applications would likely specify their own for most applications. We believe that the remaining components of PCS would operate well even with significantly different utility functions. Similarly, assessing data quality is a highly application specific task. In our evaluation, to examine data quality we provide a range of often used types of sensor data processing, however clearly there are many others. We acknowledge this may not hold for all possible uses of crowdsourced data.

In an effort to keep energy costs low we offer only coarse mobility modeling. As a result, PCS only coarsely knows where the user is when sampling occurs meaning that samples may not be taken with good geographic spread (if this is what a crowdsourcing application specifies). The problem is estimating location, prior to sensing, is prohibitively costly to approach differently without lifting the energy cost significantly. In the current design a user can address this by increasing the energy budget assigned to the mobility modeling component. However, we also plan to examine this problem as a future work item.

Privacy is key facet of crowdsensing, which we do not completely address in this work. Nevertheless, we do treat privacy seriously and provide complete and easy to use privacy controls on the PCS smartphone client (see §5). Clearly, this complex issue will require careful study before a system like PCS could ever be widely deployed to the public. Towards addressing privacy concerns we will investigate the use of automatically obfuscating collected data, for example, masking faces and certain words in collected data.

The benefits of piggyback crowdsensing do not extend to all forms of crowdsourcing. For example, many forms of crowdsourcing exploit *human computation* – where people do certain tasks machines are unable to perform effectively. In such cases PCS does not offer any benefit, except in those cases where the system is a hybrid, involving both sensing and human computation (e.g., [41]). Another case where crowdsourcing may not strongly benefit from PCS is when data is required at times when users rarely use applications. For example, if an application must sample data only from drivers while they are driving their car. Finally, many crowdsourcing scenarios require the tracking or search for moving items (e.g., cars, people) in a city; PCS is not suited to such dynamic tasks, and is much better equipped to sample based on simple static geographic or temporal constraints.

9. RELATED WORK

PCS has relevance to a number of research areas, in this section we survey work most closely related to our own.

Crowdsourcing. The power of crowdsourcing has been appreciated for a number years [39]. The crowdsourcing of mobile sensor data, in particular, is becoming an active area for both research communities and startups. Various systems are being built that depend on large-scale mobile sensor data gathered by the public. Such systems are being applied to problems in city congestion [13], localization [6, 3], noise pollution [34], or even optimizing traffic control systems [26].

Smartphone Sensing. Complementary to the rise of crowdsourcing has been the steadily increasing interest in smartphone sensing (e.g., [16] [15] [11] [28]). Recent smartphone systems investigate a variety crowdsourcing applications, such as Ear-phone [34], which collects audio samples from mobile phones to construct a noise map. These systems are also beginning to more tightly couple phone sensing and crowdsourcing. For example, [41] intelligently combines the use of sensing, image classification and human intelligence

to produce a system that reaches new operating points of accuracy and system performance (e.g., latency).

Opportunistic Sensing. Our approach to the collection of sensor data is opportunistic in nature; and related to the more general concept of *opportunistic sensing* [20, 16], which proposes to collectively leverage sensors in consumer devices to form large-scale sensor networks. However, PCS proposes new techniques to address challenges not considered by opportunistic sensing, for instance, lowering energy consumption by "piggybacking" sensing in combination with smartphone app use.

Sensor Networks. Finally, PCS shares similarities with the substantial amount of research that explores optimal scheduling and sensor placement in sensor networks (e.g., [17], [30]). Like PCS this body of work considers selecting opportunities to sample, has strong consideration for energy budgets and how these factors relate to observing larger phenomena. However, the platform, scenarios and related assumptions differ dramatically. Effective solutions for both domains, consequently, are quite different – still, as part of future work we anticipate investigating some of the optimization formulations used in this area to consider how we might improve our selection and scheduling process.

10. CONCLUSION

In this paper we have presented PCS, a system for crowdsourcing mobile sensor data that is designed to intelligently exploit opportunities to sense, compute and upload – at a low energy cost – presented by everyday phone app usage (i.e., smartphone app opportunities).

To evaluate PCS we have performed a comprehensive set of mobile crowdsourcing experiments. We used controlled benchmarks to better understand energy and data quality trade-offs of the piggyback approach. Based on a large-scale trace of smartphone app usage, we compared the performance of PCS with a number of representative baselines under a range of crowd system scenarios. To test the end-to-end performance of PCS, we built and evaluated a crowdsourcing application that constructs indoor WiFi fingerprint databases. Collectively, our findings validate the design of PCS and show it is able to outperform existing approaches to collecting mobile sensor data at scale in an energy-efficient manner.

11. REFERENCES

[1] Amazon mechanical turk. `http://mturk.com`.

[2] AnTuTu. `http://www.antutulabs.com/AnTuTu-Benchmark`.

[3] Apple iOS Location SERVICE. `support.apple.com/kb/HT4995`.

[4] NenaMark. `http://nena.se/nenamark`.

[5] NeoCore. `http://play.google.com/store/ /apps/details?id=com.qualcomm.qx.neocore`

[6] Skyhook wireless. `http://www.skyhookwireless.com/`.

[7] Stringfly. `www.stringfly.com/`.

[8] CMU Sphinx Speech Recognition Engine. `http://cmusphinx.sourceforge.net/`.

[9] Waze. `http://www.waze.com/`.

[10] Windows azure. `http://www.windowsazure.com/en-us/`.

[11] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, J. Reich. Mobiscopes for human spaces. *IEEE Pervasive Computing*, 6(2):20–29, 2007.

[12] P. Bahl, V. N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In INFOCOM '00.

[13] R. K. Balan, K. X. Nguyen, L. Jiang. Real-time trip information service for a large taxi fleet. In MobiSys '11.

[14] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, August 2006.

[15] J. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, M. B. Srivastava. Participatory sensing. In WSW '06.

[16] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson. People-centric urban sensing. In WICON '06.

[17] Q. Cao, T. Abdelzaher, T. He, J. Stankovic. Towards optimal sleep scheduling in sensor networks for rare-event detection. In IPSN '05.

[18] Y. Chon, N. D. Lane, F. Li, H. Cha, F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In UbiComp '12.

[19] T.-M.-T. Do, D. Gatica-Perez. By their apps you shall understand them: mining large-scale patterns of mobile phone usage. In MUM '10.

[20] S. B. Eisenman, N. D. Lane, A. T. Campbell. Techniques for improving opportunistic sensor networking performance. In DCOSS '08.

[21] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, H. Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. In MobiSys '08.

[22] Z. Fang, Z. Guoliang, S. Zhanjiang. Comparison of different implementations of mfcc. *J. Comput. Sci. Technol.*, 16(6):582–589, 2001.

[23] Y. Freund, R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In EuroCOLT '95.

[24] P. Jacko. *Dynamic Priority Allocation in Restless Bandit Models*. Lambert Academic Publishing, 2010.

[25] P. Jacko, J. Nino-Mora. Time-constrained restless bandits and the knapsack problem for perishable items. *Electronic Notes in Discrete Mathematics*, 28:145–152, 2007.

[26] E. Koukoumidis, L.-S. Peh, M. R. Martonosi. Signalguru: leveraging mobile phones for collaborative traffic signal schedule advisory. In MobiSys '11.

[27] A. Krause, E. Horvitz, A. Kansal, F. Zhao. Toward community sensing. In IPSN '08.

[28] N. D. Lane, M. Mohammod, M. Lin, X. Yang, H. Lu, S. Ali, A. Doryab, E. Berke, T. Choudhury, A. Campbell. BeWell: A Smartphone Application to Monitor, Model and Promote Wellbeing. In *PervasiveHealth '11*.

[29] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, F. Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In UbiComp '12.

[30] F. Lin, P. Chiu. A near-optimal sensor placement algorithm to achieve complete coverage-discrimination in sensor networks. *Communications Letters, IEEE*, 9(1):43–45, 2005.

[31] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, A. T. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In SenSys '10.

[32] N. C. Oza, S. Russell. Online bagging and boosting. In *In Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

[33] J. Papastavrou, S. Rajagopalan, A. J. Kleywegt. The dynamic and stochastic knapsack problem with deadlines. *Operations Research*, 42:1706–1718, 1996.

[34] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, W. Hu. Ear-phone: an end-to-end participatory urban noise mapping system. In IPSN '10.

[35] S. Reddy, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Determining transportation mode on mobile phones. In ISWC '08.

[36] K. Ross, D. Tsang. The stochastic knapsack problem. *Communications, IEEE Transactions on*, 37(7):740–747, 1989.

[37] C. Shin, J.-H. Hong, A. K. Dey. Understanding and prediction of mobile application usage for smart phones. In UbiComp '12.

[38] A. Thiagarajan, L. R. Sivalingam, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, H. Balakrishnan. VTrack: Accurate, Energy-Aware Traffic Delay Estimation Using Mobile Phones. In Sensys '09.

[39] L. von Ahn, B. Maurer, C. Mcmillen, D. Abraham, M. Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, pages 1160379+, August 2008.

[40] B. Yan, G. Chen. Appjoy: personalized mobile application discovery. In MobiSys '11.

[41] T. Yan, V. Kumar, D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In MobiSys '10.

[42] C. Yoon, D. Kim, W. Jung, C. Kang, H. Cha. Appscope: application energy metering framework for android smartphones using kernel activity monitoring. In ATC'12.