Crowdsensing Under (Soft) Control

John P. Rula
EECS
Northwestern University
Evanston, Illinois 60201
john.rula@eecs.northwestern.edu

Fabián E. Bustamante
EECS
Northwestern University
Evanston, Illinois 60201
fabianb@eecs.northwestern.edu

***Abstract*—Crowdsensing leverages the pervasiveness and power of mobile devices, such as smartphones and tablets, to enable ordinary citizens to collect, transport and verify data. Application domains range from environment monitoring, to infrastructure management and social computing. Crowdsensing services' effectiveness is a direct result of their coverage, which is driven by the recruitment and mobility patterns of participants. Due to the typically uneven population distributions of most areas, and the regular mobility patterns of participants, less popular or populated areas suffer from poor coverage.**

In this paper, we present Crowd Soft Control (CSC), an approach to exert limited control over the actions of participants by leveraging the built-in incentives of location-based gaming and social applications. By pairing crowdsensing with location-based applications, CSC allows sensing services to reuse the incentives of location-based apps to steer the actions of participating users and increase the effectiveness of sensing campaigns. While there are several domains where this intentional movement is useful such as data muling, this paper presents the design, implementation and evaluation of CSC applied to crowdsensing. We built a prototype of CSC and integrated it with two location-based applications, and crowdsensing services. Our experimental results demonstrate the low-cost of integration and minimal overhead of CSC.

I. Introduction

Smartphones and tablets are rapidly becoming a ubiquitous and powerful computing and sensing platform. It is estimated that over 96% of the world's population today (128% in developed nations) has a mobile phone and the fraction of smartphones among them is increasing daily [26]. Today's smartphones are commonly equipped with several gigabytes of storage, multiple cameras, microphones, motion and location sensors, as well as networking options including cellular, Bluetooth, and WiFi.

The pervasiveness and power of these devices have inspired a variety of community sensing services that crowdsource the monitoring, data transport and reporting tasks to device carrying participants. Examples of such mobile services range from WiFi mapping, waste disposal [5], place and route characterization [8], [36], transportation analysis [13] and noise pollution monitoring [24], to cite just a few.

For many of these services, their effectiveness is a direct result of their coverage. A noise pollution map should at least include every area people visit. Coverage, in turn, is driven by the scale and mobility patterns of participants. Given the typically uneven population distribution, and the regular mobility patterns of participants, areas will suffer from poor coverage [7].

The challenge is to ensure coverage without depending on large-scale adoption or particular mobility patterns. We address this challenge building on a simple idea – reuse the incentives mechanisms in location-based gaming and social networking apps to exert limited control over people's actions. We call this approach called *Crowd Soft Control (CSC)*.

In order for CSC to become viable within the domain of crowdsensing, CSC needs to provide high utility to sensing researchers with minimal impact to the hosting applications. In light of these, we designed CSC around the following three guidelines: it should be easy to integrate, not impact application user experience and not limit the functionality of crowdsensing services. CSC must be simple to incorporate into existing mobile applications in order for it to be adopted by developers. Similarly, CSC also cannot hinder the intended functionality or user experience of the host application. This includes any performance penalties imposed from the background sensing service, and any location or incentive transformations. Finally, CSC must allow fully capable crowdsensing services, and support resource sharing between multiple sensing campaigns. While we present the implementation and evaluation CSC applied in the context of crowdsensing services, there are several domains where this intentional movement could be beneficial (such as data-muling).

Contributions: We make three main contributions: (1) We extend Crowd Soft Control, first presented in an earlier position paper [34], an approach to leverage the incentives of existing location-based application to steer participants in crowdsourced sensing. (2) We describe the design (§ III) and implementation of a full CSC prototype and its integration in different example applications (§ IV). (3) We report real-world evaluation results of our ideas with 2 different applications, and example crowdsensing tasks to demonstrate the low-overhead and high effectiveness of Crowd Soft Control (§ V).

II. Crowd Soft Control

Crowdsensing – defined as individuals with sensing and computing devices collectively sharing information to measure and map phenomena of common interest [14], has seen rapid growth over the last few years. Much of this growth has come from the foreseen and eventual availability of pervasive,

Fig. 1. An example mobile location-based game – "Rescue Rush". Users (represented as the giant cat) earn points by passing over game objects projected to onto their physical locations. CSC proposes to couple game objectives with crowdsensing tasks.

always-connected, resource-rich mobile devices (e.g., [5], [8], [36], [13], [24]).

Crowdsensing services have typically followed either a *participatory* or *opportunistic* approach. Participatory crowdsensing explicitly involves users' actions, while opportunistic crowdsensing operates without requiring user involvement. Independent of the adopted approach, the effectiveness of most crowdsensing campaigns is largely a function of the coverage of its participants. For instance, the value of noise pollution studies for city planners and the effectiveness of the resulting policy decisions are a direct function of the fidelity and coverage of the resulting noise maps [11]. To ensure the necessary coverage, previous crowdsensing services have assumed, explicitly or not, large recruitments and either random or semi-random mobility patterns for its participants [42], [10], [30]. Most deployments, however, tend to have limited coverage/recruitment. Limited scale combined with the mobility patterns of humans, which have been shown to be highly predictable independently of the traveled distance [38], means that most areas of interest will go unmapped.

The same pervasiveness of sensor rich devices have enabled a growing number of location-based gaming and social applications [27], [12], [19], [29]. These applications bridge the gap between the virtual world and the physical world. Location based games such as Parallel Kingdom or Google's Ingress have mobile players interact with an augmented reality. Ingress is a massively multiplayer location based game. There have been suggestions that Google may be using this game not only for entertainment, but for collecting geo-tagged information as well [17]. Parallel Kingdom is another recent location-based, multiplayer game set in a virtual, augmented world. In these parallel kingdoms, users can claim territories based on their GPS location or by making friends who invite them to travel to new places. Check-in services by Foursquare and Facebook allow users to "check-in" to locations such as restaurants and points of interest, earning rewards such as badges and business discounts in the case of Foursquare. CSC leverages the incentives of this growing class of apps to address key challenges in crowdsensing.

A. *Incentive-reuse for Control*

A number of projects have began to explore incentive mechanisms to aid recruitment and to influence participant behavior in mobile crowdsourcing. While some studies have attempted to characterize part of the space (e.g., [32], [35]), largely looking at different macro incentive structures such as micro-payments, the variety and effectiveness of incentive mechanisms in this context is a topic of open research.

Rather than designing new incentive mechanisms, CSC reuses existing incentives mechanisms *already known to work*. Again using Ingress as an example, a recent survey of its users found that 79% of them were willing to travel up to 10km as part of the game [23]. In a related concept, gamification tries to provide incentive to users by adding certain gaming elements such as badges and leaderboards to traditionally ungamed actions. In contrast, CSC reuses the built-in incentives in existing games and social network applications, at the same time leveraging their existing user-base and recruitment efforts [42].

Consider the following scenario. Alice, an environmental scientist, is trying to generate a noise pollution map of a growing urban area. Given the large area of study, she decides to implement this using crowdsensing to leverage the high-end microphones available in many smartphones today. Alice quickly identifies, in her first try, that despite a decent participant base most of the noise samples seem to come from a few, clearly popular, areas in the city. For her second try, Alice turns to CSC and begins collaborating with Bob, the author of a virtual scavenger hunt game. Alice provides the areas and times from which she needs noise samples for her noise pollution mapping service to CSC. These sensing requests are translated into requests for individual players of Bob's game and inserted into the game space at the target locations and times. When a player achieves an objective as part of gameplay, the CSC run-time library captures the Alice's requested noise sample and reports this to her noise mapping service.

While we have focused our discussion on steering mobility, the *soft* control afforded through CSC can help crowdsensing services in other ways such as improving the quality of the sensing samples (e.g., longer noise samples, in our example) or the efficiency of sensing data transport through a data-muling service. In Park et al. [28], for instance, the authors illustrate the impact of mobility and loiter time of participants in the success and performance of a data muling service and show that *intentional* mobility, in contrast with opportunistic mobility, may be necessary for many common sensor deployments.

The following sections outline the design and implementation of CSC, and illustrate our ideas in the context of a crowdsensing application.

III. CSC Design

CSC is based on the idea of pairing mobile application incentives to the needs of the underlying crowdsensing service.

In order to provide high utility to sensing researchers with minimal impact to hosting applications, CSC follows three key guidelines: it should be easy to integrate, not impact application user experience and not limit the capability of the crowdsensing service. CSC must be simple to incorporate into existing mobile applications in order for developers to adopt it. Similarly, CSC also cannot hinder the intended functionality or user experience of the host application. This includes any performance penalties imposed from the background sensing service, and any location or incentive transformations. Finally, CSC must allow fully capable crowdsensing services, and permit resource sharing between multiple sensing campaigns.

Figure 2 illustrates the key CSC architectural components and a typical workflow for a researcher's sensing campaign requirements. Researchers submit a set of *Campaign Requirements* to the *Experiment Manager*, a part of the *CSC Cloud Service*. When an augmented *Host Application* makes a request to CSC, it passes through the *CSC Device Service* on the mobile device, where the request it received by the *Broker*. The Broker matches the request to a set of sensing campaigns, and distributes at set of discrete *Sensing Tasks* back to the application.

In the following section, we describe the problem of Crowd Soft Control, and define its application to crowdsensing. We the describe the three main design components, the *Researcher*, the *CSC Cloud Service*, the *CSC Device Runtime* and the *Host Application*, and, define the interfaces which exist between them.

A. Problem Description

It is possible to formulate CSC as an optimization problem: to maximize the value of each paired application location to the associated sensing campaigns, while minimizing the cost incurred to the hosting application.

Each mobile application $\mathcal{A}$ has a set of locations L bound to the application's context, and related incentives $\mathcal{I}$ (e.g. the value of a ghost in a GhostHunting game). Each sensing campaign C has a particular phenomena which it wishes to monitor through participatory sensing, and its own set of locations to test, L_s.

Each sensing location, ℓ, has a value $v(\ell)$ to each campaign. This value is determined by each campaign C and depends highly on the phenomena being investigated. This value is also dependent on the existing measurements recorded by the sensing service, and the temporal properties of the phenomena itself. For instance, the authors of [18] show that the value of each measurement in a Gaussian process can be derived through the mutual information of each new measurement.

Each CSC modification to a host application incurs a cost, potentially zero, by deviating from that host application's original execution. Depending on the nature of the underlying host application, locations can be translated to a more preferred location, ℓ_s, for a sensing campaign. The cost of each translation, $c_L(\ell, \ell_s)$, depends on the behavior of each host application. For example, the cost for changing locations can be the distance between the original point ℓ and the preferred point ℓ_p, normalized against an upper bound radius.

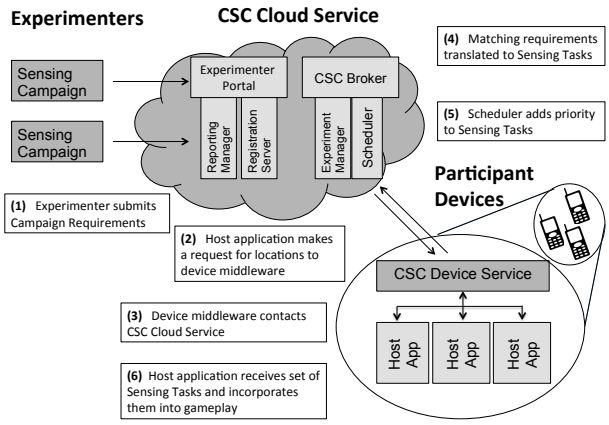


Fig. 2. CSC Architecture and workflow.

CSC also modifies or reassigns the assignment of incentives to host applications. Again, this changes the original application execution, incurring a cost, $c_{\mathcal{I}}(i, i_s)$. The magnitude of this cost is also dependent on the nature of the host application.

For each set of locations used by an application, CSC attempts to minimize the net cost for each set of points, $net\ cost = \sum c_L(\ell, \ell_s) + c_{\mathcal{I}}(1, i_s)$.

B. CSC Interfaces

CSC presents two interfaces: the researcher submitting sensing campaign descriptions (*Campaign Requirements*) and the host application that accepts CSC location recommendations (*Sensing Tasks*) and incorporates them into the application space.

1) Interface with Sensing Researchers: Since CSC modifies host application execution through a combination of location translations and incentive modifications, *application developers are required to modify their application code*. To ease this requirement, we have designed the interface between CSC and host applications to be as simple as possible.

Crowdsensing researchers, such as Alice in our previous example, issue a set of *Campaign Requirements* associated with a given sensing campaign. Campaign requirements in CSC, similar to those in [9], [30], specify the physical bounds of their sensing campaign and the parameters of the sensors used. Figure 3 shows an example set of requirements for the noise sampling campaign.

We chose a simplified set of requirements. These requirements state the *location*, *time* and *action* to be taken by a CSC registered device when sensing, as well as the number of samples needed (*frequency*).

Location is stated in terms of latitude and longitude, and optionally altitude, heading, and pitch. *Time* includes date and clock time for launching a sensing action. Both location and time are stated as ranges (e.g., start and end time), rather than specific points

```
<requirements>
    <area>
        <location>
            <point>42.3;-87.65</point>
            <point>42.56;-87.554</point>
            <point>42.23;-87.345</point>
            <point>42.3,-87.55</point>
        </location>
    </area>
    <start-time>2011-08-03 11:23:54</start-time>
    <end-time>2011-9-03 11:59:59</end-time>
    <action>
        <sensor>microphone</sensor>
        <duration units="seconds">3</duration>
    </action>
    <frequency units="hour">25</frequency>
</requirements>
```

Fig. 3. Example Campaign Requirements for a sensing campaign capturing noise pollution.

Actions states the parameters for the sensing action requested which depends on both the sensing campaign and particular devices. For example, the microphone action contains parameters such as the duration of the recording and the sampling rate.

2) Interface with Host Application: These set of broad requirements are translated into individual tasks and distributed to users of integrated *host applications*, such as players of Bob's virtual scavenger hunt game. Bob's scavenger hunt game first retrieves the set of points which it would display to the user, and passes them to the CSC Device Runtime as a set of locations. From this set of original locations, the CSC Device Runtime generates a new set of preferred locations. These *Sensing Points* contain the new preferred locations, as well as the priority of each location as a number between 0 and 1. The host application incorporates this updated location and incentive information into its current context.

Host applications become part of the CSC platform by adding a few lines of code (25 SLOC in an example application § V-B). CSC provides interfaces for the developer to incorporate spatial and temporal constraints and priority of a sensing task into its execution. Incorporating the spatial and temporal constraints of a sensing task is relatively straightforward for location-based applications. Sensing task location and temporal targets can be incorporated as new or modified targets (e.g., changing the location of a game objective) in an application. In addition, application developers can account for sensing task priorities by selecting among possible objectives or adjusting relative incentives (e.g., points in a scavenger hunt) in their application.

The host application has no knowledge of the underlying sensing task. All information regarding which sensor to use, and sensing campaign information is maintained in the CSC Device Runtime.

C. CSC Cloud Service

The CSC Cloud Service consists of four main components: the Experiment Manager, the Registration Service, the Broker and the Report Manager. The *Registration Service* handles the registration of the mobile device, researcher, and host application, and distributes authentication keys to each party be used within CSC services. The *Experiment Manager* accepts requests submitted from researchers, holds budgeting information about the number of sensing tasks distributed and completed, and can be queried from other CSC services to return experiments within a spatial or temporal range. The *Report Manager* tracks the state information of assigned sensing tasks as well as the corresponding sensing reports issued by devices. These reports, which can be aggregated similarly to AnonySense [9], are used to respond to queries from researchers. The *CSC Broker* uses sensing requirements, device context and CSC platform policies to coordinate resource allocation between researchers and participant devices. The Broker combines the set of experiment requirements (from the Experiment Manager), the history of current and past sensing task assignments (from the Report Manager), and the current advertised state from the mobile device to generate a set of task assignments.

1) Matchmaking and Scheduling: CSC implements a 3-part scheduling between the CSC Cloud Service and the CSC Device Runtime: (i) the scheduler first matches campaign requirements between the capabilities of CSC hosting applications, (ii) calculates preferred locations, and (iii) determines optimal incentive modifications.

The Matchmaking returns a set of experiments which are compatible with the requesting mobile device. Within the CSC cloud service, the Experiment Manager offers an interface to query for compatible requirements. This query takes a state vector containing a location, time and set of available actions, and returns a set of experiments. CSC treats each experiment requirement as a *multidimensional volume with each axis represented by experiment parameters* (i.e. latitude, longitude, time, available actions). Each request represents a multidimensional point in space. Compatible experiments are found by determining if the point is surrounded by the polygon in each dimension as described in Sutherland et al. [39].

With the set of campaign requirements selected, the next step of CSC scheduling is to find the location translations from the set of original locations provided by the CSC hosting application, which maximize the net value given the value of each point to the sensing campaign, $v(\ell)$, and the cost to the host application $c_L(\ell, \ell_s)$.

When computing the optimal set of preferred locations, the value of each sensing location is highly dependent on phenomena being sensed. There exist several methods to determine the value of a measurement obtained from optimal sensor placement problems, including the use of entropy [31], mutual information [16] and spatial metrics [15]. For simplicity of implementation, our system assumes a uniform distribution of each phenomena in space, and calculates the value based on the geographic distance between each proposed point and each existing measurement. While this may not be optimal for many sensing phenomena, the system can be easily extended to include any of the methods described above.

Once the set of preferred locations has been obtained, the scheduler then assigns normalized incentive values based on the results of the value function for each preferred location. These results are then returned to the CSC Device Runtime where they are delivered to the host application for integration.

Our design focuses on generalizing the interface between sensing campaigns and host applications. However, one can imagine the use of predictive mobility models similar to Reddy et al. [33], or history based models which track compliance and mobility under different mobile applications used with CSC, however, these enhancements are out of the scope of this paper.

D. CSC Device Runtime

The CSC Device Runtime is made up of the *CSC Device Service* (daemon) and the *CSC Device Libraries*. The CSC Device Service handles device registration, communication with CSC clouds services, and all sensing actions within CSC. The Device Libraries are integrated into each host application and communicate between that application and the CSC device service.

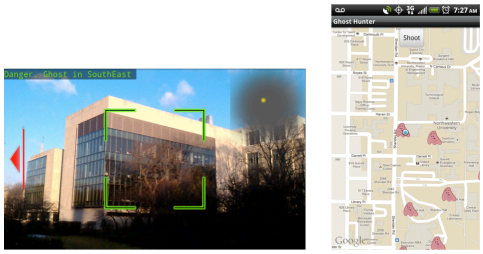
At initialization, CSC determines the sensing capabilities of the device and registers it with the CSC Cloud Service. The heterogeneity of mobile devices means that different devices contain different sensors and possess different capabilities. Phones resources can range in the number and quality of microphones and cameras, the presence of compass, accelerometers, gyroscopes, light and proximity sensors. Registration allows a device to announce its presence to CSC and report its capabilities. The device capabilities are used for matching sensing task to devices. We are exploring alternative approaches to ensure user privacy, including cloaking and measurement aggregation [9].

The Device Libraries interact with the hosting application in two instances: (i) when a new request for sensing locations is issued and (ii) when the application detects the sensing tasks requirements are satisfied. This interaction requires modifying the hosting application at the corresponding points in its source code. The first is the during the querying of in-game locations, the $Request$, and when the application detects the user context matches a task requirements, the $Callback$. In Sec. V-B we show that these changes are generally small and localized.

E. Hosting Application

CSC has two main requirements of hosting mobile applications, the use of physical locations with bound incentives. Within the classes of location-based applications, there are those with flexible locations, including games such as Parallel Kingdom [29], and fixed locations which involve existing landmarks and points of interest, such as a photo hunting game.

Depending on the flexibility of locations within each application, CSC is able to offer different utility to sensing campaigns. For applications with a large degree of location flexibility, such as role-playing, adventure games or point-capture games where users must collect items around them, CSC can have more freedom to translate the application's original locations toward more preferred locations, and provide greater value for associated sensing campaigns.



(a) Augmented Reality (b) Map

Fig. 4. The augmented reality and map screen of our GhostHunter game.

Unfortunately, those applications without flexibility in their location, and that lack variable incentives for users are unable to be used as CSC host applications.

Additionally, CSC hosting applications can offer different levels of sensing services, depending on the level of control offered by the application itself. For example, the two host applications we evaluate in Section V enact different levels of control over user movements. For instance, our GhostHunter application incorporates an augmented reality element into gameplay. In order to achieve an objective, the user must point their device's camera at a particular heading and zenith in a particular location; whereas the geocaching application only affects a user's location with no control over device orientation. Therefore, campaigns requiring the camera would only be accessible to users of the GhostHunter application.

IV. Implementation

We developed a fully operational prototype for CSC, consisting of the CSC Cloud Service and the CSC Device Runtime system. We implemented the CSC Cloud Service in Python (1,020 SLOC) as a webservice running within the Django [1] framework, backed by a MySQL server. All functions are accessed through A REST interface. Our prototype implementation executes requirement matching, location translation, task dissemination, measurement recording and scheduling.

The CSC device runtime system is implemented as a library for the Android 4.2 operating system consisting of 4,258 lines of Java code. The library consists of a background service which handles registration of the device with CSC, communication with the CSC cloud service, and executes each sensing action. A list of currently implemented sensing modules is shown in Table I.

A. Hosting Applications

We integrated CSC into two different hosting applications – a homegrown augmented reality game (GhostHunter) and an open-source geocaching game (c:geo[2]).

a) GhostHunter: GhostHunter is a proof-of-concept augmented reality game for Android. In GhostHunter, a player

[1]http://www.djangoproject.com
[2]http://cgeo.com

Name	Description
camera	Takes Photographs
microphone	Records noise measurements
wifi-scan	Records access points and RSSI
ping	Performs a ping to host
traceroute	Performs a traceroute to host
ndt	Runs a Network Diagnostic Test

TABLE I
AVAILABLE SENSORS WITHIN SENSING REQUIREMENTS

chases ghosts and other monsters around their neighborhood and "zaps" them by capturing their photo through an augmented reality display. All targets have an associated reward value. We chose this augmented-reality game model as it supports the most control over a user's device; in addition to a user's location and time, the exact position of a ghost allows us to control the heading and orientation of the device as well.

The game consists of two parts: a map screen and an augmented reality display. To play, the user walks around toward the location of "ghosts" which are overlaid on the map screen. When sufficiently close, the game switches to an augmented reality mode in which the player follows arrows on the screen to place the ghost in their cross hairs. Once the ghost is within their sights, the player is able to capture it. GhostHunter is implemented in about 3,500 lines of Java code. Screen shots of our GhostHunter app are shown in Figure 4.

b) Geo-caching: We used an open-sourced geo-caching application named c:geo[3] to illustrate the cost of integrating CSC into an unfamiliar codebase. In geo-caching players place caches, hidden containers holding small treasures, all over the globe, and publish their geographic coordinates for others to find. When one finds a cache, the tradition is to swap one of the existing treasures for one of your own, thus creating a connection with others in the geo-caching community.

Each cache exists as a physical object and a fixed location, so, unlike GhostHunter, we cannot merely change the location of the objectives, otherwise the player will find no caches. We instead must use the given points to take measurements.

The incentive structure of c:geo geo-caching is slightly different than GhostHunter since there are no in-game rewards. The incentive is the enjoyment of finding the cache and connecting with those previously there. Within the c:geo online database are ratings and comments from previous finders. We modify the ratings of the caches based on the proximity to sensing tasks. While this alters the particular ratings of a cache location, it does not fundamentally change the user experience.

V. EVALUATION

In this section, we present results from our evaluation of CSC, including the potential of our approach for steering user actions, the overhead of integrating and using our CSC prototype and its potential benefits in three concrete studies [4].

[3]http://cgeo.org

[4]A series of case studies representing a fourth evaluation measure were conducted, but were omitted for space constraints

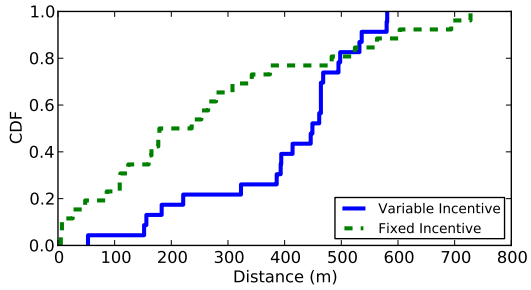


Fig. 5. Distance travelled by participants for each game item from their determined baseline path. The majority of users on phase 3 travelled further.

A. Potential of Crowd Soft Control

Our first experiments aimed to characterize the flexibility of users' mobility patterns during game play (i.e., how far and how long are users willing to travel for a game objective?). Through this experiment we show (i) that in-game rewards can motivate people outside of their baseline mobility paths and that (ii) greater in-game rewards result in higher movement distances.

We run the experiment over the course of four days in April 2013, engaging 22 undergraduate students using their personal Android mobile device. We collected data through the use of our instrumented GhostHunter application which recorded and reported user-game interaction, our CSC mobile device framework which reported task measurements, along with a logging application which recorded a user's mobility profile through out the experiment. Participants were given a $20 gift card for their successful participation in the experiment.

The experiment was split into three phases, a baseline, a fixed-incentive and a variable-incentive phase, each lasting 24 hours.

Baseline Phase: The first phase of the experiment sets the mobility baseline for each user for later comparison. A logging application recorded each device's GPS location at 5 minute intervals. This collection of GPS traces constitute a user's baseline mobility profile.

Fixed Incentives: Users were instructed to play GhostHunter for a total of 30 minutes during the day. The experiment game took place entirely on the university's campus, and game objectives were bounded by its geography. Game locations were chosen at random, with a total of 20 objectives on the map at any given times. A new objective was created upon completion. All game objectives were worth 100 points.

Variable Incentives In this last phase, we measure the user response to different incentive values for game objectives. Participants were asked to play for an additional 30 minutes during that day. Point values of each objective was determined by its distance to that particular user's baseline path recorded during Phase 1. The values ranged from 50 to 400 points depending on the distance calculated.

c) Findings: We compute the traveled distance as the shortest Euclidean Distance between the baseline mobility path and the measurement point. Figure 5 plots these distances. As the figure shows, users travelled an average of 264 meters

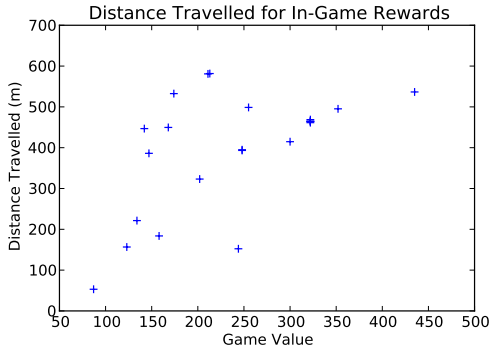


Fig. 6. Distance travelled by users in phase 3 plotted against the point value offered within the game. Distance traveled was strongly correlated with game points.

between their baseline paths and each measurement with Fixed Incentives (phase 2) and an average of 392 meters, 48% increase, when varying the point value of the different objectives.

Figure 6 plots the distance travelled from each user's baseline mobility path to each objective against the game value of a point. As it is clear from the figure, users were willing to travel the extra distance for the additional in-game rewards.

B. Cost of Integration

We measure the cost of integration based on the extent of changes necessary to incorporate CSC in a hosting application. We use the total number of SLOC added or modified as our metric.

Each location-based application communicates with the CSC device service through Android interprocess communication. At different times while executing (e.g., as the game progresses), an application requests locations from the CSC daemon using `getCSCLocations([location set])`. Where *[location set]* is a set of (latitude, longitude) pairs of original application locations. A collection containing the original location, the new CSC preferred location, and the location priority.

When the user reaches the specified location, a callback is fired to CSC through `performCSCAction()` to carry out the predetermined action from the sensing task (e.g., taking a sound sample). A callback mechanism lets us reuse the location detection capabilities existent in location-based applications and is more efficient than an alternative CSC driven pull model.

To integrate CSC one must include calls and associated handlers for both tasks, and add logic to incorporate sensing task priorities (if applicable). This can be done with minimal changes. Table II reports the number of lines needed to integrate CSC in both GhostHunter Game and the open source c:geo geo-caching application. The maximum number of SLOC added/modified was 25 (22 for c:geo). While the specific numbers depend on the particular hosting application, we believe the integration task to have relatively low-overhead given its minimal interface.

Application	Function	Num. of lines
GhostHunter	Location Request	2
GhostHunter	Device Action	10
GhostHunter	Other	13
c:geo	Location Request	7
c:geo	Device Action	15
c:geo	Other	0

TABLE II
COST TO INTEGRATE SOFT CONTROL INTO OUR EXAMPLE GAME IN TERMS OF NUMBER OF ADDITIONAL LINES OF CODE USED. CSC WAS INTEGRATED WITH LESS THAN 25 LINES OF ADDITIONAL CODE.

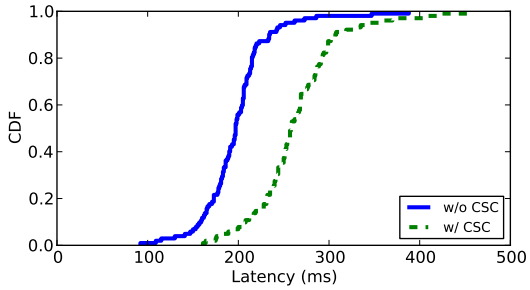


Fig. 7. Additional application latency induced with CSC. The integration of CSC results on a median latency difference of 61 ms, imperceptible to users [4].

C. Micro Benchmarks

We estimate the impact of using the CSC library to the hosting applications in terms of additional processing delay and network usage. These tests were performed on a Samsung Galaxy S3 phone running Android 4.2.

1) Induced Application Latency: One of the main concerns with CSC integration is noticeable performance differences between original applications and those integrated with CSC. We estimate the perceived impact of CSC integration on an application by measuring the additional latency in the application's execution during CSC activities. Performance penalties manifest themselves as delayed responsiveness within the application such as the application's response time to user input or the time between game screens.

We measured this application responsiveness within GhostHunter recording the time taken to capture a Ghost in the game both with and without CSC integration. When the player "captures" the ghost, the capture is recorded as part of the game and the game returns the user to the Map screen. When CSC is involved, at the time of the ghost capture a callback to the CSC runtime library results in a sensing measurement sampling.

The time between when a user presses the capture button and when the new screen loads is the event latency in GhostHunter. Any additional time incurred by CSC activities is the performance impact of CSC integration. We measured this additional latency from when a user presses the capture button, to when the map screen completed loading. We ran 100 trials within GhostHunter with and without CSC, and measured the response time. Figure 7 plots the results of this experiment. As the figure shows, CSC added delays are minimal and imperceptible to users [4]. On average, the additional latency

Function	Mean	Min	Max
Task Request	3.27	1.15	5.73
Measurement - camera	492.8	258	680
Measurement - microphone	4.56	1.8	7.5
Measurement - wifi	2.2	0.89	4.34

TABLE III
NETWORK USAGE FOR CSC DEVICE RUNTIME FUNCTIONS PER INVOCATION (IN KB).

imposed is only 66 ms, and a 90th percentile additional latency of only 72 ms.

2) Network Usage: Last, we measured CSC overhead with regard to additional network usage incurred by the framework. We used the network profiler included in the Android 4.2 SDK to capture the amount of data transferred to our servers. We distinguish between measurement and non-measurement traffic data since the amount of data transferred depends on the type of measurement desired (e.g. a photo contains more information that a wireless network measurement). Table III shows that while certain measurements such as photos can take on average of nearly 500 kB, prolonged use of CSC could potentially add significant network traffic; however, since measurement data is not time sensitive, it would be trivial to require that all measurement traffic be transferred over Wifi interfaces.

VI. DISCUSSION

By coupling crowdsensing services with third-party mobile applications, CSC diverges from traditional crowdsensing applications and frameworks. In particular, we discuss how CSC recruits participants for measurements and how it maintains the privacy and security of its users.

In terms of user recruitment, CSC shifts the target up from individual users towards mobile application developers. We believe this may be a preferable approach to more common pure crowdsourcing models that require each individual participant to be "enlisted". With CSC, a single developer has the potential to reach a large number of users depending on the popularity of her application. In addition, we see a possibility for models like CSC as a way for developers to be compensated for their mobile applications as an alternative for ad-supported models.

As is the case with all location based systems, including mobile micro-task markets, existing mobile crowdsourcing systems and location based games/applications, privacy remains an open concern for users. By design, CSC retains no identifiable information from the users. All information is stored with a unique ID that is randomly assigned to each client. There is still the opportunity for identification to be inferred from geographic context [37]. However, CSC adds no additional loss of privacy beyond that of existing host application.

Similarly, there is a potential risk of allowing researchers to "lure" users toward potentially dangerous areas. We argue that CSC does not add to the risks already present in any location-based application. As augmented reality gamers can be trusted to exercise their best judgement during play, users of extended location-based applications should be trusted to apply that same judgement to the suggestions made through CSC.

VII. RELATED WORK

This paper builds upon the earlier work of Rula et al. [34], and is inspired by much previous work on mobile sensing systems [18], mobility analysis and incentives in the context of crowdsensing. To the best of our knowledge, CSC is the first approach to mate existing applications and their incentives with crowdsensing tasks.

Crowdsensing platforms. Several mobile sensing platforms have been proposed and implemented in recent years (e.g. [9], [10], [22], [30]). Anonysense [9] primarily focuses on mobile user privacy. It uses a pull based method of distributing sensing tasks which is shared by CSC. PRISM [10], Bubble Sensing [22], and Medusa [30], three other related frameworks, distribute sensing tasks to participating users. PRISM allows the execution of remote code on participating devices, Bubble Sensing introduces an abstraction to tie physical locations to tasks, and Medusa generates crowdsensing work flows across multiple devices. Like CSC, these platforms aggregate sensing tasks and disseminate them to mobile users based on contextual information.

Mobility patterns and sensing. Previous studies have shown the benefits controlled (or predictable) mobility can have on sensing coverage and routing efficiency [21], [2], [3], [6], [28]. Reddy et al. put forward a closely related idea leveraging participant's previous location traces to guide scheduling in participatory sensing campaigns [33]. While predictive methods can help optimize existing mobility patterns, they are unable to direct measurements to currently uncovered areas. These platforms do not offer any form of "control" over participants' actions relying on either opportunistic interactions or participatory volunteer response for coverage. Our work attempts to gain partial control over participants actions through the reuse of applications built-in incentives.

Incentive Design. Recent work has started to explore the use of incentive mechanisms in crowd sensing applications to address recruitment and data collection issues found in early sensing platforms [35]. Sik et al. [20] proposes a market mechanism for pricing participatory sensing data. Yang et al. [43] devises both a platform-centric and user-centric model of incentive mechanisms for crowdsourcing to smartphones. Reddy et al. [32] investigated the effects of different micro-payment incentive structures on participatory sensing performance. While our work avoids the difficult challenge of modeling human response to incentives, and instead relies on incentives already known to work in existing applications, it could benefit from better incentive mechanisms.

Games with a purpose. Similar to *Games with a Purpose* [41], CSC attempts to utilize indirect incentives to help motivate users to complete system-level goals. Several applications have used the idea of games as platforms for crowdsourcing activities by developing games which specifically exploit human computation during game play [40], [1],

[25]. Unlike this previous work, CSC exploits mobile game play from applications *not* designed for crowdsourcing.

VIII. Acknowledgements

We would like to thank the anonymous reviewers for their helpful and insightful comments. This work was supported in part by the National Science Foundation through Award CNS 1218287 and 1211375.

IX. Conclusion

We have presented Crowd Soft Control and demonstrated the benefits of this approach in crowdsensing. By pairing the sensing requirements of crowdsensing services with location-based applications, CSC allows researchers to leverage an application's incentives (e.g. game objectives) to control the movement of participating users, increasing the effectiveness and efficiency of sensing campaigns. We implemented and evaluated CSC in the context of two mobile applications. Our evaluation shows the low cost of integrating CSC into existing applications and, through micro-benchmarks, the minimal overhead it imposes our run-time system on the mobile device and hosting application.

References

[1] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande. Foldinghome: Lessons from eight years of volunteer distributed computing. 2009.

[2] D. R. Bild, Y. Liu, R. P. Dick, Z. M. Mao, and D. S. Wallach. Using predictable mobility patterns to support scalable and secure MANETs of handheld devices. In *Proc. of MobiArch*, 2011.

[3] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: Sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1), 2004.

[4] S. K. Card, A. Newell, and T. P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.

[5] U. center for Embedded Networked Systems. Participatory sensing / urban sensing projects. http://urban.cens.ucla.edu/projects/garbagewatch/, January 2011.

[6] A. Chakrabarti, A. Sabharwal, and B. Aazhang. Using predictable observer mobility for power efficient design of sensor networks. In *Proc. of IPSN*, 2003.

[7] Y. Chon, N. D. Lane, Y. Kim, F. Zhao, and H. Cha. Understanding the coverage and scalability of place-centric crowdsensing. In *Proc. of UbiComp*, 2013.

[8] Y. Chon, N. D. Lane, F. Li, J. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. September 2012.

[9] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos. Anonysense: privacy-aware people-centric sensing. In *Proc. of MobiSys*, 2008.

[10] T. Das, P. Mohan, V. N. Padanabhan, R. Ramjee, and A. Sharma. PRISM: platform for remote sensing using smartphones. In *Proc. of MobiSys*, 2010.

[11] European Commission. The environmental noise directive. Technical Report 2002/49/EC, European Commission, 2002. Adopted June 25, 2002.

[12] Foursquare. foursquare. http://www.foursquare.com.

[13] R. K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T. F. Abdelzaher. Greengps: a participatory sensing fuel-efficient maps application. In *Proc. of MobiSys*, 2010.

[14] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: Current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[15] H. González-Banos. A randomized art-gallery algorithm for sensor placement. In *In Proc. SCG*, 2001.

[16] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *In Proc. ICML*, 2005.

[17] H. Hodson. Why google's ingress game is a data gold mine. *New Scientist*, 2893, Dec 2012.

[18] A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Toward community sensing. In *Proc. of IPSN*, 2008.

[19] R. R. Labs. Red robot — location is everywhere. http://www.redrobot.com.

[20] J.-S. Lee and B. Hoh. Sell your experiences: a market mechanism based incentive for participatory sensing. In *Proc. of PerCom*, 2010.

[21] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley. Mobility improves coverage of sensor networks. In *Proc. of MobiHoc*, 2005.

[22] H. Lu, N. D. Lane, S. B. Eisenman, and A. T. Campbell. Bubble-sensing: Binding sensing tasks to the physical world. *Pervasive Mob. Comput.*, 6(1):58–71, Feb. 2010.

[23] Lui. The demographics of ingress. http://simulacrum.cc/2013/01/23/the-demographics-of-ingress.

[24] N. Maisonneuve, M. Stevens, M. Niessen, and L. Steels. Noisetube: Measuring and mapping noise pollution with mobile phones. *Information Technologies in Environmental Engineering*, pages 215–228, 2009.

[25] S. Matyas, C. Matyas, C. Schlieder, P. Kiefer, H. Mitarai, and M. Kamata. Designing location-based mobile games with a purpose: collecting geospatial data with CityExplorer. In *Proc. ACE*, 2008.

[26] mobiThinking. Global mobile statistics 2012. http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#subscribers, May 2013.

[27] NianticLabs@Google. Ingress. http://www.ingress.com.

[28] U. Park and J. Heidemann. Data muling with mobile phones for sensornets. In *Proc. of SenSys*, 2011.

[29] PerBlue. Parallel kingdom. http://www.parallelkingdom.com.

[30] M.-R. Ra, B. Liu, T. la Porta, and R. Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proc. of MobiSys*, 2012.

[31] N. Ramakrishnan, C. Bailey-Kellogg, S. Tadepallit, and V. N. Pandey. Gaussian processes for active data mining of spatial aggregates. In *In Proc. SIAM*, 2005.

[32] S. Reddy, D. Estrin, M. Hansen, and M. Srivastava. Examining micro-payments for participatory sensing data collections. In *Proc. of UbiComp*, 2010.

[33] S. Reddy, D. Estrin, and M. Srivastava. Recruitment Framework for Participatory Sensing Data Collections. In *Proc. of PerCom*, 2010.

[34] J. P. Rula and F. E. Bustamante. Crowd (soft) control: moving beyond the opportunistic. In *Proc. of HotMobile*, 2012.

[35] J. P. Rula, V. Navda, F. E. Bustamante, R. Bhagwan, and S. Guha. No "one-size fits all": Towards a principled approach for incentives in mobile crowdsourcing. In *Proc. of HotMobile*, 2014.

[36] S. Shah, F. Bao, C.-T. Lu, and I.-R. Chen. Crowdsafe: Crowd sourcing of crime incidents and safe routing on mobile devices. In *Proc. of ACM SIGSPATIAL GIS*, November 2011.

[37] K. Shilton. Four billion little brothers?: Privacy, mobile phones, and ubiquitous data collection. *Commun. ACM*, 52, November 2009.

[38] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási. Limits of Predictability in Human Mobility. *Science*, 327(5968):1018–1021, Feb. 2010.

[39] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys (CSUR)*, 6(1):1–55, 1974.

[40] K. Tuite, N. Snavely, D.-Y. Hsiao, N. Tabing, and Z. Popović. PhotoCity: training experts at large-scale image acquisition through a competitive game. In *Proc. CHI*, May 2011.

[41] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *Proc. CHI*, 2004.

[42] Y. Xiao, P. Simoens, P. Pillai, K. Ha, and M. Satyanarayanan. Lowevering the barriers to large-scale mobile crowdsensing. In *Proc. of HotMobile*, 2013.

[43] D. Yang, G. Xue, X. Fang, and J. Tang. Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing. In *Proc. of MobiCom*, 2012.