PRISM: Platform for Remote Sensing using Smartphones

Tathagata Das
Microsoft Research India
Bangalore 560080, India
tathadas@microsoft.com

Prashanth Mohan *
University of California, Berkeley
Berkeley, CA 94720, USA
prmohan@cs.berkeley.edu

Venkata N. Padmanabhan
Microsoft Research India
Bangalore 560080, India
padmnab@microsoft.com

Ramachandran Ramjee
Microsoft Research India
Bangalore 560080, India
ramjee@microsoft.com

Asankhaya Sharma *
Microsoft India Development Center
Hyderabad 500046, India
asankhs@microsoft.com

ABSTRACT

To realize the potential of opportunistic and participatory sensing using mobile smartphones, a key challenge is ensuring the ease of developing and deploying such applications, without the need for the application writer to reinvent the wheel each time. To this end, we present a Platform for Remote Sensing using Smartphones (PRISM) that balances the interconnected goals of *generality*, *security*, and *scalability*. PRISM allows application writers to package their applications as *executable binaries*, which offers efficiency and also the flexibility of reusing existing code modules. PRISM then pushes the application out automatically to an appropriate set of phones based on a specified set of predicates. This *push* model enables timely and scalable application deployment while still ensuring a good degree of privacy. To safely execute untrusted applications on the smartphones, while allowing them controlled access to sensitive sensor data, we augment standard software sandboxing with several PRISM-specific elements like *resource metering* and *forced amnesia*.

We present three applications built on our implementation of PRISM on Windows Mobile: *citizen journalist*, *party thermometer*, and *road bump monitor*. These applications vary in the set of sensors they use and in their mode of operation (depending on human input vs. automatic). We report on our experience from a small-scale deployment of these applications. We also present a large-scale simulation-based analysis of the scalability of PRISM's push model.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server,Distributed applications*

*The authors were with Microsoft Research India during the course of this work.

MobiSys'10, June 15–18, 2010, San Francisco, California, USA.

General Terms

Design, Experimentation, Security, Performance

Keywords

Mobile Platform, Mobile Sandbox, Participatory Sensing, Opportunistic Sensing, Smart Phones

1. INTRODUCTION

Mobile phones are proliferating with more than 4 billion phones in use worldwide [1]. Programmable smartphones constitute a significant and growing fraction of these phones. For instance, 172 million of the 1.2 billion phones sold worldwide in 2009 were smartphones [2]. While smartphones today are used largely in isolation, operating as individual units serving their respective users (e.g., enabling users to check their email), there is a nascent interest in community applications, which leverage the resources of a potentially large and distributed set of mobile smartphones.

One such class of community applications that has received much research attention recently is *community sensing* [26, 25, 37, 4, 7]. Community sensing using mobile smartphones is motivated by the observation that such phones include not only computing and communication capabilities but also a range of sensing capabilities, such as provided by the microphone, camera, GPS, and accelerometer, among other sensors. The idea, then, is to orchestrate the computing, communication, and sensing capabilities of a population of mobile phones, which happen to be at the right place at the right time, to enable large-scale sensing purely through software running on this existing hardware base. A community sensing application could either be *participatory*, involving explicit user action (e.g., taking photographs), or *opportunistic*, operating without user involvement (e.g., recording a GPS trace) [8, 27].

A key challenge in realizing the potential of community sensing is ensuring the ease of developing and deploying such applications, so that the application writer does not have to reinvent the wheel each time. To this end, we present PRISM, a Platform for Remote Sensing using Smartphones, which addresses a number of common challenges that such applications face. PRISM tries to balance three interconnected goals:

1. *Generality:* support a wide range of applications with flexibility to reuse existing code.
2. *Security:* ensure that the participating phones, belonging to individual users, remain secure and that applications do not misuse sensitive sensor information.

3. *Scalability:* allow the system to scale to a large number of nodes (say hundreds of thousands) without placing an undue burden on the PRISM infrastructure.

Generality demands the ability to run arbitrary code on the mobile smartphones. However, doing so exposes the phone to security attacks, including the possibility of the phone being compromised. To provide generality, PRISM supports the execution of applications in binary form. This allows applications to reuse existing code modules such as libraries. However, to ensure security, the (untrusted) application code is run in a software sandbox, which intercepts and mediates system calls.

The mobile sensing context, however, raises unique challenges that go beyond standard software sandboxing. On the one hand, the applications of interest, by their very nature, require access to sensor data. On the other hand, such sensor data (e.g., audio, images) could be sensitive from the viewpoint of the user. Furthermore, access to the sensors and the processing performed by the application could drain precious battery energy. To address these concerns, we augment the sandbox with several PRISM-specific features: (i) *resource metering* to measure and limit the amount of battery energy that the application can consume and also to prevent the applications from leaking out sensitive sensor data that it may have access to, (ii) *forced amnesia* to disallow the sensing application running on the phone from retaining long-term state, and (iii) *sensor taint tracking and access control* to allow the participating users to set simple policies on the kinds of applications that they are willing to run on their phones. These mechanisms help PRISM mitigate the risks associated with allowing access to sensor data. Avoiding all risk would require fully blocking access to sensors, which would curtail functionality.

The key question with regard to scalability is how tasks are distributed. There is an interplay between the number of tasks, the specificity of their requirements, and the number of participating nodes. On one hand, one could employ a *pull* approach (as in AnonySense [13]), wherein sensing tasks are posted on a server and the participating nodes download all of the published tasks, before deciding locally which tasks to actually run. This approach has the advantage that the nodes do not reveal anything about their context (e.g., their location), but it imposes a high overhead since many or even all of the tasks downloaded by a node might not match the node's local context. In PRISM, we employ the alternative *push* approach, wherein the participating nodes register with the server and the server only pushes matching tasks out to the individual nodes. This approach avoids the scalability bottleneck of the pull approach and allows tasks to be distributed in a timely manner. The downside, however, is that the server can track the mobile node. We limit the extent of such tracking in PRISM by expiring registrations, and requiring fresh registrations, from time-to-time.

We have implemented PRISM on Windows Mobile, with the infrastructural component running on Windows 7. We report on our experience and findings from building and deploying, on a small scale, three different community sensing applications. The intention is not to present novel applications but rather to showcase the generality of the PRISM platform.

The first one is a *citizen journalist* application, which is inspired by Micro-Blogs [22] and involves *participatory* sensing, wherein PRISM provides location-based triggers to alert human users to take pictures or provide other information from the scene of interest. The second application is a *party thermometer*, which allows a user to query others who are at bars about how "hot" the party is. In addition to the locations of the bars, this application uses the microphone to sense music and thereby only target users who are *in* a party. In doing so, the application uses an off-the-shelf Fast Fourier Transform (FFT) library and operates within the constraints of PRISM's resource metering and forced amnesia policies. The third application is a *road bump monitor*, which is inspired by Pothole Patrol [18] and Nericell [31], and involves *opportunistic* sensing, wherein phones equipped with GPS and accelerometer sensors are used to detect and locate road bumps automatically. We show that this application running on PRISM (within constraints such as forced amnesia) is as effective in detecting bumps accurately as a native application such as Nericell [31] without such constraints.

To summarize, we make the following contributions:

- The design and implementation of the PRISM platform for supporting community sensing applications, which balances the goals of generality, security, and scalability.
- A demonstration of the generality of PRISM through the development and deployment, on a small scale, of three participatory or opportunistic sensing applications.
- An evaluation of PRISM's push model of task distribution through large-scale simulation.

2. RELATED WORK

The authors in [3] introduce a new term called *Mobiscope* to characterize an infrastructure composed of federation of distributed mobile sensors/phones that can execute sensing tasks to build these applications. PRISM is targeted as a framework that simplifies the task of building applications using Mobiscope. A wide range of Mobiscope applications have been built and deployed on generic or custom-enhanced mobile phones. The goals have been varied: monitoring air pollution [25], evaluating soot emission levels [37], assessing environmental impact of individuals [32], noise mapping [7], matching passengers to cars for dynamic ride sharing [36], urban gaming [12], sensing enhanced social networking [30], and rich monitoring of road and traffic conditions [31]. PRISM is designed to enable developers of such applications to easily harness the appropriate set of phones with the required sensing resources, without having to be concerned with distributed operation, resource management, security, or privacy.

PRISM draws inspiration from the rich body of prior work on frameworks for building sensing applications over a distributed but dedicated set of sensor nodes. Motelab [41] and Kansei [19] provide a generic framework for web-based scheduling of tasks on dedicated sensor platforms. CitySense [33] provides a testbed for city-wide sensing applications. However, the fact that the sensors are dedicated means that, unlike in PRISM, user security and privacy are not an issue nor is mobility (except for mobility under the control of the programmer, as with the robotic nodes in Kansei).

There has also been recent work, more closely related to PRISM, on frameworks for mobile phone based community sensing. Some of this work has been in the context of the Metrosense project [10] on people-centric sensing. Specifically, Bubble Sensing [29] allows sensing tasks to be posted at specific physical locations of interest. The tasks are broadcast over a local-area radio by an anchor node that is at the location of interest. When other mobile nodes pass by and hear the broadcasts, they can help fulfill the sensing tasks. Unlike PRISM, bubble sensing avoids the need for phones to report their location to a server in the infrastructure. However, this also means that the ability to satisfy sensing tasks depends on the simultaneous availability of a local anchor and sensing nodes at the location of interest. Furthermore, the tasks in bubble sensing are participatory (i.e., human) actions, such as "take a photo", and *not* executable code as in PRISM.

Micro-blogs [22] is another system for participatory sensing, where users upload "blogs" annotated with sensed information (e.g., photos) to a micro-blog server. The users' mobile devices also upload their locations to the server periodically; privacy concerns are sidestepped by assuming that users trust the micro-blog service. When another user posts a location-specific query, it is answered either based on information that has already been uploaded or by directing the query to one or more users who are in the desired location.

AnonySense [13] is an alternative framework for sensor tasking and reporting that goes beyond participatory sensing to also encompass opportunistic sensing, wherein the application accomplishes its sensing task by pushing code onto the mobile nodes. However, to ensure safety, the code must be written in a constrained, special-purpose language called AnonyTL, which is in contrast to PRISM, where an application can execute a generic binary on the mobile phone. Each of these approaches has its advantages. Having a special-purpose, interpreted language would ensure safety and often also reduce the size of the application code. On the other hand, executable binaries allow generality, including the flexibilty to reuse existing code modules. For instance, to build the party thermometer application presented in Section 7.2, we were able to reuse an existing Fast Fourier Transform (FFT) module.

AnonySense also lays strong emphasis on privacy. To this end, AnonySense adopts a polling model for task distribution, where each mobile node periodically polls and downloads all tasks from the infrastructure (no filtering is done, say based on location, to avoid any privacy leak). While such a task "pull" approach does not reveal the mobile node's location to the infrastructure, it can be burdensome and wasteful when the number of tasks or clients is large (see Section 8). In contrast, PRISM employs a "push" approach, which allows limited tracking of a mobile node.

Both AnonySense and PRISM suffer from privacy risks arising from the access that applications have to sensor data (e.g., audio). PRISM includes sandbox mechanisms to mitigate these risks.

Sandboxing [35] is a well-known mechanism for securely executing untrusted applications and can be supported using a variety of techniques such as system call interposition [6], virtual machine monitors [42], or capability-based systems [38]. While the PRISM sandbox employs system call interposition, it goes beyond standard software sandboxing by providing mechanisms motivated by the sensing context, specifically to track and control an application's access to sensor data. Given the potential risk to security posed by sensor data, we believe that such mechanisms for control would be needed even if the untrusted application were run within a pocket hypervisor [14].

Another mechanism that the PRISM sandbox includes is energy metering. A fair amount of work has focused on energy monitoring of applications with the goal of allowing them to adapt better to energy usage [20, 28]. There is also work on measuring energy usage in a fine-grained manner on sensor platforms, either using hardware support [40] or by monitoring hardware power states in low-level software (e.g., modified device drivers) [17, 21]. PRISM could leverage such accurate energy monitoring if it were to become available widely on mobile phones. However, our current design focuses on coarser-grained monitoring of energy, which is accurate enough for our purposes (enforcing energy limits on untrusted applications) yet generic enough to run on any mobile smartphone, with some calibration but without requiring low-level support in hardware or software.

There has also been work on programming models that include support for building resource-aware (including energy-aware) applications. Examples include Eon [39] and Pixie [28]. We view this body of work as being orthogonal to our current focus in PRISM, which is on monitoring energy usage and enforcing limits. A future version of PRISM could leverage such advances in energy-aware programming models.

Finally, providing incentive mechanisms for participation is a challenge shared by many community-based sensing applications. Designing appropriate incentive mechanisms is an active area of research [9, 11, 15] and is orthogonal to the research issues addressed by PRISM.

System	Generality	Security	Scalability	Privacy
Bubble-Sensing	No	Yes	Yes	Yes
AnonySense	OK	Yes	No	Yes
Micro-Blog	No	Yes	Yes	No
PRISM	Yes	Yes	Yes	OK

Table 1: PRISM compared to prior work

2.1 Placing PRISM in Context

As discussed above, PRISM builds on a large body of prior work in sensing systems but yet differs from it in various ways. These differences arise from PRISM's goal of providing a *flexible* platform for *participatory* as well as *opportunistic* sensing applications on a *substrate of mobile phones* that are *not dedicated* to sensing. A qualitative comparison of PRISM to other community sensing systems is shown in Table 1. Specifically, PRISM:

- Supports generality by enabling the execution of untrusted application binaries on mobile phones. This is in contrast to systems that only support participatory (i.e., human) sensing tasks or only allow applications written in a restricted, special-purpose language.
- Ensures security by running untrusted applications in a software sandbox that supports novel features such as forced amnesia, sensor taint tracking and access control, and energy metering based on coarse-grain monitoring.
- Employs a push-based model for task dissemination that achieves scalability by sacrificing some privacy (specifically, anonymous users could be tracked over a short interval), relative to a pull-based model [13].

3. PRISM DESIGN

The scenario we envision for PRISM is as follows. Users who would like to participate in and contribute to community sensing install the PRISM runtime on their mobile smartphones and register with the PRISM infrastructure. These phones are then available to run community sensing applications. Note that the PRISM runtime is a middleware that runs on top of an existing mobile phone operating system (OS). We now discuss our assumptions and then outline the design of PRISM.

3.1 Assumptions

We assume that PRISM is trusted in that users are willing to install and run the PRISM runtime on their phones. However, we require that the PRISM infrastructure not be in a position to identify users. Since a user's location itself could compromise their identity in private areas such as homes, we assume that the PRISM runtime on the phones is disabled in all but designated "public" areas.

We assume that entities that submit applications to be run on PRISM have identities certified by a trusted authority. This helps ensure that we are in a position to exercise control over the number of applications submitted to PRISM by any entity. However,

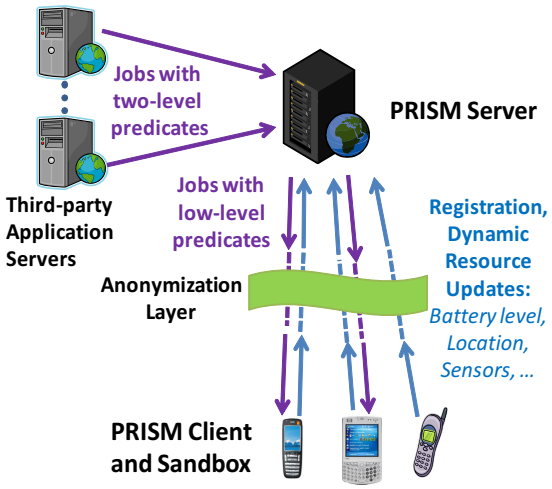


Figure 1: PRISM architecture

we do not require the applications themselves to be certified since certifying application binaries is a hard problem and furthermore expensive, especially in a community sensing context.[1] Protecting the mobile phones that host such untrusted applications is a key task for the PRISM runtime.

We assume that the participating nodes are trusted, which means that the OS running on these phones is trusted and that we can count on standard OS mechanisms such as memory and file system protections to work as intended.

Finally, we assume that each participating phone has wide-area network connectivity (say over a wireless WAN), which allows it to communicate directly with nodes in the infrastructure (see below). In other words, there is no direct peer-to-peer communication between the phones.

3.2 Architecture

The PRISM architecture is shown in Figure 1 and consists of the following three components:

- **Application server (supplied by third parties):** submits jobs to PRISM servers, for deployment onto a desired set of mobile phones
- **PRISM server:** accepts jobs from the application servers and deploys them onto an appropriate set of mobile phones
- **PRISM client and sandbox on mobile:** registers with PRISM servers and supports the execution of the jobs in a specially-designed sandbox

Push-based Model: Applications can be deployed on mobile phones in two ways: a pull-based approach [13] where all mobile clients independently pull/download jobs from a server or a push-based approach where a server pushes jobs to only a desired set of mobile phones. Since PRISM is targeted towards city-scale sensing applications, scalability and efficiency of operation is critical, necessitating our choice of a push-based framework for the PRISM platform.

[1]Apple's iPhone application certification simply ensures that the developer can be identified; all liabilities arising from the application are passed onto the developer.

A push-based model requires some amount of tracking of mobile phones in order to able to "push" applications onto phones. Having a common framework — PRISM — track phone resources on behalf of all applications, has the following benefits:

- **Fast Response:** Tracking phone resources allows PRISM to deploy applications immediately, as and when the desired set of phones are available.
- **Efficiency:** Since phones are potential candidate hosts for multiple applications, PRISM eliminates the need for each application to track phone resources independently.
- **Scalability:** The amount of tracking can be modulated to the load of application arrivals and the density of available phones.

In order to support a push-based model, two key challenges must be addressed by PRISM. First, a generic and flexible application programmer interface (API), exposed by PRISM to applications, needs to be designed so that applications can be effectively pushed to a desired set of mobile phones. Second, an efficient mechanism to continuously track a large number of mobile phones without impacting energy usage on the mobile phones is essential. To address the first challenge, we design a new API with two-level predicates and a choice of two deployment modes. To address the second challenge, we design an adaptive and predictive registration and update mechanism.

Next, we describe the registration mechanism that allows tracking of phone resources (Section 3.3), followed by the API that enables applications to be pushed onto the tracked phones (Section 3.4) and finally our optimizations for making the tracking process efficient (Section 3.5).

3.3 Registration

The *registration* process enables a phone to inform PRISM server of its presence and its availability to run PRISM applications. Registrations are maintained as soft-state and automatically expire after the registration period. We set the registration period to one hour to balance the overhead of the registration process with privacy risks where tracking phones long enough could reveal the identify of the user [23, 24].

The registration includes both static and dynamic resource information. Static information comprises the hardware resources, such as sensors and radios, on the phone available to PRISM. Dynamic information comprises information that is time-varying, specifically the location of the phone and the battery energy remaining. The dynamic information is kept updated at the PRISM server through *update* messages.

Privacy: Note that tracking of phones can be accomplished without significantly weakening privacy, though, PRISM does trade-off some privacy for scalability compared to a pull-based approach like AnonySense [13]. First, PRISM server's ability to track mobile phones is limited to a short registration interval. After the expiry of registration period, phones wait for a random amount of time, picked from an exponential distribution, before registering again. Further, PRISM servers are prevented from tracking phones across registration periods by employing an independent anonymization service [16] (see Figure 1), thus avoiding any tracking through client IP addresses. Finally, privacy can be further strengthened by adopting cloaking techniques presented in [23, 24] such that registrations from any given region are sent to PRISM servers only when the number of registering phones exceeds a given threshold.

3.4 API

The API between the application server and PRISM server is designed to enable the application server to *accurately and quickly identify* the set of mobile phones that can run the application. Accurate identification is enabled by a two-level predicate mechanism while quick deployment is enabled by a choice of deployment modes. We describe these next.

3.4.1 Two-level predicate

The identification of phones is achieved through a two-level predicate-based API: the top-level predicate is coarse-grained and used to identify phones where the jobs are deployed but not activated while the second low-level predicate is fine-grained and decides when the jobs on the phones begin execution.

The top-level predicate specifies the capabilities desired of the phones in terms of sensors, the number of phones needed, and their coarse-grain locations. The number of phones and their locations could be specified in one of two ways: (a) as a list of specific locations and radii, with the desired number of phones at each (e.g., 3 phones in the vicinity of the clock tower and 2 phones near the palace), or (b) as a region, with the desired (uniform) density of phones. Since the mobile component is a binary executable, the application server can either supply binaries for a range of mobile hardware/OS platforms and/or include the desired mobile hardware/OS platform as part of the predicate specification. The low-level predicate is more fine-grained in nature and can consist of locations, specified at finer granularity, or be based on derived attributes, such as speed (e.g., deploy only if phone is moving at pedestrian speed to minimize distraction for human-related queries). The low-level predicate also includes a time-out parameter which determines how long the PRISM client monitors for a match of the fine-grain predicate before cancellation.

Based on early deployment experience of one of our applications, we decided to split predicates into two levels for the following reasons. First, maintaining the PRISM server updated with fine-grain attributes such as precise location or speed can result in prohibitive amount of updates generated from the phone. Second, the two-level mechanism allows applications to be deployed onto phones with ample time ahead of actual execution, side-stepping issues with spotty network connectivity. Thus, application servers can use the top-level predicate with a coarse-grain specification allowing the application to be deployed on potential candidate phones and be ready for execution whenever the precise predicate is matched at a later time. Given that community sensing applications have little control over the sensors' (phone) mobility, a two-level predicate mechanism *ensures that any limited sensing window of opportunity is not missed*. Third, by having the PRISM client track predicates on behalf of all the pushed applications and allowing their execution only when their respective fine-grained low-level predicates are matched, application developers can reduce the risk of spam, especially in cases where the final sensor is the person. Thus, the two-level predicate mechanism *decouples deployment coverage from desired execution coverage*.

Finally, one useful clause as part of the top-level predicate is whether the desired phones are static or mobile. Thus, in the citizen journalist application, the top-level predicate can ensure deployment to only those phones that are recently mobile (maximizing chances of fine-grain predicate being matched), while for the party thermometer application, deployment occurs to only those phones that are deemed static (ensuring that phones just passing-by near the party location are not involved unnecessarily).

3.4.2 Deployment mode

A choice of two deployment modes is available to the application server. These two modes can be used for quick deployment of applications, with different trade-offs.

One option is to use a *deploy-or-cancel* mode. In this mode, the PRISM server deploys the application immediately only if the top-level predicate is matched. An alternative is the *trigger* mode, wherein the application server sets a trigger with the PRISM server for the desired predicate. The triggers are reevaluated continuously as updates are received from the phones, ensuring deployment as soon as the top-level predicate is satisfied.

The deploy-or-cancel mode can be used when a quick deployment is necessary. For example, by specifying a "large" area as part of the top-level predicate, quick deployment can be ensured for critical queries in the citizen journalist application (Section 7.1), albeit at the cost of deployment to a large number of phones. On the other hand, the trigger mode can be used for applications that are targeting regions with low density of PRISM clients. For example, a latency-insensitive query in the citizen journalist application that requires deployment to, say, a low-density rural region, can avail of the trigger mode.

3.5 Update optimization

As mentioned earlier, the mobile client sends update messages to the PRISM server to keep the dynamic information updated. By default, the update messages are sent periodically. Given that these update messages are an overhead, we now discuss two techniques for cutting down on the number of update messages.

Adaptive updates: In this approach, the update frequency is adapted based on the density of phones and the arrival rate of sensing applications. The PRISM server notifies each client with a parameter p at the time of registration, computed as

$$p = min(1, \rho * n/N)$$

where ρ is the job arrival rate, n is the average number of phones requested by a job and N is the total number of phones registered, for each geographic region. The clients then send periodic updates with probability p. The intuition behind the above equation is simple: the update frequency (probability) can be reduced in a given region if either a large number of phones (N) are available to PRISM or there is little application demand ($\rho * n$) for phones.

Prediction-based Suppression: In this approach, the mobile node and the PRISM servers run identical predictors for each dynamic resource of interest. At the mobile end, since the ground truth on the dynamic resource is also known (e.g., the current location or battery energy level), updates are sent to the servers if and only if there is a significant deviation between the ground truth and its local prediction. While such predictors could be very sophisticated in general, we only consider two simple predictors:

1. *Constant Predictor:* This predictor predicts that the new value is the same or "close" to the previous value. We use this predictor for the location resource and evaluate its efficacy in Section 8. For instance, if a phone remains close to user's office for several hours (e.g., during a workday), the constant predictor would suppress all location update messages during this period.

2. *Affine predictor:* This predictor predicts the new value as an affine function of a quantity (e.g., time) that is shared by both the mobile node and the PRISM server, with the previous value being the constant term in the affine function. In other words, $newValue = \alpha * time + previousValue$, assuming that time is the domain of the function. An affine predictor

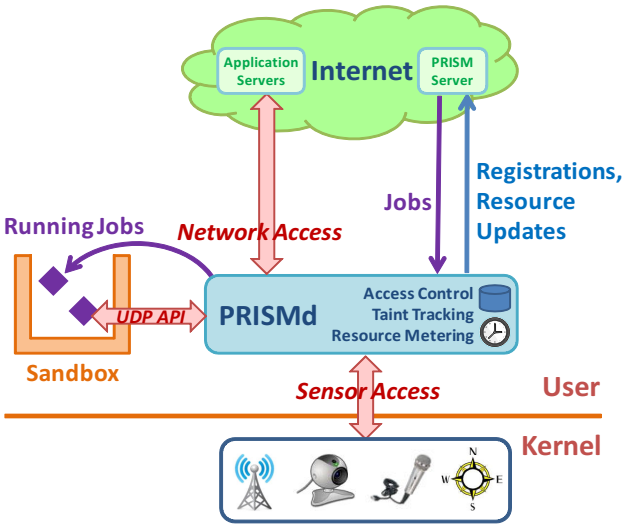


Figure 2: Client Sandbox

would be suitable for the battery energy resource, where exact tracking is not essential.

4. SOFTWARE SANDBOX

We now turn to the mobile phone end of PRISM. The key goal here is to enable the safe execution of untrusted application binaries on a mobile phone. Our basic approach is to run the untrusted application inside a software sandbox. To realize the software sandbox, we use the standard technique of system call interposition [6]. For simplicity, all system calls pertaining to sensor device, file system, and network operations are blocked rather than being modulated inside of the OS kernel. In the place of the blocked system calls, applications use a set of library APIs that interface with a user-level `PRISMd` daemon, which mediates all accesses to the sensor devices, file system, and network. Figure 2 shows the architecture of the software sandbox.

While the software sandbox addresses basic security concerns, there are a number of additional issues that arise from the sensing context. These pertain to security and privacy concerns arising from access to sensitive sensor data and the risk of resource depletion, specifically with regard to battery energy. We focus our discussion here on these novel aspects of the software sandbox.

4.1 Mitigating Privacy Risks

In general, any access to sensor data could pose a threat to user privacy. For instance, access to the microphone would allow recording of private conversations. Access to a seemingly less sensitive sensor such as GPS could also compromise privacy by allowing the (anonymous) user to be tracked and eventually identified [24]. Even access to a seemingly less sensitive sensor such as the WiFi could compromise privacy because of manufacturing artifacts that may allow the device to be fingerprinted.

We argue that such privacy risks are inherent to community sensing applications. If we want to ensure perfect privacy, all access to sensors would have to be blocked. Given this, our goal in PRISM is to mitigate the risks while retaining useful sensing functionality.

One approach to mitigating the risk is "dumbing down" the sensor data that is passed on to the application, say by quantizing it to a coarse-grained level. However, since application needs cannot be anticipated, it is hard to do such dumbing down without impacting generality.

The mechanisms included in PRISM's sandbox are aimed at mitigating privacy risks while still allowing useful sensing functionality. As the first line of defence, sensor access control (Section 4.2.1) gives the user broad control over which sensors, if any, applications may access. When the user permits access to a sensor, PRISM mitigates the risks by constraining the computing and communication that the application may perform, using the resource metering (Section 4.3) and forced amnesia (Section 4.4) mechanisms. These controls are designed to match the requirements of typical community sensing applications, where sensor data is processed and significantly reduced on the mobile node before being transmitted (e.g., noise mapping [7] or pollution monitoring [25]). Where significant amounts of raw data is transmitted, it typically happens with human involvement (e.g., taking a picture or recording an audio clip and then uploading the raw data). Thus, we believe that the combination of tight controls by default and human-controlled overriding, offers the flexibility needed for a range of community sensing applications.

4.2 Regulating Access to Sensors

In PRISM, we use two complementary approaches to regulate access to sensors.

4.2.1 Sensor Access Control

The most direct way of addressing user concerns pertaining to privacy is via *sensor access control*, i.e., blocking access to sensitive sensors (e.g., the microphone). While the sensor access control policies could, in general, be complex, we restrict ourselves here to three simple yet natural policies:

1. *No sensors:* Direct access to all sensors is blocked. While this severely restricts functionality, there are sensing applications that would fit this mould, since sensor information could still be used indirectly as part of the predicate. For example, an application could prompt users at a particular location and have the human users do the sensing (e.g., report back on the food at a restaurant). Note that, while the untrusted application does not have direct access to the location sensor, the PRISM runtime would still have access to location information, thereby allowing the application to target phones in the desired location.

2. *Location only:* Only access to location information (e.g., from GPS) is allowed. This would, for instance, enable a traffic flow monitoring application that requires knowledge of location and derived quantities such as speed.

3. *All sensors:* Access to all sensors (including multimedia sensors such as camera and microphone) is allowed, which provides the maximum flexibility.

An alternative to these coarse-grained access control policies would be to prompt the user and seek authorization for each application that wishes to access a sensor. While offering greater control, this alternative runs the risk of overloading the user.

4.2.2 Sensor Taint Tracking

An alternative to sensor access control is to place severe resource limits on PRISM applications that access sensitive sensors, instead of blocking such accesses entirely. By diminishing an application's ability to process or transmit sensed data, we could significantly diminish privacy risks while providing greater flexibility compared to blocking access to sensors.

Sensor taint tracking is a mechanism to enable the above. Since `PRISMd` mediates accesses made by a PRISM application to all

sensitive resources, it is in a position to track which sensors the application has been *tainted* with. For example, if the application has accessed the microphone, then it is tainted with microphone data; otherwise, it is not. Such taint tracking is coupled with policy information to drive resource metering decisions, as discussed next.

4.3 Resource Metering

Besides the privacy risks of granting untrusted applications access to sensor data, PRISM also has to contend with the security risk of resource depletion by the untrusted applications. A particularly constrained resource is battery energy. It would be unacceptable for the user's voluntary participation in community sensing to cut down the battery life of the user's phone significantly. Resource metering is designed to address the problem of resource depletion and, as our discussion below makes clear, it also helps mitigate privacy risks.

`PRISMd` tracks and enforces limits on the resource usage of PRISM applications. For resources such as network, sensors, and files, which it mediates access to, `PRISMd` is in a position to directly track resource usage and enforce controls, e.g., by not returning the sensed data. For other resources, in particular, CPU and memory, `PRISMd` does *not* mediate access. Instead, it monitors the usage of these resources by the PRISM application using the appropriate system calls and, if any limits have been exceeded, it terminates the application.

From the viewpoint of the user (i.e., the owner of a PRISM node), it is the overall energy consumed by PRISM applications that matters. However, PRISM also imposes per-application limits, both to ensure that a single application does not hog the resources and also to mitigate privacy risks, as we discuss in Section 4.3.2 below.

4.3.1 Energy Metering

Enforcing limits on the use of the individual network and sensor devices is likely to be cumbersome. Instead, we translate the usage of each resource, such as the CPU, network, and sensor, into energy. To accomplish this translation, we use a simple model where the energy consumed is estimated as a linear function [28] of (a) the amount of time that a particular device is active, and (b) the amount of data read and/or written.

The per unit time and per byte energy costs can either be *measured* empirically, through controlled benchmarking, or *estimated* in normal course, using linear regression [5]. For simplicity, our current prototype uses active measurements and we defer automatic estimation based on passive measurements to future work.

We emphasize that our goal here is to prevent runaway applications from depleting battery energy rather than building an accurate software-only energy measurement tool. As such, we use conservative estimates, where appropriate. For example, CPU frequency scaling would impact the energy consumed in a given amount of CPU time. To be conservative, however, we use the energy cost corresponding to the highest clock frequency, although if one could infer the time spent in the various CPU states [28], the estimate would likely be more accurate.

4.3.2 Bandwidth Metering

While network communication is also factored in to the energy computation, we also meter network bandwidth separately for two reasons. First, network communication can incur a monetary cost, in addition to an energy cost, because of service provider tariffs. Second, and more importantly, network access has implications for privacy. For example, an application that is *tainted* with microphone data and also allocated a generous slice of the network bandwidth could simply record a user's private conversations and ship these out, which would clearly be unacceptable from a privacy viewpoint. On the other hand, if the application's access to network bandwidth were severely limited (say to just a few bits per second), then the application could still perform a useful sensing function (e.g., honk detection [31]), while not posing a similar threat to privacy.

Item	Infrastructure	Mobile	Common
PRISM	1901	1436	1197
Sandbox	-	3129	-
CitizenApp	328	330	347
PartyApp	88	200	-
(Off-the-Shelf FFT)	-	356	-
BumpApp	88	448	-

Table 2: Lines of code for various components

4.4 Forced Amnesia

In the example noted above, severely limiting the bandwidth usage of an application would preclude a direct attempt to stream out sensitive information. However, a malicious application could still accumulate some sensitive data (e.g., a 10-second long recording of a user's private conversation) and then dribble it out over the network over an extended period of time (say an hour).

To prevent this and other such "resource accumulation" attacks, `PRISMd` employs a forced amnesia mechanism to wipe out a PRISM application's state periodically, say every minute. This mechanism is inspired by the observation that generally a *sensing* application running on a mobile phone should not need to perform long-running computations. Rather, the typical application would sense data, possibly perform some local processing aimed at data reduction, and then ship condensed information back to the application server.

In practice, wiping out the application's state can be achieved effectively, even if not very elegantly, by terminating the application and then restarting it within a fresh sandbox environment. Since all of a PRISM application's network communication is routed via `PRISMd`, such an application restart will not disrupt WAN connections to the application server.

5. IMPLEMENTATION

We now discuss our implementation of PRISM. We focus primarily on the various quirks we encountered and on shortcomings of our implementation. Also, we focus on the PRISM platform itself here, deferring discussion of three applications we have prototyped on PRISM to Section 7.

5.1 Computing Platform

We use Microsoft Windows Mobile 5.0 (WM 5.0) and 6.1 (WM 6.1) as the OS platforms for our mobile phone implementation. The infrastructure components run on a Windows 7 PC.

Our testbed comprises 15 smartphones: 4 HP iPAQ hw6965 running WM 5.0 and 8 Samsung SGH-i780, 2 HTC Advantage 7501, and 1 HTC Advantage 7510, each running WM 6.1. Each of these phones includes the microphone, camera, and GPS sensors. Each of the 3 HTC Advantages also has an external accelerometer sensor attached to it.

Each of the 15 phones has Bluetooth, 802.11b, and GPRS/ EDGE/ 3G radios. However, we used the GPRS/ EDGE/ 3G radio on each phone for all network communication, with network access being split across three service providers.

All of our implementation is in C#, except for some of the sandbox-related components, which are written in C++ and the FFT code

which is written in C. Table 2 shows the lines of code for the different components.

5.2 PRISM: Infrastructural Component

We have prototyped the PRISM server application. The PRISM server supports the two-level predicate-based API and the deploy-or-cancel and trigger modes (Section 3.4). The top-level predicate also allows application to specify whether the desired set of phones are static or mobile; a phone is determined to be static if it does not have a GPS lock (e.g., indoors) or if it has not updated the server with a new GPS location (due to update suppression, Section 3.5) for an application-specified time interval.

5.3 PRISM: Mobile Phone Component

The mobile phone component comprises the software sandbox, which includes implementations of `PRISMd` and the system call interposition layer (shim layer).

When a process makes a system call, the shim layer checks to see if the process is a PRISM application, a determination that is made based on the parent process ID check. If it is, then system call interposition is applied to block the following calls, except where noted otherwise:

- *Network communication*: all calls are blocked except for sends to and receive from the `localhost:9500` UDP port, which corresponds to `PRISMd`.
- *Device access*: e.g., `ioctl`
- *File system*: for efficiency reasons, we only shim and block calls that *return* a handle (e.g., `CreateFile()`). Blocking calls that merely *use* a handle (e.g., `FileRead()`) is unnecessary since the sandboxed process would not be able to obtain a valid handle in the first place.
- *Registry access*
- *Process calls*: e.g., spawning off child processes is disallowed, to avoid complicating the task of monitoring the resource usage of an application.

The implementation of `PRISMd` has to contend with several issues pertaining to resource monitoring. First, once a process terminates, information on its CPU usage is no longer available. Hence `PRISMd` samples the CPU usage of a PRISM application process periodically (once every 2 seconds, by default) and, when the process terminates, it makes a conservative choice by adding the inter-sample interval (e.g., 2 seconds) to the CPU usage of the terminated process.

Second, Windows Mobile does not provide a way to directly determine the memory usage of a process. In our current implementation, `PRISMd` traverses the page tables periodically to add up the total memory usage of a process. A more efficient alternative would be to shim the memory allocation and deallocation system calls, and include hooks to keep track of the total memory usage of a process. In our current implementation, however, we have chosen to keep the in-kernel shimming layer very simple, hence it does not include support for such accounting.

Third, when `PRISMd` receives a resource access request from a process over its UDP socket, it would need to know the ID of the requesting process in order to perform access control and resource accounting on a per-application basis. To facilitate this, the system call shim layer includes the ID of the requesting process with the message while intercepting inter-process communication over UDP between a PRISM application and `PRISMd`.

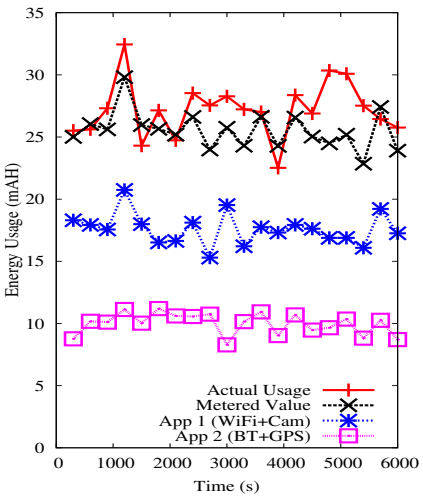


Figure 3: Energy metering

6. EVALUATION: MICRO-BENCHMARKS

We start our experimental evaluation of PRISM by presenting results from micro-benchmarks run in the lab.

6.1 Energy Metering

First, we evaluate the efficacy of energy metering performed by the PRISM sandbox. Recall from Section 4.3 that energy consumption is estimated by tracking the extent to which each resource (e.g., sensor) is used and feeding this into a simple linear model.

In our experiment, we emulate two applications. One application cycles through using the camera sensor, performing WiFi scans, and remaining idle, in turn. Each phase lasts for a randomly chosen length of time. The other application operates similarly except that it uses the GPS sensor and performs Bluetooth scans instead of the camera and WiFi operations. We run these applications on an HP iPAQ hw6965 phone, to emulate two concurrently running PRISM applications.

PRISM separately estimates the energy consumption of each of these two applications using the linear model, based on the time (measured by `PRISMd`) that each sensor is accessed by each application and the empirically-measured unit energy cost of each sensor. The sum of the estimated energy consumption of the two applications yields an estimate of the total energy consumption. We also record the ground truth for the total energy consumption, using the GetSystemPowerStatusEx2() API in Windows Mobile to measure the voltage and the average current drawn. (Note that the system does not provide us a way to establish the ground truth on a per-application basis.)

Figure 3 shows the estimated (i.e., metered) energy costs of each of the two applications, the estimated total, and also the ground truth for the total. Each sample point is obtained over a 5-minute bucket. We see that the total metered value tracks the ground truth closely for the most part. However, the metered value undershoots at times because of the cost of non-PRISM related activity on the phone, which PRISM's metering mechanism does not account for.

We conclude that PRISM's simple approach to energy estimation is adequate for the purposes of enforcing limits on the energy consumption of PRISM applications.

6.2 Overhead of `PRISMd` Mediation

Next, we turn to quantifying the overhead of having `PRISMd` mediate all accesses to sensors. Such mediation imposes overhead

	Direct	Via PRISMd	Overhead
GPS	804.3 mW	821.2 mW	2.10%
Mic	312.6 mW	315.0 mW	0.76%

Table 3: Average power drawn with direct access to sensors versus access that is mediated by PRISMd

in terms of context switching between the PRISM application process and PRISMd, and data copies involved in UDP-based interprocess communication between the two processes.

To quantify the overhead, we perform micro-benchmarks to compare the energy consumed when an application accesses sensor data via PRISMd versus when it accesses it through a direct system call. We perform these measurements for two sensors: GPS and microphone. In the case of GPS, the application alternates between looking up the GPS lat/long information and sleeping for 1 second; each location report is 20 bytes in size. In the case of the microphone, the application alternates between obtaining a 1-second long 8-bit PCM audio sample and sleeping for 1 second; each audio sample is 22 KB in size.

We use a hardware energy meter connected to an HP iPAQ hw6965 phone, to accurately measure the average power drawn by the phone during each experiment. The results are summarized in Table 3. We find that the overhead of mediation by PRISMd is low. This lends support to PRISM's choice of encapsulating sensor access control and tracking in a separate user-level daemon rather than the more efficient but also more complex alternative of stuffing this functionality into the in-kernel system call interposition layer.

7. EVALUATION: APPLICATIONS BUILT ON PRISM

We have prototyped three simple applications on the PRISM platform. Our goal is to demonstrate the flexibility of the platform rather than present novel applications. Our deployment thus far has been on a very small scale, limited to the 15 phones in our testbed and a handful of volunteers.

7.1 Citizen Journalist

This application is inspired by Micro-Blogs [22] and involves *participatory* sensing, wherein PRISM provides location-based triggers to alert human users, who are in the vicinity of a location of interest, to respond to the application. Responses could take the form of answering simple queries, taking pictures of interesting events, etc.

Such an application could be used, for example, by small news organizations to collect information for particular events of interest. In this context, two types of application requirements exist: 1) critical queries where fast response to application queries is necessary (e.g., someone reports an accident and the news organization requires a citizen to take a photograph of the scene); 2) queries that are not latency sensitive but may need answers from areas that are sparsely populated (e.g., someone writing an article about the condition of school buildings in remote villages and requires a recent photograph of the same).

The application requests PRISM to deliver the sensing task to a certain number of camera-equipped phones in the vicinity of the desired location. The location is specified by (lat, long) and includes a coarse radius for deployment and a fine radius for actual execution. If matching phones are not readily available, PRISM's trigger mechanism is used to deploy at the location as and when PRISM clients register/send updates from the desired location.

Figure 4 shows the pseudocode for the distributed aspects of the application. The simplicity of the pseudocode is striking. Indeed, we believe that the PRISM infrastructure relieves the programmer of the details of distributing their application, thereby letting them focus on the core application tasks — local processing on the phones (e.g., image capture, GUI) and centralized processing on the application server (e.g., collating all of the queries/pictures received).

```
// set up the first level coarse-grained predicate
L1pred = new FirstLevelPredicate();
L1pred.location = <desired location>;
L1pred.radius = <desired coarse radius>;
L1pred.stationary = false;
L1pred.cameraPresent = true;
L1pred.numOfPhones = <desired number of phones>;

// set up the second level fine-grained predicate
L2pred = new SecondLevelPredicate();
L2pred.location = <desired location>;
L2pred.radius = <desired fine radius>;

// set up the application with the predicates
PRSIMapp = new PRISMApplication();
PRSIMapp.Init();
PRISMapp.SetPredicates(L1pred, L2pred);
PRISMapp.SetBinary(<path to 'phoneapp.exe'>);
PRISMapp.TriggerMode = true;
PRISMapp.DistributeToPhones();

// read and process data sent by phones
while (appData = PRISMapp.GetData()) {
    <process the received data>;
}
```

Figure 4: Pseudocode for Citizen Journalist Application

Coarse-grain Radius →	30m		75m		125m	
Network →	2G	3G	2G	3G	2G	3G
User Speed ↓						
Walking (4kmph)	5/5	5/5	5/5	5/5		
Driving (30kmph)			5/5	5/5		
Driving (40kmph)				5/5	2/5	5/5
Driving (50kmph)				3/5		5/5

Table 4: Micro-benchmark results: success rate of application launch with 30m fine-grain radius against varying coarse radii, user speed and network type. Number x/y indicates x successful launches out of y trials. Cells colored black indicates no success, gray indicates partial success and white indicates complete success.

Micro-benchmark: Recall that when a PRISM phone with the appropriate sensors is within the identified coarse-grain radius of an application predicate, the application is *deployed* onto the phone and only when the phone enters the fine-grain radius, the application is *launched* for execution. We first conduct experiments to quantify the impact of the choice of coarse-grain radius on successful deployment and launch of PRISM applications for different user mobility speeds and network types (2G vs 3G). We use the citizen journalist application (the size of the executable was about 35KB) for these experiments and set the fine-grain radius to 30m around a chosen center of interest.

Table 4 summarizes the results of our experiments. From the results, we make the following observations. First, at walking speeds, the application was successfully launched within the fine-grain radius for all choices of coarse-grain radius (30m, 75m, and 125m) and networks (2G, 3G). However, we would like the launch to oc-

Item	Count
Deployed	417
Launched	274
Total Responses	235
Response Time in seconds (avg., max)	46, 149
Photo Responses	141
Total Cancelled	38
Cancelled (TooFarAway)	9
Normalized Deployed Distance (avg., max)	71%, 443%
Normalized Launched Distance (avg., max)	83%, 100%

Table 5: Statistics from the pilot deployment

cur when the user is approaching, rather than receding from the point of interest, to notify the user of a sensing opportunity sufficiently in advance. For 2G networks with 30m coarse-grain radius, we found that most of the application launches occurred beyond the center point of interest. This indicates that for pedestrian users on 2G cellular networks, a larger coarse-grain radius (e.g., 75m) is required. Second, as expected, for a given coarse-grain radius and user speed, the success rate in 3G networks is higher than in 2G networks. This is because the lower latency and higher bandwidth of 3G networks allow for a faster deployment of the application than on 2G networks. Third, as the user speed increases, a larger coarse-grain radius becomes necessary for achieving successful application launches. These experiments validate the benefits of two-level predicates for successful application launches while catering to a range of user speeds and network types.

Small-scale pilot deployment: The citizen journalist application was deployed on a small-scale using ten volunteers, including three authors of the paper. The volunteers were given windows mobile phones with GPRS (2G) data subscription and they carried the phones whenever they left work (e.g., to go home or for walks).

A total of 30 locations of interest was identified in an area of few square kilometer in the vicinity of the Microsoft Research India lab in Bangalore. Custom tasks seeking responses from users at these locations, were generated periodically by the application server and sent to the PRISM server for deployment. For example, a task would ask the user how heavy the traffic is at an intersection and also ask them to optionally take a picture. Given that the speed limit in the area of interest was 30kmph and many volunteers used the system at walking speeds, a fine-grain radius of 30m and a coarse-grain radius of 75m were chosen for vast majority of the tasks. These choices ensure a high application launch success rate with ample notification time based on the micro-benchmark results reported earlier.

When the application launches based on the coarse-grained and fine-grained predicates, the user is notified of the launch by generating a ringtone on the phone. If the user chooses to ignore it, then the application will get cancelled automatically based on a timeout period. Otherwise, the user has the option of responding and performing the requested task, or manually cancelling the application. In the latter case, the user can optionally provide a reason for cancellation (e.g., too busy or location too far away).

Key takeaways from the pilot: Table 5 presents several statistics from the pilot. A total of over 400 application instances were deployed out of which 274 were launched and 235 responses, majority of which included a photo attachment, were received during the trial. The average response time for applications where phones matching the top-level predicate were immediately available was 46 seconds (including 10 seconds of deployment delay over GPRS), demonstrating the value of a push-based framework such as PRISM.

1. Value of two-level predicates: As mentioned earlier, for most

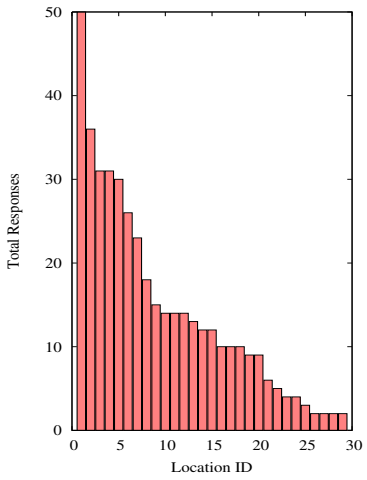


Figure 5: Responses to applications indexed by locations

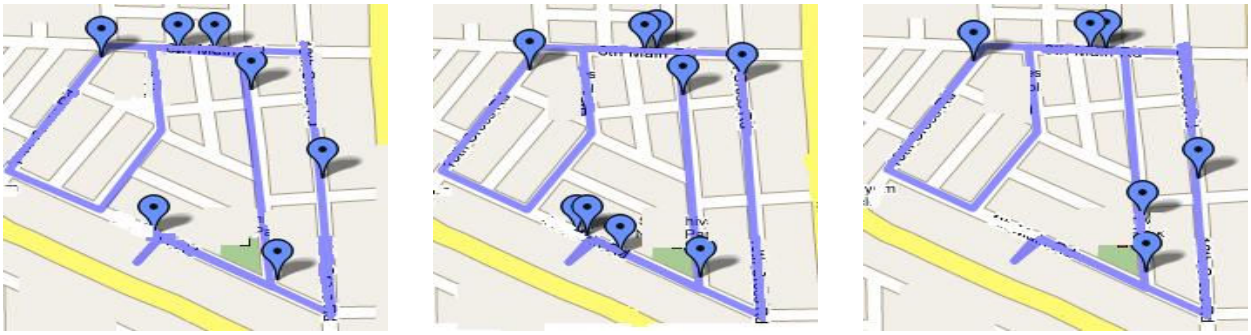
regions of interest, we set the fine-grain radius to 30m and coarse-grain radius to 75m. For a few regions of interest that spanned a large area, the fine-grain radius was set to 50m and the coarse-grain radius to 100m.

Table 5 shows the average and maximum value of the normalized distance at which the applications were deployed and launched (relative to the specified radii). The average normalized deployed distance is only 71% of the coarse radius. This is due to the lack of precise GPS information at the server and the time taken to download the application through GPRS. Thus, the application is deployed well inside the coarse radius. Instead of using two-level predicates, if we had restricted ourselves to a single-level predicate that specifically targets the small precise region of interest, the deployments would often have been too late to be of value, as discussed in the micro-benchmark results.

The maximum normalized deployment distance is over 4 times the coarse-grain radius. The reason for this discrepancy is because of the lag between the location that has been registered at the server and the precise current location of the mobile phone. However, since the applications are launched only after the fine-grain predicate check, these deployments do not impose any cognitive load on the user. In contrast to the deployment, the launch distance is more tightly controlled, the average and maximum normalized launch distance being 83% and 100% of the specified fine radius. This again demonstrates the value of the two-level predicates because the actual launch is initiated locally on the phone with full and precise knowledge of the location by the PRISM client.

2. Need for tight integration with maps: Out of the 38 instances of application launches that were cancelled, users had indicated that 9 of the requests were too far away and thus, these launches could be interpret as spam. However, from the launch distance statistic in Table 5, we see that the application never launched outside the fine-grained predicate. Upon further investigation, users revealed that they were either on an adjacent street behind the point of interest or they were moving away from the point of interest when the application launched. This clearly indicates that tight integration with maps (streets, direction of traffic, etc.) and heading information would help reduce such unintended spam possibilities.

3. Trigger mode Figure 5 depicts the total user responses to applications indexed by locations over the duration of the pilot which lasted approximately one week. From the graph, it is clear that jobs posted to a few locations received a large number of responses (popular locations visited by many volunteers) while a few loca-



(a) Ground truth (b) Bumps detected without forced amnesia (c) Bumps detected with forced amnesia

Figure 6: Pushpins marking road bumps detected on a 2.5 km long drive.

tions received hardly any responses. The latter locations are prime candidates for the trigger mode of application deployment.

Finally, we relate an interesting anecdote. One of the volunteers was quite pleased with the way the system worked and requested us to post a query task at a new location on her behalf. The query was to determine if a certain product was available at a store. She was surprised to find her query answered the following day (by another volunteer who was unaware of this recently created task). This anecdote points to the value of a community sensing system in general; we believe the flexibility and scalability of PRISM will make the deployment of such community sensing systems practical.

7.2 Party Thermometer

The second application is also a human-query application, where queries are directed to users who are at parties. For example, a query could be how hot a particular party is. Like in the citizen journalist application, location is a key part of the predicate used to target the queries. However, unlike in the citizen journalist application, location alone is not enough for targeting because there is a significant difference between a person who is actually in a party and a person who is just outside, possibly having nothing to do with the party. Thus, in addition to location, we employ (party) music detection using the microphone sensor to establish the user's context more precisely.

The location predicates used in the party thermometer application must be more precise than, say, in the citizen journalist application since the application has to perform an energy intensive activity (detecting music through microphone sensing) before even determining whether to involve the user. Avoiding such unnecessary sensing would be essential for efficiency. To enable a precise fine-grain predicate, we choose the location of the party down to a building as the top-level predicate and further require that the phone be stationary (e.g., within the building) before deploying applications to the phone. The latter check helps avoid deploying to phones of users that are simply passing by.

Once the application is deployed, the second level predicate requires music to be heard for the application to launch, thus ensuring that the user is prompted only when he is present in the party. To detect music, one simple heuristic is to perform a Fast Fourier Transform (FFT) of the audio samples and examine the spikes in the frequency domain for harmonics. Since we require this operation to be done efficiently, we use an off-the-shelf efficient FFT code written in C to build a dynamic linked library that is downloaded with the party application. This demonstrates the generality and flexibility of the PRISM platform; applications deployed on platforms that only supports a scripting language such as AnonySense [13] will be forced to wait for the scripting language to support such functionality.

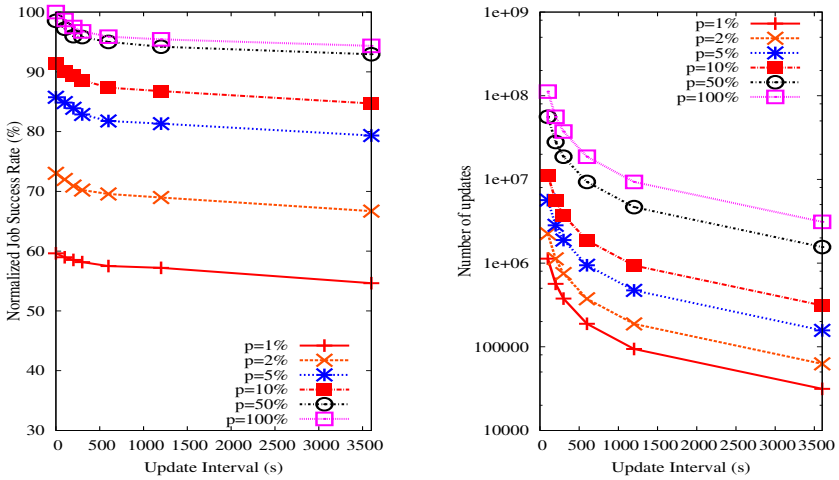
We built and tested this application. We verified that the application was deployed only to users' phones inside the predefined party location and not to phones with users that are merely passing by the location. In our limited testing, while the above music detection heuristic worked reliably, despite the constraint of the forced amnesia interval of one minute, we did not do extensive testing of this aspect since this is an application specific feature and is orthogonal to the capabilities of PRISM.

7.3 Road Bump Monitoring

The final application is inspired by Pothole Patrol [18] and Nericell [31] and involves *opportunistic* sensing, wherein phones equipped with GPS and accelerometer sensors are used to detect and locate road bumps automatically without any user involvement. Compared to the citizen journalist application, the region of interest here is large (e.g., a section of a city or even an entire city), so the application server uses the deploy-or-cancel mode rather than the trigger mode. Also, the sensed (accelerometer) data is processed locally on the phones to extract the desired information (the location of road-bumps), before it is shipped back to the server.

We use this application to quantify the potential drawback, if any, of the forced amnesia feature of PRISM for sensing applications. To evaluate this application, we conducted an experiment where the 3 accelerometer-equipped phones at our disposal were taken on a 2.5 km long drive through a neighbourhood. We established the ground truth by manually recording the actual locations of the road bumps. We then opportunistically deployed the road bump monitoring application on the phones and compared the bumps detected by the application with the ground truth. Figure 6 shows the results. We find that of the 9 bumps detected by the application, 6 match bumps in the ground truth set within 12m (Figures 6(a) and (b)).

We also turned on forced amnesia on one of the phones, which meant that the application was terminated every 60 seconds and then restarted immediately, a process that took about 5 seconds. As a result, the application running on this phone found fewer bumps — 6 bumps out of which 4 matched the ground truth (Figures 6(a) and (c)). However, even with forced amnesia, the missed sensing opportunities on any one phone are likely to be masked by the aggregate sensing performed by a large population of phones for this application.



(a) Normalized Job Success Rate (b) Job Updates

Figure 7: Impact of periodic update interval

8. EVALUATION AT SCALE USING SIMULATION

Finally, we turn to simulation to evaluate PRISM at scales far larger than is permitted by our limited deployment. Specifically, we focus on evaluating the efficiency of PRISM's update strategies (Section 3.5). The key metric to be optimized is the number of dynamic resource updates generated by the participating phones. Frequent updates ensure that the information at the PRISM server is current, resulting in better success of matching incoming jobs to phones. However, each update also adds to the overhead in terms of energy and bandwidth usage on the phones, and processing at the server.

The PRISM server is responsible for balancing between efficiency in resource updates and success in satisfying the needs of various applications. We use two metrics to quantify this trade-off: (1) *total number of resource updates* and (2) *normalized job success rate*, i.e., the percentage of arriving jobs that find matching phones right away, normalized to the case where PRISM server has perfect knowledge. In this evaluation, we focus on location as the dynamic resource of interest. In addition, we present a comparison of push-based PRISM with a pull-based architecture such as AnonySense. For this evaluation, we use a third metric: *number of jobs downloaded per phone*.

In order to evaluate the performance of the update optimizations in a wide-area setting, we use a custom simulator that is driven by large-scale mobility traces from [34]. The trace represents about 260,000 vehicles on real roads in an area of around 250 km X 260 km in the canton of Zurich, Switzerland. We assume that each vehicle carries one smartphone, which is PRISM-enabled with a probability that we vary in the range 1-100%. The mobility trace is 24 hours in duration and contains over 27 million mobility events. We use the first few hours of the trace to warm up the simulator and then evaluate our metrics on about 14 hours of the trace, spanning the hours from 7AM to 9PM.

Motivated by the citizen journalist application, we model sensing jobs as seeking between 1 and 10 phones, uniformly distributed within a radius of 1 km of the target location. The target location is chosen randomly from the 250 km X 260 km mobility region, weighted based on the density of phones. The job arrival process is assumed to be Poisson and each job is set to expire one hour after its arrival.

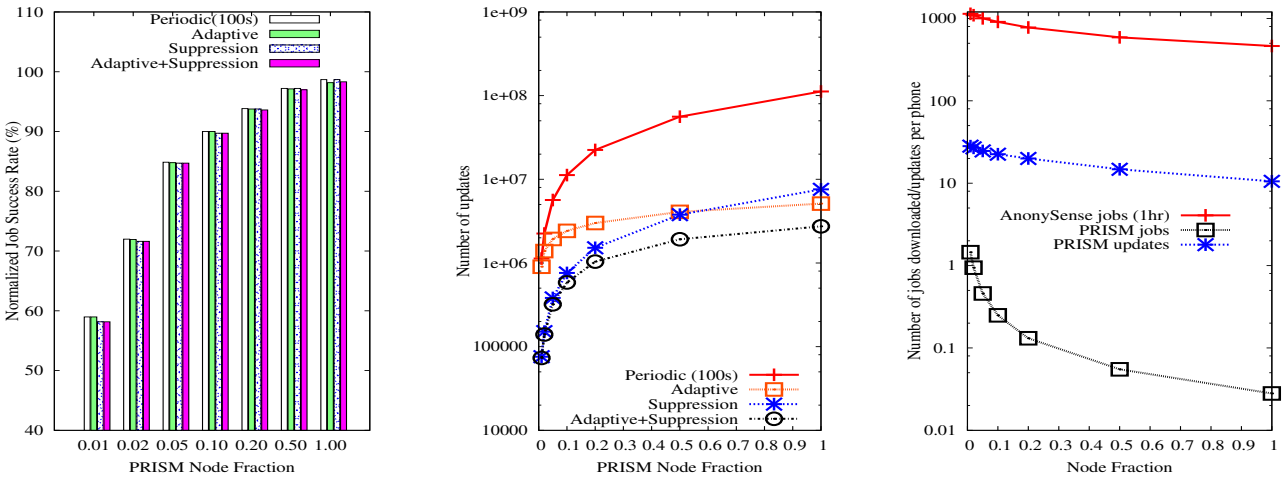
Before we evaluate the adaptive and predictive suppression algorithms (Section 3.5), let us first consider the simple periodic update policy. Figures 7(a) and (b) show the impact of the periodic update interval on the normalized job success rate and total number of updates, respectively. The individual curves in each figure correspond to different percentages (1-100%) of all phones being PRISM-enabled.

From these figures, we make the following observations. As expected, the smaller the update interval, the more current the information at the server and hence the greater the success rate in matching incoming jobs to phones (update interval of zero corresponds to perfect knowledge). On the other hand, a smaller update interval means a larger volume of updates. The success rate with an update interval of 100 seconds is within 2% of the optimal. However, the volume of updates is quite high, an issue we consider next.

With the update interval set to 100 seconds, we evaluate the effectiveness of the adaptive and predictive optimizations described in Section 3.5 in cutting down the volume of updates. Recall that the adaptive optimization means that the higher the density of phones in a region, the less frequent the updates from those phones. On the other hand, the predictive optimization suppresses updates if the resource of interest has not deviated too far from the prediction. Specifically, for the location resource, we use the constant predictor and a threshold of 500m for "too far".

Figures 8(a) and (b) show the impact of the adaptive and predictive optimizations on the normalized job success rate and total number of updates, respectively. The figures show the baseline (periodic updates with a 100-second interval) plus 3 other cases: adaptive only, predictive suppression only, and both the adaptive and predictive suppression optimizations. From Figure 8(a), we see that the success rate is not significantly impacted, with reductions of at most 2% due to the optimizations. However, Figure 8(b) shows that the impact of the optimizations on the volume of updates is dramatic. Predictive suppression alone cuts down the number of updates by an order of magnitude. The adaptive optimization also offers similar gains when the phone density is high enough. Finally, when both adaptive and predictive optimizations are applied, the volume of updates is *cut down by a factor of up to 40*.

Finally, we compare a pull-based approach to distributing jobs



(a) Normalized Job Success Rate (b) Job Updates (c) AnonySense vs PRISM

Figure 8: Impact of adaptive and predictive suppression optimizations, and (c) comparison with pull-based AnonySense

(as in AnonySense [13]) with a push-based approach (as in PRISM). In the case of AnonySense, we assume that clients pull all unexpired jobs once every hour, assuming that jobs can tolerate an average latency of half-an-hour. In the case of PRISM, we use the adaptive and predictive optimizations for updates and push jobs immediately, if possible, to the phones that match the desired predicates of the application. Figure 8(c) presents, on a log scale, the number of jobs or updates per phone for AnonySense and PRISM for the case where the average job arrival rate is 100 jobs per hour. First, it is clear that the number of jobs pushed in PRISM is several orders of magnitude lower than the number of jobs pulled in AnonySense. This comes at the expense of increased updates per phone in PRISM. Second, as the number of available phones increase, PRISM results in significantly fewer jobs or updates per phone but the reduction in number of jobs downloaded per phone is marginal with AnonySense. This is a potential scalability issue for the infrastructure. To place these results in context, AnonySense was designed primarily with a focus on providing strong privacy properties rather than on scaling. However, we argue that, by trading off a little privacy (i.e., allowing phones to be tracked within a registration interval), the push-based architecture of PRISM is able to achieve significant improvements in scalability.

9. CONCLUSION

In this paper, we have presented PRISM, a platform for supporting the growing class of mobile smartphone based participatory and opportunistic sensing applications. The key focus of PRISM is on flexibility in terms of supporting a large class of applications, easing both their development and their deployment. The flexibility that PRISM seeks to provide raises a number of challenges, specifically with regard to security, scalability, and resource exhaustion concerns on what remain phones belonging to individual users.

PRISM is designed to address the challenges noted above. It includes both an infrastructural component to effect distributed orchestration of phones, and a mobile phone component that provides a software sandbox, with several novel features, for executing untrusted sensing applications in binary form. Our evaluation of PRISM is based on three applications that we have prototyped and deployed on a small scale as well as laboratory experiments and simulations.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Rick Han, for their insightful comments. We also acknowledge Jay Taneja and Pei Zhang for discussions during the early stages of the project.

10. REFERENCES

[1] 4.1 Billion Mobile Subscribers Worldwide Helps Reduce Digital Divide (Slightly), March 2009. www.moconews.net/entry/419-4.1-billion-mobile-subscribers-mobile-helping-reduce-digital-divide-sli/.

[2] Worldwide Mobile Phone Sales in 2009, February 2010. http://www.gartner.com/it/page.jsp?id=1306513.

[3] T. Abdelzaher et al. Mobiscopes for human spaces. *Pervasive Computing, IEEE*, 6(2):20–29, 2007.

[4] E. Agapie et al. Seeing our signals: Combining location traces and web-based models for personal discovery. In *HotMobile*, 2008.

[5] G. Ananthanarayanan et al. COMBINE: Leveraging the Power of Wireless Peers through Collaborative Downloading. In *ACM MobiSys*, 2008.

[6] A. Berman, V. Bourassa, and E. Selberg. TRON: process-specific file protection for the UNIX operating system. In *USENIX ATC*, 1995.

[7] M. Bilandzic et al. Laermometer - a mobile noise mapping application. In *NordiCHI 2008*, 2008.

[8] D. Burke et al. Participatory Sensing. In *World Sensor Web Workshop*, 2006.

[9] L. Buttyan and J.-P. Hubaux. *Security and Cooperation in Wireless Networks*. Cambridge Univ. Press, 2007.

[10] A. Campbell et al. The Rise of People-Centric Sensing. *IEEE Internet Computing*, Jul/Aug 2008.

[11] L. Cheng, C. Chen, J. Ma, and Y. Chen. A group-level incentive scheme for data collection in wireless sensor networks. In *"IEEE Conference on Consumer Communications and Networking Conference"*, 2009.

[12] Come out and Play. http://www.comeoutandplay.org/.

[13] C. Cornelius et al. AnonySense: Privacy-Aware People-Centric Sensing. In *ACM MobiSys*, 2008.

[14] L. P. Cox and P. M. Chen. Pocket Hypervisors: Opportunities and Challenges. In *HotMobile*, February 2007.

[15] L. Deng and L. Cox. Livecompare: Grocery bargain hunting through participatory sensing. In *HotMobile*, 2009.

[16] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. of the USENIX Security Symposium*, Aug 2004.

[17] A. Dunkels, F. ï£¡sterlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *EmNets*, 2007.

[18] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *MobiSys*, 2008.

[19] E. Ertin, A. Arora, R. Ramnath, and M. Nesterenko. Kansei: A Testbed for Sensing at Scale. In *IPSN/SPOTS*, 2006.

[20] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-Tuned Remote Execution for Pervasive Computing. In *HotOS*, 2001.

[21] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking Energy in Networked Embedded Systems. In *OSDI*, 2008.

[22] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-Blog: Sharing and Querying Content through Mobile Phones and Social Participation. In *MobiSys*, 2008.

[23] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Preserving privacy in gps traces via density-aware path cloaking. In *ACM CCS*, 2007.

[24] B. Hoh et al. Virtual trip lines for distributed privacy-preserving traffic monitoring. In *MobiSys*, 2008.

[25] R. Honicky, E. Brewer, E. Paulos, and R. White. N-smarts: Networked suite of mobile atmospheric real-time sensors. In *NSDR*, 2008.

[26] A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Toward Community Sensing. In *IPSN*, 2008.

[27] N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo, and A. T. Campbell. Urban Sensing Systems: Opportunistic or Participatory? In *HotMobile*, 2008.

[28] K. Lorincz et al. Resource aware programming in the pixie os. In *ACM SenSys*, 2008.

[29] H. Lu, N. D. Lane, S. B. Eisenman, and A. T. Campbell. Bubble-sensing: A new paradigm for binding a sensing task to the physical world using mobile phones. In *Intl. Workshop on Mobile Devices and Urban Sensing*, 2008.

[30] E. Miluzzo et al. Mobile social networks: The design, implementation and evaluation of the cenceme application. In *ACM SenSys*, 2008.

[31] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *SenSys*, 2008.

[32] M. Mun et al. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *MobiSys*, 2009.

[33] R. N. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers, and M. Welsh. CitySense: An Urban-Scale Wireless Sensor Network and Testbed. In *IEEE International Conference on Technologies for Homeland Security*, May 2008.

[34] V. Naumov, R. Baumann, and T. Gross. An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces. In *Mobihoc*, 2006.

[35] D. S. Peterson, M. Bishop, and R. Pandey. A flexible containment mechanism for executing untrusted code. In *USENIX Security*, August 2002.

[36] Piggyback Mobile. http://www.piggybackmobile.com/.

[37] N. Ramanathan, S. Reddy, J. Burke, and D. Estrin. Participatory sensing for project surya. In *SenSys 2007 Workshop on Sensing on Everyday Mobile Phones in Support of Participatory Research*, 2007.

[38] J. S. Shapiro, J. M. Smith, and D. J. Farber. Eros: a fast capability system. In *SOSP*, 1999.

[39] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. Berger. Eon: A Language and Runtime System for Perpetual Systems. In *SenSys*, 2008.

[40] T. Stathopoulos, D. McIntire, and W. J. Kaiser. The Energy Endoscope: Real-Time Detailed Energy Accounting for Wireless Sensor Nodes. In *IPSN*, 2008.

[41] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *IPSN/SPOTS*, 2005.

[42] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the Denali isolation kernel. In *OSDI*, 2002.