AnonySense: Privacy-Aware People-Centric Sensing

Cory Cornelius, Apu Kapadia,
David Kotz, Dan Peebles, Minho Shin
Institute for Security Technology Studies
Dartmouth College, USA

Nikos Triandopoulos
Department of Computer Science
University of Aarhus, Denmark

ABSTRACT

Personal mobile devices are increasingly equipped with the capability to sense the physical world (through cameras, microphones, and accelerometers, for example) and the network world (with Wi-Fi and Bluetooth interfaces). Such devices offer many new opportunities for cooperative sensing applications. For example, users' mobile phones may contribute data to community-oriented information services, from city-wide pollution monitoring to enterprise-wide detection of unauthorized Wi-Fi access points. This people-centric mobile-sensing model introduces a new security challenge in the design of mobile systems: protecting the privacy of participants while allowing their devices to reliably contribute high-quality data to these large-scale applications.

We describe AnonySense, a privacy-aware architecture for realizing pervasive applications based on collaborative, opportunistic sensing by personal mobile devices. AnonySense allows applications to submit sensing *tasks* that will be distributed across anonymous participating mobile devices, later receiving verified, yet anonymized, sensor data *reports* back from the field, thus providing the first secure implementation of this participatory sensing model. We describe our trust model, and the security properties that drove the design of the AnonySense system. We evaluate our prototype implementation through experiments that indicate the feasibility of this approach, and through two applications: a Wi-Fi rogue access point detector and a lost-object finder.

Categories and Subject Descriptors

C.2.4 [**Communications**]: Distributed Systems; D.4.6 [**Operating Systems**]: Security and Protection; H.3.4 [**Information Systems**]: Systems and Software

General Terms

Design, Experimentation, Measurement, Security

Keywords

mobile sensing, opportunistic sensing, privacy, anonymity

MobiSys'08, June 17–20, 2008, Breckenridge, Colorado, USA.

1. INTRODUCTION

Opportunistic sensing has been gaining popularity, with several systems and applications being proposed to leverage users' mobile devices to collectively measure environmental data, sometimes used as *context* in pervasive-computing applications. In these systems, applications can task mobile nodes (such as a user's sensor-equipped mobile phone or vehicle) in a target region to report context information from their vicinity. In this model, the system opportunistically hands the task to mobile nodes that choose to participate, and the nodes report sensor data through opportunistic network connections (such as third-party access points they encounter). Examples of such systems include *CarTel* [18], *Mobiscopes* [1], *Urbanet* [31], *Urban Atmospheres* [40], *Urban Sensing* [8], *SenseWeb* [32] and our own *Metrosense* [7] at Dartmouth College. Applications of opportunistic sensing include collecting traffic reports or pollution readings from a particular street or part of a university campus [18, 7], finding parking spots [31], locating lost Bluetooth-enabled objects with the help of other users' mobile devices [13], and even inferring coffee-shop space availability [36].

Although opportunistic sensing is a best-effort service, it can be useful if there is no fixed-sensor infrastructure in the places where an application desires sensor data. Indeed, by leveraging personal mobile devices as its sensor nodes, this model collects context information closely related to real-life experience "by the humans, for the humans" and potentially with little or no infrastructure cost.

In short, opportunistic sensing introduces a new, *people-centric*, *dynamic* and *highly mobile* communication and computation model. It raises three major challenges.

First, it depends on a large-scale, inherently heterogeneous and unpredictable collection of users' personal devices forming the sensing infrastructure, in which available mobile nodes are tasked with specific context requests and expected to later report the sensed data.

Second, this new model for tasking, sensing, and reporting is likely to be implemented across administratively autonomous wireless access points and the public Internet.

Third and most importantly, opportunistic sensing poses a new security dimension. Users offer the services of their devices for sensing the environment and usually do not gain a direct benefit from reporting such data; they, therefore, will be reluctant to participate in environmental sensing if their privacy is at risk, or if it consumes too many resources on their mobile device. Since reports usually include the time and location of the sensor reading, the report could trivially reveal the user's location at a particular

time. Furthermore, since context is produced by volunteer users and is queried and collected through third-party access points, higher-level applications will require certain guarantees about the integrity of the system and the reliability of reports. The new challenge is to ensure reliable, efficient, and privacy-preserving context tasking and reporting. In this paper, we address this challenge and describe how our system, AnonySense, incorporates new privacy-aware techniques for secure tasking and reporting, and demonstrate that our solution consumes few device resources. Other major systems note these challenges but offer no solutions [1, 7, 18, 31, 8, 32].

We should note that simply suppressing or anonymizing the node's identity in reports [36] is insufficient to protect users' privacy—if multiple reports from nodes can be *linked* as being from the same user, they may reveal the identity of the user [24]. Cryptographic unlinkability [4] is insufficient if various reports uploaded by a user can be linked based on the timing of the reports. Care must be taken, therefore, to ensure that multiple reports from the same user are unlinkable through timing attacks. Prior approaches provide only part of the solution for anonymous tasking and reporting, because they focus either on a narrow set of pre-defined applications, or only parts of the tasking and reporting lifecycle.

To this end, we present AnonySense, an application-independent infrastructure for realizing *anonymous tasking and reporting* that is designed from the ground up to provide users with privacy. AnonySense provides a new tasking language that can express a rich set of context queries; applications can deliver tasks to anonymous nodes, and eventually collect verifiable, yet unlinkable, reports from anonymous nodes. We use a stringent threat model that adopts minimal trust assumptions: in particular, we assume that the mobile-device *carriers* do not completely trust the *system*, the *applications*, or the *users* of the applications, with the anonymity of the reports produced by their mobile devices. AnonySense is designed so that no entity should be able to link a report to a particular carrier and, additionally, so that no intermediate entity can infer information about what is reported, tamper with tasks, or falsify reports.

To the best of our knowledge, AnonySense is the first secure implementation of this fundamental *task-report* model for collecting sensor data, providing flexibility to applications while simultaneously ensuring the anonymity of users against strong adversarial models that include timing attacks on reported data. We discuss the tradeoffs for achieving such anonymity, which comes at the cost of higher latency in receiving reports. For example, higher latencies may preclude certain real-time applications if privacy is to be maintained.

To demonstrate our tasking and reporting architecture, we developed AnonySense within the Metrosense project [7], and built two applications of interest to the mobile-computing community. RogueFinder is our application to task users' mobile devices to detect unauthorized Wi-Fi access points in and around the Dartmouth College campus, and ObjectFinder leverages Bluetooth "sensors" on mobile devices to locate Bluetooth-enabled objects. We use these applications to measure the performance overhead of our security protocols, and to demonstrate the feasibility of privacy-aware opportunistic sensing. We discuss other possible applications in Section 5.

We note that AnonySense focuses on anonymous tasking and reporting, but does not address the leakage of private information through the reported data. For example, a report containing Alice's office as the location where the sensor reading was taken leaks information about Alice's identity. There is a wealth of research focused on this problem, and our architecture can be extended to include techniques such as k-anonymity and spatio-temporal cloaking [35, 15, 14, 19, 17] to reduce the leakage of private data through reports. Indeed, we propose one such cloaking technique in the context of AnonySense in another paper [23]. These techniques are, however, orthogonal to the work presented in this paper, which focuses on anonymous tasking and reporting. In the remainder of the paper, therefore, we assume that such techniques are used in conjunction with our tasking-reporting protocol.

Our contributions.

We summarize the contributions of our work in the context of the design, implementation, and evaluation of mobile systems and applications as follows:

- We present AnonySense, a *general-purpose* framework for anonymous opportunistic tasking and reporting, which, by design, allows applications to query and receive context through an expressive task language and by leveraging a broad range of sensor types on users' mobile devices, and at the same time respects the privacy of users.
- We have implemented AnonySense and through experiments we show that our privacy-aware tasking and reporting approach can be realized efficiently, that is, consuming little CPU time, network bandwidth, and battery energy.
- We demonstrate the flexibility and feasibility of AnonySense in supporting collaborative-sensing applications by presenting two such prototype applications: RogueFinder and ObjectFinder.

Paper outline.

We present the architecture of AnonySense, formalize our privacy and security goals and state our trust assumptions in Section 2. We present our protocols for anonymous tasking and reporting in Section 3, followed by an evaluation of AnonySense in Section 4. We discuss several important issues related to AnonySense in Section 5, and conclude in Section 6. Focused on anonymity, this paper omits some design and implementation details of our system.

2. ANONYSENSE ARCHITECTURE

In this section, we present the AnonySense architecture for anonymous tasking and reporting. We begin by presenting the system design, followed by a description of our tasking language *AnonyTL*, the security properties we desire for our design, and the underlying trust assumptions and threat model.

2.1 System design

AnonySense has three major design principles: (1) to allow a broad range of sensor types and application tasks, (2) to provide anonymity for participating carriers, and

(3) to provide applications with confidence in the integrity of the sensor data. The first principle recognizes our goal to provide a general-purpose framework that can serve a variety of applications, and can leverage a broad set of mobile platforms. The second principle recognizes that people will only participate if the design respects their privacy [21]; we provide anonymity for the carrier, with respect to both the system components and the applications and application users. The third principle recognizes the need for applications to receive high-quality information.

Overview.

The foundation of the AnonySense architecture rests on a dynamic set of *mobile nodes* (MNs) that volunteer to participate. We necessarily assume that there are a variety of node platforms, including mobile phones, PDAs, laptops, or other personal mobile devices, with widely varying capabilities for sensing the physical and network environment around them. We assume that all mobile nodes have wireless access to the Internet, at least intermittently, through wireless *access points* (APs) distributed across the sensing area. We assume the existence of an open-access Wi-Fi infrastructure, operated by any number of individuals and organizations. In many cities, there are municipal Wi-Fi networks and Wi-Fi hotspots, as well as privately-owned open APs, and opportunistic connections are readily available for applications that can tolerate delay [18].

In AnonySense, applications request the desired context through *tasks*, which specify when and where to sense, and what sensor readings to report. After receiving a task, a mobile node decides whether to accept the task depending on certain criteria (related to the node's ability to correctly and privately serve the task). If a node accepts a task, the node produces a series of *reports*, each of which is a tuple containing a unique task ID (assigned by the system, described below) and a set of task-specified data fields (such as timestamp, location, and sensor readings). The nodes retrieve tasks, and submit reports, through an anonymity-preserving protocol.

Since there may be a large population of MNs, a task may list criteria to limit which MNs may accept the task. These "acceptance conditions" may reference static *attributes* of the MN or its carrier. This method allows an application to task, for example, only MNs that are mobile phones and are carried by students. The AnonySense design does not impose any specific meaning on the attributes, which are defined by convention with string names and with string or numeric values. As we explain below, the system verifies the attributes of an MN and its carrier when the MN first registers as an AnonySense node. The system will use its knowledge of the MNs' attributes to ensure that applications cannot violate a carrier's privacy by crafting narrow attribute-based conditions (see *Task verification* in Section 3.1).

The AnonySense architecture is demonstrated in Figure 1. The applications and mobile nodes communicate via the Internet with the system's four core services: the registration authority, the task service, the report service, and a Mix network. These services can be administratively independent, in the sense that they are operated by autonomous entities, with trust relationships defined by our trust model (below).

Components.

The *Mobile Nodes* (MNs) are devices with sensing, computation, memory, and wireless communication capabilities. Mobile nodes are carried by people, or are attached to objects such as vehicles. We can support stationary nodes, but consider mobile nodes as the general and common case. The *carrier* of a node is the person carrying the node, or the owner of the vehicle.

The *Registration Authority* (RA) is responsible for registering nodes that wish to participate, and for issuing certificates to the task service and the report service so that applications and nodes can verify the authenticity of the services. Mobile-node registration serves three purposes: (1) the RA verifies that the task interpreter is properly installed on the node and the node's sensors are properly calibrated, (2) the RA verifies the attributes of the mobile node and its human carrier and records them in the RA database for use in future tasking decisions, and (3) the RA installs a private "group key" on the node, so that the node may later sign reports anonymously as described below, along with the public keys of the task and report services.

We do not specify here how the RA should verify attributes of the MN carrier. The RA may accept digital certificates issued by known authorities (e.g., universities, governments, or professional organizations) or the RA's human operators may verify paper documents to the same effect.

The *Task Service* (TS) receives task descriptions from applications, performs some consistency checking (related to the carriers' privacy requirements and the feasibility of the task), and distributes the current tasks to mobile nodes when they ask to download new tasks. It returns to the application a token that may be used in retrieving the tasked data.

The *Report Service* (RS) receives reports from mobile nodes, aggregates them internally to provide additional privacy, and responds to queries from applications who present a token to collect their task's sensor data.

The *Mix Network* (MIX) serves as an anonymizing channel between mobile nodes and the report service: it de-links reports submitted by MNs before they reach the RS. In general, a MIX allows users to send messages anonymously. Chaum originally proposed a remailing scheme [9], where clients sent multiply-encrypted messages in such a way that each subsequent MIX node could "peel off" a layer of encryption and forward the embedded encrypted message to the next MIX node as prescribed by the message. In general, MIX nodes wait for enough incoming messages before sending the messages to the next MIX node. Such delaying and mixing of messages makes it difficult for passive adversaries to correlate incoming and outgoing messages. Assuming that there are enough users sending messages via the MIX, a recipient cannot link multiple messages to the same sender. Mixmaster [26] is the most popular MIX in use today, and operates as a remailer network. In our AnonySense implementation, MNs use Mixmaster to send reports to the RS, although any MIX supporting standard email protocols (SMTP) would suffice.

2.2 Task language

We defined a simple and expressive language called *AnonyTL* for applications to specify their tasks. AnonyTL allows an application to specify a task's behavior by providing a set of acceptance conditions, report statements, and termination conditions. The acceptance conditions are evaluated by the MN after retrieving tasks from the TS. The ac-

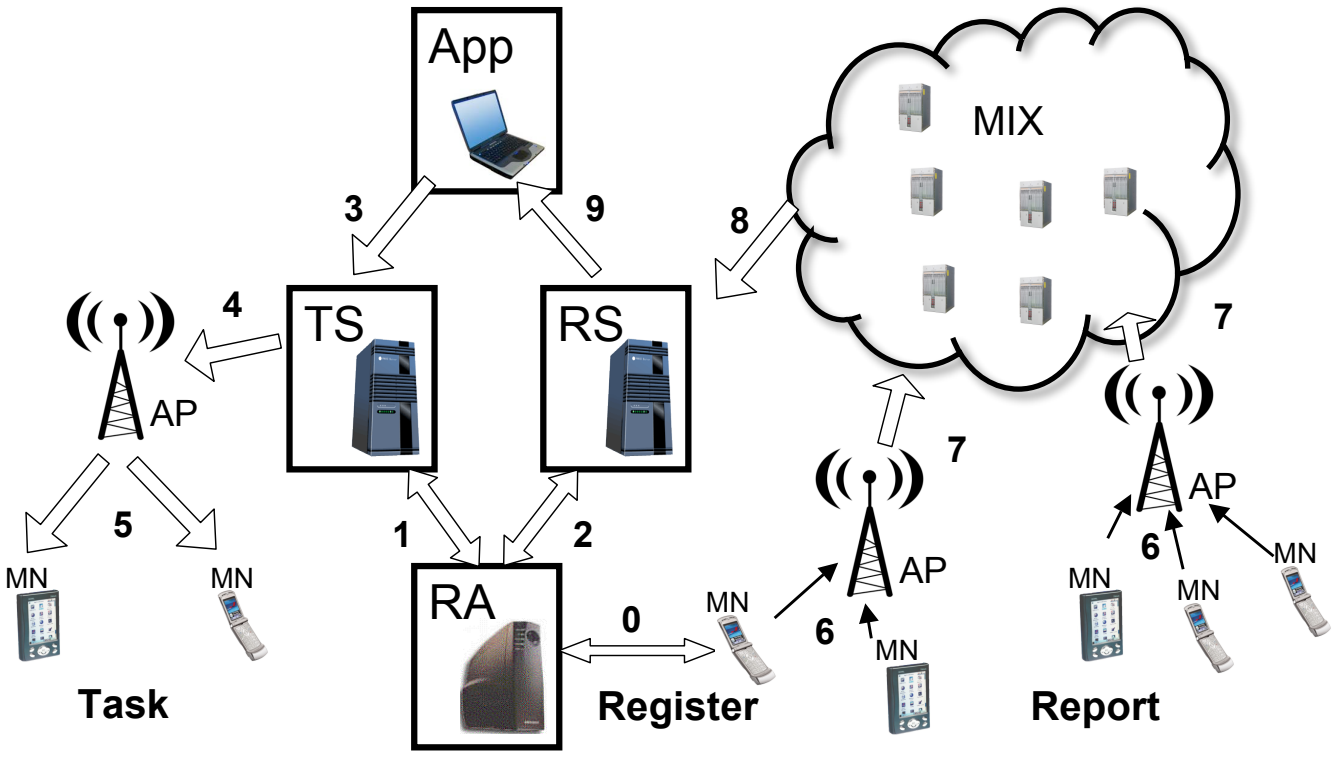

Figure 1: The AnonySense architecture and overview of the communications model. A collection of sensor-equipped mobile nodes (MNs) register (0) as volunteers with the registration authority (RA). The RA also certifies the authenticity of (1) the task service (TS) and (2) report service (RS). Applications (App) submit (3) tasks to the task service; the MNs occasionally download (4,5) new tasks from the TS using the Internet and any handy wireless access point (AP). The task specifies when the MN should sense information, and under what conditions to submit reports. MNs report (6) sensed data via any AP and through (7) a Mix network (MIX), such that the report eventually arrives (8) at the RS. At its convenience, the App fetches (9) the data from the RS.

ceptance conditions indicate that the MN must have certain attributes; the report statements implicitly indicate that the MN must have the necessary sensors. As specified by the report statements, MNs periodically check a set of report conditions against polled sensor values, and if the conditions are met, prepare a report containing application-specified fields to send to the RS. These periodic evaluations continue until a termination condition is satisfied or the task reaches its expiration date, at which point the task is removed from the MN's task pool. We note that tasks are *not executable code*; tasks specify sensor readings desired at a particular granularity, and under what conditions an MN should report data.

We chose to develop a small new language, rather than using an existing language such as SQL or XQuery, because we wanted concise task descriptions, a small interpreter footprint, easy portability to embedded platforms, and a clean fit with the semantics of sensing and reporting. Had we used a query language, we would have needed to wrap it or extend it to obtain the desired semantics.

We chose a Lisp-like syntax for AnonyTL, with parenthesized statements throughout and a prefix notation for operators, to allow construction of small interpreters. Applications can specify acceptance and reporting conditions by using arbitrary logical expressions relating sensor values, MN attributes, and constants with a set of operators. The language is strongly typed, and provides meaningful operators for each type. For example, the spatial point data type returned by a location sensor can be checked for containment in a polygon literal using the In operator. The following task (of our RogueFinder application) demonstrates a simple use of such a comparison:

```
(Task 25043)(Expires 1196728453)
(Accept (= @carrier 'professor'))
(Report (location SSIDs) (Every 1 Minute)
 (In location
   (Polygon (Point 1 1) (Point 2 2)
        (Point 3 0))))
```

This task uses two sensors: "location," which provides the coordinates of the node's estimated location, and "SSIDs," which returns a list of SSIDs responding to a Wi-Fi beacon probe. The task starts by specifying its taskID = 25043, and the timestamp (Unix time) at which it expires. The acceptance condition corresponds to the second line: only MNs in which the attribute "carrier" has value "professor" (attribute names are indicated with @), will accept this task. Furthermore, the reference to a "location" sensor and an "SSIDs" sensor add implicit acceptance conditions; it would be useless to accept a task that required sensors the MN did not possess. A report statement covers the remaining lines, in-

dicating what to report (the current location and the SSID list), the sensing interval (once a minute), and a report condition (when the current location is inside the given triangle). There is no explicit termination statement in this task, but the expiration timestamp serves as an implicit one.

The following task illustrates some additional behavior:

```
(Task 25044)(Expires 1210392000)
(Accept (< temperature 0))
(Report (location time temperature)
 (Every 5 Minute)
 (and (< temperature 0) (< humidity 20)))
(Report (location time temperature humidity)
 (Every 10 Minute)
 (and (> temperature 20) (> humidity 80)))
```

This task restricts the number of MNs accepting it to those whose temperature is below 0°C (and, as mentioned earlier, those MNs who have the sensors needed by the task). It then polls every five minutes, reporting the sensing location, time, and sensed temperature if is still below freezing, and if the humidity is below 20%. It additionally tests every 10 minutes if the temperature is above 20°C and the humidity is above 80%, reporting humidity in addition to the earlier values if the conditions are met.

We note that the reports never contain the name of the MN's carrier, or a unique identifier for the MN, and are thus superficially anonymous. AnonySense addresses deeper threats to anonymity, as described below.

2.3 Threat model

In our design goals we set out to provide two key security properties: anonymity for the carriers and integrity for the sensed data. Here we consider the likely threats related to each goal. In this context, we recognize that an adversary can come in many forms: as one of the AnonySense applications or its user, as one of the AnonySense components, or as a third party (including another MN).

Carrier anonymity.

We seek to provide anonymity for participating carriers, i.e., the humans who carry MNs. An adversary seeks to de-anonymize a carrier by linking a given report to a given carrier or his MN, thus obtaining detailed information (time, place, or sensor data) about a given MN. For now, we assume that a carrier's attributes (e.g., "a professor" and "a Dartmouth person") are public information.

We assume that the adversary may eavesdrop on communications between the MN and the APs, collude with an AP or MIX node to intercept the MN's traffic, attempt to impersonate the TS to hand out bogus tasks, or attempt to impersonate the RS to receive reports. The adversary may submit tasks, via the TS, and receive reports back from those tasks. The adversary may register as an MN, and receive tasks from the TS. Finally, we assume the adversary is free to observe the carrier's activities, over time, except those that generated the report the attacker desires to link to the carrier. Thus, the adversary is knowledgeable but not omniscient.

The adversary may attempt to link an MN's actions (accepting tasks or submitting reports), and correlate this linked history of time/location events to known carrier patterns and identify which carrier's MN generated those reports.

The adversary may attempt to discern the activity of an MN, or small group of MNs, by submitting tasks with a specific combination of attributes.

Finally, we imagine that the RS or TS may be an adversary, that is, attempting to link reports to individual MNs, or to collude with APs to build linkable location histories that could later be correlated with individual MNs.

Data integrity.

We seek to provide applications with some confidence in the integrity of the sensor data. An adversary seeks to tamper with the sensor data received by applications, or to insert false data.

We assume that the adversary has all the capabilities described for the anonymity threats above. The adversary may also attempt to submit bogus reports, may intercept and replay old reports, may collude with an AP or MIX node to tamper with reports on the way to the RS, or may attempt to impersonate the RS to deliver bogus reports to the applications. The adversary may attempt to tamper with the MN hardware or software.

Other threats.

We make no guarantees about an adversary who tampers directly with the MN sensor or its environment. For example, an adversary may hold the MN's temperature sensor next to a heat source, skewing its reading of the local temperature. In future work, we plan to address the broader question of how the system, or applications, can verify the integrity of sensor data.

We do not consider denial-of-service threats in this paper. In such cases an adversary seeks to deny applications the opportunity to submit tasks or to receive reports, to overload the services with excessive tasks and reports, or to cause MNs to consume excessive resources through burdensome tasks.

Finally, we recognize that a sophisticated adversary with a physical infrastructure may track a target mobile device, e.g., by analyzing RF characteristics. This threat exists regardless of AnonySense, although any techniques to mitigate this threat could easily be employed by AnonySense.

2.4 Trust model

Our system design and implementation, as in any system, depends on assumptions about who trusts whom for what purpose. Here, we define our trust model, with each paragraph identifying what that entity trusts about the others.

Carrier.

The human carrier of a mobile node trusts the node software to properly implement the AnonySense protocols and trust relationships, as described below.

Mobile nodes.

We assume that all MNs communicate with the TS and RS using Wi-Fi APs. (Use of other networking technologies is possible, but the subtle differences are beyond the scope of this paper.) MNs trust APs to allow Internet connections, but do not trust the APs (or eavesdroppers) with the confidentiality or integrity of their network traffic, or with their identity and location (beyond the fact that the MN is in the AP's vicinity); in the worst case, a malicious AP or eavesdropper may collude with the TS, RS, or the application.

MNs trust the Registration Authority (RA) to certify the identities of TS and RS. The MN, therefore, can authenticate the TS (e.g., with SSL) and encrypt reports for the RS. Likewise, the RA certifies each MN, which can then prove to the TS and RS that it is a valid node in the system. As we explain below, MNs can prove their validity anonymously using a cryptographic construct called *group signatures*. The MNs trust the RA to certify the authenticity of each task whose attributes would not target too narrow a set of MNs. Last but not least, MNs trust the RA to not conspire (with any component) against the carriers' privacy.

Access Points.

The APs, which we assume are owned and operated by various third parties, need not trust anything about any component.

Applications.

Applications (Apps) trust the RA to certify the TS and RS, so that they may authenticate the TS before sending it tasks and the RS before downloading reports. Apps trust the TS to deploy tasks as requested, and to not divulge the report-retrieval token to any other party. Apps trust the RS not to drop or corrupt reports, or to give a task's reports to another App that does not present the right token. Apps trust the RA to calibrate MNs, protecting the integrity of the sensor data. Apps trust that MNs will execute tasks correctly, and that the carrier of the MN does not tamper with the node. (See below for more about MN integrity.) For now, we do not certify or authenticate Apps. In the future we may require Apps to authenticate, or require the *querier* (user of the App) to authenticate, if we wish to control access to certain kinds of sensor data or limit the right to task mobile nodes.

Registration Authority.

The RA trusts nothing about any component; it is a root of trust. Additionally, we assume the RA is administratively independent from the task or report services.

Task Service and Report Service.

The TS and RS trust the RA to certify only valid MNs in the system, and they verify an MN's credentials (see below) before sending tasks or accepting reports. The TS and RS need not trust the applications; the TS validates any task submitted by an application, and trusts the RA to certify only those tasks that target a sufficiently large subset of the MNs (see Section 3.1). The RS validates the token provided by the application before sending it sensor data.

Certifying MNs.

The RA is responsible for certifying valid MNs. It first verifies that the MN is running the proper version of the AnonyTL interpreter and that the MN's hardware and software are untampered. It verifies the attributes of the MN and the carrier, and calibrates the MN's sensors. Once complete, it provides the MN with a credential that allows the MN to produce signatures that appear to come from *some* valid MN, but the signatures do not reveal *which* MN produced that signature. MNs can, therefore, maintain anonymity and yet prove that they are valid, calibrated MNs (the credentials are updated upon recalibration).

We use the short-group-signature scheme by Boneh et al. [2] to allow MNs to generate group signatures. We use Boneh's scheme because group signatures are "cutting-edge" cryptography technology, and Boneh's scheme was the only publicly available implementation at the time of writing.[1]

Note that schemes based on group signatures allow for different levels of privacy for carriers. Some schemes allow MNs to be deanonymized by a special entity such as the RA, whereas some schemes maintain unconditional anonymity for carriers. For a discussion on various techniques that balance anonymity with accountability, we point the interested reader to a recent paper on Blacklistable Anonymous Credentials (BLAC) [39]. AnonySense makes no assumptions on the revocation capabilities of the RA; some systems may want the option to identify misbehaving carriers, while others may want to guarantee unconditional privacy.

We assume the MNs are tamper-resistant with respect to software attacks, for example, by leveraging a Trusted Platform Module (TPM) [38]; the RA can test for the presence of, and proper configuration of, the TPM as part of the certification process. (Although TPMs are currently available only on laptops, desktops, and servers, we expect to see TPMs in smart phones soon— the Trusted Computing Group released a complete specification of the Mobile Trusted Module (MTM) in June 2007 [25, 37]. MTMs add to the TPM's functionality for use in mobile devices.) The TPM protects the group-signature key, so that only RA-certified nodes will be able to generate AnonySense reports acceptable to the RS. We note that TPMs do not offer strong guarantees against hardware attacks; nevertheless, TPMs raise the bar since most users would be unwilling to tamper with the hardware.

3. PROTOCOL

In this section we describe our privacy-preserving protocols for anonymous tasking and reporting, which provide security despite the threats of Section 2.3 under the trust assumptions of Section 2.4.

There are two major protocols: one for getting tasks from applications to mobile nodes, and the other for mobile nodes to report sensor data back to applications. We assume that relationships with the RA are handled offline; for example, an MN may be registered with the RA at the time of purchase, or by the service provider, when the MN's sensors can be calibrated and the AnonySense software can be securely installed.

3.1 Tasking protocol

We first consider the protocol for anonymously assigning tasks to MNs.

Step 1. Task generation. The App generates a task using the tasking language and sends the task to the TS using a server-authenticated channel (SSL, in our implementation). The App, therefore, ensures that the true TS receives the task without being tampered by a third party. As part of the task, the application specifies an expiration date, after which the task is deleted by the TS and MNs. The TS generates a unique task ID for the task.

[1]Idemix [5] is an alternative, but is currently unavailable because of its transition to the Higgins project. See http://www.eclipse.org/higgins/.

Step 2. Task verification. If the task syntax is valid, the TS sends the task to RA over a mutually authenticated channel. The RA computes the value of k, the number of unique MNs that satisfy the attribute criteria and sensor capabilities required by this task. If $k \geq k_g$, where k_g is a global parameter, the RA prepares a certificate stating that at least k_g MNs satisfy the task criteria. (Without such a safeguard, Apps might craft tasks that target a small set of users, thereby reducing the privacy of users.) The RA sends this certificate, which includes a hash of the task and the task ID, back to the TS. Note that this protocol insulates the TS from knowledge about individual MNs or their attributes. MNs and their carriers need only trust the RA to check the attribute conditions against k_g.

Step 3. Response to App. If the task is semantically or syntactically incorrect, or $k < k_g$, the TS replies to the App that the task is invalid. Otherwise, the TS replies to the App in a message that contains the task ID along with a TS-signed certificate for the task ID. The application later uses this certificate as a token to retrieve data from the RS, or to tell the TS to cancel the task once it has enough data.

Step 4. Tasking nodes. When MNs have Internet access, they poll the TS for tasks over a server-authenticated channel (using a new address as discussed below). For each connection, the MN uses anonymous authentication to prove to the TS that it is a valid MN in the system, without revealing its identity. To do this, the MN uses a group signature to sign a nonce challenge provided by the TS. The TS delivers all recent, unexpired tasks to the MN. For example, the TS will deliver all tasks that have been created in the past three hours and have not yet expired. This approach means that some nodes will repeatedly retrieve the same tasks. We sacrifice performance here for privacy: if MNs were to reveal the ID of the last downloaded task, or the time they last downloaded tasks, it becomes easier to link the MN's connections. This approach also means that some older tasks will not be delivered; we assume that most nodes will poll the TS during the "recent" window so that few tasking opportunities will be missed. We sacrifice some task deliveries here for performance, to reduce the time for each poll of the TS.[2] The TS drops a task when it reaches its expiration date or when it is canceled by the App.

The MN ignores any tasks it has considered in an earlier download, and considers the acceptance conditions of new tasks. The MN accepts any tasks for which it meets the criteria. Although the AnonyTL engine allows any number of tasks to "run" on an MN, in future work we will allow MN policies to limit resources consumed by AnonyTL tasks.

3.2 Reporting protocol

The MN processes the task using the AnonyTL interpreter, reading sensors when required and generating reports as necessary. The MN includes a hash of the task within the report package, so the application can later verify that the correct task was provided to the MN. The MN includes a large random nonce within the report package, so the RS can detect report replays. The MN signs each report using the group-signature scheme, encrypts it with the RS public key, and stores it in an outgoing queue. When the network is available, and there are queued reports, the MN connects to an AP anonymously (using a new address as discussed below), and delivers reports to the RS, each report a separate message sent through the MIX. The RS verifies the group signature on each report when it arrives, thereby verifying that the report was sent by a valid node. The MIX ensures that when the reports arrive at the RS, they are "mixed" in with reports from other users, and hence the RS cannot link multiple reports to a single user. The mixing step is important to prevent the RS from trivially linking "anonymous" reports together using timing analysis. For the same reason, a low-latency anonymizing network (such as Tor [11]) is inappropriate, because it allows for timing attacks. On the other hand, since a MIX like Mixmaster delays messages until they can be reliably mixed with other messages, reports may arrive late and out of order. AnonySense, therefore, trades off reporting latency for improved privacy for users.

[2]We may consider alternatives in future work: MNs may download a random subset of tasks, or the MN may reveal certain attributes to the TS, reducing the number of tasks downloaded at the expense of (some) privacy.

Data fusion.

The RS aggregates reports from a task before delivering the aggregated results to the application. Reports can be combined using standard k-anonymity techniques, and the specific aggregation method depends on the type of data sensed (such as a picture, an audio file, or temperature reading) and the needs of the App. A detailed discussion of aggregation methods is beyond the scope of this paper.

Data retrieval.

The App polls the RS for available context data using a server-authenticated and encrypted channel. The application presents the TS-issued token with the task ID, proving that it is authorized to access the data from that task. Encryption prevents eavesdroppers from receiving sensor data they do not deserve, and ensures the integrity of data transfer from the RS to the App.

MAC address recycling.

Using anonymous authentication is useless if an MN can be tracked using its static MAC address, because MNs must assume the APs may collude with the TS or RS. We assume the MN changes its MAC and IP addresses using one of the standard mechanisms [16, 20] so that an MN's report and task actions may not be linked. Addresses should be recycled for *each* tasking or reporting session, for maximum privacy. The MIX serves to defeat timing attacks that link reports submitted in the same session, and the use of random intervals between connections to the TS protects the MN from attempts to link tasking operations. Recently, Pang et al. show how users' privacy can be reduced through 802.11 fingerprinting [30], and therefore better mechanisms are needed to prevent such inferences.

3.3 Security properties

The above AnonySense protocol, given the trust model in Section 2.4, provides anonymity to the MN users and ensures integrity of the sensor data reported to the applications, as follows.

- An adversary (even an AP) can learn little by eavesdropping on the MN's communications, because all communications with the TS or MIX are encrypted, and the MN rotates its MAC and IP addresses for each

such session. We note that MAC and IP rotation provides unlinkability at the granularity of an AP. After a MAC and IP rotation, the user appears to be a new, possibly different, user in the vicinity of the AP.

- An adversary may not pose as the TS or the RS, because both the MNs and the Apps have a certificate from the RA to the public key for the TS and the RS, and they use an authenticated channel (such as SSL) or encrypted messages to secure these communications.

- The TS may not link an MN's tasking operations together, because each poll arrives from a new IP address and the polling interval is randomized.

- An adversary can learn little by submitting tasks to the system, because any task must satisfy $k \geq k_g$ to be distributed to mobile nodes. Adversaries, therefore, cannot construct tasks to target a narrow set of users.

- An adversarial MN may receive many of the tasks in the system, but since (a) we ensure that the TS validates an MN before providing it tasks, (b) the RA certifies MNs as valid only if they have some tamper-resistant features (such as TPM) and have our software installed, and (c) our software will not divulge the tasks (except possibly to its human carrier), an adversary cannot see the tasks. The tasks are encrypted in the air and thus cannot be sniffed by a third party.

- An adversary may attempt to link reports together, or link task actions to report actions, but we foil these attempts. The MN rotates its MAC address every time it connects to the system, it uses random TS-polling intervals, and it uses a MIX to temporally separate its reports sent to the RS. These methods prevent even the RS or TS from linking tasks or reports.

- Any report submitted by an adversary will be rejected, because an adversary does not have the group-signature key necessary to sign reports, and cannot easily extract it from an MN due to its tamper-resistant features.

- An adversary cannot replay an intercepted report because of the nonce encoded within the signed and encrypted report, and the RS memory of reports already submitted.

- An adversary cannot tamper with reports between the MN and the AP, within an AP, or within a MIX node, because all reports are signed by the MN and encrypted with the RS public key.

- If TPMs are used, an adversary cannot tamper with the MN software because the TPM hardware will detect software tampering (at load time) and fail to provide the group-signature key for signing future reports. As mentioned earlier, using TPMs would provide security against adversaries unwilling to perform hardware-based attacks.

A more detailed proof of these properties is beyond the scope of this conference paper.

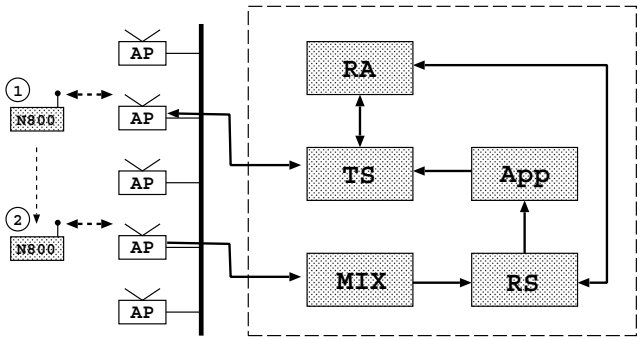

Figure 2: Implementation

4. EVALUATION

We implemented AnonySense. The services run on generic Linux servers. The mobile-node software runs on a Linux PDA (the Nokia N800) and the Apple iPhone, and can be easily ported to any other Unix-based platform.

In this section we describe our implementation and evaluation. Our evaluation focuses on the cost the implementation imposes, particularly on the mobile nodes' resources: network bandwidth, CPU time, and battery life.

4.1 Implementation

Figure 2 illustrates the overall architecture of our prototype implementation. The AnonySense services (RA, TS, and RS), a single-node MIX, and an application component run on a Linux desktop. To minimize network effects during testing, the services were connected to Dartmouth's Computer Science wired network (100 Mbps switched Ethernet), and the MNs were connected to Dartmouth's wireless network, which consists of about 1500 APs spread throughout the campus. Although a real AnonySense deployment would require multiple MIX nodes, we needed only one node for the purpose of our measurements; we obtain realistic MIX latency information from the literature.

Communications.

Our implementation leverages standard protocols implemented by open-source libraries, resulting in compact and robust code. The Apps POST their tasks to the TS over an SSL-encrypted HTTP channel, after authenticating the TS using the certificate from the RA. In response, TS provides the App a URI and token. The MN also communicates with the TS using an SSL-encrypted HTTP channel, requesting and downloading tasks after authenticating to the TS using the group signature. The TS communicates with the RA using a mutually-authenticated SSL-encrypted HTTP channel to submit a task to the RA and receive a digital signature certifying that the RA has verified the task (see Section 3.1). The MN communicates with the MIX by encrypting its report with known MIX node keys, and then sends the message using SMTP, which the MIX ultimately forwards to the RS. The App finally polls the RS over HTTP with the TS-provided URI and token to retrieve new reports, again validating the authenticity of the RS using the certificate from the RA.

Servers.

The AnonySense services are written in the Ruby pro-

gramming language (v1.8). Both the TS and RA are implemented using Camping [6], a micro-framework for developing small HTTP servers, with the actual SSL and HTTP handling done by Mongrel [27]. The RS is a simple Ruby script that processes emails as forwarded by the postfix email server. The RS verifies and decrypts emails forwarded to it by the MIX, storing reports for later retrieval by the App. Each server is backed by a SQLite3 database for persistence.

Mobile node.

We use a Nokia N800 (with 330 MHz TI OMAP 2420 processor and 128 MB DDR RAM) equipped with IEEE 802.11b and Bluetooth 2.0 interfaces. Although this popular device is not a mobile phone, its features are comparable to many "smart" phones; we chose this device because its OS (Linux) eases system development. There are no PDAs or mobile phones with TPM support, yet, so our implementation currently trusts the MN to correctly implement the AnonyTL interpreter and the AnonySense protocols; our evaluation does not include the cost of operating with a TPM. The MN software also compiles and runs on the iPhone, but due to the current closed nature of the platform APIs, its sensing capabilities are limited to 802.11-related measurements (including localization) only.

The AnonySense MN software is written in C++. It downloads tasks from the TS using libcurl, and verifies task signatures using the RSA and SHA-1 functions provided by OpenSSL. To parse downloaded tasks, the AnonyTL interpreter uses a Bison/Flex-generated parser. Wireless SSID scanning capabilities are provided by the wireless tools (libiw version 28) library, and the equivalent Bluetooth capabilities are provided by BlueZ's hcitool utility. Localization is currently performed by Skyhook's Wi-Fi library [34], but it is possible to use any other such library with minimal effort.

When reporting, the MN generates a random AES key and uses OpenSSL to encrypt the prepared XML[3] report with it, appending a copy of the key encrypted with the RSA public key for the RS. It then signs the package with Boneh's short group signatures using a modified version of Stanford's PBC_sig library (version 0.0.2, using the included d159 pairing parameters). We used the Mixmaster utility (version 3.0rc1, slightly modified) to prepare a MIX message and send it using libsmtp 0.8.5.

For simplicity, our current implementation does not implement MAC-address rotation. We were able to change the MAC address on our N800s, and thus do a MAC address rotation; we recognize this trick may not be possible on all platforms due to limitations of their hardware or software. We also do not currently implement the group signature-based authentication step of the tasking protocol. This change, however, should not affect the performance of the tasking step significantly, adding only a group-signature computation (the cost of which we evaluate separately in the context of reporting) and some additional data transfer over the network for the signature.

4.2 Applications

To demonstrate AnonySense operation, we implemented two simple applications. Each uses the network interfaces of the N800 as sensors, although we have also written similar applications that use the N800's physical sensors (its microphone, for sound-level measurements). Although these are just two applications, AnonySense is designed and able to support a broad range of application types.

Application RogueFinder*.*

The RogueFinder application is used to detect rogue APs in a given area. To accomplish this, RogueFinder tasks the AnonySense system to report all APs visible to the MNs. The sensor in this case is the MN's Wi-Fi interface; the interface sends a *probe request* on every Wi-Fi channel and listens for *probe responses* from APs. RogueFinder then checks the list of APs reported against a list of known deployed APs to determine which are rogues. When a rogue AP is detected, then RogueFinder can display a marker on a map that is the approximate location of the rogue AP.

Granted, given an extensive static infrastructure, it might be possible to detect most rogue APs. A large deployment, however, typically has gaps in its coverage, particularly near the edges, in which it is difficult or impossible for static methods to effectively detect rogues; MNs tasked by RogueFinder can potentially cover the gaps in the static infrastructure.

Application ObjectFinder*.*

Our inspiration for ObjectFinder comes from a similar application described earlier [13]. If a person loses one of their Bluetooth devices, they can use ObjectFinder to task AnonySense to find a specific Bluetooth MAC address. When an MN detects the specified MAC address, it then reports the current location. The App is then able to mark on a map where the Bluetooth-enabled object was detected. Although the positioning may be crude, one could easily imagine ObjectFinder being extended to include other information such as signal strength so triangulation can be used for more accurate object positioning.

Because a Bluetooth scan takes a significant amount of time (10.5 sec on average in our experiments), a recent paper provides a method for computing the optimal probing interval [41]. Rather than specifying a fixed sensing interval, as in our current implementation, ObjectFinder could request an adaptive interval and leave it to the MN to adjust its probing interval to its environment.

4.3 Experimental Results

Our tests were conducted in the Dartmouth Computer Science building, with around 60 distinct Wi-Fi BSSIDs visible from the testing station, and around 3–7 discoverable Bluetooth devices in the vicinity.

Methods.

We ran the RogueFinder application with a single Nokia N800 registered with the AnonySense system. We measured the CPU time by logging timestamps between different operations. Data transfer between the MN and servers was captured by WireShark and analyzed by tcptrace to extract statistics of TCP flows of interest. We measured the energy consumption of the device by measuring the voltage and current between the battery and the device across a test resistance of 0.5 Ω using an Agilent 34401A multi-meter. We also measured the base energy consumption of the device without

[3]We use XML for reports because (unlike with AnonyTL) we have no special requirements for the report format. It needs to be able to encode key/value pairs for sensed values, and XML is a well-recognized standard for that purpose.

Table 1: Data Transfer in Bytes

TCP Stream	Bytes
MN → TS	478
MN ← TS	3086
Total	3564
MN → MIX	29164
MN ← MIX	297
Total	29461

any applications running. We subtracted these base values from the measured energy consumptions to get the net energy consumption of various applications. We found that measurement results for RogueFinder in this section are similar to the ObjectFinder application, except that the cost of the Wi-Fi scan is replaced with that of a Bluetooth scan.

Overall results.

After walking around our building with an MN for several minutes, the MN detected 84 unique APs, of which RogueFinder determined 12 to be rogues, that is, not part of the official campus infrastructure. Out of 50 repetitions, it took 15.5 seconds on average for the MN to receive the RogueFinder task, perform one scan, and issue a report. In our experiment, the average power cost was 6.64 mW and a complete task-scan-report cycle cost 0.11 Joule on average. As a rough benchmark, this power consumption is 17 times smaller than MP3-quality audio streaming on the N800. Although in this experiment the RogueFinder task was set to report only once, a more realistic operation would send a task that senses and reports periodically, over a given period, so the costs for tasking (bytes, energy) would be amortized across many reports.

Data Transfer.

We analyzed the data exchanged over the lifetime of a single task from the point at which the MN contacts the TS to when it reports to the MIX. We are concerned only with the number of unique bytes sent over the TCP streams, so we ignore retransmits. The total number of bytes exchanged was 33,025 bytes (32.3 Kbytes). Table 1 shows the detailed data transfer for each direction. The total data exchanged when the MN communicates with the TS will scale linearly as the number of tasks increases with some fixed overhead for the SSL encryption headers. Data transmitted from MN to RS will grow linearly in the number of reports submitted, but in discrete increments due to the MIX protocol requiring large message paddings to evade statistical attacks on message contents.

Overall energy consumption.

Table 2 illustrates how the energy cost of a whole cycle of tasking, sensing, and reporting compares with the consumption of various multimedia applications. The purpose of this comparison is to give an intuitive sense of the magnitude of energy consumption in AnonySense compared to other common activities someone might do on their MN. For example, the energy consumed by one RogueFinder tasking is equivalent to the energy consumed by playing a local MP3 file for 46.8 seconds. We also note that the power consumption by RogueFinder tasking is larger than streaming radio (33 kbps) but less than streaming MP3 (128 kbps) over the network. In another experiment, we found that RogueFinder reduced the battery lifetime of a fully charged N800 by 5.26% (from 285 minutes without RogueFinder to 270 minutes with RogueFinder). In this experiment we simulated a network-heavy usage scenario by playing streaming audio continuously while downloading 20 emails per hour. The tasking operation was also heavy; one RogueFinder cycle per minute. We ran two sets of experiments and report the average.

Table 2: Multimedia job equivalent to one cycle of a RogueFinder task (15.5 sec. with 6.64 mW)

Application	Power	Job
Local MP3 play	2.34 mW	46.8 s
Streaming Radio	4.55 mW	24.0 s
Streaming MP3	7.61 mW	14.4 s
Local Video play	9.23 mW	11.8 s
Streaming Video	16.88 mW	6.4 s
Download	22.92 mW	746.1 KByte

Detailed energy consumption.

One sensing task can be divided into several sub-operations. First, the MN retrieves tasks from TS (Tasking), executes the task once (Sensing), generates and signs the report (Signing), and sends it to RS (Reporting). Table 3 shows the cost of each operation. As shown in the table, the execution of a RogueFinder task took the most time (46.6%) and energy (49.1%). The energy consumption of the sensing operation depends on the type of application being tasked. With RogueFinder, the MN needs to probe all channels to retrieve a list of open access points, which is the most expensive operation, in terms of Joules. The next most expensive operation is computing group signatures of reports. In general, Tasking is more expensive than Reporting due to the SSL connection with the TS. Finally, the table lists the ObjectFinder sensing cost (BT Sensing); other ObjectFinder costs are equivalent to those in RogueFinder.

Table 3: Energy cost of task sub-operations

Operation	Time	Power	Energy	Fraction
Tasking	1.1 s	11.26 mW	12.1 mJ	11.0 %
Wi-Fi Sensing	7.2 s	7.44 mW	53.8 mJ	49.1 %
Signing	5.2 s	5.16 mW	26.6 mJ	24.3 %
Reporting	2.1 s	8.34 mW	17.1 mJ	15.6 %
BT Sensing	10.5 s	2.87 mW	30.0 mJ	

5. DISCUSSION

In this section we discuss subtle issues of our design or implementation.

Scalability.

The vision of people-centric urban sensing seeks to achieve a metropolitan scale, with thousands of devices being tasked

and sending reports. We expect AnonySense to scale well with careful implementation. The TS, RS, and RA can all be replicated by borrowing load-balancing techniques from the web [10, for example]. The MIX easily scales by adding more MIX nodes, and encouraging MNs to choose MIX nodes randomly.

As the number of concurrent tasks grows, the burden on MNs may increase. There are two simple refinements. First, to reduce the download burden when an MN requests tasks from the TS, the TS could provide a random subset of tasks rather than all tasks. Second, to reduce the overhead of executing tasks, an MN can impose resource constraints that cause it to reject some tasks for which it otherwise is qualified.

Task dissemination.

The flip side of too many tasks is too many reports: an App may submit a task only to find that it receives far more reports than needed or desired. AnonySense allows Apps to explicitly remove a task, e.g., after receiving the desired number of reports. For long-running tasks that may not report quickly, we could adjust the language so an App can code the task to send a null report immediately `(Report (timestamp) (once))`, thus allowing the App an early indication of how many MNs have accepted the task. (We cannot allow the TS to count task acceptances, by expecting the MN to indicate acceptance during tasking, since that allows TS to link information about the carrier's identity to its MN's IP address.)

Attribute-based tasking.

The number of MNs satisfying the requirements of an attribute-based task may be small enough to be a privacy concern. AnonySense uses the parameter k_g to ensure that at least k_g MNs satisfy the specified attributes. Reported data, however, includes information such as time and location where the reading was taken. External knowledge of movement patterns may be used to reduce the reporter's purported k_g-anonymity. For example, even if there are 100 professors on campus, a professor reporting from the CS building may have only 10-anonymity (instead of 100-anonymity) since there are known to be only 10 CS professors on campus. As mentioned in Section 1, we defer to existing techniques such as spatio-temporal cloaking and k-anonymization at the server. For example, the RS could filter or aggregate reports to provide additional k-anonymity against the Apps, using a k_l parameter provided by the MN as part of its report. Specific strategies for setting the anonymity parameters (k_l and k_g) is an interesting problem, outside the focus of this paper. We propose another alternative based on "statistical k-anonymity," where MNs can perform spatio-temporal cloaking on the reported data for added privacy [23].

Carrier policy.

Currently, an MN accepts and executes tasks automatically, with no intervention from the human carrier. Clearly, it would be inconvenient to repeatedly prompt the carrier about which tasks to accept. We intend to explore mechanisms for carriers to configure a personal policy about which kinds of tasks to accept, and which kinds of information to divulge. From our experience [22], such an interface must be extremely simple to be effective.

TPM for data integrity.

To provide confidence in sensor data and to protect private group keys, AnonySense leverages TPM-protected mobile phones. This RA-certified, TPM-controlled platform may, however, be cumbersome to the carrier. For example, carriers may want the freedom to install applications without being constrained by TPM-based attestation. In future work, we plan to explore the data integrity issue under a more modular sensing paradigm that does not require TPMs.

Data aggregation.

We de-emphasize in-network aggregation in the AnonySense architecture. We believe that urban mobile-sensing systems, as opposed to traditional sensor networks, will feature devices (such as mobile phones) with frequent Internet connectivity. With such devices, bandwidth at the link layer and network layer does not appear to be a major impediment.

Delay tolerance.

We make use of a MIX to allow clients to upload reports efficiently in a single network connection, while maintaining the unlinkability of reports. As a consequence, reports arrive at the Report Service after being delayed by the MIX. The amount of delay depends on the population of MIX users and the message flow rate. Current deployments of Mixmaster show that messages can arrive in a few minutes, or may take hours. In general, as the number of messages passing through the MIX increases, the latency goes down because the MIX queue fills up faster. Thus, as more carriers join AnonySense and report, the latency of reports will go down. If the application is sensitive to delay, and needs low-latency reports, nodes could rotate their MAC and IP addresses before sending each report directly to the RS. Given a queue of reports, however, rotating MAC addresses could take time. For example, Jiang et al. [20] found that a 12-minute "silent period" was needed before rotating the MAC address to provide reasonable unlinkability. We believe, therefore, that using a MIX to send reports is more pragmatic, especially since it reduces the window during which the MN must maintain an active network connection, making it more able to take advantage of brief connectivity [18]. If an application needs to know the specific time at which data was sensed, it can request a timestamp from the MN, keeping in mind as mentioned above that the timestamp reading may be blurred by the MN to provide suitable k-anonymity [23].

Using either approach, applications would need to wait for several minutes before receiving reports for their tasks. Under our set of trust assumptions, AnonySense is best suited to delay-tolerant applications. We have demonstrated how applications such as ObjectFinder and RogueFinder can tolerate delays and find objects or rogue Wi-Fi access points successfully.

Data quality.

It is intuitive that providing better, more accurate data leads to less privacy for carriers, and that more privacy necessarily means less accurate data. The specifics of this tradeoff may be predicted for time and location data based on historical movement patterns, so we could allow an application to request a certain granularity of either time or location, and have the MN blur the other dimension appropriately so

that k-anonymity is respected with high probability. Using this idea, it may also be possible for an application to specify desired granularities for both time and space dimensions, allowing the MN to reason about the k-anonymity such parameters would afford it and make a policy decision about whether to report. We explore some of these tradeoffs in our work on statistical k-anonymity [23].

Wi-Fi vs. cellular networks.

An alternative to the AnonySense architecture would be to rely on cellular-phone service providers to track carriers at all times (as they already do), and route tasks and reports through the cellular network. We believe, however, in an architecture that preserves carriers' privacy without placing as much trust in the provider. (There have been cases where U.S. providers have handed over sensitive data about users without a subpoena [28].) AnonySense, like CarTel [18], leverages the growth of open-access Wi-Fi networks, and AnonySense is designed to ensure carriers' anonymity while contributing sensing data for community use.

Privacy risks in RogueFinder*.*

AnonySense has been carefully designed to protect the anonymity of carriers who accept and report on tasks. No component of the system knows which MNs or which carriers have accepted a task, or which MN submitted a given report. In certain conditions, and for certain types of tasks, however, it may be possible for an external observer to learn something about the MN.

RogueFinder uses active Wi-Fi probes to scan for nearby APs. Suppose an adversary (Alice) wishes to learn more about the attributes of a nearby AnonySense carrier (Bob). Alice submits a task asking MNs to probe Wi-Fi every 1.23 minutes, say, and includes attribute conditions to limit the set of MNs that accept the task. Sniffing the Wi-Fi network, Alice can tell if Bob's MN accepted the task by observing whether Bob probes the network every 1.23 minutes, in addition to its pre-tasking probing interval. By submitting a series of tasks, with carefully constructed attribute-based acceptance conditions, Alice may eventually be able to learn many of Bob's attributes. Since we currently assume attributes are not sensitive information, this attack is not a serious threat. The solution to this problem is for the MN to add a random time delay before using any "active" sensor, that is, whose operation can be externally observed. It remains future work to consider more general cases and more general solutions.

Privacy risks in ObjectFinder*.*

ObjectFinder depends on the lost object being "discoverable" in Bluetooth. Many people commonly leave their devices in discoverable mode, and thus it is possible to harvest the MAC addresses carried by a person simply by standing nearby. Later, one could use ObjectFinder for stalking that person, by tasking thousands of MNs to help track the location of the victim's MAC addresses. One solution is for people never to leave devices in "discoverable" state, but then ObjectFinder cannot help find lost devices. One alternative, assuming a person carries multiple paired Bluetooth devices, would be for a device to periodically become discoverable when it has been out of contact with paired devices for several minutes. More sophisticated solutions are possible, but require changes to the Bluetooth protocols.

Another potential risk is a race to find a lost device. An adversary may use ObjectFinder to start his or her own search in an attempt to find the device first. Although the tasking protocol and tamper-resistant MN could prevent carriers from learning the contents of tasks, as a design decision we believe it is important to allow carriers to inspect tasks, and thus a savvy AnonySense participant may discover the Bluetooth address of an object being sought. Later, when we develop a carrier-configurable policy module, we may choose to hide the task from the carrier but expose the task to the policy module, reducing this risk.

An observant third-party adversary may notice many Bluetooth devices issuing discovery probes, but since these do not expose the MAC address of the lost item they cannot learn the MAC address to seek.

MN safety.

As mentioned above, we do not consider denial-of-service attacks in this paper. One concern of any cooperative application is excessive use by greedy or malicious users. An application, or its users, may issue many tasks, or tasks that require excessively frequent sensing or reporting. We imagine two aspects to a solution. Globally, the TS can throttle aggressive users or applications by authenticating users or applications and then applying rate limits, or requiring some form of (anonymous) payment. Locally, an MN can analyze the task before accepting it, rejecting tasks with short sensing or reporting intervals; the MN can also drop tasks, sensing duties, and drop or delay reports when it is low on resources. The specific mechanisms remain as future work.

Other applications.

There are many exciting possible applications for a system like AnonySense. We mention a few here, some of which have been imagined or even prototyped by others.

A small modification to RogueFinder could map both 802.11 coverage and quality around campus.

We implemented a QuietFinder, which maps the sound levels around campus. The task is just like RogueFinder, except using the N800's microphone as a sound-level sensor.

For runners or bikers [12], one could use an accelerometer and GPS to detect running or biking activity and have an application identify the popular routes and their difficulty. Variant: use an outboard Bluetooth sensor (such as pulse or respiration) to sense physical exertion. Or, contribute location data from bikers toward a street map of the world [29].

One could task mobile nodes to send images or video from locations of interest; one set of researchers use peer-to-peer communications for nearby cellphones to coordinate video capture and analysis [33]. (We have concerns about privacy from any image-based tasks, however!)

Suppose public infrastructure (such as street lamps, parking meters, fire hydrants) were instrumented to transmit beacons (or respond to probes) when they need service. Then tasks could report the location and serial number of the broken object; as long as the timestamp is blurred (e.g., reporting the date but not time) the carrier's location privacy would be reasonably preserved.

There are many potential opportunities related to wellness or health care. The Continua alliance lists some use cases [3].

For public safety, imagine if MNs had radiation detectors. In addition to a local application that informs the carrier about their own personal exposure to radiation, tasks can

provide health officials information about when and where radiation is detected, enabling better tracking of a plume resulting from a dirty bomb.

Other papers about people-centric sensing or urban sensing imagine yet more applications [1, 7, 8, 18, 31, 32].

6. SUMMARY

We present AnonySense, a comprehensive system aimed at preserving the privacy of users in opportunistic-sensing environments. AnonySense allows a variety of applications to request sensor data using a flexible tasking language, and later receive by the system the sensor data from personal mobile devices. Data is collected in an opportunistic and delay-tolerant manner, in which a large and dynamic set of mobile nodes can volunteer to accept tasks and send back reports, both reliably and anonymously.

We implemented and evaluated our system in the context of two applications, ObjectFinder and RogueFinder, and our results show that sensor data can be reliably obtained, from anonymous users, without much overhead. We believe a privacy-aware architecture will make opportunistic sensing infrastructures more acceptable, since users will see little risk to their privacy by participating in applications that provide them with indirect benefits.

Acknowledgments

This research program is a part of the Institute for Security Technology Studies, supported by Grants 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance, 60NANB6D6130 awarded by the U.S. Department of Commerce, and 2006-CS-001-000001 awarded by the U.S. Department of Homeland Security. Triandopoulos was supported by the Institute for Information Infrastructure Protection (I3P) under an award from the Science and Technology Directorate at the U.S. Department of Homeland Security, and by the Center for Algorithmic Game Theory at the University of Aarhus under an award from the Carlsberg Foundation. The views or opinions in this paper do not necessarily reflect the views of the sponsors.

We thank the anonymous reviewers, the Metrosense team at Dartmouth College, Patrick Tsang, Vijay Bhuse, and our shepherd Urs Hengartner, for their helpful comments.

7. REFERENCES

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

[2] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Proceedings of Crypto '04*, volume 3152 of *LNCS*, pages 41–55. Springer-Verlag, 2004.

[3] Continua alliance. http://www.continuaalliance.org/use_cases/. Use cases available on the web.

[4] G. Calandriello, P. Papadimitratos, J.-P. Hubaux, and A. Lioy. Efficient and robust pseudonymous authentication in VANET. In *VANET '07: Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks*, pages 19–28. ACM Press, 2007.

[5] J. Camenisch and E. V. Herreweghen. Design and implementation of the *idemix* anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 21–30. ACM Press, 2002.

[6] Camping. http://code.whytheluckystiff.net/camping/. Available on the web.

[7] A. Campbell, S. Eisenman, N. Lane, E. Miluzzo, and R. Peterson. People-centric urban sensing. In *The Second Annual International Wireless Internet Conference (WICON)*, pages 2–5. IEEE Computer Society Press, August 2006.

[8] CENS Urban Sensing project, 2007. http://research.cens.ucla.edu/projects/2006/Systems/Urban_Sensing/.

[9] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), Feb. 1981.

[10] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *COMPCON '96: Proceedings of the 41st IEEE International Computer Conference*, page 85, Washington, DC, USA, 1996. IEEE Computer Society.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[12] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. The BikeNet mobile sensing system for cyclist experience mapping. In *Proceedings of the 5th ACM Conference On Embedded Networked Sensor Systems (SenSys)*, pages 87–101, Nov. 2007.

[13] C. Frank, P. Bolliger, C. Roduner, and W. Kellerer. Objects calling home: Locating objects using mobile phones. In *Proceedings of the 5th International Conference on Pervasive Computing (Pervasive)*, pages 351–368, May 2007.

[14] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 620–629. IEEE Computer Society, 2005.

[15] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the First International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 31–42. ACM Press, 2003.

[16] M. Gruteser and D. Grunwald. Enhancing location privacy in wireless LAN through disposable interface identifiers: a quantitative analysis. *Mobile Networks and Applications*, 10(3):315–325, 2005.

[17] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Preserving privacy in GPS traces via uncertainty-aware path cloaking. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 161–171. ACM, 2007.

[18] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. K. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 125–138, Nov. 2006.

[19] G. Iachello, I. Smith, S. Consolvo, M. Chen, and G. D.

Abowd. Developing privacy guidelines for social location disclosure applications and services. In *Proceedings of the 2005 Symposium on Usable Privacy and Security (SOUPS)*, pages 65–76, July 2005.

[20] T. Jiang, H. J. Wang, and Y.-C. Hu. Preserving location privacy in wireless LANs. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 246–257. ACM Press, 2007.

[21] P. Johnson, A. Kapadia, D. Kotz, and N. Triandopoulos. People-Centric Urban Sensing: Security Challenges for the New Paradigm. Technical Report TR2007-586, Dartmouth College, Computer Science, Hanover, NH, February 2007.

[22] A. Kapadia, T. Henderson, J. J. Fielding, and D. Kotz. Virtual walls: Protecting digital privacy in pervasive environments. In *Proceedings of the Fifth International Conference on Pervasive Computing (Pervasive)*, volume 4480 of *LNCS*, pages 162–179. Springer-Verlag, May 2007.

[23] A. Kapadia, N. Triandopoulos, C. Cornelius, D. Peebles, and D. Kotz. AnonySense: Opportunistic and privacy-preserving context collection. In *Proceedings of the Sixth International Conference on Pervasive Computing (Pervasive)*, May 2008.

[24] J. Krumm. Inference attacks on location tracks. In *Proceedings of the Fifth International Conference on Pervasive Computing (Pervasive)*, volume 4480 of *LNCS*, pages 127–143. Springer-Verlag, May 2007.

[25] Mobile Phone Work Group, Trusted Computing Group. https://www.trustedcomputinggroup.org/groups/mobile.

[26] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003.

[27] Mongrel. http://mongrel.rubyforge.org/. Available on the web.

[28] E. Nakashima. Cellphone tracking powers on request: Secret warrants granted without probable cause. *Washington Post*, page A01, 23 November 2007.

[29] Open street map. http://www.openstreetmap.org/. Available on the web.

[30] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 99–110. ACM Press, Sept. 2007.

[31] O. Riva and C. Borcea. The Urbanet revolution: Sensor power to the people! *IEEE Pervasive Computing*, 6(2):41–49, 2007.

[32] Microsoft Research SenseWeb project, 2007. http://research.microsoft.com/nec/senseweb/.

[33] T. Simonite. Cellphones team up to become smart CCTV swarm. *New Scientist*, 31 October 2007.

[34] Skyhook wireless, 2007. http://www.skyhookwireless.com/.

[35] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 10(5):557–570, October 2002.

[36] K. P. Tang, J. Fogarty, P. Keyani, and J. I. Hong. Putting people in their place: An anonymous and privacy-sensitive approach to collecting sensed data in location-based applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 93–102, 2006.

[37] TCG Mobile Trusted Module Specification, Revision 1. https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf.

[38] Trusted Computing Group (TCG), May 2005. https://www.trustedcomputinggroup.org/home.

[39] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. Blacklistable anonymous credentials: Blocking misbehaving users without TTPs. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 72–81, 2007.

[40] Urban atmospheres project, 2007. http://www.urban-atmospheres.net.

[41] W. Wang, V. Srinivasan, and M. Motani. Adaptive contact probing mechanisms for delay tolerant applications. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 230–241. ACM, Sept. 2007.