AnonySense: A System for Anonymous Opportunistic Sensing

Minho Shin [a], Cory Cornelius [a], Dan Peebles [a], Apu Kapadia [b]
David Kotz [a], Nikos Triandopoulos [c]

[a] *Institute for Security, Technology, and Society, Dartmouth College, USA*
[b] *School of Informatics and Computing, Indiana University Bloomington, USA*
[c] *Department of Computer Science, Boston University, USA*

Abstract

We describe AnonySense, a privacy-aware system for realizing pervasive applications based on collaborative, opportunistic sensing by personal mobile devices. AnonySense allows applications to submit sensing *tasks* to be distributed across 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 security goals, threat model, and the architecture and protocols of AnonySense. We also describe how AnonySense can support extended security features that can be useful for different applications. We evaluate the security and feasibility of AnonySense through security analysis and prototype implementation. We show the feasibility of our approach through two plausible applications: a Wi-Fi rogue access point detector and a lost-object finder.

Keywords: opportunistic sensing, urban sensor networks, privacy, anonymity

1 Introduction

Opportunistic sensing has been gaining popularity, with several systems and applications being proposed to leverage users' mobile devices to collectively measure social or 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* [1], *Mobiscopes* [2], *Urbanet* [3], *Urban Atmospheres* [4], *Urban Sensing* [5], *SenseWeb* [6], and *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 [1,7], finding parking spots [3], locating lost Bluetooth-enabled objects with the help of other users' mobile devices [8], and even inferring coffee-shop space availability [9].

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

An opportunistic sensing system depends on a large-scale, heterogeneous and unpredictable collection of users' personal devices. Furthermore, the tasking and reporting mechanism depends on administratively autonomous wireless networks and the public Internet. Therefore, the first challenge for opportunistic sensing is to support *reliable tasking and reporting* mechanisms, and *energy-efficient mobile sensing* protocols.

Second, since sensor data is produced by volunteer users, and requested and collected through third-party access points and the Internet, it is difficult to guarantee the *integrity and quality of reports*.

Third and most importantly, *user privacy* is hard to protect in opportunistic sensing. Since reports usually include the time and location of the user, location privacy of the user can be compromised by an adversary who can access the reports. Even if reports are kept confidential, the adversary may de-anonymize a user by simply observing the user's various activities; the adversary may analyze what tasks a mobile node downloads, when it submits reports, or what IP address it is using. Such information may reveal the identity of the user and other sensitive information. In opportunistic sensing, users usually offer the resource of their devices without direct benefit; they, therefore, will be reluctant to participate if their privacy is at risk, or if it consumes too many resources on their mobile device.

In this paper, we address these challenges 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 [1–3,5–7].

Toward that goal, we first propose a basic framework for large-scale opportunistic sensing and set forth our security goals and threat model in Section 2. We also propose a new tasking language that can express a rich set of context queries. Based on this framework, Section 3 presents our design for AnonySense, a privacy-preserving sensing infrastructure with reliable sensor data. We use a stringent threat model that adopts minimal trust assumptions: people who volunteer their mobile devices do not completely trust the system or the application users for respecting their privacy.

We evaluate the security of AnonySense by analyzing the defense mechanisms of AnonySense against possible attack scenarios in Section 4.

To demonstrate our tasking and reporting architecture, we developed AnonySense and built two applications of interest to the mobile-computing community. RogueFinder is our application to task users' mobile devices to report Wi-Fi access points to detect those that are unauthorized, 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 in Section 5. We discuss various design and implementation issues in Section 6 and conclude in Section 7.

We summarize our contribution as follows:

- We present AnonySense, a *general-purpose* framework for anonymous opportunistic tasking and reporting. At the same time, AnonySense respects the privacy of users and protects the integrity of reports, and optionally allows for user or application authentication, confidential reports, and incentive-based reporting.
- We implemented AnonySense. Our experiments 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.

2 AnonySense Design and Security Goals

In this section, we present a high-level design for people-centric sensing that can distribute a variety of tasks to a large number of heterogeneous mobile nodes. When we design this *base architecture*, we focus on the functionality of the system, leaving security challenges unsolved. Given the base architecture, we formalize our security goals and perform a threat analysis, based on which we design our security solution in Section 3.

2.1 Base architecture

Our base architecture consists of *Mobile Nodes* (MN), *Applications* (App), the *Registration Authority* (RA), the *Task service* (TS), and the *Report Service* (RS). We also introduce a tasking language *AnonyTL* [10] to support various types of sensing tasks. Figure 1 illustrates the base architecture.

The mobile nodes (MNs) are devices with sensing, computation, memory, and wireless communication capabilities in various platforms such as smartphones, PDAs, or laptops. Each MN is carried by a person called *carrier*, and can access personal sensors either on board (e.g., an accelerometer in the iPhone) or attached to the carrier. We assume the MN has wireless access to the Internet, at least intermittently, through some open-access Wi-Fi infrastructure

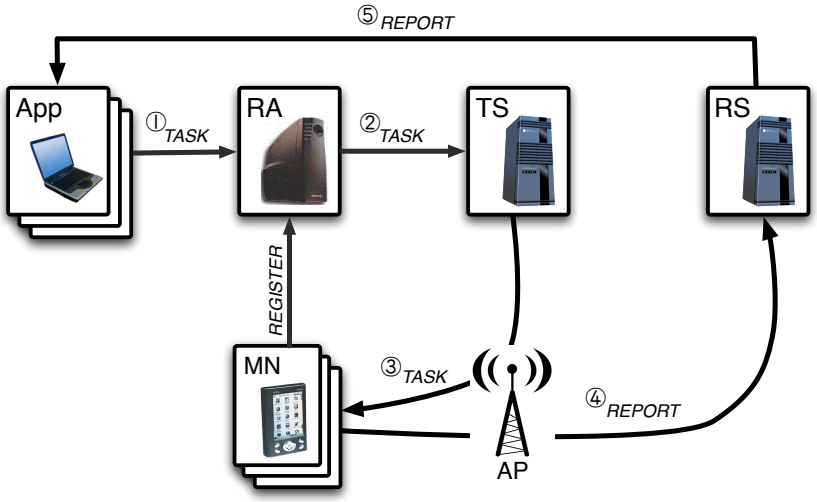

Fig. 1. Base architecture for large-scale people-centric sensing

operated by any number of individuals and organizations [1]. [1]

When a carrier wants to participate in AnonySense, she has to register her mobile device with the registration authority (RA). During registration, the AnonySense software is installed in the MN along with the IP addresses and certificates of the task service (TS) and the report service (RS).

When an application (App) wants to collect sensor data from MNs, it creates a new task, written in AnonyTL, and submits the task to the RA (message ① in Figure 1). After validity checks, the RA forwards the task to the TS (message ②). The MN contacts the TS to download the available tasks (message ③). The MN chooses tasks to run based on the *acceptance condition* of the task and the MN's own policy. The acceptance condition limits which MNs may execute the task (see Section 2.2 for more detail), and the MN's policy determines whether to contribute its resources for the task. If the MN accepts a task, it collects sensor data according to the task description and later submits reports to the RS (message ④). Later the App retrieves the reports from the RS (message ⑤).

In the proposed architecture, tasks are *pulled* by an MN from the TS rather than pushed to the MN. We chose a pull-based protocol for scalability and for carrier privacy, as the connection can be anonymous. Likewise, reports are pulled by the App from the RS for scalability.

2.2 AnonyTL: a general-purpose task language

We defined a simple and expressive language called AnonyTL for applications to specify their tasks. Instead of using an existing language such as SQL or

[1] Although we assume direct communication between MNs and access points, multi-hop connections can be leveraged if necessary. Since MNs do not trust access points, the use of multi-hop connections does not impact the security of AnonySense.

XQuery, we designed AnonyTL with Lisp-like syntax to allow concise task descriptions, a small interpreter, easy portability to embedded platforms, and a clean fit with the sensing/reporting semantics. For the formal definition of AnonyTL, please refer to the AnonyTL specification [10].

AnonyTL defines a task by providing acceptance conditions (who can accept this task), report format (what to report), reporting frequency (how often to report), reporting condition (when to report), and termination conditions (when to stop reporting). The following example defines a RogueFinder task in AnonyTL:

```
(Expires 1196728453)
(Accept (= @WiFi 'a/b/g'))
(Report (LOCATION APLIST)
 (Every 60 seconds)
 (In ((1 1) (2 2) (3 0))))
```

The first line defines when to stop the task: "stop reporting after given Unix time." The second line describes an acceptance condition that "only MNs in which the Wi-Fi interface supports 802.11 a, b, and g will accept this task." The third line defines what to report: location of the MN (using a location sensor) and a list of access points observed by the MN (using an APLIST "sensor," based on the Wi-Fi interface). This report format also implicitly defines another acceptance condition: "only MNs with a location sensor and a Wi-Fi interface will accept this task." The fourth and fifth lines respectively describe how often and under what condition the MN will generate a report: "report every 60 seconds whenever the MN is located within the polygon defined by coordinates (1,1), (2,2), and (3,0)." AnonyTL supports a broad range of conditional expressions and an extensible set of sensor types whose description we omit due to space limitations.

2.3 Security Goals

The base-architecture presented obviously does not provide any privacy or integrity guarantees: reports may reveal sensitive information about the carrier and the reports may be manipulated by an adversary. We therefore design AnonySense to meet the following security goals.

(SG1: Carrier Privacy) AnonySense aims to prevent *identity disclosure* of carriers: an adversary cannot link an AnonySense activity to the identity of a carrier. AnonySense activities include any action of the MN (e.g, tasking and reporting) and any result of those actions (e.g., reports). Identity disclosure is undesirable because it can lead to the disclosure of sensitive information about the carrier: location and time (e.g., where was John last night?), or sensitive sensor data (e.g., what was John doing this morning?).

(SG2: Report Integrity) AnonySense aims to protect *report integrity*: the integrity of a report is protected if the report has been generated by a legiti-

mate MN, according to the task definition, and the report was not modified in transit.

2.4 Threat model

We design AnonySense to protect carrier privacy and report integrity under the following threat model.

We consider a powerful adversary who has the ability to compromise some system components, or public infrastructure such as access points (AP). An adversary can also become an App by creating its own task, or become a legitimate carrier by registering its own mobile device to the RA. We also consider a local adversary, near an MN, who may already know the carrier's identity, but is curious about the carrier's private information. We assume the adversary can do all these things, at the same time.

However, we assume that the adversary cannot compromise certain trusted components or the software platform of an MN (such attacks trivially violate the privacy of the user, with or without AnonySense). In Section 3 we discuss which components are assumed to be trusted.

Such an adversary may attempt the following attacks against carrier privacy and report integrity.

Threats to Carrier Privacy

Narrow tasking. A malicious application may attempt to learn about a victim carrier by submitting a task with such restrictive acceptance conditions that the victim and only a few other carriers will accept the task. For example, the adversary may submit a task only for the mobile user who carries a heart-rate sensor paired with an iPhone; then, the reports may contain John's locations with high probability.

Tasking de-anonymization. An adversary may attempt to de-anonymize an MN during a *tasking action*, that is, when the MN connects to the TS and downloads some tasks. Through tasking actions, the MN may reveal to the TS where and when it connects and what kind of tasks it wants to download. The carrier's preference for tasks can reveal attributes of the MN and types of carried sensors. Linking multiple tasking actions allows the adversary to trace a carrier. By combining a trace with known facts about the victims, the adversary may be able to identify the carrier.

Reporting de-anonymization. An adversary may attempt to de-anonymize an MN during *reporting actions*, that is, when the MN connects to the RS and submits reports. In this attack, the adversary focuses on reporting actions (the next threat considers report content). Through reporting actions, the MN may reveal to the RS where and when they are reporting and what kind of tasks they performed. Linking multiple reports may allow the adversary to identify the carrier more easily, for example, by analyzing patterns of where and when

the reports are submitted.

Selective tasking. To link between multiple reports, the adversary, controlling the TS, may attempt to distribute a task only to one or a few MNs so that reports for the task are easily linked with each other. For example, if only one MN is known to have downloaded a task, all reports for the task are easily linked to the same MN. Note that selective tasking attack restricts the tasked MN-set for report linkability by controlling task distribution, while narrow tasking attack does so for identity disclosure through narrow acceptance conditions.

Report analysis. An adversary may attempt to de-anonymize an MN by analyzing information contained in one or more reports. For example, the adversary tries to recognize multiple reports as originating from the same MN by analyzing the content of reports. Once linked, the series of reports may reveal a location trace of the carrier, which may be combined with known information to identify the carrier. If lucky, the adversary might be able to de-anonymize an MN from only one report, if, for example, a report arrives from an IP address known to be in John's house.

Local eavesdropping. A local adversary may attempt to overhear a nearby MN's wireless communication to learn what tasks the carrier is performing or what sensor values the carrier is reporting.

Eavesdropping by collusion. In this attack, a local adversary may attempt to learn the MN's communication by colluding with the TS or the RS. The TS or RS may be able to provide a local adversary with the contents of the victim MN's communications, while the local adversary can provide time and location information. To succeed in this attack, the adversary and its colluding parties should be able to recognize the victim's communication among the many communication sessions of the colluding TS or RS.

Threats to Report Integrity

Report tampering. An adversary may attempt to degrade the quality of the reported sensor data by manipulating reports submitted by a legitimate MN.

Report replay. An adversary may attempt to degrade the quality of the reported sensor data by duplicating an existing report submitted by a legitimate MN.

Report forgery. An adversary may attempt to degrade the quality of the reported sensor data by injecting bogus reports.

Threats outside the scope

We do not focus on *denial of service* (DoS) attacks. For example, we do not consider DoS attacks against system components or MNs. We also assume that the adversary (or even the malicious carrier) cannot compromise legitimate MNs to launch report tampering or report forgery attacks.

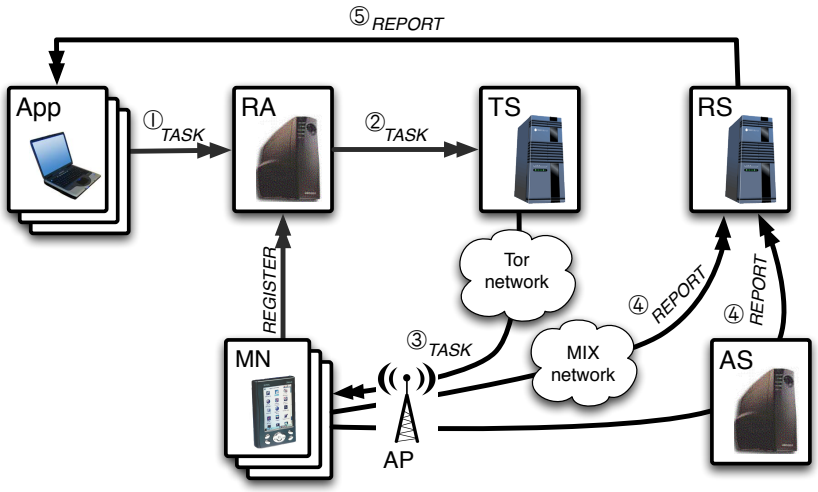

Fig. 2. AnonySense: secure architecture for large-scale people-centric sensing. Double arrows indicate AnonySense protocols for carrier privacy and report integrity.

3 AnonySense: Architecture and Protocol

In this section, we extend the design proposed in Section 2.1 by adding new security features and introducing new security components, aiming to achieve our security goals (Section 2.3) under the above threat model (Section 2.4).

3.1 Architecture

AnonySense adds three components to the base architecture — an anonymization service (AS), a synchronous anonymizing network (such as Tor [11]), and a (asynchronous) MIX network — as well as custom security protocols. Figure 2 illustrates the new secure architecture of AnonySense.

AnonySense works as follows. To bootstrap, the RA certifies system components such as TS, RS, AS, and MNs in terms of correct software installations and security features if necessary. See below for detailed requirements for each component. When an App submits a task, the RA verifies whether the task respects carrier privacy (see Section 3.3.1) before releasing tasks to the TS. At random intervals, the MN connects to the TS (through an anonymizing network), to download new tasks. [2] The MN chooses which tasks to accept. When reports are ready, the MN sends the reports either to RS or to AS depending on the sensitivity of the reports. If the report contains sensitive information, the reports are sent to AS, which anonymizes the reports before forwarding to the RS. If not, reports are sent directly to the RS via a MIX network. An App can retrieve reports from the RS, and can verify the integrity of the reports. Overall, AnonySense provides secure protocols for task submission, task distribution, report submission, and report retrieval (see Section 3.3).

[2] Any unpredictable pattern of connections will suffice; the purpose is to make it difficult for the TS to link together a sequence of connections to a given MN.

AnonySense Components

At the center of AnonySense security architecture is the registration authority (RA). The RA is the root of trust in the sense that every component trusts the certificates issued by the RA, the RA always behaves correctly, and the RA never becomes malicious. The RA is responsible for (i) certifying system components and MNs, (ii) certifying privacy-safe tasks, and (iii) releasing tasks securely (see Section 3.3.2 for more detail). To register an MN, the RA ensures that (i) the AnonySense software is installed in the MN, (ii) the public keys of the TS, RS, and the AS are installed, (iii) the sensors are properly calibrated, (iv) the attributes of the MN are recorded in the MN database, and (v) a digital-signature key and a group-signature key are stored securely (see Section 3.3.3 for their use).

The *Anonymizing network* (Tor) serves to protect the network identity and location of the MN when it connects to the TS to download new tasks. Tor [11] allows clients to anonymously connect to servers through multiple Tor relays using onion encryption. Traffic is delivered as quickly as possible and the location (IP address) of the client is hidden from the server. Each MN connects to the TS with a randomized interval, making it harder for the adversary to link between tasking connections.

The *Mix Network* (MIX) serves as an anonymizing channel between MNs and the RS, routing reports through multiple servers, inserting delays and mixing reports with messages from other sources and to other destinations. No adversary (the RS, a MIX node, or an eavesdropper) can link an MN's reports together, identify which MN sent the report, or learn the time and location of the reporting action. Our AnonySense implementation uses Mixmaster [12], the most popular MIX in use today, although any remailer-type MIX network supporting SMTP email protocols would suffice.

Normally MNs send reports to the RS via the MIX network. MNs can optionally first send their reports to a trusted *Anonymization Service* (AS) if the MN desires additional privacy measures. The AS will apply anonymization techniques to ensure that each report is mixed with reports generated by other carriers with similar information that may serve as "quasi identifiers." The AS could also apply *blurring* techniques [13,14] by adding uncertainty to the location within reports (or even a series of reports) to ensure that the MN cannot be uniquely identified. The AS can also blur reporting times by acting as a MIX node [14]. (Although we show how AnonySense MNs can locally blur location and time [15], a trusted AS can be more effective by mixing reports from other MNs.) These techniques are out of the scope of this paper, but we provide support for such systems within our architecture.

AnonySense uses group signatures by Boneh et al. [16] to authenticate an MN or sign a report without revealing the identity of the signing MN. (Boneh's scheme was the only publicly available implementation of group signatures,

at the time of implementation.) If an AS is used as part of the AnonySense architecture, we assume that Apps trust the AS to verify the integrity of reports from the MNs. Reports modified or aggregated by the AS are signed with the AS's own key. The App need only verify that a report is correctly signed by either the AS or with the MNs' group signature.

3.2 Trust model

AnonySense assumes minimal trust assumptions on its components.

The carrier of a mobile node trusts the RA to correctly perform its certification process and trusts the AS to perform its role of protecting the reports from de-anonymization as described in Section 3.1. The carrier also trusts the MIX network and the anonymizing network to perform their roles, protecting communication privacy.

The application trusts the RA to certify MNs as described in Section 3.1, trusts the AS to verify the integrity of reports it processes, and trusts certified MNs to produce only valid reports (consistent with our threat model in Section 2.4, which assumes the adversary does not compromise MNs).

Note that no components trust the TS, RS, App, and other peripheral components (such as APs), to protect privacy or anonymity; any or all of these components may be malicious at any time.

3.3 Protocol details

We now present a detailed description of the AnonySense protocols. In what follows, we denote by $X \Rightarrow Y$ that X sends a message to Y over a secure channel (i.e., a channel that guarantees the integrity and confidentiality of exchanged messages). When an endpoint is underlined, we further mean that the opposite endpoint has authenticated that endpoint; that is, $X \Rightarrow \underline{Y}$ means that X has authenticated Y, $\underline{X} \Rightarrow Y$ means that Y has authenticated X, and $\underline{X} \Rightarrow \underline{Y}$ means that X and Y are mutually authenticated. Additionally, $\overset{\text{Tor}}{\Rightarrow}$ denotes that the communication is performed over Tor.

3.3.1 Task registration

In Algorithm 1, an App submits a task t to the RA (line 1) through an RA-authenticated encrypted channel. Upon receiving the task, the RA determines whether the task can be accepted by at least k_a MNs in the MN-set $\mathcal{M}$, where k_a is a system parameter that defines the minimum anonymity-set size for tasking (line 2). If the test succeeds, the RA generates a unique task-id for the task (line 3), stores the task and the task-id in the task repository T (line 4), and notifies the App of the task-id so that the App later can claim reports (line 5). If the task acceptance condition is too restrictive, the RA rejects the task (line 7).

Algorithm 1 Task registration

Notation:
k_a : threshold for narrow tasking

1: App $\Rightarrow$ RA : $\langle t \rangle$ ▷ App submits a new task t

{RA runs the following}

2: **if** $|\{\text{MN} \in \mathcal{M} \mid \texttt{AcceptTask}_{\text{MN}}(t) = \texttt{true}\}| \geq k_a$ **then** ▷ Wide tasking
3: $t_{id} := \texttt{GenerateNonce}(t)$ ▷ Generate task-id
4: $T := T \cup \{\langle t_{id}, t \rangle\}$ ▷ Store in task repository
5: RA $\Rightarrow$ App : $\langle$"accept", $t_{id}, t\rangle$
6: **else** ▷ Narrow tasking
7: RA $\Rightarrow$ App: $\langle$"reject", $t\rangle$
8: **end if**

3.3.2 Task distribution

The goal of task distribution is to ensure that MNs are confident that the tasks they receive are (a) k_a-anonymous, (b) untampered copies of the task submitted by the App, and (c) not a subset produced by the TS in a "selective tasking" attack. We design the protocol so that any deviation of the TS from the protocol is detected by MNs. The RA's signature guarantees (a) and (b); (c) is the core challenge we address here.

Our task-distribution algorithm comes in two parts: task-release (Algorithm 2, in which the RA releases tasks to the TS) and task-download (Algorithm 3, in which the MN downloads tasks from the TS). The prerequisites of this approach are (i) the clocks of MNs are reasonably synchronized with the RA (e.g., within one minute) and (ii) the MN knows when the RA *releases* tasks to the TS, called *release-times*. The first prerequisite is easily achieved; for example, most mobile phones maintain fairly accurate clocks through network providers. For the second prerequisite, the MN obtains from the RA (during registration) details of its release schedule (e.g., an initial time and an interval). (Other, more sophisticated release schedules are possible, but outside the scope of this paper.)

Let $r_1, r_2, \ldots$ be a sequence of release times announced by the RA. At each release time $r = r_i$ for some $i > 0$, let T_r be all the unexpired tasks (Algorithm 2 line 1). The RA creates a *release message* m_r that defines its release time (line 2) followed by the list of task-id and task-hash pairs of each task in T_r (lines 3–5). The release message is signed (line 6) and sent to the TS along with the signature and the set T_r. The TS needs only remember the latest release (line 8). (Although line 7 implies that the RA transmits all tasks in T_r on every release, this step can of course be optimized if RA–TS bandwidth is a concern. The RA would transmit only $\langle m_r, \sigma_r \rangle$, then the TS requests only those tasks needed to update its local task cache to T_r.)

Algorithm 2 Task release

Notation:

$r_1, r_2, r_3, \ldots$: release times announced by the RA

{**RA runs the following at time** $r \in \{r_1, r_2, \ldots\}$}

1: $T_r := \{\langle t_{id}, t\rangle \in T \mid t \text{ is unexpired}\}$ ▷ unexpired tasks in T
2: $m_r := r$
3: **for each** $\langle t_{id}, t\rangle \in T_r$ **do**
4: $\quad m_r := m_r \mid\mid t_{id} \mid\mid \texttt{Hash}(t_{id}||t)$
5: **end for**
6: $\sigma_r := \texttt{Sign}_{\text{RA}}(m_r)$ ▷ RA signs rth release message m_r
7: $\underline{\text{RA}} \Rightarrow \underline{\text{TS}} : \langle m_r, \sigma_r, T_r\rangle$

{**TS runs the following**}

8: $L := \langle m_r, \sigma_r, T_r\rangle$ ▷ TS remembers only the latest release

Algorithm 3 shows how the MNs download tasks from the TS, assumed to run at some random time $t = \texttt{now}$. To pick a random time, the MN can wait for a while uniformly at random before next download, or alternatively the MN can choose a moment uniformly at random within each predefined period (e.g., a day) for download. This is to prevent the TS from linking two different downloads.[3] We wish to avoid the cost of downloading all the tasks, every time, over the MN's wireless network link. Thus, the MN sends the TS a random subset-index $j \in 0, \ldots, p-1$ (lines 1–2) where p is a system parameter. Given this subset-index, the TS extracts a task-subset as indicated by the index (lines 3–4). The TS finally forwards this set of filtered tasks, the release message and its signature to the MN (line 5). Note that one can define task-subset differently, and our algorithm only shows one example of such methods (i.e., modulus p).

Given downloaded tasks and the release message, the MN checks to be sure the release time is the latest (line 6), the signature is valid (line 7), the set of downloaded tasks equals the jth subset of released tasks (line 8), and the hash values of downloaded tasks match the release message (line 9). If any of these checks fail, the MN reports the TS as fraudulent to the RA (line 11). If all succeed, then the MN executes the acceptable and unseen tasks (lines 16–18). Note that function $\texttt{StartTask}_{\text{MN}}$ runs the task in the background so that the MN can perform multiple tasks simultaneously.

3.3.3 Report submission

Suppose the MN decides to submit report d for the task t with task-id t_{id}. When d does not contain any sensitive data, the MN sends a report message to the RS via a MIX network; the message contains the task-id and the report,

[3] To choose the best min-max waiting time (or predefined period) that maximizes privacy, one can follow the approach proposed in [17].

Algorithm 3 Task download

Notation:
p: the number of task-subsets

{MN does the following at random intervals}

1: pick a random $j \in \{0, \ldots, p-1\}$ ▷ random subset index
2: MN $\overset{\text{Tor}}{\Rightarrow}$ TS: $\langle j \rangle$

{TS does the following}

3: $\langle m_r, \sigma_r, T_r \rangle := L$ ▷ The latest release L
4: $T_r^{(j)} := \{\langle t_{id}, t \rangle \in T_r \mid t_{id} \equiv j \pmod p\}$ ▷ Compute jth subset of T_r
5: TS $\overset{\text{Tor}}{\Rightarrow}$ MN: $\langle m_r, \sigma_r, T_r^{(j)} \rangle$

{MN does the following}

6: **if** $r \neq \max\{r_i \in \{r_1, r_2, \ldots\} | r_i \leq \mathsf{now}\}$ ▷ Check freshness of release
7: **or** $\mathtt{Verify}_{\mathrm{RA}}(m_r, \sigma_r) = \mathsf{false}$ ▷ Verify RA's signature
8: **or** $\{t_{id} | \langle t_{id}, t \rangle \in T_r^{(j)}\} \neq \{t_{id} \in m_r | t_{id} \equiv j (\mathrm{mod}\ p)\}$ ▷ Mismatch $T_r^{(j)}$ and m_r
9: **or** $\exists \langle t_{id}, t \rangle \in T_r^{(j)}$ s.t. $t_{id} || \mathtt{Hash}(t_{id} || t) \notin m_r$ ▷ Wrong hash in m_r
10: **then**
11: $\sigma_{fraud} := \mathsf{GSig}_{MN}(\text{“tasking fraud”} \ || \ TS)$
12: MN⇒RA: $\langle \text{“tasking fraud”}, TS, \sigma_{fraud} \rangle$ ▷ Report fraud
13: **exit**
14: **end if**
15: **for all** $\langle t_{id}, t \rangle \in T_r^{(j)}$ **do**
16: **if** t_{id} was not seen before ▷ Check task duplication
17: **and** $\mathtt{AcceptTask}_{\mathrm{MN}}(t) = \mathsf{true}$ ▷ Check acceptance condition
18: $\mathtt{StartTask}_{\mathrm{MN}}(t_{id}, t)$ ▷ Execute the task in the background
19: **end if**
20: **end for**

signed by the MN's group-signing key (Algorithm 4 line 2–3). The symbol $\perp$ indicates that there is no explicit nonce value; with Boneh's group-signature method [16], the group signature serves the purpose of a nonce (preventing replay attacks), because the group signature is different every time the message is signed. This value also indicates that the report was not anonymized by the AS.

When d contains sensitive information,[4] the MN sends to AS its identity, the task-id, the report, a nonce, and a digital signature and, optionally, the corresponding certificate. The MN sends its identity because the AS needs that information for enforcing k-anonymity and for path confusion. Unlike

[4] A report contains sensitive information if part of the report is sensitive (e.g., location information) or it includes static attributes that may collectively identify the user if external information is given (called quasi-identifiers [18]). The decision algorithm depends on the exact anonymization techniques available in the AS.

Algorithm 4 Report submission

{**MN does the following to report** d **for task** $\langle t_{id}, t\rangle$}

1: **if** d has no sensitive information **then**
2: $\sigma_d := \texttt{GSign}_{\text{MN}}(t_{id} \,||\, d)$ ▷ Compute a group signature
3: MN $\overset{\text{MIX}}{\rightarrow}$ RS : $\langle t_{id}, d, \perp, \sigma_d\rangle$
4: **else**
5: $N_d := \texttt{Nonce}()$ ▷ Generate a nonce
6: $\sigma_d := \texttt{Sign}_{\text{MN}}(\text{MN} \,||\, t_{id} \,||\, d \,||N_d)$ ▷ Compute a digital signature
7: MN $\Rightarrow$ AS : $\langle \text{MN}, t_{id}, d, N_d, \sigma_d, \text{Cert}_{MN}\rangle$ ▷ Encrypted for AS
8: **end if**

{**AS does the following after line 7**}

9: **if** $\texttt{Verify}(\text{MN}||t_{id}||d||N_d, \sigma_d) = \texttt{true}$ **then** ▷ Verify the signature
10: $d^* := \texttt{Anonymize}(\text{MN}, t_{id}, d)$ ▷ Anonymize the data
11: $\sigma_{d^*} := \texttt{Sign}_{\text{AS}}(t_{id} \,||\, d^* \,||\, N_d)$ ▷ Digital signature of AS
12: AS $\Rightarrow$ RS : $\langle t_{id}, d^*, N_d, \sigma_{d^*}\rangle$
13: **else**
14: $\texttt{Discard}(\text{MN}, t_{id}, d, N_d, \sigma_d)$ ▷ Reject the report
15: **end if**

{**Upon receiving a tuple** x **in line 7 and 12, RS does**}

16: $R := R \cup \{x\}$ ▷ RA stores the reports

the previous case, a digital signature is used because the MN trusts the AS not to compromise its privacy.

When an AS receives a report, it first verifies the signature to make sure it came from a valid MN. We skip details on the one-time verification of the RA's signature from the MN's certificate. Then, the AS anonymizes the report (line 10) and signs a new message (line 11) before sending it to the RS (line 12). The anonymization technique is out of the scope of this paper and thus left as general as possible to support a variety of different anonymizing schemes.

When the RS receives a report, it simply stores the report in its database.

3.3.4 Report retrieval

When the App wants to receive reports, it sends the task-id of interest to the RS (Algorithm 5 line 1). Upon receiving *tid*, the RS responds with tuples of the task-id, report, nonce, and signature for the requested task-id.

If N_d is $\perp$ (line 3), the App verifies the group signature (line 4) and checks for duplicates (caused either by a replay attack or simply by repeated queries to the RS). When all is well, the App stores the report along with the task-id and the signature (as a nonce).

If N_d is not $\perp$, the App verifies the AS's signature (line 11) and checks for

Algorithm 5 Report Retrieval and Verification

1: App $\Rightarrow$ RS : $\langle tid \rangle$ ▷ request reports for a task
2: RS $\Rightarrow$ App : $\{(t_{id}, d, N_d, \sigma_d) \in R \mid t_{id} = tid\}$
{**App does the following for each received report** $\langle t_{id}, d, N_d, \sigma_d \rangle$}
3: **if** $N_d = \perp$ **then** ▷ No AS involved
4: **if** $\texttt{GVerify}(t_{id}||d, \sigma_d) = \texttt{true}$ ▷ Verify MN's group sig.
5: **and** $\langle t_{id}, d, \sigma_d \rangle \notin \mathcal{D}$ **then** ▷ Check replay attack
6: $\mathcal{D} = \mathcal{D} \cup \{\langle t_{id}, d, \sigma_d \rangle\}$ ▷ Accept d as a valid report
7: **else**
8: $\texttt{Discard}(t_{id}, d, N_d, \sigma_d)$
9: **end if**
10: **else** ▷ AS involved
11: **if** $\texttt{Verify}_{\text{AS}}(t_{id}||d||N_d, \sigma_d) = \texttt{true}$ ▷ Verify AS's signature
12: **and** $\langle t_{id}, d, N_d \rangle \notin \mathcal{D}$ **then** ▷ Check replay attack
13: $\mathcal{D} = \mathcal{D} \cup \{\langle t_{id}, d, N_d \rangle\}$ ▷ Accept d as a valid report
14: **else**
15: $\texttt{Discard}(t_{id}, d, N_d, \sigma_d)$
16: **end if**
17: **end if**

duplicates (line 12). If success, it stores the tuple; otherwise, it discards.

3.4 Extended Features

AnonySense can be easily extended to support any combination of the following extensions.

Replicated TS or RS. Although the architecture and protocols described above assume there is only one TS server and one RS server, it is possible to improve scalability and availability by replicating the TS or RS services on multiple servers. Given multiple TS replicas, the RA releases the same set of tasks to each TS (Algorithm 3); each MN can download tasks from any TS. Indeed, by contacting a different TS for each download, an MN may increase its anonymity by making it more difficult for an adversary to link its download activity. Given a system with multiple RS replicas, the RA selects an RS at random when it processes the task. The RA encodes that RS address, in its response to the App (Algorithm 1) and in the task delivered to the TS (Algorithm 2), enabling the MN to submit reports to that RS (Algorithm 4).

Closed Tasking and Reporting. To limit task submissions to some set of "authorized" applications, we extend Algorithm 1 so that the RA first authenticates the App and confirms its credentials for each submitted task. Symmetrically, to ensure that only the submitting App is allowed to retrieve its task's reports, we extend Algorithms 4 and 5 so that an MN encrypts reports under the public key of the App, which is included in the task. Moreover, to allow tasking specific MNs without exposing the task details or destination to other parties,

we extend our protocol so that the App adds a label identifying the target MN, and encrypts the task using the RA's public key. The RA decrypts the task and then encrypts it using a secret key k it shares with that carrier's MN at the time of certification; it prepends a nonce n, and a one-way keyed hash of the target MN's identity along with the nonce, i.e., $H(\text{MN}, n, k)$. This approach provides efficiency and privacy: the target MN can quickly check whether the task is for itself; other MNs do not learn the identity of the target, and waste no time in decrypting the task. The nonce ensures that multiple tasks targeted at the same user appear no different from tasks targeted at different users, and the key prevents dictionary attacks.

Identifiable Tasking and Reporting. Anonymity is not always desired; one might wish to identify the tasking App or the reporting MN. AnonySense can support either or both. To allow a carrier to identify the tasking App or user, we extend our protocols as follows. The App signs the task with an application-specific or user-specific private key. The RA can verify the signature, and state (in its own signature about the task) the identity and the fact that the signature was verified. Any MN may now use this identity, e.g., in deciding whether to accept the task or submit identifiable reports. To allow identifiable reports, the application simply includes "identity" as one of the "sensor values" to be returned by a task, as part of a report statement. If the MN chooses to accept such tasks, it includes the carrier's identity, signs the reports with the carrier's private key, and attaches the certificate for that key. The App can verify the signature and thus learn and validate the origin of the report. As an optimization, the MN can include the certificate in only the first report, expecting the App to cache the certificates for validating future reports. As a generalization, the App may request "sensors" such as "age" or "gender"; the MN attaches the relevant attribute certificates to the report.

Incentive-based reporting. To provide incentives for participation in opportunistic sensing, we extend our protocols to support a direct reward mechanism. [5] We assume that only the application can compute appropriate rewards and pay the carriers for their contributions. On behalf of applications, a system component called the *payment service* (PS) can distribute to carriers the payments related to them. In such a scheme, the adversary should not be able to link a reward to a carrier or to another reward (as being claimed by the same carrier). Consistent with our trust assumptions above, the adversary may collude with the PS and applications. We propose the following delayed reward mechanism. The carrier occasionally contacts the PS to claim rewards for his or her contributions made in the past. Although any cryptographic trapdoor function will do, our scheme uses a one-way function (e.g., SHA2) for performance reasons. Let $F(\cdot)$ be the one-way function and $h(\cdot)$ be a hash function.

[5] Although the literature abounds with incentive-based systems [19–22], only a few support privacy [23,24], and none of them supports the rewarding model of AnonySense.

Given a report d, the MN picks a nonce n_d and computes $\lambda_d = F(h(d)||n_d)$ when reporting to the RS, or $\lambda_d = F(h(N_d)||n_d)$ when reporting to the AS where N_d is the nonce in Algorithm 4 line 5. The MN appends λ_d to the report message and signs it along with the rest of the message (in the lines 2, 3, 6, and 7 in Algorithm 4). After verifying the signature and evaluating the value of the report, the application sends the PS a self-signed payment $(h(d), \lambda_d, p_d)$ or $(h(N_d), \lambda_d, p_d)$ where p_d is an e-cash payment appropriate for report d. To redeem its contribution, the carrier anonymously sends the PS a claim message $(h(d), \lambda_d, n_d)$ or $(h(N_d), \lambda_d, n_d)$. If the PS can verify λ_d using function $F(\cdot)$, it forwards the payment to the carrier. The carrier can verify the payment using the application's public key. The above scheme protects carrier privacy as long as λ_d is fresh for each report and the one-way function is secure.

4 Security Evaluation

In this section, we examine how AnonySense (Section 3) indeed achieves our security goals (Section 2.3). We do so by showing how adversaries will fail in their attacks discussed in Section 2.4. As is standard in such analysis, we assume the underlying security components and tools are secure based on standard assumptions for those tools (e.g., sufficient key lengths are used for cryptographic schemes, only a small fraction of Tor nodes are malicious, at least one MIX node is honest, the RA operator is honest, and so on).

AnonySense prevents the *narrow tasking* attack (i.e., submit tasks with a narrow acceptance condition) because the RA approves only tasks with non-restrictive acceptance conditions and the MN can verify that a task was approved by the RA. To decide whether an acceptance condition is narrow, the RA maintains an attribute database for all MNs and checks whether there are enough MNs (i.e., at least k_a MNs) that meet the acceptance condition (Algorithm 1). To verify that a downloaded task is approved by the RA, the MN checks if the release message, accompanying the task, is signed by the RA (Algorithm 3 line 7) and the hash value of the task matches the one in the release message (Algorithm 3 line 9). Therefore, the narrow tasking attack cannot succeed as long as the attribute database remains up-to-date, the RA private key remains secret, and the RA remains trustworthy.

AnonySense prevents the *tasking de-anonymization* attack (i.e., identify the MN when it downloads some tasks from the TS) by making it hard to link download events with the MN or other download events. Each download event occurs at a random interval, uses a Tor-anonymized source IP address, asks for a random subset of tasks, and does not reveal which tasks are accepted by the MN.

AnonySense prevents the *reporting de-anonymization* attack (i.e., identify the MN when it submits reports) using the MIX-network, the AS, and the RA's task verification. By observing the task-id, the time of receipt, and the sending

IP address (Algorithm 4 lines 3 and 12), the adversary tries to link report actions to an MN, or to other report actions, and then may use external data to identify the MN. In AnonySense, the IP address and report-receipt time do not reveal the MN's identity because reports are sent through a MIX network, which hides the IP address and adds a random delay (Algorithm 4 lines 3), or the AS submits the report on behalf of the MN with the same effect as a MIX network (Algorithm 4 lines 12). The task-id cannot identify the carrier because there are enough MNs that can accept the task (i.e., narrow tasking is prohibited).

For the same reasons, the adversary cannot link multiple reporting actions by only observing reporting actions; IPs are hidden, times are randomized, and there exist enough other MNs for the same task-id). The attacker may try colluding with the TS for selective tasking attack, i.e., arranging for the TS to distribute a specific task only to one MN. Then, any reports for that task arriving at RS are generated by the same MN. AnonySense prevents this selective tasking attack because the TS has no control over who gets what tasks (Algorithm 3 lines 1–5). More precisely, since the MN randomly chooses one of p subsets of all the available tasks, each task t is downloaded by more than N_t/p MNs on average, where N_t denotes the number of MNs that connects to the TS during the lifetime of the task. Depending on the task lifetime and the task-download frequency of MNs, one can decide on the value of p to control how many MNs are given each task on average.

AnonySense prevents the *report analysis* attack (i.e., identify the carrier by looking at the content of the reports) because of the AS. Many techniques have been proposed to anonymize reports with sensitive data [13,14,18,25–27,15,28,29]. The anonymization server (AS) is a trusted component of AnonySense that is responsible for anonymizing reports with sensitive information. The AS could implement any state-of-the-art anonymization algorithm, and its privacy guarantees vary with the algorithm. For example, the AS can guarantee k-anonymity for each report, perturb sensitive fields, or induce confusion among mobility paths. For some anonymizations to work well, the AS needs access to the MN attribute database.

In AnonySense, the MN decides whether anonymization is necessary for its reports (Algorithm 4 line 1) and, if so, sends the report to the AS (line 7). A report is assumed to be sensitive if it contains the location of the carrier or other static attributes (called *quasi-identifiers* [18]) that may potentially identify the carrier.

AnonySense prevents the *local eavesdropping* attack because the communication with the TS, the RS, and the AS is encrypted due to onion encryption by Tor/MIX or the secure channel with the AS. Because of the encryption, the eavesdropper does not know whether the MN is contacting the TS, RS or any other non-AnonySense destination. A local eavesdropper may recognize

an MN's connection to the AS, but (due to encryption) sees none of the report and (by trust assumption) cannot collude with the AS.

AnonySense prevents the *report tampering* and the *report forgery* attack by the MN's group signature (Algorithm 4 line 2), the MN's digital signature (Algorithm 4 line 6), the AS's digital signature (Algorithm 4 line 11), and the App's trust in the AS. Therefore, the application can detect any unauthorized changes in reports due to the signatures (Algorithm 5 lines 4 and 11).

AnonySense prevents the *report replay* attack by the use of nonces: the group signature σ_d (used as a nonce as well) in Algorithm 4 line 2 and the nonce N_d in Algorithm 4 line 5. Note that the group signature can work as a nonce because the group-signing algorithm produces a different signature for each signing — even for the same message. The application and AS check nonces to detect duplicated reports (Algorithm 5 lines 5 and 12). Any attempt to replace a nonce is detected via mismatching signatures (Algorithm 5 lines 4 and 11).

In the above analysis, we showed that the AnonySense architecture and protocols can effectively prevent the plausible threats identified in Section 2.4. In the context of the considered attacks and the adversary's assumed capabilities, AnonySense achieves our security goals – carrier privacy (SG1) or report integrity (SG2) – set forth in Section 2.3, fulfilling the privacy and security requirements of opportunistic people-sensing systems.

5 Performance Evaluation

We implemented the complete AnonySense system, except for the anonymization server (AS). The services run on generic Linux computers. 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 full implementation and evaluation. Our performance evaluation focuses on the cost the implementation imposes, particularly on the mobile nodes' resources: network bandwidth, CPU time, and battery life.

5.1 Implementation

Figure 3 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. We use the Tor service provided by Torproject.org. Although a real AnonySense deployment would require multiple MIX nodes, we needed only one node for the purpose of our measurements. Realistic MIX latency information is easily obtained from a pinger. [6] The services were connected to Dartmouth's wired network (100 Mbps switched Ethernet), and the MNs were connected to Dartmouth's Wi-Fi wireless network.

[6] http://pinger.mixmin.net/

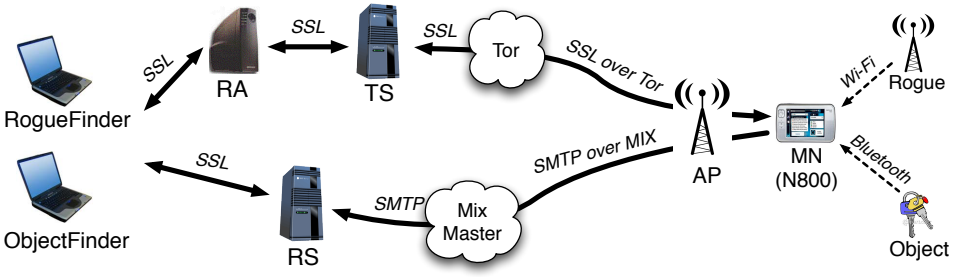

Fig. 3. The architecture of our prototype implementation.

We do not implement the AS because anonymization necessarily depends on the nature of the report data; one could easily deploy existing implementations of anonymization algorithms in AnonySense. For example, given an anonymization module, one can add the interface with MNs and the RS as described in Algorithm 4.

Communications. Our implementation leverages standard protocols that use open-source libraries, resulting in compact and robust code. The Apps, the RA, the TS, and the RS communicate with each other over SSL-authenticated HTTPS channels (double arrows in the pseudo-code). Some connections require mutual authentication while other connections require one-way authentication; the authenticated entities are underlined in the protocol descriptions. The MN communicates with the TS through an SSL channel over the Tor network while it communicates with the RS through SMTP over the MIX network.

Servers. The AnonySense services are written in the Ruby programming language (v1.8). Both the TS and RA are implemented using Camping [30], a micro-framework for developing small HTTP servers, with the actual SSL and HTTP handling done by Mongrel [31]. 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. 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).

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 [32], but it is possible to use any other such library with minimal effort.

When reporting, the MN generates a report in XML,[7] 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.

Finally, we note that our AnonySense implementation has been successfully used by another research group, who integrated it with their system [33].

5.2 Applications

To demonstrate AnonySense operation, we implemented two simple applications. Each uses the network interfaces of the N800 as sensors.[8] 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 do so, 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. After collecting the reports, 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, RogueFinder can display a marker on a map that is the approximate location of the rogue AP. Using mobile MNs, RogueFinder can potentially detect rogues where static sensors may not see them.

Application ObjectFinder. Our inspiration for ObjectFinder comes from a similar application described earlier [8]. 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 might be used for more accurate object positioning.

[7] 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.

[8] We have also written similar applications that use the N800's microphone for sound-level measurements, but we do not include the results in this paper.

5.3 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 between the battery and the device across a test resistance of 0.5 Ω using an Agilent 34401A multi-meter. The net energy consumption of AnonySense was computed by subtracting the base energy use (i.e., no application was running) from the energy use when AnonySense was running. [9] 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. During one walk 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. We then conducted controlled experiments in the lab to measure resource consumption; in what follows, each measurement is the average of 50 repetitions unless indicated otherwise. It took 14.5 seconds on average for the MN to perform one scan and issue a report. In our experiment, the average power cost was 495 mW and a complete scan-report cycle cost 7.44 Joules on average. As a rough benchmark, this power consumption was 17 times smaller than streaming a song in MP3-quality on the N800. Since the MN downloads a batch of tasks, we measured the cost of downloading 1000 tasks of the same length as RogueFinder. In our experiment, the average power consumption for downloading 1000 tasks was 2.23 Joule. The time for downloading tasks depends on the bandwidth of the wireless connection. In our experiment, it took 8.2 seconds to download 1000 tasks with a bandwidth of 113 Kbps.

Benchmarking. Table 1 illustrates how the energy cost of RogueFinder compares with the cost of various multimedia applications to give an intuitive sense of the magnitude of energy consumption in AnonySense. For example, the energy consumed by one cycle of RogueFinder is equivalent to the energy consumed by playing a local MP3 file for 26.1 seconds. We also note that the power consumption by RogueFinder is similar to that of streaming radio (in 33 kbps quality). In a separate experiment, we found that RogueFinder had a minimal effect on the overall battery lifetime of a fully charged N800: in one experiment, RogueFinder reduced battery lifetime from 279 to 262 min-

[9] These numbers differ from our conference paper [34], where we made errors in the calculation of the energy measurements. These tables provide correct measurements.

Table 1
Jobs equivalent to one cycle of a RogueFinder task (7.44 Joule)

Application	Power	Job
Local MP3 play	284.6 mW	26.1 s
Streaming Radio	377.2 mW	19.7 s
Streaming MP3	524.0 mW	14.2 s
Local Video play	636.6 mW	11.7 s
Streaming Video	820.2 mW	9.1 s
Download	955.5 mW	7.8 s

Table 2
Energy cost of task sub-operations

Operation	Time	Power	Energy	Fraction
Wi-Fi Sensing	7.2 s	525.2 mW	3.78 J	50.8 %
Group Signing	5.2 s	530.5 mW	2.76 J	37.1 %
Reporting	2.1 s	429.7 mW	0.90 J	12.1 %
RSA Signing	0.02 s	350.4 mW	0.008 J	
BT Sensing	10.7 s	280.5 mW	3.03 J	

utes (by 6.1%), and in another experiment, it reduced battery lifetime from 291 minutes to 278 minutes (by 4.5%). [10] In this experiment we simulated a network-heavy usage scenario by playing streaming audio continuously while downloading 20 emails per hour. The scanning operation was also heavy; one RogueFinder cycle per minute.

Detailed energy consumption. One sensing task can be divided into several sub-operations: *sensing* (Wi-Fi scanning for RogueFinder or Bluetooth scanning for ObjectFinder), *signing* (Group signing for insensitive data or RSA signing for sensitive data), and *reporting*. Table 2 shows the cost of each sub-operation. The energy fraction of each sub-operation is with respect to the total cost of RogueFinder using group signatures. As shown in the table, the sensing took the most time (about 50%) and energy (50.8%). The next most expensive operation was computing group signatures of reports. However, when RSA signature was used, i.e., for sending reports to the AS for anonymizing the data, the signing cost was negligible.

[10] Due to the length of battery-lifetime experiments, it was not feasible to run more than two times.

6 Discussion

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

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 RS 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, changing the MAC address for each report could take time. When the report is sent to the AS, the MIX delay is replaced with the delay imposed by the AS, during which it aggregates or mixes the report with other reports for anonymity or trace confusion. We believe, therefore, that AnonySense is best suited to delay-tolerant applications.

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 [35].) AnonySense, like CarTel [1], leverages the growth of open-access Wi-Fi networks, and AnonySense is designed to ensure carriers' anonymity while contributing sensing data for community use.

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 [36], 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 [37].

One could task mobile nodes to send images or video from locations of interest; one set of researchers uses peer-to-peer communications for nearby cellphones

to coordinate video capture and analysis [38]. (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.

For public safety, consider MNs with 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.

There are many potential opportunities related to wellness or health care [39], social networks [40,41], city dynamics [42], traffic monitoring [43], and other applications [1–3,5–7].

Finally, the COPSE project at Duke University has integrated the AnonySense system into their code base and they are developing applications [33].

7 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 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. AnonySense can optionally identify the user submitting the task, the carrier to be tasked, or the carrier submitting the report; the application decides which form(s) of identification are needed, but the mobile-device carrier decides whether to accept such tasks.

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.

References

[1] 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 *Proc. of SenSys*. ACM, Nov. 2006, pp. 125–138. DOI 10.1145/1182807.1182821

[2] 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 Comp.*, vol. 6, no. 2, pp. 20–29, 2007. DOI 10.1109/MPRV.2007.38

[3] O. Riva and C. Borcea, "The Urbanet revolution: Sensor power to the people!" *IEEE Pervasive Comp.*, vol. 6, no. 2, pp. 41–49, 2007. DOI 10.1109/MPRV.2007.46

[4] (2009, Dec.) Urban atmospheres. Intel Research. Available online: http://www.urban-atmospheres.net

[5] (2008, Dec.) CENS Urban Sensing project. Center for Embedded Network Sensing. Available online: http://research.cens.ucla.edu/projects/2006/Systems/Urban_Sensing/

[6] A. Kansal, S. Nath, J. Liu, and F. Zhao, "SenseWeb: An infrastructure for shared sensing," *IEEE Multimedia*, vol. 14, no. 4, pp. 8–13, 2007. DOI 10.1109/MMUL.2007.82

[7] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson, "People-centric urban sensing," in *Proceedings of theInternational Wireless Internet Conference (WICON).* ACM, Aug. 2006, pp. 18–31. DOI 10.1145/1234161.1234179

[8] C. Frank, P. Bolliger, C. Roduner, and W. Kellerer, "Objects calling home: Locating objects using mobile phones," in *Proc. of Pervasive*, ser. LNCS, vol. 4480. Springer, May 2007, pp. 351–368. DOI 10.1007/978-3-540-72037-9_19

[9] K. P. Tang, P. Keyani, J. Fogarty, and J. I. Hong, "Putting people in their place: An anonymous and privacy-sensitive approach to collecting sensed data in location-based applications," in *Proc. of SIGCHI.* ACM, Apr. 2006, pp. 93–102. DOI 10.1145/1124772.1124788

[10] D. Peebles, C. Cornelius, A. Kapadia, D. Kotz, M. Shin, and N. Triandopoulos, "AnonyTL specification," Dartmouth College, Tech. Rep. TR2010-660, Jan. 2010. Available online: http://www.cs.dartmouth.edu/reports/TR2010-660.pdf

[11] (2009, Dec.) Tor: anonymity online. The Tor Project. Available online: http://www.torproject.org

[12] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. (2004, Dec.) Mixmaster protocol — version 3. IETF Internet Draft. Available online: http://www.ietf.org/internet-drafts/draft-sassaman-mixmaster-03.txt

[13] B. Hoh and M. Gruteser, "Protecting location privacy through path confusion," in *Proc. of SecureComm.* IEEE Computer Society, 2005, pp. 194–205. DOI 10.1109/SECURECOMM.2005.33

[14] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proc. of MobiSys.* ACM, May 2003, pp. 31–42. DOI 10.1145/1066116.1189037

[15] A. Kapadia, N. Triandopoulos, C. Cornelius, D. Peebles, and D. Kotz, "AnonySense: Opportunistic and privacy-preserving context collection," in *Proc. of Pervasive*, ser. LNCS, vol. 5013. Springer, May 2008, pp. 280–297. DOI 10.1007/978-3-540-79576-6_17

[16] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Proc. of CRYPTO*, ser. LNCS, vol. 3152. Springer, Aug. 2004, pp. 41–55. DOI 10.1007/b99099

[17] T. Jiang, H. J. Wang, and Y.-C. Hu, "Preserving location privacy in wireless LANs," in *Proc. of MobiSys*. ACM, Jun. 2007, pp. 246–257. DOI 10.1145/1247660.1247689

[18] L. Sweeney, "k-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, Oct. 2002. DOI 10.1142/S0218488502001648

[19] M. Jakobsson, J. P. Hubaux, and L. Buttyan, "A Micro-Payment Scheme Encouraging Collaboration in Multi-Hop Cellular Networks," in *Financial Crypto 2003*, 2003.

[20] L. Buttyán and J.-P. Hubaux, "Stimulating cooperation in self-organizing mobile ad hoc networks," *Mob. Netw. Appl.*, vol. 8, no. 5, pp. 579–592, 2003. DOI http://dx.doi.org/10.1023/A:1025146013151

[21] S. Zhong, Y. Yang, and J. Chen, "Sprite: A simple, cheat-proof, credit-based system for mobile ad hoc networks," in *Proceedings of IEEE INFOCOM*, 2002, pp. 1987–1997. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.1213

[22] N. B. Salem, L. Buttyán, J.-P. Hubaux, and M. Jakobsson, "A charging and rewarding scheme for packet forwarding in multi-hop cellular networks," in *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*. New York, NY, USA: ACM, 2003, pp. 13–24. DOI http://doi.acm.org/10.1145/778415.778418

[23] J. Camenisch and E. Van Herreweghen, "Design and implementation of the *idemix* anonymous credential system," in *Proc. of ACM CCS*. ACM, 2002, pp. 21–30. DOI 10.1145/586110.586114

[24] E. R. Verheul, "Self-blindable credential certificates from the Weil pairing," in *Proceedings of theInternational Conference on the Theory and Application of Cryptology and Information Security*. Springer-Verlag, Dec. 2001, pp. 533–551.

[25] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Preserving privacy in GPS traces via uncertainty-aware path cloaking," in *Proc. of ACM CCS*. ACM, Oct. 2007, pp. 161–171. DOI 10.1145/1315245.1315266

[26] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," in *Proc. of ICDCS*. IEEE, Jun. 2005, pp. 620–629. DOI 10.1109/ICDCS.2005.48

[27] G. Iachello, I. Smith, S. Consolvo, M. Chen, and G. D. Abowd, "Developing privacy guidelines for social location disclosure applications and services," in *Proc. of SOUPS*. ACM, Jul. 2005, pp. 65–76. DOI 10.1145/1073001.1073008

[28] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preserving anonymity in location based services," National University of Singapore, Department of Computer Science, Tech. Rep. TRB/06, 2006. Available online: http://en.scientificcommons.org/13004931

[29] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new Casper: query processing for location services without compromising privacy," in *Proc. of VLDB*. VLDB Endowment, May 2006, pp. 763–774. Available online: http://www.vldb.org/conf/2006/p763-mokbel.pdf

[30] (2009, Nov.) Camping. Open source code. Available online: http://github.com/camping/camping

[31] (2009, Dec.) Mongrel. Open source code. Available online: http://mongrel.rubyforge.org/

[32] (2007, Sep.) Skyhook. Skyhook Wireless. Available online: http://www.skyhookwireless.com/

[33] L. Cox. (2008, Nov.) The COPSE project. http://copse.cs.duke.edu/. Available online: http://copse.cs.duke.edu/

[34] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "AnonySense: Privacy-aware people-centric sensing," in *Proc. of MobiSys*. ACM, Jun. 2008, pp. 211–224. DOI 10.1145/1378600.1378624

[35] E. Nakashima, "Cellphone tracking powers on request: Secret warrants granted without probable cause," *Washington Post*, p. A01, Nov. 2007. Available online: http://www.washingtonpost.com/wp-dyn/content/article/2007/11/22/AR2007112201444.html

[36] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "BikeNet: A mobile sensing system for cyclist experience mapping," *ACM TOSN*, vol. 6, no. 1, pp. 1–39, 2009. DOI http://doi.acm.org/10.1145/1653760.1653766

[37] (2009, Dec.) Open street map. OpenStreetMap Foundation. Available online: http://www.openstreetmap.org/

[38] T. Simonite, "Cellphones team up to become smart CCTV swarm," *New Scientist*, Oct. 2007. Available online: http://technology.newscientist.com/article/dn12861-cellphones-team-up-to-become-smart-cctv-swarm.html

[39] "Continua alliance," http://www.continuaalliance.org/use_cases/, use cases available on the web. Available online: http://www.continuaalliance.org/use_cases/

[40] 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 *Proc. of MobiSys*. ACM, Jun. 2008, pp. 174–186. DOI 10.1145/1378600.1378620

[41] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, S. Eisenman, H. Lu, M. Musolesi, X. Zheng, and A. Campbell, "Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application," in *Proc. of SenSys*. ACM, 2008, pp. 337–350. DOI 10.1145/1460412.1460445

[42] J. Froehlich, J. Neumann, and N. Oliver, "Measuring the pulse of the city through shared bicycle programs," in *Proc. of UrbanSense*, Nov. 2008. Available online: http://sensorlab.cs.dartmouth.edu/urbansensing/papers/froehlich_urbansense08.pdf

[43] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Proc. of SenSys*. ACM, 2008, pp. 323–336. DOI 10.1145/1460412.1460444