A Platform for Privacy Protection of Data Requesters and Data Providers in Mobile Sensing

Ioannis Krontiris[a], Tassos Dimitriou[b,c,*]

[a] *Goethe University Frankfurt, Grueneburgplatz 1, 60323 Frankfurt, Germany*
[b] *Computer Engineering Dept., Kuwait University, Kuwait*
[c] *Research Academic Computer Technology Institute, Patras, Greece*

Abstract

In typical mobile sensing architectures, sensing data are collected from users and stored in centralized servers at third parties, making it difficult to effectively protect users' privacy. A better way to protect privacy is to upload sensing data on personal data stores, which are owned and controlled by the users, enabling them to supervise and limit personal data disclosure and exercise access control to their data. The problem however remains how data requesters can discover the users who can offer them the data they need. In this paper we suggest a mobile sensing platform that enables data requesters to discover data producers within a specific geographic region and acquire their data. Our platform protects the anonymity of both requesters and producers, while at the same time it enables the incorporation of trust frameworks, incentive mechanisms and privacy-respecting reputation schemes. We also present extensive experimental results that demonstrate the efficiency of our approach in terms of scalability, load balancing and performance.

Keywords: Mobile Ubiquitous Sensing, Privacy, Anonymity, Cloud, Auctions, Reputation

1. Introduction

The increasing availability of sensors on today's smartphones and other everyday devices, carried around by millions of people, has opened up diverse kinds of information gathering by people and their devices. Eventually, researchers envision the creation of a unified data-sharing infrastructure, where people and their mobile devices provide their collected data streams in accessible ways to third parties interested in integrating and remixing the data for a specific purpose. This trend is often named *mobile crowd sensing* [1].

Several of the works on mobile crowd sensing systems started differentiating very early between two data collection models [2]. In the participatory model, users are actively involved in the collection process by deciding on the spot when to report data, while in the opportunistic model, sensor sampling occurs whenever the state of the device (e.g. geographic location) matches the application's requirements described in a sensing task, without the knowledge of the individual phone user.

Independently from the collection model however, what is common in the majority of existing architectures is that the sensing data collected from the mobile phones are stored in centralized servers at third parties, creating massive databases of individuals' location, movements, images, and even health data. After collecting the data, the entity controlling the database aggregates, processes and releases them through various interfaces (e.g. statistical data on a map).

This paradigm raises several challenges concerning information access and reciprocity. Who controls data collection and who owns the data or benefits from them? In most cases, the data are collected, stored, and analyzed by data processors typically out of view of the individual whose life they describe. The collection of the data is not always restricted to the purpose for which they were collected. Also, individuals cannot pose restrictions on the collection and processing of their data and the data collected from them are not made available back to them through proper interfaces.

To deal with the power imbalance created in such paradigms, architectures taking a more user-centric approach started to appear [3]. What these architectures try to do is to enable individuals to supervise and limit personal data disclosure and exercise access control to their data by third parties. Several existing solutions in crowd sensing applications suggest a vault-like entity to provide an online trusted storage and processing. Mobile phones sense and upload data to this vault, which is owned and controlled by the individual. The process of storing personal data streams is decoupled from the sharing of that information. After the collection and archival of data, the users can define their own privacy policies and review/control who can see which kind of data.

The problem that remains largely unexplored in these

*Corresponding author
Email address: tassos.dimitriou@ieee.org (Tassos Dimitriou)

Authors' copy: Please cite this article as: I. Krontiris, T. Dimitriou, A platform for privacy protection of data requesters and data providers in mobile sensing, Computer Communications, Elsevier, Vol. 65, Pages 43-54, (2015), http://dx.doi.org/10.1016/j.comcom.2015.02.005

architectures is that of *information discovery* from data consumers. We refer specifically to the case where data requesters, either being applications or physical persons, are interested in retrieving information according to some requirements (location area, time frame, sensor type, etc.) from multiple data contributors that satisfy these requirements. That means they need to search data from data contributor's individual data stores, since there is not a central place where all data is gathered. Given the distributed nature of data stores controlled by the corresponding data producers, this is not trivial to do.

But discovering data providers in a specific geographic area is not sufficient for a requester, because not all of them can provide the same quality of data. Data providers can inadvertently position their device in the wrong place while collecting sensor readings or they could deliberately contribute bad data. So, requesters want to select specific data contributors based on criteria like their reputation gathered in previous participations, in order to guarantee some quality in the sensed data. It is also usual that the requestor offers some micro-payments to the producers for motivating them to contribute. Then an additional selection criterion could be based on the amount that the producers require for their data.

What makes the problem more challenging is the growing requirement of protecting requester's data access privacy; a user may want to keep confidential whether (and when) she accessed the sensed data, the data types she was interested in, or from which nodes she obtained the data, as the disclosure of such information may be used to infer additional context about the user and used potentially against her interest. This means that data requests cannot be linked to the real identity of the requester. However, some access control mechanism is needed, so that not anybody can take benefit of the platform's services without demonstrating some sort of permission. How this permission can be obtained depends on the business model of the platform provider. For example, it could be that the requester has to pay for each "*sensing quantum*". To preserve the requester's privacy, the process of acquiring such a sensing quantum and the process of demonstrating it to the data providers for enabling the processing of the request should be unlinkable to each other.

Our Contribution. In this paper, we suggest a platform that enables data requesters to discover data providers in a specific geographic region of interest and retrieve data from them, while protecting the privacy of both against each other and the platform providers. Data providers maintain location privacy according to their own preferences and access policies to the data they own, while requesters may contact directly the mobile users in the area of interest and select the ones to get sensing data from, based on their own criteria.

Organization. In Section 3, we discuss the system and threat model and highlight the major components of the platform. In this work cloud agents represent the mobile users to the outer world according to the user's location privacy preferences. They interconnect with each other in such a structure that enables data requesters to discover mobile users in a specific geographic area. This structure (Section 4) is not stored by a central entity, but it is maintained by the agents in a *distributed* fashion, thus avoiding the bottlenecks and the privacy implications of centralised approaches. Requesters obtain *tokens* from the service provider in order to have access to the data provided by the selected data providers. Using appropriate cryptographic mechanisms that we describe in Section 5, the validity of the token can be verified without leaking the identity of the requester to the node or to the application owner. Sections 6 and 7 describe how the platform can incorporate incentives and reputation management mechanisms. Finally, in Section 8 we conduct extensive experiments demonstrating the efficiency of our approach in terms of scalability, load balancing and performance.

2. Related Work

In the participatory sensing domain, various *centralized* solutions for distributing tasks or queries to sensor nodes have been proposed. In PRISM [4], participating nodes (i.e. mobile phones) register with the server and the server tracks the nodes and pushes only matching tasks to them, based on their context (e.g. location). For example, Alice may be assigned the task "measure temperature in area X", when she is entering this area. However, this solution does not consider privacy for any of the involved entities, queriers or mobile nodes.

A solution that offers a privacy-friendly way of task distribution is AnonySense [5]. Sensing tasks are posted on a server and the nodes download the tasks and match them to their context to decide which one to execute. This approach has the advantage that the nodes do not reveal anything about their context to the service provider, in order to receive the sensing task. Still, AnonySense does not consider privacy for the entities posting the tasks.

Recently, PEPSI [6] was suggested as a system designed with the privacy of the queriers in mind, queriers being entities external to the platform, who are interested in some specific sensing information. PEPSI is based on a *centralized* solution and to protect the privacy of the queriers, it introduces a Registration Authority, a trusted third party which collects queries from the queriers and provides back the corresponding cryptographic material. In that sense, the queries reach the platform in an encrypted form. However, the problem is shifted to the Registration Authority, where essentially all queries are known in advance, along with the identities of the queriers, leading to the assumption that this entity must be trusted.

To protect the privacy of data providers, we employ the concept of user-owned proxies. The use of stationary proxies has been already suggested in some participatory sensing systems so far, where they are used as data

vaults or brokers for the user [7]. For example, Mun et al. proposed Personal Data Vault (PDV) [8], which functions as individual data storage with fine-grained access control mechanism, privacy rule recommender, and trace audit. Choi et al. also presented SensorSafe [9], an architecture that consists of multiple remote data stores and a broker enforcing a fine-grained access control by supporting privacy rules with context/behavior conditions and control for levels of inferences. What is common in all of the above systems is that the mobile phones sense and upload their data to their corresponding proxies *proactively*. Then the proxy is responsible to help the user manage this data and make it available to third parties, functioning as an access control mechanism. In our work, mobile nodes perform sensing operations only *reactively*, when there is a specific query to which they can respond to.

Drosatos et. al. [10] presented recently a privacy-respecting solution following the same reactive model, where mobile agents on the cloud store the data encrypted and execute a cryptographic protocol based on a homomorphic encryption scheme in order to aggregate the data and make them available. However, this setting does not consider the privacy of the data requesters, neither does it enable them to apply selection criteria on specific mobile nodes.

3. System and Threat Model

3.1. Main entities of the system

The Data Provider. Users carry personal sensors, either embedded in their mobile phones or part of wearable devices, and collect contextual data from their immediate environment. We will use the term *Data Provider* to refer to these entities in the system who are actively sharing their sensing data with others. Part of the data provider is the mobile object (MO), which represents the device that the data provider is carrying, as well as a corresponding software agent running on a cloud, which operates on behalf of the MO and represents its interests (see Figure 1).

The Cloud Agent. A cloud agent serves as a proxy that extends and enhances the capability of the MO. As it resides on a stationary infrastructure, it has much higher availability and has not significant limitations on storage, communication and energy. It can thus offload tasks from mobile devices, that can be easily performed outside the device, and at the same time appear as the device's front-end to the rest of the world. This idea is well supported by the recent directions in cloud computing for resource-constrained mobile devices [11]. The device can upload personal information opportunistically to the agent, while the agent takes over the role of presenting these data to third-parties according to the owner's privacy preferences. In this way the device preserves its limited resources for the direct needs of its owner.

The Data Requester. In our model, there is also the data requester, who is interested in collecting data that satisfy some specific requirements. These requirements consist of the definition of a geographic area (range query), but also of additional parameters, like the kind of sensors needed, the time frame, etc. For simplicity, in this paper we assume that the geographic area is the only parameter, but the system can easily be extended to support additional ones. The goal of the system is to provide the data requester with a list of all the MOs that satisfy these requirements.

This comes down to providing the data requester with the contact details of the corresponding agents. In our solution, all the agents interconnect with each other and form a quadtree, preserving a hierarchy according to location and location precision they want to reveal. In the next sections we will explain how this structure is being built and updated. What is important to emphasise here is that there is *no central entity* maintaining the topology information, but instead the network is distributed, with all the agents storing and maintaining its structure.

Following that, the requester can narrow down the list of possible contributors by applying additional criteria, such as for example their trustworthiness. Then, the next step is to contact the MO agents and forward them the data request.

3.2. Nature of communications

Communication between the various entities of the system is mostly wireless which can lead to violation of privacy by (say) identifying the access point through which the network connection was made. In such a communication paradigm, user behavior leaves a lot of traces regarding who accessed what and at what time. Providing anonymity at the first hop of communication, i.e. between the user and the mobile operator, is a problem that is beyond the scope of this work as the involved parties (mobile providers and service providers) can potentially violate user privacy by *colluding* with each other.

In this work, we consider only attackers who are able to observe traffic over the Internet, between the access point and the platform or data providers. Obviously, in such a case additional mechanisms are required to ensure that a connection cannot be traced back to a requesting user. At this level the goal is to provide communication anonymity, which means hiding the network identifiers in the network layer. In this case, data requests must be sent with a help of a proxy or through an anonymization network such as TOR [12] in order to hide the origins of reports, anonymize the connection with the platform and prevent an identification of the various entities from their IP addresses or their access points.

3.3. Nature of attackers

Ensuring anonymity of communications provides security only against *outside* attackers, i.e. third parties trying to find information about a mobile user's access patterns

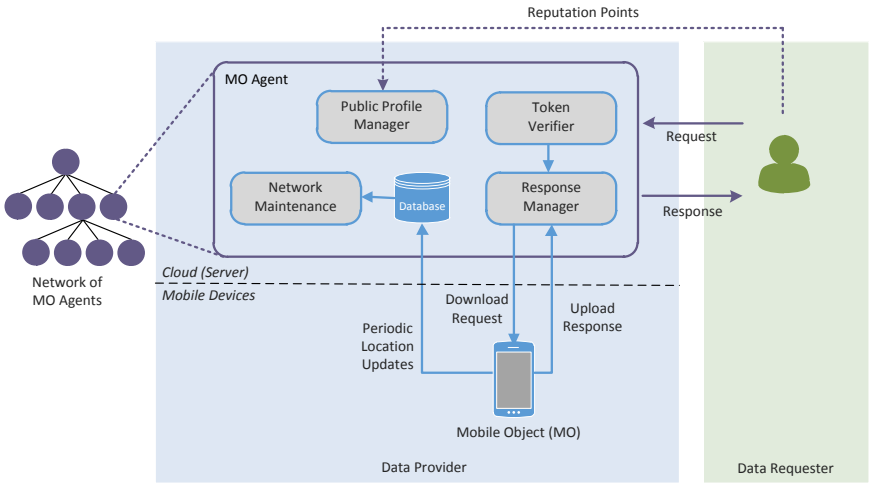


Figure 1: Query privacy in the context of mobile sensing.

and locations. However, our threat model needs to be extended beyond this simple case to include possible malicious *insiders*.

While security and privacy of transactions against external attackers (any entity eavesdropping on data communications) can be handled with appropriate cryptographic mechanisms, security against internal attackers (platform provider or other entities interacting with the platform) is more difficult to provide. Questions that need to be answered include: How the privacy preferences of data providers are ensured? How they remain anonymous when providing for sensed data? Can a data provider find the real identity of the data requester? How can the system preserve the privacy of the requester, by allowing her to stay anonymous (or pseudonymous) throughout the process? Can a token purchase (request for data) be linked to a particular data requester? Can two tokens be linked to the same requester? And so on. All these issues are summarized in the following requirements:

- Data providers should remain anonymous against both the platform and the data requesters. Location privacy of data providers should be also protected.
- Data requesters should remain anonymous against both the platform and the data providers. They should be able to provide evidence that they have permission to request data from the data providers, however, without revealing any other information about them.

An implicit assumption we make here is that the cloud infrastructure provides enough security guarantees so that an attacker (internal or external) cannot compromise the user agent, including the cloud provider itself. As a consequence, any data and processing associated with a MO agent can be accessed and controlled *only* by its associated user.

We close this section by outlining the agent's structure and responsibilities that will ensure the security goals mentioned above. The main components of an agent will be further analyzed in subsequent sections.

3.4. Structure of a MO agent

Figure 1 depicts the general structure of a MO agent. The MO periodically updates its own agent about the changes of its location. Given that both entities are controlled by the user, this update can consist of accurate location information, depending on the technology used at the mobile device (GPS, Wi-Fi, etc.) The agent uses the received information to update its position in the quadtree. As we will see shortly, the agent does not reveal the location in full precision, but it obfuscates it according to the privacy preferences of the user. As a result, the MO agent resides in a place in the tree that corresponds to the given location and the given level of obfuscation.

Overall, an agent has several responsibilities with regard to its MO, as well as the maintenance of the network structure. The following modules of the agent are responsible for these operations.

Network Maintenance. Each MO agent takes some share from the task of maintaining the quadtree structure of all the MO agents. They participate in building and hosting the information about the network topology and the links between the peers. The operations in this module are explained in Section 4.

Token Verifier. The requester presents a token to the MO agent together with the data request. This token protects the requester's anonymity, while at the same time it proves that he is an eligible user of the platform. Once the Token Verifier is convinced that the token is both valid and fresh, it forwards the attached request to the Response Manager for providing the data. In Section 5 we explain in detail the protocol of token issuance and verification.

Response Manager. The MO agent is in charge of processing the received data requests and enforce the sharing policies set by the user. Such policies could restrict sharing for example to a specific time of the day, or within specific areas, etc. The filtering could also be based on the incentives offered by the data requester. In Section 6, we discuss this in more detail.

Public Profile Manager. After receiving the data, the data requester has the opportunity to evaluate its quality and provide feedback to the data provider in form of reputation points. These reputation points are published on the public profile of the provider and it can be used in the future by data requesters to filter out those who provide data of bad quality. The process of updating the reputation must maintain the anonymity of the data provider. We discuss this further in Section 7.

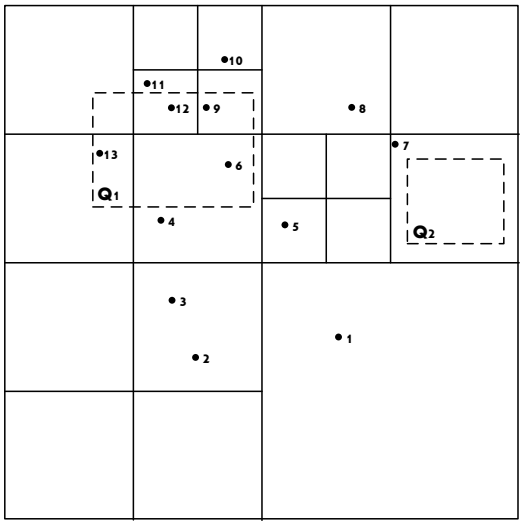
4. Network Manager

As mentioned above, the goal of the system is to enable the data requester to come in contact with the mobile nodes that are currently within a specific region. Additionally, the system allows users to obfuscate their location through their agents using spatial cloaking, as a way to protect their location privacy.

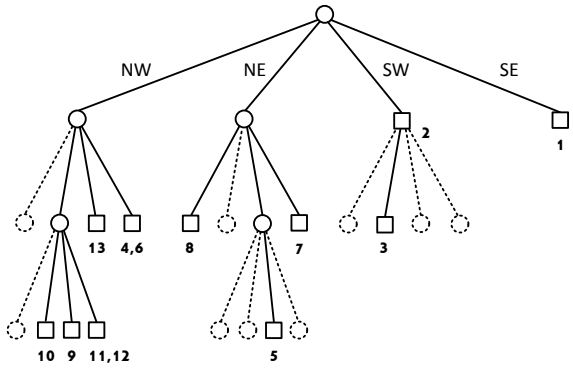
Obfuscation is the process of degrading the quality of information about a person's location, with the aim of protecting that person's location privacy. Most research to date has looked at the use of imprecision to degrade the quality of location information (e.g., [13, 14]). The concept of location obfuscation for mobile users was also studied within the EU-Project PICOS, where it was implemented and tested with actual users through extensive user trials. The experiments showed that the concept was actually the most desirable one amongst a set of measures that helped mobile users protect their privacy [15].

The need to serve spatial queries on one side and to enable location obfuscation on the other is best served by partitioning geographic space with a fixed space grid and use a region quadree structure as location data indexing method [16]. In particular, the space is partitioned in two dimensions by decomposing the region into four equal quadrants, subquadrants, and so on until a predefined limit is reached. MOs obfuscate their location by spatial cloaking, that is by declaring their position based on these fixed quadrants, choosing the one with size closer to the obfuscation level they want. So, we move away from the typical solution of obfuscating location by defining a circle or square around the current location of the MO, which would move along with the MO and would make location profiling still possible. The choice of pre-defined quadrants as obfuscation areas better protect location privacy.

For example, Figure 2a shows an example of 13 mobile objects scattered in a region. Each of these MOs obfuscate their real location at different levels, choosing different



(a)



(b)

Figure 2: (a) A region and its decomposition in quadrants. Mobile objects and queries are also depicted. (b) Quadtree representation of the same region.

size squares. So, for instance, MO_1 has the biggest level of obfuscation, while other objects like MO_{11} and MO_{12} declare their location being in more granular squares. This way of obfuscation implementation is different than classical solutions, where users are allowed to simply blur their location by declaring a region of a specific radius around their true location. Instead, here blurring is done using fixed squares of different sizes.

The above partition of space can be represented by a quadtree [17]. In the tree representation, the root node corresponds to the entire region we want to cover (e.g., a whole city). Each child of a node represents a quadrant (labelled in order NW, NE, SW, SE) of the region. As we go further down in the tree, nodes represent more accurate location information, until a predefined limit f_{max} (granularity) is reached. This limit f_{max} is predefined by

the system and it corresponds to the maximum height of the tree. When a MO agent declares its position within a region, then it is also registered inside the corresponding tree node.

Each tree node stores two kinds of information: the addresses of the MOs registered by it and the pointers to its children. It can be the case that a node has no registered MOs and we call these kind of nodes *control nodes*. For example, Figure 2b shows the quadtree that corresponds to the region in Figure 2a. In the figure, a tree node with registered agents is represented with a square, while control nodes are represented with a circle. We have represented the missing children of a node with a dashed line. In this particular example of Figure 2, the maximal allowed location granularity is $f_{max} = 3$ and therefore the height of the corresponding quadtree is also 3.

There are two differences here, compared to the typical definition of a quadtree in the bibliography. First, we allow MOs to be stored in intermediate nodes and not just in the leaves. Second, we do not create a complete tree, i.e., if a leaf node has no registered MOs, we do not construct it.

What is important to realise is that the quadtree is not stored in a central place, but it is maintained in a distributed fashion among all participating MO agents (peers). Each peer stores its own share of the tree, while the tree decomposition is implicitly known by all the peers in the system, without the need for any communications. Below we explain in detail the basic operations on the quadtree, in order to make clear how it is structured and maintained.

4.0.1. Initialize

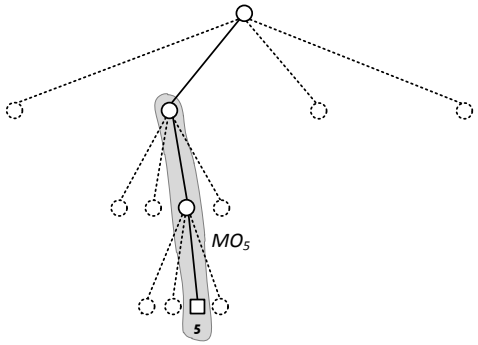
At the beginning only the root node exists, which represents the whole city area. The root is the only node stored centrally in the service provider, so that everybody can use it as an entry point to the rest of the tree. We do not create any node in the tree, until a MO arrives that wants to register with the corresponding quadrant. In that case, the node is stored by that MO, according to the "insert" operation described below.

4.0.2. Insert

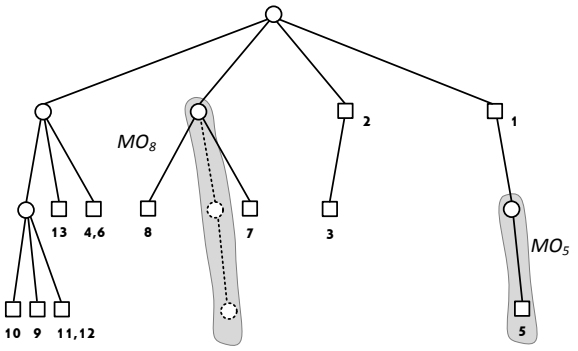
When a MO agent wants to join the network, it has to first identify the node that corresponds to its location and has the right level of granularity, according to its obfuscation level. So, starting from the the root it traverses the tree until it reaches the node representing the desired region and registers with it.

In case the MO agent reaches a point where there is no node representing the next level in the path, it takes over the responsibility to store and maintain that node. It continues doing so, expanding the tree until it reaches the desired level. Then, it registers itself with the last node and remains responsible for maintaining the new part of the tree.

As an example, let us assume that MO with $Id = 5$, namely MO_5, is the first one that registers with the tree. Let us assume that MO_5 has defined its obfuscation level



(a)



(b)

Figure 3: (a) Insertion of MO_5 in an empty tree. The agent of the MO creates and stores the branch shown in the gray area. (b) Example of an *Update*. MO_5 moves to the SE branch of the root. The grey area denotes the nodes stored by its agent.

to be $f = 3$. That corresponds to level 3 in the quadtree. Figure 3a shows the exact position where MO_5 needs to register. Since the intermediate nodes under the root do not exist, MO_5 creates them together with the leaf node, where it registers itself and has responsibility for this part (shaded path). That means, the root node points to the address of MO_5 for the NE part of its subtree. Later, MO_7 and MO_8 join the same part, as we saw in Figure 2b. They only need to create and maintain the corresponding leaf nodes, where they register themselves. Their parent node is stored by MO_5.

4.0.3. Delete

MOs are always registered with the quadtree node that corresponds to their quadrant. When a MO wants to go offline, it needs to contact the MO that is responsible for maintaining that node and unregister. Also in the case the MO moves to another quadrant, a *Delete* has to be performed before the object registers with the new node, as we will see in the *Update* operation.

If a MO is responsible for maintaining some tree nodes,

it keeps doing so, even if it goes offline (i.e. not available for responding to queries) and is not registered with any node. The MO agent keeps running on the cloud and participates in maintaining the structure of the tree.

4.0.4. Update

As MOs move in the city, the quadtree is changing dynamically and MO agents need to update its structure. As long as a MO stays inside the same quadrant no updates are necessary, but once it moves outside it and into the next one, the following actions need to be performed. First, it needs to unregister from the previous quadrant, executing the *Delete* operation above. Then, it need to execute an *Insert* operation, as if the MO just joined the network. So overall, the update operation consists of the sequence of the previous two operations, i.e. a *Delete* and then an *Insert*.

Let us assume for example that MO_5 moves to the SE part of the map, close to MO_1, maintaining the same obfuscation level $f = 3$. As shown in Figure 3b, MO_5 creates two new tree nodes and registers itself to the leaf node. It also keeps storing the three tree nodes from its previous position (see Figure 3a), so in total it maintains 5 nodes, shown in the grey area of Figure 3b.

The fact that an agent may be responsible for more than one part of the three has no impact on the MO itself. Since the agent is decoupled from the MO and acts as a proxy of it, it is not limited by the resource constraints of the MO.

4.0.5. Cleanup

As MOs move around and they change quadrants, it can be the case that some nodes in the quadtree are left with no registered MOs and with no children. In that case, the corresponding MO agent that stores this node can delete it from its memory. MO agents periodically invoke this check locally in their storage, as part of the quadtree maintenance. For example, in Figure 3b, MO_5 stores two tree nodes (shown with a dashed line), which will be deleted as a result of two successive *Cleanup* operations.

4.0.6. Query

In order for a data requester to submit a request, she first needs to look up the nodes that might be able to answer that request. The *Query* operation is responsible for this. The procedure is very similar to the case where a new MO wants to join the tree. The requester starts from the root and traverses the tree till she reaches the node that is representing the region of interest. The MO agent maintaining that node will provide the list of the registered MOs, possibly after verifying the corresponding credentials from the requester. Now she can communicate with the agents of MOs residing in the area of interest and see if they can help her with the data request.

However, we allow the query region to be an arbitrary region, as shown in Figure 2a by the dashed squares. In this case, the query region overlaps partially with several regions of the quadtree, which are at different levels. Since we don't know where exactly each MO is inside its region (as the user decides the level of obfuscation and the precision of its location), the answer to the Query will have to return all MOs in the quadrants that overlap with the query region. But this does not necessarily mean that they will *all* be inside the query region.

In the example shown in Figure 2a, the *Query* operation for Q_1 will return a list of all objects in the overlapping quadrants, including MO_4 and MO_{11}, even though these two are outside the query area. Similarly, in the example of Q_2, there is actually no MO inside the query area, but the operation will return MO_7.

In this way, we have prevented attacks where an attacker could reduce the location privacy of MOs by posting a series of Query requests. The location accuracy inferred for each MO will always be at the obfuscation level chosen by its agent. The output of multiple queries could change only when the MO moves from one quadrant to the next, allowing some inference for the movement of the MO, but this is still in accordance to the privacy protection level chosen by the agent (the accuracy of the location has not increased).

5. Token Verifier

In this section, we present a privacy-respecting access control protocol for requesting data from MOs. A high-level overview of the steps is shown in Figure 4.

A data requester $\mathcal{Q}$ wishes to query the sensing network for data. She first contacts the platform owner $\mathcal{S}$ and retrieves a token $\mathcal{T}$ which can spent with any data provider (mobile phone user) $\mathcal{P}$ (Section 5.1). Here we concentrate on the requester getting one token but nothing prevents $\mathcal{Q}$ from obtaining more than one (as in a batch request) from $\mathcal{S}$. Each token reveals no information about either $\mathcal{Q}$ or the desire of $\mathcal{Q}$ to spend it with any specific $\mathcal{P}$. Once $\mathcal{P}$ retrieves the token, it first has to verify its validity and then test whether the token is an attempt of doublespending (Section 5.2). For that reason it contacts an appropriate witness service $\mathcal{W}$. The evidence provided by $\mathcal{W}$ again reveals nothing about the identity of $\mathcal{Q}$, only the fact that the token has not been used before. Finally, the provider $\mathcal{P}$ can redeem the token for credit or additional services from $\mathcal{S}$.

5.1. Purchasing a token

To make a request for sensor data, $\mathcal{Q}$ must first obtain a valid token from the platform provider. In order to make the token untraceable and protect the privacy of $\mathcal{Q}$, the token will not be associated with a particular data requester, however it should contain such information as date, expiration date and amount of data to be retrieved as well as the signature of the platform provider $\mathcal{S}$ on it. This information is included in the *common part* of the

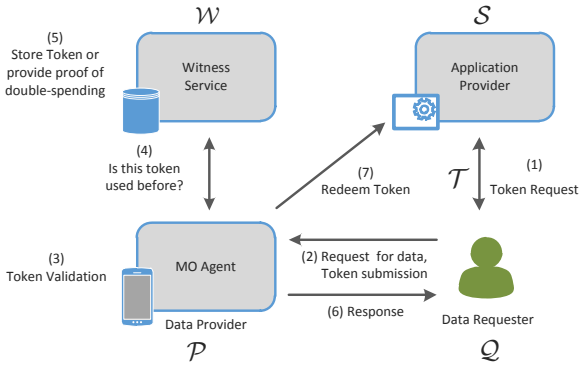


Figure 4: Token request and verification for accessing data.

token which is necessary in order for the mobile user to be able to provide a commensurate amount of information and perform an initial test on the validity of the token (i.e. check expiration date and signature of $\mathcal{S}$).

However, in order to provide a mechanism to detect double spending, the token must also contain some *identifying* piece of information, denoted by $\langle UniqueInfo \rangle$, which will be supplied by $\mathcal{Q}$. We stress here that this uniquely identifying piece of information is in no way related to the identity of $\mathcal{Q}$; its only purpose is to deter double spending. However, in order to prevent possible tracking by $\mathcal{S}$, it will be *blinded* before it is signed by $\mathcal{S}$. An instantiation of this process using RSA and generic values for $\langle CommonInfo \rangle$ and $\langle UniqueInfo \rangle$ is described below. Later on we will substitute these with concrete values.

Let k be the security parameter. Let p' and q' be two large primes of size $k/2$ such that $p = 2p'+1$ and $q = 2q'+1$ are also prime. Let $N = pq$ be an RSA modulus, (e, N) be the public key of the platform provider $\mathcal{S}$ and (d, N) its corresponding private key in the RSA key generation process. Following the general method outlined in [18], one way to include common information to any message m is to *embed* it in the signer's key and generate new, *per message* signing keys (in our case per token keys). So, let $h()$ be a secure cryptographic hash function. The new public (and private) exponent e_T (resp. d_T) for token $\mathcal{T}$, generated from e, is obtained as follows:

Algorithm 1
Creating partial signature keys from (e, N)

$$\begin{aligned} h_1 &\leftarrow h(e, \langle CommonInfo \rangle) \\ e_T &\leftarrow h_1 \parallel h(h_1) \parallel 00000001 \\ d_T &\leftarrow 1/e_T \mod \phi(N) \end{aligned}$$

The existence and the security of the inverse, signing keys d_T is proved in [19]. A data requester $\mathcal{Q}$ can ask the service provider for a token using the procedure shown in Protocol 1. Recall that $\langle UniqueInfo \rangle$ is the part that has to be blinded and contains identifying information to prevent double spending (the exact format of this part will become clear in the next section). Let $r \in Z_N$ be a random number chosen by $\mathcal{Q}$ and $m = h(\langle UniqueInfo \rangle)$.

Protocol 1
Obtaining a blind signature of $m = h(\langle UniqueInfo \rangle)$ from $\mathcal{S}$

1. $\mathcal{Q}$ sends $m^* = mr^{e_T} \mod N$ to $\mathcal{S}$ by evaluating the public key e_T from public information available (e.g. e and $\langle CommonInfo \rangle$).
2. The platform owner $\mathcal{S}$ returns the signature $\sigma^* = (m^*)^{d_T} \mod N$ to $\mathcal{Q}$.
3. $\mathcal{Q}$ computes $\sigma = r^{-1}\sigma^* \mod N$, which is the platform owner's signature on $h(\langle UniqueInfo \rangle)$.

Due to the blinding factor r, the network owner cannot derive m and σ from m^*. In other words, given $\langle m, \sigma \rangle$, the network owner cannot link it to $\mathcal{Q}$. Additionally, since the $\langle CommonInfo \rangle$ part is clear to $\mathcal{Q}$ and is negotiated at the beginning of (or before) the protocol, $\mathcal{S}$ cannot include any information in it and hence trace the requester[1]. Once the signature σ is retrieved, the final token becomes $\mathcal{T} = \langle CommonInfo, UniqueInfo, m, \sigma \rangle$.

5.2. Spending and Redeeming a Token

Let $\mathcal{P}$ be the producer of data (mobile phone user) that $\mathcal{Q}$ has decided to spend the token to. In order to provide the required amount of sensed data, $\mathcal{P}$ has to be able to tell if the token $\mathcal{T}$ has been used before. For that reason, $\mathcal{P}$ will contact a *witness* service $\mathcal{W}$ (which can also be run by the application provider or any other trusted party) in order to attest on the validity of $\mathcal{T}$ or provide proof that the token has been used before. Crucial to the above is the structure of the $\langle UniqueInfo \rangle$ existing in $\mathcal{T}$. This element will allow $\mathcal{P}$ (as well as $\mathcal{W}$) to provide the necessary evidence for the token's validity.

Let P and Q be primes such that $Q|P-1$ and g a generator of order Q in the group Z_P^*. Typically, P and Q will have length 1024 bits and 160 bits, respectively. The data requester $\mathcal{Q}$ will select two secret values $s, r \in Z_P$ and compute $v = g^{-s} \mod P$ and $x = g^r \mod P$. The $\langle UniqueInfo \rangle$ element will consist of the two values v and x, which will be signed by $\mathcal{S}$ as explained in Section 5.1.

When $\mathcal{Q}$ wishes to spend the token $\mathcal{T}$, it will first have to demonstrate the *validity* of the coin by proving knowledge of the two secret values s, r using appropriate zero-knowledge proofs. This protocol is based on the identification scheme of Schnorr [20] (see also Okamoto [21] for a

[1] An implicit assumption here is that the parameters in the common part do not contain any one-time elements that may help distinguish the transaction and narrow down the search (e.g. a strange combination of date and amount of data, etc.). If this is a concern, users may use a trusted third party to purchase token or use alternative schemes such as anonymous gift cards.

generalization and [22] for an instantiation on the e-cash setting) and can be made non-interactive by making the challenge of the verifier equal to the hash value of the token and committed protocol parameters. The interaction between $\mathcal{Q}$ and $\mathcal{P}$ is shown below.

Protocol 2
Checking the validity of a token $\mathcal{T}$ supplied by $\mathcal{Q}$

1. $\mathcal{Q}$ sends $\mathcal{P}$ $\langle \mathcal{T}, y, date/time \rangle$, where $y = r + es \mod Q$ and $e = h(\langle \mathcal{T}, date/time \rangle)$.
2. $\mathcal{P}$ verifies the signature of $\mathcal{S}$ on the token and checks that $x = g^y v^e \mod P$.

If the tests in Step 2 of Protocol 2 succeed, $\mathcal{P}$ considers the token *valid*. However, $\mathcal{P}$ still has to determine whether the token is *fresh* or an attempt of doublespending. For that reason it has to contact the witness service $\mathcal{W}$ that can attest on the freshness of the token. The interaction between $\mathcal{P}$ and a $\mathcal{W}$ is shown below:

Protocol 3
Checking for doublespending

1. $\mathcal{P}$ sends $\mathcal{W}$ the transcript of the interaction with $\mathcal{Q}$, i.e. $\langle \mathcal{T}, y, date/time \rangle$.
2. $\mathcal{W}$ verifies the signature of $\mathcal{S}$ on $\mathcal{T}$ and checks that $x = g^y v^e \mod P$. Then, based on the expiration date of the token, searches its records for a token containing the same $\langle UniqueInfo \rangle$ element. If no match is found, the token is considered fresh and $\mathcal{W}$ records the values $\langle \mathcal{T}, y, date/time \rangle$.
   If a match is found then $\mathcal{W}$ returns evidence that the token has been used before. This evidence has the form of the secret values s, r selected by $\mathcal{Q}$ in forming the values v and x contained in the $\langle UniqueInfo \rangle$ element.

To see why $\mathcal{W}$ can provide evidence[2] that a token has been used before, notice that in this case it will have two transcripts $\langle \mathcal{T}, y, date/time \rangle$ and $\langle \mathcal{T}, y', date'/time' \rangle$ such that $x = g^y v^e \mod P$ and $x = g^{y'} v^{e'} \mod P$. $\mathcal{W}$ can then compute $s = (y - y')/(e - e') \mod Q$ by solving

$$y = r + es \mod Q,$$
$$y' = r + e's \mod Q.$$

In a similar manner $\mathcal{W}$ can obtain r. *Notice that these values are not connected with the ID of $\mathcal{Q}$, hence they are not used in identifying $\mathcal{Q}$. They are only used to provide evidence that a token has been double-spent. Hence the privacy of $\mathcal{Q}$ is maintained even in this case.*

Once $\mathcal{P}$ is convinced that the token is both valid and fresh, it can provide the data requested by $\mathcal{Q}$. Then it can go on redeeming the token in exchange for other services or credit provided by the platform provider.

[2] Alternatively, $\mathcal{W}$ returns the previous transcript $\langle \mathcal{T}, y', date'/time' \rangle$ and $\mathcal{P}$ can check itself if the token is not used before.

6. Response Manager

The Response Manager is responsible for controlling how much information is revealed in response to a data request and enforcing the sharing policies of the user. Such policies could restrict sharing for example to a specific time of the day, or within specific areas, etc. [23].

How are data providers going to decide whether to share a particular sensory stream, and up to which point, or granularity? We see two broad factors that would influence people's disclosure preferences and decisions:

- The first factor is the degree of *trust* in the information receiver. This trust decision is influenced by past experience – or, in the absence of it – the degree of trust that other community members have placed in the information receiver.
- The second factor is the *risk/benefit trade-off* associated with information usage, as perceived by the individual. This assessment will depend on the perceived value of incentives, e.g., monetary incentives for higher disclosure and the social context (the reputation of the individual in the community).

From the viewpoint of requesters that collect sensing data, monetary incentives could increase user participation. It could help them attract a large number of participants and thereby increase not only the collected amount of data, but also its quality, in order to offer a higher quality of service back to the users [24]. Monetary incentives can be of the form of micro-payments, or even in the form of coupons [25].

One of the challenges here is to determine the price that users expect to receive for their effort of collecting and sharing sensing data. This price may depend on the individual preferences of the users and how they perceive the cost of participating in the process. Sensing takes time, interrupts other activities, consumes additional battery power and data traffic bandwidth. In general, each user has her own valuation and thereby a minimum price that he expects to receive, depending on how much effort and personal information he has to give out, in order to collect and provide sensing data for a requester. This valuation naturally differs among individuals and based on the context or situation they are currently in.

An attractive solution to the problem above is the use of reverse auctions, meaning that an auction takes place with the data providers being the sellers and the data requester being the buyer [26, 27, 28]. This is attractive because it allows the requester to avoid the challenge of determining the price a user expects to receive for his data.

Figure 5 shows how a reverse auction works. On the data provider's side the logic is implemented at the Response Manager. First, the data requestor announces an auction round for a given sensing task. Users bid for selling their sensing data and the requestor selects the user with the lowest bid. The winner gets notified and only

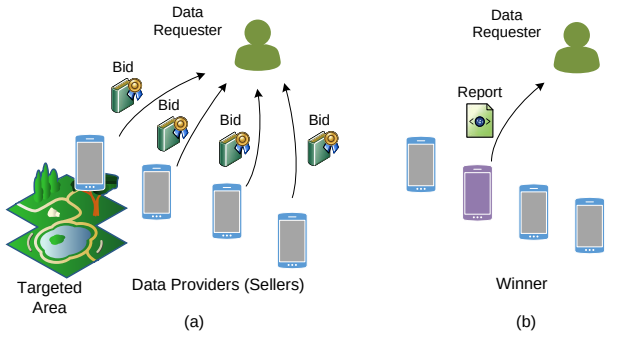


Figure 5: A reverse auction round in the context of mobile sensing.

then downloads the Response Manager the sensing task to the mobile device and the sensing takes place. The data is uploaded to the MO Agent (see Figure 1) and the Response Manager submits the data to the data requester. The rest of the users who lost the auction round do not perform any sensing, but their probability to win in the next round is increased, so that they keep participating. Finally, the winner receives the bid price for the data. Reverse auctions lead to the decrease of prices, since sellers (mobile users) compete with each other and continuously decrease their bids to increase their chances to win.

Let us emphasize here that the data requester should select the data provider not only based on the price, but also based on the data quality. There is a way to tune the reverse auction in such as way that the winner is not the one who offers the data at the cheapest price, but the one who has the best combination of data quality and price. This kind of auctions are called multi-attributive auctions because they integrate additional attributes of a good into the auction bid, besides the price. So, each bid is represented by an n-dimensional vector Q of both monetary and non-monetary relevant attributes.

Using an additive utility function $S(x_i)$, the bid of the data provider can be expressed as $x = (x_1, \ldots, x_n)$. The data requester evaluates each relevant attribute x_i through the utility function. As a result, the function $S : Q \to \mathbb{R}$ translates the value of each attribute into a *utility score*. At the end, the overall utility $S(x)$ for bid x constitutes the sum of all individual utility scores resulting from each attribute. If applicable, the individual utility scores can be weighted, with the weights $w_1, \ldots, w_n$ summing up to one. The overall utility of bid is given by Equation 1. For a detailed description of a multi-attributive auction protocol see [26].

$$S(x) = \sum_{i=1}^{n} w_i S(x_i) \text{ and } \sum_{i=1}^{n} w_i = 1 \, . \qquad (1)$$

So what kind of attributes could one add to account for data quality? One can include here technical aspects of sensing, like the location accuracy, the distance from the targeted area and the sampling frequency. But in addition to that, the overall data provider's credibility (i.e. trustwothiness) should also be included. In our scheme, this is a value that is calculated and published for each data provider by the Public Profile Manager (see Section 7). Finally, as mentioned before, the number of previously lost auction rounds should also be included as an attribute, so that the utility score improves the chances of those who have lost previous rounds.

The process is the same as described above for the reverse auctions, only that the Response Manager calculates the utility score and submits its value as the data provider's bid. The data requester collects all submitted bids and determines the winner to be the one with the best utility score.

7. Public Profile Manager

The Public Profile Manager is responsible for maintaining the public profile of data providers. After retrieving a list of MOs that fall inside the geographic region of interest, data requesters can use additional information published on their public profiles to identify and select well-suited participants under additional criteria. An important aspect to be considered is for example the trustworthiness of the providers, which is connected to the quality of their data. This can be also used as input to the auction mechanisms, as we saw in Section 6.

A way to evaluate the trustworthiness is to employ a reputation-based framework, in which a reputation score is calculated and published for each data provider based on historical information [29]. More specifically, the reputation points are collected when submitting data to a data requester. The requester evaluates the quality of the data and assigns reputation points of either positive or negative value, depending on their quality.

The process of acquiring reputation points for a given report and displaying them on a public profile would clearly compromise the anonymity of the contributor. So, a protocol must be in place that satisfies the following properties:

- The process of acquiring reputation points is independent from the process of updating the reputation value on someone's public profile.
- The process of acquiring reputation points for two successive reports should be unlinkable to each other.

An approach that satisfies this requirement is to award reputation points to anonymous data providers using cryptographic tools, such as blind signatures [30]. Androulaki et al. [31] has shown how these cryptographic tools can be combined to build a reputation system. One of the drawbacks of this protocol is that negative reputation is not supported. That is, users can only increase their reputation and eventually the system will reach a final state, where all users have the maximum reputation. To address this problem, Schiffner et al. [32] proposed a solution that supports non-monotonic reputation. To make sure that ratees are forced to deposit negative reputation coins, a

trusted server is used to guarantee that each interaction is actually rated, possibly also in a negative way.

Later such schemes were further adapted specifically for mobile sensing systems [33, 34]. Wang et al. [35] recently suggested one such scheme, which does not require a trusted third party and both positive and negative reputation updates can be enforced. Here we assume that one of these schemes is used to implement the Public Profile Manager, since they can all be used without any incompatibility with the rest of the modules.

Another issue to be considered here is how the requester can evaluate the contributor's data. In many cases the requester might not be in the best position to evaluate the quality due to not being aware of the situation in the targeted area. In such cases, methods such as majority consensus could be employed to come up with a reliable rating [29]. This should be complemented by additional parameters that can provide insights into the behavior of the devices over a long time period, in order to reinforce the requester's confidence. As argued by Amintoosi and Kanhere [36], one can consider the trust of participant (ToP), which is a combination of personal and social factors. Personal factors can be for example the experience of the contributor, how promptly they submit their data and whether they are locals or passing-by from the targeted area. Social factors could include the friendship duration between contributor and requester, or their interaction frequency. However, it is an open problem how to integrate social aspects in a system where contributors and requesters want to maintain their anonymity.

8. Experimental Analysis

In this section, we present an experimental analysis related to the Network Maintenance and Token Verifier modules, since these are the two modules where we introduced new protocols.

8.1. Network Maintenance

To evaluate our algorithm of building and maintaining the quadtree, we implemented a simulator that takes as input the movements of MOs in the city over a period of time (corresponding to 100 location updates for each MO) and builds the corresponding quadtree. To generate a set of moving objects and their tracks, we used the Network-based Generator of Moving Objects [37]. As input to the generator we used the road network of two different cities, namely Oldenburg in Germany and San Joaquin in CA, USA. Oldenburg's road network has a width of about 9 Km, while San Joaquin extends to 20 Km. So, choosing these two cities we cover the case of a small and a medium sized city for our simulations.

The output of the generator is a random set of moving objects that move on the road network of the given city. The distribution of the MOs in the spatial space is not uniform throughout the map, but it is correlated to the density of the network (i.e. the city centre is more busy than the outer region). In order to test the algorithms, we maintain the number of the MOs in the city constant. The obfuscation level f of all MOs is decided randomly following the uniform distribution between zero and f_{max}. For simplicity, we assume that MOs do not change their obfuscation level. Finally, for each experiment below, we run the generator of MOs 10,000 times and we averaged these results in order to produce statistically stable values.

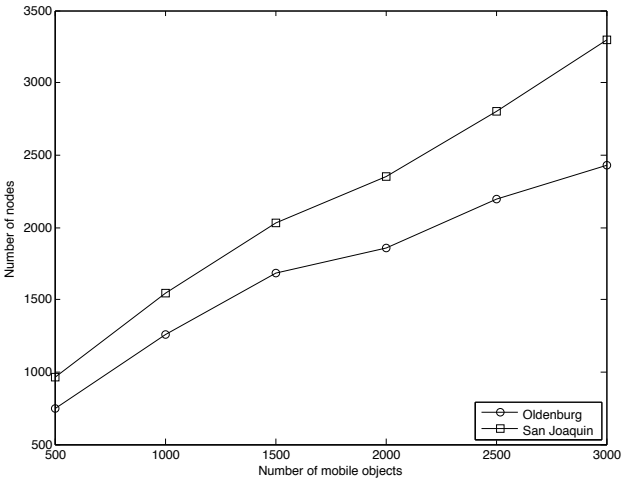


Figure 6: Size of the quadtree.

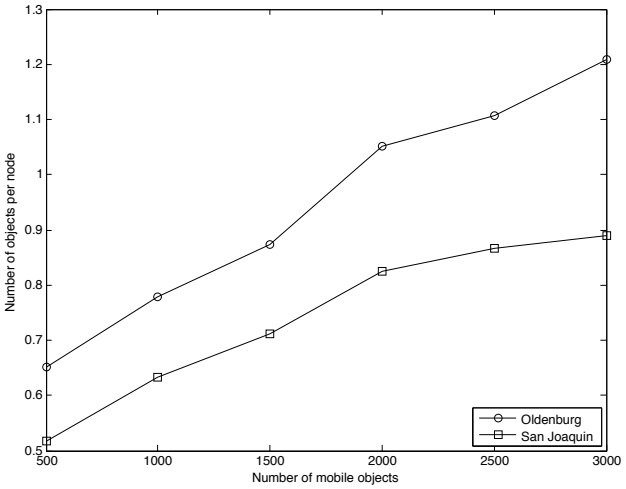


Figure 7: Number of MOs registered per tree node.

8.1.1. Load-balancing and scalability

Figure 6 shows how the quadtree grows in number of nodes, as the population of MOs grows in the city. For Oldenburg we have assumed $f_{max} = 9$ and for San Joaquin $f_{max} = 10$. In this way, for both cities the quadtree extends to the same minimum supported quadrant size, equal to about 15 m. So, as it is expected, for San Joaquin, a bigger quadtree is needed compared to Oldenburg, for the same number of participating MOs.

In Figure 7 the average number of MOs registered with

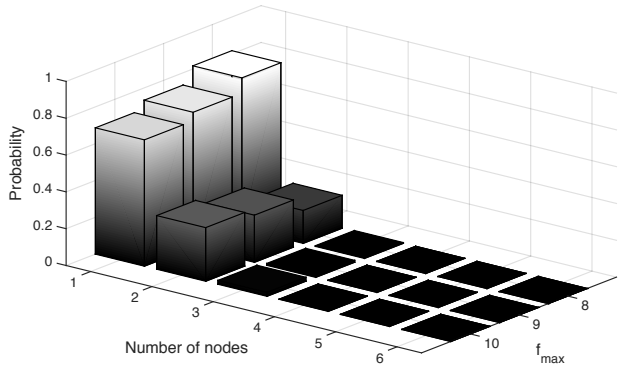


Figure 8: Probability mass function of the number of tree nodes maintained by each MO.

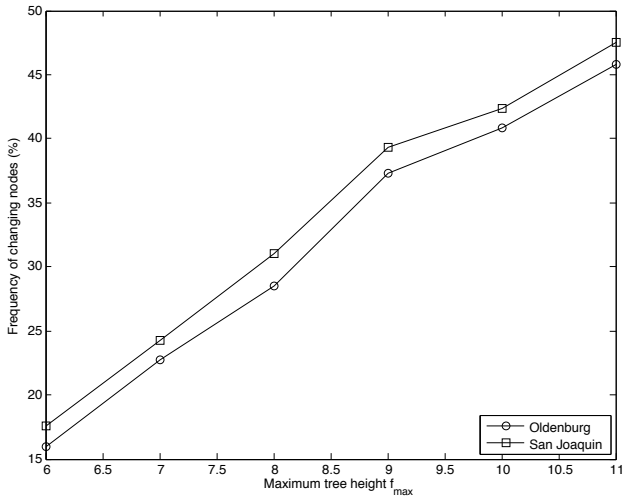


Figure 9: Frequency of MO update operations compared to the overall location updates.

each tree node is depicted. As we see, the number of nodes is almost the same with the number of MOs, following a dependency close to 1-to-1. What is of particular importance here in terms of performance is the load placed on each MO for storing parts of the quadtree. Our simulations showed that the average number of nodes that each MO has to store is very low and remains unaffected by the total number of MOs. That is to be expected, since as the MOs increase, the size of the tree also increases (see Figure 6). Of course, if the number of MOs approaches and exceeds the number of maximum tree nodes $4^{f_{max}}$, then there will be more MOs per tree node, but again the dependency will be linear (albeit not 1-to-1).

Figure 8 shows the probability mass function (pmf) of the number of quadtree nodes maintained by each MO agent for three different values of f_{max}. The experiment has assumed 500 MOs in Oldenburg. In all three cases one can see that the vast majority of agents store one or two nodes. For larger values of f_{max}, the tree is allowed to grow deeper and therefore it needs more nodes. So, the probability of two and three nodes per MOs is slightly increased.

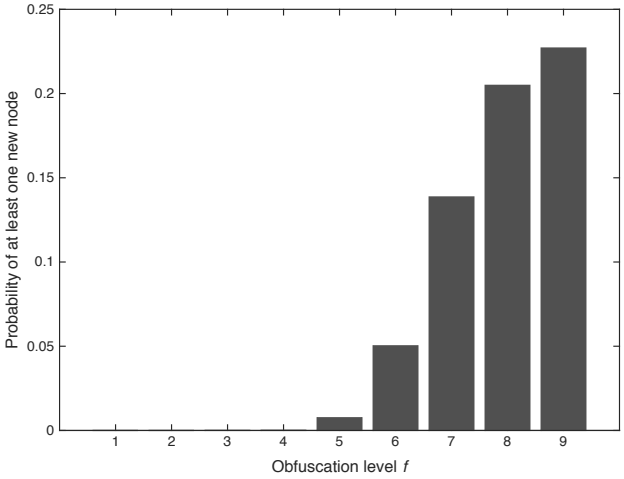


Figure 10: Probability of creating a new tree node as a result of a MO update operation.

8.1.2. Updates

Figure 9 shows the frequency that a MO unregisters from the current node and registers in a new one, as it moves in the city. For both cities we have assumed 2000 participating MOs. The values in the y-axis are calculated as a percentage of the overall location updates sent by the MOs to their agents. So, for Oldenburg the MOs move to a new quadrant 37.3% of the time on average and their agent needs to update its position in the quadtree. As shown in the figure, this frequency can vary a lot and depends on how small the quadrants are allowed to become (i.e. on the value of f_{max}). This, however, does not have an impact on the MOs as the agents are separate entities living in the cloud and can easily handle the update operations.

It is also interesting to see the probability that during an update operation the tree node that corresponds to the new quadrant does not exist and the MO needs to create a new one. Let's take for example Oldenburg and assume 2000 MOs. Let's also assume that $f_{max} = 9$. Figure 10 shows the probability that at least one new node needs to be created in the quadtree as a result of an update operation. This probability depends at which level of the tree the MO moves. For nodes with $f < 4$ the probability is practically zero and it reaches up to 22% for $f = 9$.

8.1.3. Queries

We are interested to see what is the delay of performing a query. Since this is implementation dependent, we decided to use the simulations to measure a corresponding parameter that would give the scale of how the delay would vary, depending on the size of the query (i.e. the size of the targeted region). Specifically we are interested to see what percentage of the whole tree a requester has to traverse in order to discover the MOs in the overlapping quadrants. For our experiment, we assume Oldenburg, with a population of 3000 MOs.

To perform the experiment, we post several queries,

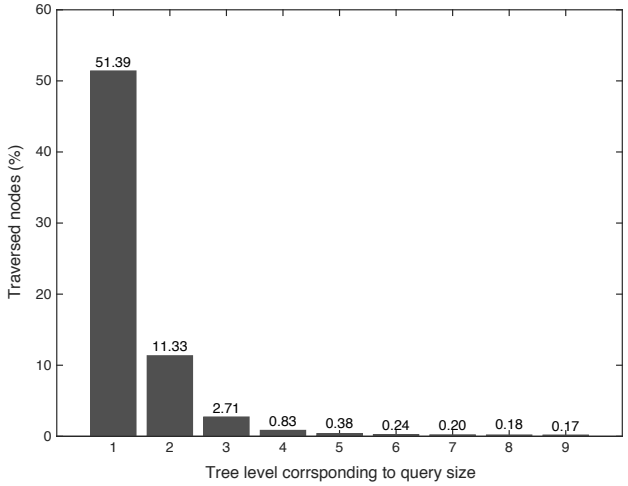


Figure 11: Percentage of traversed tree nodes as a function of query size.

choosing some points randomly on the map and considering them as the centre of the queries' square. The size of the squares varies from large ones to smaller ones. In particular, we take nine different sizes that correspond to the size of the quadrants on nine different levels of the quadtree. In the case of Oldenburg taken in this experiment, level 1 in the tree corresponds to quadrants with diameter of 4.5 Km, level 2 to 2.25 Km, and so on, until level 9 where quadrants reach a diameter of 17.57 m. So, we experimented with queries of sizes that correspond to this scale (i.e. the tree levels).

Figure 11 shows what is the percentage of the total quadtree nodes that were traversed by the requester, depending on the size of the queries that she posts. By traversed we mean nodes that the requester visits as she executes the *Query* command described in Section 4. As the figure depicts, for the biggest queries, the percentage reaches up to 51.39%, but it drops by almost a factor of four for every next step. This is to be expected, since each level divides the space in four equal quadrants. So, a conclusion one can draw from Figure 11 is that the presented solution is efficient in case of queries that are targeted and more granular, compared to queries that cover large areas of the map.

8.2. Token Verifier

To evaluate the efficiency of the token generation and verification processes we relied on the Java built-in libraries to implement and test our solution on an Intel Core2 Duo E7600 3.06GHz machine. We used a public key of size 1024 bits as was described in Section 5, while each experiment was repeated 20 times in order to obtain more accurate results.

These results are shown in Table 1. Clearly, the token generation process is more computationally expensive compared to token verification. This is due to the fact that at generation time, a data requester $\mathcal{Q}$ is required to perform a number of modular exponentiations in the computation of s, x and in the verification of the signature σ. This, however, is not a problem since the token generation process can take place any time, even before the actual request of data ($\mathcal{Q}$ just needs to have purchased a valid token in order to use it later on).

Table 1: Efficiency of token generation and verification

Token generation time taken by $\mathcal{Q}$	1.14 seconds
Token verification taken by $\mathcal{Q}$	0.001 seconds

The time to spend a token is considerably less than the time to generate it. According to Protocol 2, $\mathcal{Q}$ must only evaluate $y = r + es$ and send it to $\mathcal{P}$. Similarly $\mathcal{P}$ must verify that $x = g^y v^e$, which takes time proportional to token generation. In summary, these results show that the time to generate and spend tokens can be tolerated, since these tasks might as well be offloaded to the corresponding agents which are not restricted by the computational capabilities of mobile phones.

9. Conclusions

In this paper we presented a platform that is designed to protect the privacy of both data requesters and data providers in mobile sensing systems. Our platform uses mobile agents on the cloud that represent the mobile users to the outer world according to the user's location privacy preferences. This structure makes users discoverable by data requesters who are seeking data from a geographic region of interest. Data requesters remain anonymous throughout the process, but they prove to the data providers that they are legitimate users of the platform.

After data requesters come in contact with the mobile nodes inside the geographic area of interest, the former can apply further selection criteria, based e.g. on reputation scores, for guaranteeing a quality in the responses. At the same time data providers can decide with which requesters to share data and how much, depending on their sharing preferences and on the incentives given to them. Overall our platform proves that protecting privacy of both sides is possible without the need to trust third parties and without hindering mechanisms that facilitate trustworthiness.

10. Acknowledgements

The authors would like to thank the reviewers for their many useful comments that helped improved the readability of the paper. The second author would like to acknowledge support of this work by Kuwait University, Research Grant No. QE 01/13.

References

[1] R. K. Ganti, F. Ye, H. Lei, Mobile Crowdsensing: Current State and Future Challenges, IEEE Communications Magazine 49 (11) (2011) 32–39.

[2] N. D. Lane, S. B. Eisenman, M. Musolesi, E. Miluzzo, A. T. Campbell, Urban sensing systems: Opportunistic or participatory?, in: Proceedings of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile '08), 2008, pp. 11–16.

[3] I. Krontiris, M. Langheinrich, K. Shilton, Trust and privacy in mobile experience sharing: future challenges and avenues for research, IEEE Communications Magazine 52 (8) (2014) 50–55.

[4] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, A. Sharma, PRISM: platform for remote sensing using smartphones, in: Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10), San Francisco, California, USA, 2010, pp. 63–76.

[5] M. Shin, C. Cornelius, D. Peebles, A. Kapadia, D. Kotz, N. Triandopoulos, AnonySense: A system for anonymous opportunistic sensing, Journal of Pervasive and Mobile Computing.

[6] E. De Cristofaro, C. Soriente, Short paper: PEPSI – privacy-enhanced participatory sensing infrastructure, in: Proceedings of the fourth ACM conference on Wireless network security (WiSec '11), 2011, pp. 23–28.

[7] R. Cáceres, L. Cox, H. Lim, A. Shakimov, A. Varshavsky, Virtual individual servers as privacy-preserving proxies for mobile devices, in: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds (MobiHeld '09), Barcelona, Spain, 2009, pp. 37–42.

[8] M. Y. Mun, D. H. Kim, K. Shilton, D. Estrin, M. Hansen, R. Govindan, PDVLoc: A personal data vault for controlled location data sharing, ACM Transactions on Sensor Networks 10 (4) (2014) 58:1–58:29.

[9] H. Choi, S. Chakraborty, Z. M. Charbiwala, M. B. Srivastava, Sensorsafe: a framework for privacy-preserving management of personal sensory information, in: Proceedings of the 8th VLDB International Conference on Secure Data Management (SDM '11), Seattle, WA, 2011, pp. 85–100.

[10] G. Drosatos, P. Efraimidis, I. Athanasiadis, E. D'Hondt, M. Stevens, A privacy-preserving cloud computing system for creating participatory noise maps, in: 36th Annual IEEE Computer Software and Applications Conference (COMPSAC 2012), Izmir, Turkey, 2012, pp. 581–586.

[11] N. Fernando, S. W. Loke, W. Rahayu, Mobile cloud computing: A survey, Future Generation Computer Systems 29 (1) (2013) 84–106.

[12] R. Dingledine, N. Mathewson, P. Syverson, Tor: The second-generation onion router, in: Proceedings of the 13th Conference on USENIX Security Symposium (SSYM'04), 2004, pp. 21–21.

[13] M. Gruteser, D. Grunwald, Anonymous usage of location-based services through spatial and temporal cloaking, in: Proceedings of the 1st international conference on Mobile systems, applications and services (MobiSys '03), San Francisco, California, 2003, pp. 31–42.

[14] M. Duckham, L. Kulik, A formal model of obfuscation and negotiation for location privacy, in: Proceedings of the Third international conference on Pervasive Computing (PERVASIVE '05), Munich, Germany, 2005, pp. 152–170.

[15] M. Tschersich, C. Kahl, S. Heim, S. Crane, K. Böttcher, I. Krontiris, K. Rannenberg, Towards privacy-enhanced mobile communities - architecture, concepts and user trials, Journal of Systems and Software 84 (11) (2011) 1947–1960.

[16] I. Krontiris, T. Dimitriou, Privacy-respecting discovery of data providers in crowd-sensing applications, in: IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2013), 2013, pp. 249–257.

[17] A. Klinger, M. L. Rhodes, Organization and access of image data by areas, IEEE Transactions on Pattern Analysis and Machine Intelligence 1 (1) (1979) 50–60.

[18] M. Abe, E. Fujisaki, How to date blind signatures, in: ASIACRYPT'96, 1996, pp. 244–251.

[19] T. Dimitriou, I. Krontiris, A. Sabouri, PEPPeR: A Queriers Privacy Enhancing Protocol for PaRticipatory Sensing, in: Security and Privacy in Mobile Information and Communication Systems (MobiSec 2012), Vol. 107 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2012, pp. 93–106.

[20] C. P. Schnorr, Efficient signature generation by smart cards, Journal of Cryptology 4 (3) (1991) 161–174.

[21] T. Okamoto, Provably secure and practical identification schemes and corresponding signature schemes, in: CRYPTO, 1993, pp. 31–53.

[22] S. Brands, Untraceable off-line cash in wallets with observers (extended abstract), in: CRYPTO, 1993, pp. 302–318.

[23] R. Schlegel, A. Kapadia, A. J. Lee, Eyeing your exposure: quantifying and controlling information sharing for improved privacy, in: Proceedings of the Seventh Symposium on Usable Privacy and Security (SOUPS '11), Pittsburgh, Pennsylvania, 2011.

[24] N. Thepvilojanapong, K. Zhang, T. Tsujimori, Y. Ohta, Y. Zhao, Y. Tobe, Participation-aware incentive for active crowd sensing, in: Proceedings of the 11th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2013), Zhangjiajie, China, 2013, pp. 2127 – 2134.

[25] A. Albers, I. Krontiris, N. Sonehara, I. Echizen, Coupons as monetary incentives in participatory sensing, in: Proceedings of the 12th IFIP Conference on e-Business, e-Services, e-Society (I3E 2013), Athens, Greece, 2013.

[26] I. Krontiris, A. Albers, Monetary incentives in participatory sensing using multi-attributive auctions, International Journal of Parallel, Emergent and Distributed Systems.

[27] D. Yang, G. Xue, X. Fang, J. Tang, Crowdsourcing to smartphones: Incentive mechanism design for mobile phone sensing, in: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (Mobicom '12), Istanbul, Turkey, 2012, pp. 173–184.

[28] J.-S. Lee, B. Hoh, Sell your experiences: a market mechanism based incentive for participatory sensing, in: Proceeding of the IEEE International Conference on Pervasive Computing and Communications (PerCom '10), 2010, pp. 60–68.

[29] K. L. Huang, S. S. Kanhere, W. Hu, On the need for a reputation system in mobile phone based sensing, Ad Hoc Networks 12 (2014) 130–149.

[30] D. Chaum, Blind signature system, in: Proceedings of CRYPTO'83, 1983, p. 153.

[31] E. Androulaki, S. G. Choi, S. M. Bellovin, T. Malkin, Reputation systems for anonymous networks, in: Proceedings of the 8th international symposium on Privacy Enhancing Technologies (PETS '08), Leuven, Belgium, 2008, pp. 202–218.

[32] S. Schiffner, S. Clauß, S. Steinbrecher, Privacy and liveliness for reputation systems, in: Proceedings of the 6th European PKI Workshop: Research and Applications (EuroPKI '09), Pisa, Italy, 2009.

[33] D. Christin, C. Rosskopf, M. Hollick, L. Martucci, S. Kanhere, Incognisense: An anonymity-preserving reputation framework for participatory sensing applications, in: Proceeding of the IEEE International Conference on Pervasive Computing and Communications (PerCom '12), 2012, pp. 135–143.

[34] Q. Li, G. Cao, Providing privacy-aware incentives for mobile sensing, in: Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom '13), 2013, pp. 76–84.

[35] X. Wang, W. Cheng, P. Mohapatra, T. Abdelzaher, Enabling reputation and trust in privacy-preserving mobile sensing, IEEE Transactions on Mobile Computing 13 (12) (2014) 2777–2790.

[36] H. Amintoosi, S. S. Kanhere, A reputation framework for social participatory sensing systems, Mobile Networks and Applications 19 (1) (2014) 88–100.

[37] T. Brinkhoff, A framework for generating network-based moving objects, Geoinformatica 6 (2) (2002) 153–180.