Privacy Preservation for Participatory Sensing Data

Ioannis Boutsis, Vana Kalogeraki
Department of Informatics
Athens University of Economics and Business, Athens, Greece
{*mpoutsis, vana*}*@aueb.gr*

***Abstract*—Over the recent years, the proliferation of mobile networking and the increasing capabilities of smartphone devices have led to the development of "Participatory Sensing" applications, where users actively participate in data collection and sharing in a wide range of application domains from entertainment, to transportation, to environmental monitoring. One important challenge in these settings is privacy preservation for participatory sensing data, such as user trajectories. This paper develops a participatory sensing system for Android smartphones and proposes an efficient approach for privacy preservation which enables users to disclose their trajectory information without compromising their privacy. Existing approaches for privacy-aware data sharing operate under the assumption that user data is maintained in a centralized database. In our work we assume that user data is generated and stored locally on the individual smartphone devices. Our technique is distributed and low cost. We present detailed experimental results to illustrate that our approach is practical, efficient and with low overhead.**

I. Introduction

The proliferation of mobile networking and the increasing capabilities of smartphone devices in the recent years have led to the development of a new class of "community-based participatory sensing" systems, where all members of the community sense and contribute data to the system with the purpose of identifying events of interest for the community. We already see this trend in a variety of application domains including *transportation systems* such as MetroSense[1] and VTrack [2], where multiple users share their local traffic observations in order to identify traffic events such as congestion detection or for real-time delay estimation. Similar examples can be found in a number of location-based systems such as *WiFi Hotspot Discovery* using WiFi data collected by several cooperating users [3] and *earthquake warning detection systems* [4].

In a typical participatory sensing system mobile users sense and gather data through sensing devices embedded on the phones; these can be either stored on the device in a local data storage, or uploaded to a centralized server. The data can then be shared with other members of the community by issuing queries and processing the results; *i.e.,* in a traffic monitoring applications users may submit queries of the form *"Give me the trajectories from location A and B"* in order to estimate the average travel time needed between these two locations. Hence, encouraging individuals to participate in community-based data sensing and collection has important advantages in terms of improving the quality of the systems. However, one important challenge in these settings is how to preserve the user privacy when sharing participatory data, and we focus especially on the participatory data that contain locations that a user has visited and his trajectories.

The problem of privacy preservation is not new and several well-known attacks [5], [6] have been identified in the literature. These attacks can be categorized as follows:

- User Identification attack, where the goal is to expose the user's identity issuing queries about a spatial region and then with subsequent, more specific, queries involving spatial subregions of the original query.
- Sensitive location tracking, where the attacker tries to identify one or more locations that the user visits frequently. This attack can reveal information about a person's health, lifestyle, habits etc.
- Sequential tracking attack, that aims to track down the user by carrying out a set of queries involving spatiotemporal regions which are adjacent to each other and then analyzing the user trajectories to identify the places that he/she has visited.

Current solutions to preserve privacy in location-based databases focus on *centralized databases* where all data is stored, as opposed to the participatory systems settings where each user maintains an individual local database. One common technique for preserving the privacy of the user is *additive random perturbation* [7] through the addition of some noise to the geographical location of the data tuples, based on a selected distribution. However, it has been shown that once the attacker discovers the selected distribution through data analysis, he can extract the original data from the noise with high precision [8]. On the other hand, techniques that propose the addition of *fake trajectories* along with real ones [5] reduce the attacker's confidence, but at the expense of requiring a large number of fake trajectories which often produces unreliable results. *Edges distortion* techniques attempt to hide the start/end points of the trajectories by adding fake points so that the places visited by users become imprecise [6]. However, sophisticated queries would still be able to define the geographical location of the user's home within a radius and thus a simple analysis of the

points may identify the actual location. Techniques such as clustering or filtering that aim to capture similar trajectories are too costly to be executed online on mobile devices and also suffer from information loss since the actual trajectories are not provided [9]. Furthermore, we would like to stress that none of these techniques consider the unique challenges and characteristics of the Android OS, which we discuss in the next section.

In this paper, we present **LOCATE** (LOCation-based middlewAre for TrajEctory databases), the middleware that we have developed, that aims to provide privacy preservation for participatory sensing systems on Android-based devices. LOCATE allows users to locally sense and store data, as well as issue queries on data stored across the system. We propose a data exchange approach that aims to protect the user sensitive data by making the attacker consider all trajectories as equiprobable to contain sensitive data, so that leak of sensitive data is prevented. The data exchange technique we have developed aims to distribute the user data trajectories among multiple user databases, based on the local entropy, so that all local trajectories are equiprobable to be sensitive data. We examine the working of our approach under different types of attacks that can occur in the participatory sensing system, including attacks that arise from the use of the Android OS itself. Our extensive experimental results illustrate that our approach is practical, has low overhead and effectively addresses the privacy preservation problem.

II. Motivation, Background and Challenges

In this section we first present the motivation our approach, we then provide a brief introduction to the Android OS and lastly we describe the unique challenges of the Android OS for developing participatory sensing systems.

Why a distributed approach? Today's mobile devices are equipped with significant processing, storage and sensing capabilities (*i.e.,* LG Nexus 4 operates on a Quad-core 1.5 GHz processor with 2 GB RAM and utilizes sensors like GPS, WiFi, microphone, camera, accelerometer, gyroscope, barometer and compass). The ubiquitous sensing capabilities and the widespread adoption of smartphones are driving the development and adoption of applications where humans become the focal point of interest and mobile devices are used as tools for continuous data sensing, collection and analysis, turning people into producers of "personal data". In this work our aim is to take advantage of such devices, not only for producing data, but for processing data as well, since processing local data can be executed efficiently on such devices, and explore the design of a system for enabling applications with little or no infrastructure compared to client-server architectures.

Note, that, in a participatory sensing system we are interested on the data in the hands of the participants, and not in the producer of the data, in order to identify events of interest for the community. In these systems it is expected that a user cannot always retrieve all possible results for a query, as opposed to centralized approaches, rather the participation of multiple users enable these systems to operate properly. For instance, in a traffic monitoring application, we would not be able to trace all the mobile devices that produced data in a specific location and time, but we would be interested in finding a representative subset that allows us to estimate traffic events (i.e., average traffic, congestion, etc.).

Another characteristic of the participatory sensing systems is that users typically query for data which are dependent on the location of the users. Thus, bringing all data in a central location may involve high cost and storing them in the same location may not necessarily be beneficial. Considering the example of streaming systems (such as the traffic monitoring system), it is unlikely that a user will be interested in the same location multiple times, and in the case that people issue ad hoc queries, it is not clear whether collecting all data in a particular location would be more beneficial. Although our technique requires some data to be exchanged among user devices, the amount of data is considerably smaller than the totally produced ones in the system.

The Android OS. Android [1] is an open source platform, that has gained the leading market share in smartphones for mobile system development. Android relies on a Linux kernel for core system services such as security, memory management, process management, network stack, and driver model. The kernel acts as an abstraction layer between the hardware and the rest of the software stack. The runtime comprises core libraries and Dalvik, a virtual machine optimized to run under constrained memory and CPU requirements. The application framework provides APIs for developers to build their own applications. Each application in Android is typically a separate process and can be composed of the following components: **Activities** for graphical display, **Services** for background tasks, **Content Providers** to access persistent data, and **Broadcast Receivers** to receive notifications and essentially act as callbacks, when the appropriate Activity is invoked.

Component Communication in Android. Components in Android communicate through messages called **Intents**, which are routed through Android runtime and the Kernel. The Android runtime environment manages all inter-component communication. More specifically, it is responsible for the launching of the Activities and the Services, where Intents are used to initiate these components.

Intents are an abstraction for an action to be performed. They encapsulate the following fields: `Action`, `Data`, `Type`, `Category`, `Component` and `Extra`.

The fields defined in the Intents reflect the component to be invoked. Note, though, this is not always the case since a specific selection of the fields may not initiate the desired component. For example we may have set

[1] Android platform: http://www.android.com/

the `Action field` as `ACTION_DIAL` and the `Data field` as `tel:123456`. Normally this Intent would display the phone dialer with the given number filled in. However, if the identifier of another component has been specified on the `Component field`, the specific component would be invoked (Explicit Intent). By specifying the `Component field`, all of the other attributes become optional, since they are no longer used to match the Intent with the components which are able to handle the Intent (Implicit Intent).

Intent messages can be either explicit, when the naming of the target component is provided, or implicit when the Android runtime resolves the target component by matching the message against the components that can handle it. Note that the scope of the Intents is not limited to the application's components as they can trigger other components as well. A given target component can handle an Intent if it has been stated in the applications Android-Manifest.xml at the intent-filter tag. Intents are sent by the application components with the following ways: (i) initiating an activity using *startActivity(Intent)*, (ii) sending it to all interested BroadcastReceivers using *sendBroadcast(Intent)*, (iii) creating a dependency with an application service using *bindService (Intent, ServiceConnection, int)*, (iv) accessing data through ContentProviders.

Challenges for Privacy Preservation on Android Devices. In participatory sensing applications, the use of Intents play a crucial role as they can become simple tools for an attacker to compromise the Android system. Intents are used to invoke services at run-time (i.e., whenever a demand to initiate a service arrives) or to initiate periodic events (such as local sensing). Consider for example an application that receives a query along with the respective desired process (Service) requested by a remote user. The application initiates the appropriate Service to process the incoming request. Although their flexible structure makes it easy to generate them, it can also make them a tool for an adversary to exploit them [10]. For example, the attacker can initiate other Services/Activities (i.e. opening the browser with a specific URL) or even freeze the device or the application with multiple Intents (from multiple queries) or specific parameters on the Intents if these messages are not filtered. Thus, we should exclude the possibility of an attacker to gain access to the components of the user's device.

When processing queries on the local database we identify two types of attacks: (i) query selection and (ii) service selection. Queries to be executed in the user local database are received through WiFi and 3G and are then passed as a parameter in the Intent's `Extra` field so that the appropriate Service is invoked. A malformed query can cause an exception such as `SQLException`, `NullPointerException`, etc. that we need to handle. Moreover some of these exceptions may also lead to the application or device misbehavior such as forcing it to close. A worst case scenario might be that the attacker intentionally executes well-formed queries in order to corrupt the database such as "`DROP TABLE x`" or "`DELETE FROM x`". Thus, we need to investigate each query before instantiating the corresponding Intent. We resolve this by developing filters, based on regular expressions. These regular expression filters exclude all the queries that modify the database such as `DROP`, `CREATE`, `ALTER`, `DELETE`, `INSERT`, etc. when they are generated from external users. Moreover, we have developed regular expressions for the legimate commands like `SELECT` as well, to ensure that they have a valid syntax. Thus, only well formed queries and specific SQL commands can be performed on the user's database.

The second attack that we need to consider is the selection of the appropriate service. If we have multiple services that external users are able to use, we need to be very cautious with the way Intents are called. Clearly, letting an external user choose his/her desired service, enables him/her to choose from every component in the mobile device, which might cause an adverse behavior. However using implicit Intents and specific parameters on the Intents might cause similar problems as well, as he could override the selection of the Services through the parameters.

To overcome these problems we choose to use explicit Intents. Furthermore, we intercept the Intents [11] in order to investigate their target component before the Android runtime environment instantiates the components. By using interception we are able to examine the Fields of the Intent to define the component that will be invoked and thus to decide whether the specific component belongs to the application scope before invoking it. In order to reduce the overhead of the interception we examine the Intents at the application level, since the interception of the system calls has significant overhead.

III. System Architecture and Model

Assume a system with N users $user_i \in U$ where $1 < i < N$. Each $user_i$ produces data using the sensors embedded in the mobile phone (*i.e.,* GPS, accelerometer, motion sensors). The produced data is represented as sets of tuples and stored in the user's local database. Let $tuple_{ij}$ be the j tuple maintained by $user_i$. Each tuple $tuple_{ij}$ is associated with the following information: $< latitude_{ij}, longitude_{ij}, altitude_{ij}, timestamp_{ij}, point\_type_{ij}, traj\_id_{ij} >$, where $latitude_{ij}$ and $longitude_{ij}$ define the geographical location of the $tuple_{ij}$ with $altitude_{ij}$. $Timestamp_{ij}$ defines the time instance that the tuple was produced and the $point\_type_{ij}$ characterizes the point as Starting point, Destination point, etc. Finally the $traj\_id_{ij}$ is the id of the trajectory where the tuple belongs; this is a unique id among all the tuples from different trajectories (not only the local

ones) produced through a cryptographic hash function that utilizes the timestamp and the id of the $user_i$.

We classify each of these $tuple_{ij}$ as follows based on the corresponding $point\_type_{ij}$: *Starting Point* is the starting point of the trajectory, *Destination Point* is the final point of the trajectory, *Moving Points* are the tuples that correspond to the intermediate points of the trajectory and *Static Points* are the tuples produced when the user stays within a range of the Destination point (e.g. when the user makes a stop at the supermarket). Thus, the starting and destination points of the trajectories can be considered as *Stopping Points* or *Stops* and each of their locations $(latitude_{ij}, longitude_{ij})$ is denoted as $loc_{ij} \in L$.

To avoid the leak of user private information we follow an approach similar to [5], [9] where each tuple is separated from its producer ($user_i$), which is a common technique in privacy preserving approaches. Although the identity of the user that produced the data may be of some benefit, such as building user profiles, it could lead to a serious privacy breach, since it would be relatively easy for an attacker to assemble the trajectories provided for the same id and identify the user, his/her sensitive locations, etc.

Each $tuple_{ij}$ can be stored in the local database with the SQL's `INSERT` command, either when sensed by the user's device or through the Exchange Phase where another user sends sensed tuples, as we will discuss later. The tuples can subsequently be retrieved from the system upon receiving queries from other users. Different types of user queries can be supported including: (i) range queries, (ii) distance queries and (iii) spatiotemporal queries.

The trajectories are constructed based on the tuples locally maintained by the users. Consider $user_i$ with t trajectories in the local database; these have been either produced locally or have been received from another user. We denote as $Trajectory_{it}$ the set of tuples $(tuple_{ik}, tuple_{ik+1}, .., tuple_{il})$, that is, all the tuples $\{tuple_{im} \mid timestamp_{ik} \leq timestamp_{im} \ \wedge \ traj\_id_{ik} = traj\_id_{im}\}$, namely all the tuples that share the same trajectory id with $tuple_{ik}$ and produced later than $tuple_{ik}$, along with $tuple_{ik}$. Moreover, $tuple_{ik}$ should have been identified as the *Starting Point* of the $Trajectory_{it}$.

In order to detect the destination points of a $Trajectory_{it}$, and consequently the starting point of $Trajectory_{it+1}$, we assume that a destination point is identified whenever the user produces data within a radius of x meters for more than y seconds. In our experiments we have chosen x to be 100 meters and y to be 180 seconds. We made this choice because we wanted to discover locations loc_{ij} that involve a wide geographical range (e.g., visit to the park) as well as small Stops (e.g., a quick stop to the mini market to buy an orange juice). Our experiments have shown that these values can accurately define a stop. Even in the case that the trajectory represents walking, there is enough time to cover that distance without being considered as an intermediate

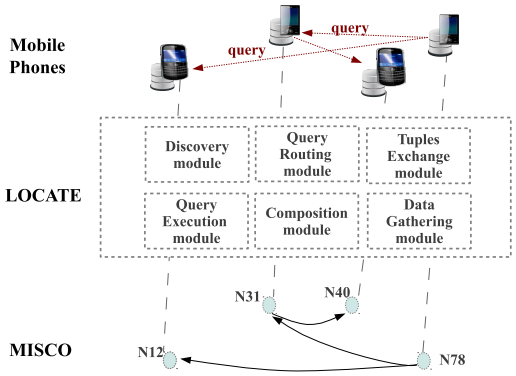

Figure 1. Our system architecture.

stop. Moreover, a new trajectory is also identified whenever the $timestamp_{ij}$ and $timestamp_{ij+1}$ are more than 3 minutes apart, since we infer that the user either closed the application or entered a building where the device's sensors (e.g., GPS) could not produce a new tuple. Moreover, we would like to stress that the values of these parameters can be tuned to reflect the types of trajectory data studied.

Each $user_i$ in the participatory sensing system can generate requests for mobility data, denoted as $query_q$. Each request is submitted to the LOCATE system along with an $amount_q$ for each $query_q$, that denotes the number of users to whom the request should be forwarded to and the $service_q$ to be executed.

The attacks can be injected from users of the system through the issuing of queries. The attackers might as well be users that receive tuples passively, through the exchanges.

System Architecture LOCATE (Figure 1) is built on top of our Misco system [12] [13], a MapReduce system tailored for cell phones and mobile devices, and thus, takes advantage of the routing and profiling functions it provides. Misco adopts a distributed task programming model based on the MapReduce model of computations of map and reduce tasks, which is a flexible and efficient way to program applications on the phones that is portable across multiple devices. To deal with a dynamic topology, device unavailability and fluctuations in network quality of a mobile setting, Misco builds distributed scheduling and reliability mechanisms that enable real-time and reliable delivery of streams of data to all appropriate stakeholders, schedules data transmissions based on their urgencies and responds to changing conditions due to failures or increasing requests.

LOCATE benefits from Misco's flexible, user-friendly way to program, develop and deploy software components on the mobile phones. In LOCATE, the task of data sensing is "crowdsourced" into a number of users; users sense and collect data independently of each other and in parallel, these data items are stored in their private local databases, but can also be shared with other members of the community and utilized for analysis or other types of data processing, to identify events of interest for the community. Such local

databases are easier to maintain, are more resource efficient as there is no need to forward all the sensed data to a centralized server thus incurring the corresponding resource costs, and are also more suitable for streaming applications where the queries typically involve the most recent data rather than historical data. All communication and data exchange between mobile nodes is achieved through TCP sockets. For mobile devices in the same WiFi network or connected through 3G, a direct TCP connection is established between the devices. In the case where the devices are connected in different WiFi networks, we make use of intermediate nodes, to enable devices to communicate with each other, due to the limitations of the private IP addresses provided by NAT.

Overview. We provide an overview of how our approach handles different types of attacks. LOCATE handles queries for participatory sensing data as follows: (1) A user executes a $query_q$ (that can be range query, distance query, etc.) by defining an $amount_q$ of workers, that will be selected by the LOCATE system, to whom the query will be forwarded and the $service_q$ to be executed (e.g. $query_q$ = "`SELECT * FROM table WHERE ( point_type = 'Starting' OR point_type = 'Destination' ) AND timestamp > 1199145600`", $amount_q$ = 10, $service_q$ = "Range Query"). (2) Every worker that receives the ($query_q$, $service_q$) information will create an Intent to instantiate the proper Service (e.g. Intent with fields {`Action = null`, `Data = null`, `Type = null`, `Category = null`, `Component = RangeQueryService`, `Extras =` $query_q$}), to execute the specific query on the Android device. In this step the robustness of the device and its database needs to be secured, as one of the threats, query selection or service selection, may occur. (3) If the received message is characterized as legimate by our system, then the Intent and the respective Service is instantiated to execute the query. (4) When executing the query we need to consider the privacy attacks (User Identification attack, Sensitive location tracking and Sequential tracking attack). Since the data is locally maintained, sensitive locations can be obtained easily, even with the use of well-known techniques for centralized databases, such as the injection of fake trajectories as we discuss on the experimental evaluation. If an attacker retrieves all the Stopping Points from a user with the specified $query_q$, he can define the frequency for every Stopping Point within a specific geographical radius and extract sensitive locations. These types of attacks are difficult to be recognized at run-time, so our technique aims to deal with them before they occur. (5) Finally, the user returns the output of the query to the requester.

IV. Trajectory Exchanges

In this section we present our approach for privacy preservation based on implementing exchanges of trajectories among users. We provide an online technique so that trajectories with sensitive locations are distributed among multiple users within the system. The selection of the trajectories to be exchanged is based on the Shannon's Entropy. The advantage is the confusion of the attackers as they will not be able to spot sensitive locations or identify the users.

A. Entropy

Entropy, in an information sense, is a measure of unpredictability. Consider that we have T trajectories, then the maximum degree of privacy is achieved when an attacker sees all the trajectories as equally probable of providing important locations and he cannot distinguish the producer of the trajectory. Therefore, the degree of anonymity depends on the distribution of the probabilities for the T trajectories. Thus, we are able to use the Entropy to measure the quality of a set of trajectories with respect to the anonymity it provides. Our model aims to maximize the Entropy so that all the trajectories would have the same probability of being important or belonging to a specific user.

For a given distribution of probabilities, the concept of entropy in information theory provides a measure of the information contained in that distribution. We denote the entropy H of a discrete random variable x with possible values x_1, ..., x_n and the probability function $p(x)$ as: $H(X) = -\sum_{k=1}^{n} p(x_k) \log_b(p(x_k))$ where b is the base of the logarithm used.

Our algorithm is executed periodically and independently on each mobile phone and the privacy increases as the time period shortens. The goal is to get rid of frequent trajectories that contain important locations.

Thus, each smartphone defines the k most frequent locations loc_{ij} from the database. Moreover, the selection of the trajectories to be exchanged is not a trivial task since one of the neighboring smartphones might be an attacker. Thus, we should not provide only a set of sensitive location trajectories. We choose m trajectories that contain m from the k ($k > m$) most frequent locations. The selection of m is based on the Shannon's Entropy. Our algorithm is flexible enough to exchange high frequent locations as well as least frequent locations. Thus, in addition, we select n trajectories that do not contain frequent locations so that the neighbor won't be aware about which of the trajectories contain sensitive data. Exchanging both types of trajectories is important as they may contain sensitive data as well (e.g. political meetings) and thus we need to ensure that the attacker will not associate a user with these trajectories.

These trajectories ($m + n$) can be forwarded either to a specific neighbor or to $m + n$ different neighbors (one trajectory to each neighbor) and the user can remove these trajectories from the database. In our experiments we distribute the set of $m + n$ trajectories to $m + n$ individual users. The reason is that this prohibits the user that receives a trajectory to be certain whether this is a sensitive location or not. On the other side, if we send all the trajectories to one user he could conclude that at least one of the trajectories

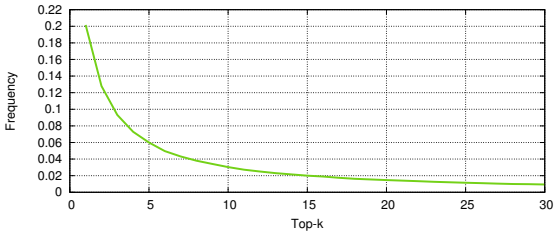

Figure 2. Frequencies of the Top-K locations.

contains a sensitive location. Furthermore, LOCATE does not allow users to respond to multiple queries originating from the same requester or from neighbors with whom the user has exchanged data in a recent time period to avoid sophisticated attacks.

B. Trajectory Exchange Phase

To find the k most frequent locations loc_{ij} we enumerate the frequencies of all the Stopping points in the user database and compute their probabilities. These represent the sensitive locations for every user. We have decided to set k to 10 due to the statistical analysis of the database, which has shown that the frequency of the locations highly decreases after the first top-10 locations. This is shown in figure 2. Thus, we first identify the top-k locations and then we exchange them whenever this exchange increases the total entropy of the database. The entropy of the database is computed by setting for every location a probability $Pr(loc_{ij}) = \frac{freq_{loc_{ij}}}{TotalFrequencyLocations}$, which is the frequency of the location loc_{ij} in the database divided by the sum of locations' frequencies, as:

$$H(loc) = -\sum_{p=1}^{L} Pr(loc_{ij}) \log_b(Pr(loc_{ij}))$$

Thus, we only exchange m out of the k locations. This exchange decreases the frequency of the specific location in the database. We select the user to whom to forward the trajectories based on historical data (to avoid contacting the same user in a short period). Afterwards, we select $n = x*m$ trajectories randomly and exchange them in parallel with the m trajectories, to prevent a privacy breach, that would occur if we sent only the sensitive trajectories, in case where there is an eavesdropper in the communication link or the user that receives the data is the attacker. We have set x to 2 in our experiments. The steps of our technique are summarized in Algorithm 1.

Obviously, when the entropy is low, the frequency of the sensitive locations would be a lot higher than the other trajectories. Using our algorithm results in maximization of the entropy after a while so that the sensitive locations frequency would become similar to the unimportant locations and thus only a few exchanges are going to be made afterwards.

Discussion. Our scheme does not assume the existence of a central server, thus, the attacker should query the devices themselves to obtain sensitive locations. In order to breach the users privacy and identify the sensitive locations, in our setting, the attacker would have to generate a very large number of queries. However this is not an easy task: (i) Due to the intermittent connectivity of the devices, we cannot assume that the devices are always on. Phone self shutdowns or manually resetting a device, make the devices unavailable and thus an attacker cannot easily collect their data. (ii) Mobile device have limited resources. Thus, when the attacker uses a mobile device to query and collect a large number of the trajectories, it will incur a high cost not only to store the data on the limited device's storage capacity (typically 32 or 64GB) but to retrieve them as well (communication cost). (iii) We cannot predict where each user's data trajectories are stored, since our scheme distributes the trajectories among all users. Nevertheless, in the unlikely case that a user collects all the data, we are led to a situation similar to having a centralized database. In that case we can use techniques that have been proposed in the literature for centralized databases (e.g. injection of fake trajectories). Our approach is flexible enough to work in concert with such well-known techniques proposed for centralized databases.

Note also, that, our technique aims to maximize the entropy, although it could be easily extended so, that, when a certain level of entropy is reached, no more trajectories are exchanged. However, this can result in reduced level of privacy since there is a trade-off among the amount of trajectories exchanged and the privacy levels that we achieve. Using such a threshold for the level of the entropy would also require a normalization since the entropy value depends on the amount and frequency of the trajectories in each individual database.

Finally we would also like to point out, that, the amount of the trajectory data exchange is smaller than the produced data. Thus, our technique does not waste communication resources, compared to a centralized approach where all the data need to be forwarded to the centralized database.

Algorithm 1 Exchange Trajectories Technique

while $(true)$ **do**
  $k = 10$, $m = 0$, $n = 0$, Sleep for $period$ seconds
  For each location loc_{ij} in the database of $user_i$ enumerate its frequency $freq_{loc_{ij}}$ and sort them.
  for $(enum = 1 : k)$ **do**
    Compute $H(loc)$
    $freq_{loc_{enum}}$ = $freq_{loc_{enum}}$ - 1
    Compute $H(loc)'$
    if $(H(loc)' \geq H(loc))$ **then**
      Mark a $Trajectory_{it}$ containing loc_{enum} for exchange
      m++, n+=x
    $freq_{loc_{enum}}$ = $freq_{loc_{enum}}$ + 1
  for $(enum = 1 : n)$ **do**
    Select randomly a $Trajectory_{it}$ that does not contain a top-k location and mark it for exchange
  Forward all tuples of the marked trajectories to other users

V. Experimental Evaluation

A. Geolife Dataset

We run experiments using the Microsoft's Geolife dataset[14], [15], [16] which is a GPS trajectory dataset collected in the context of the Geolife project using 182 users for a period of over three years (April 2007 to August 2012). A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude. This dataset contains trajectories with a total distance of about 1.2 million kilometers and a total duration of 48,000+ hours. Most of the tuples (more than 90% of the tuples) are logged in a dense representation, e.g. every 1-5 seconds. This dataset recorded a broad range of users outdoor movements, including life routines like go home and go to work as well as activities, such as shopping, sightseeing, dining, and cycling. In our experiments all of the users posses a database with at most 200 trajectories, if they were available in the Geolife database and we have a total of 22567 trajectories, which is not modified during the exchanges.

B. Experimental Setup

We have implemented our techniques on the LOCATE middleware. The system is implemented in Java and on the Android SDK with approximately 2500 lines of code. We also use a MySQL database for the user emulators as well as the native SQLite database as the data storage on our Android devices, where we used a rooted HTC Wildfire with Android 2.3.7.

When computing the frequencies for the user locations, we have set a radius of 50m, for defining close locations that need to be considered as one location. Thus, we cluster all the stop points inside a radius of 50m as one point (single user location) and the frequency of the location is set to the sum of these points.

The frequency period of our Exchange technique is set to 30 seconds, this period is enough to maximize the entropy on the mobile phones in a few minutes, without overloading the system with high overhead. However, this is a tunable parameter and we expect that after the system has been stabilized the frequency period can be increased to several minutes based on the application characteristics.

The experimental evaluation focuses on the following parameters: (i) **Entropy** that is measured as $H(loc) = -\sum_{p=1}^{L} Pr(loc_{ij}) \log_b(Pr(loc_{ij}))$ for each individual user DB, (ii) **Similarity** that shows the similarity percentage of the modified user DB's trajectories to the original user DB, computed as: $overlap/(amountOriginal + amountModified - overlap)$, where the $overlap$ is the amount of the overlapped trajectories on both databases and $amountOriginal$, $amountModified$ is the amount of trajectories on the original and the modified database respectively, (iii) **Ability to identify Sensitive Locations**, measured as the percentage of the overlap of the original top-k sensitive locations to the top-k sensitive locations of the modified user DB, (iv) **Area Coverage** where we compute the total geographical coverage for which a user has data and (v) **Overhead** where we estimate the overhead of our technique on an Android device.

We compare our approach with the **Additive Random Perturbation** technique and the **Edges Distortion** Technique. The Additive Random Perturbation technique works as follows: For each produced tuple we alter its latitude and longitude of the corresponding point, by randomly changing the location of the point within a radius of of 30 meters from the original location. The value of the radius was selected by analyzing the statistics of our databases in order to create tuples which are plausible. The Edges Distortion Technique is implemented as follows: For each produced trajectory we remove the tuples that contain stop points and each of these points is replaced with 3 new points. Every one of the 3 new points is selected within a radius of 30 meters from the previous one. Thus, we randomly develop new stopping points for each trajectory to hide the user's identity.

In all experiments we report the average values of the corresponding results from all users, unless it is explicitly stated otherwise.

C. Online Evaluation of our Exchange Technique

In the first set of experiments we evaluate the effectiveness of our technique in terms of the following parameters: Entropy, Similarity and Ability to identify Sensitive Locations.

One of our goals is to maximize entropy of the local user databases, which means that all the trajectories in the user database are equiprobable to contain sensitive locations and thus an attacker would not be able to distinguish the important ones. Figure 3 illustrates the average entropy of all users databases over time with the error bars presenting the minimum and maximum value over all users. As can be seen, the entropy increases gradually until it reaches its maximum levels after about 35 minutes and then it is stabilized. Then, all the trajectories are almost equiprobable in the local user database and their distribution through our technique would make it almost impossible for an attacker to breach the privacy of the user. The maximum entropy of each database can be computed as $\log_b Trajectories$, where $Trajectories$ is denoted as the amount of the trajectories in the database and b was defined in the previous section and set as 10. Due to the non-uniform selection of the users to the exchange of the trajectories, some of them receive more trajectories than others. That is the reason for the errorbars to have such a deviation among the maximum and the minimum value. We also computed the average value of the maximum possible entropy for all the users was 2.23, while our technique was stabilized at 2.21, which means that the entropy was almost maximized. Although we use the value 2.23 in our experiments, if we wish to achieve a

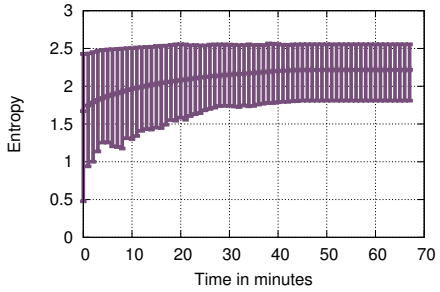

Figure 3. Average Entropy Over Time

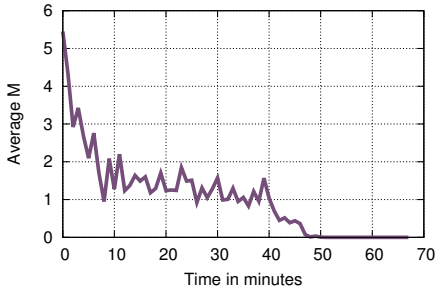

Figure 4. Average M Variable Over Time

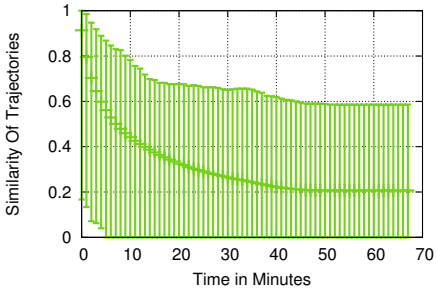

Figure 5. Average Similarity Over Time

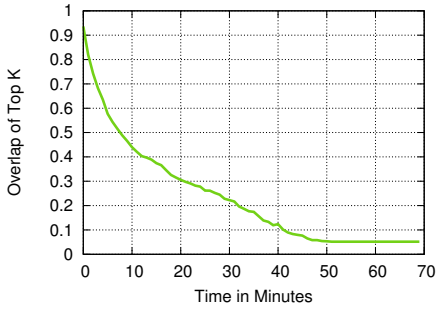

Figure 6. Ability to identify Sensitive Locations Over Time

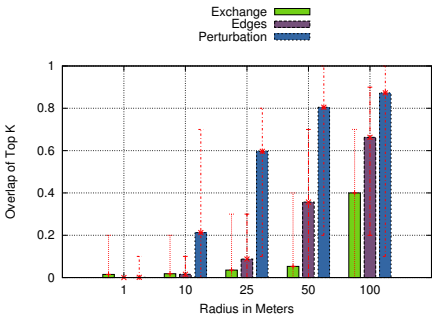

Figure 7. Ability to identify Sensitive Locations

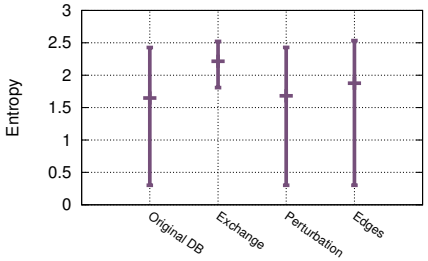

Figure 8. Entropy for the Techniques.

certain level of entropy, we could choose any value in the range [0, 2.23].

Figure 4 presents the average m variable over all users. We observe that the curve decreases until it reaches a close to zero value. This happens since at the beginning the entropy is low, meaning that the frequencies of the trajectories are not similar. Thus, there are trajectories with high frequency that need to be exchanged, so as to increase the entropy. However, when the trajectories begin to have the same frequency (possibility) in the user DB, the entropy is maximized and there is no need to exchange any other trajectories until the arrival of new trajectories. Thus, our technique can work efficiently without draining the battery with constant exchanges.

In figure 5 we observe the average similarity of trajectories with those in the original user database over time. As expected, the exchange of trajectories causes the similarity to decrease rapidly until the entropy stabilizes. This happens when almost 21% of the trajectories remain the same with the original databases.

Figure 6 illustrates the Ability to identify Sensitive Locations as the percentage of the overlap for top-10 locations among the original and the modified user DB through time. We use a radius of 50m to identify the locations. Thus, if there are 2 locations with different coordinates in the 2 sets of top-10 locations within a range of 50m this is considered as an overlap since this could lead to user identification. Hence, the overlap of the top-k locations decreases through time until it reaches 5% where the entropy has been maximized. That means that an attacker would not be able to identify the users sensitive locations even within a range of 50m.

Figures 3, 5, 6 illustrate that our technique addresses the sensitive location tracking attack, as it achieves high entropy (*i.e.,* the local trajectories are equiprobable) and the distribution of the sensitive data across the system significantly differentiates the top-10 trajectories of the user. Thus, an attacker cannot identify the frequent/sensitive locations, without retrieving the majority of the distributed data for a user, which is a difficult task as discussed earlier. Similarly, the user identification attack is addressed since the attacker is not able to obtain the user's home location, which is the most frequent location in the original database.

D. Comparison among the Different Techniques

In this set of experiments we compare the techniques in terms of Ability to identify Sensitive Locations, Entropy and Area Coverage.

Figure 7 illustrates the Ability to identify Sensitive Locations in the user's database for each of the techniques. We present the percentage of the Top-k Sensitive Locations which can be identified both in the original and the modified user DB. We set k to 10, that is, we focus on the top 10 sensitive locations. We vary the value of the radius between numbers 1m, 10m, 25m, 50m and 100m; this represents an acceptable distance among two locations to be considered as the same location. Thus we check for each of the original Top-K sensitive locations, if it can be identified as a Top-K sensitive location of the modified database inside the defined radius. By analyzing the Geolife dataset we identified 10 outliers, derived from users with a very small number of sensitive locations (2-3), while the remaining

top-10 locations were non-important locations with a small frequency. As a result these locations are not exchanged as they would not increase the entropy, but they should not be considered as sensitive locations either. Thus, even though the original top-k locations would overlap with some of the top-k locations on the modified database they would not refer to sensitive locations, so we decided not to include these users at the results of this figure. As can be observed in figure 7, for our Exchange Technique, the average overlap for all users is below 6% for each radius below 50m. One important observation is that the overlap percentage increases when the radius is set to 100m. This comes from our decision to cluster the locations inside a radius of 50m on our Exchange technique. We remark that setting the clustering distance variable to 100 meters in our online technique resulted on a decreased overlap (below 10%) for a radius of 100m, since it results on exchanging more trajectories that belong on that cluster. We can also observe the Edges Distortion technique where the stop points are being modified, with the injection of fake points. As can be seen, the Top-10 points have a low overlap due to the fact that the modified points have been moved from the original sensitive location. However, when the radius is increased, the modified stop points start to belong inside that radius and thus we are able to identify them. Finally, figure 7 also presents the overlap for the Random Perturbation Technique. We can observe that the overlap is higher than the other techniques with an overlap of 60% for a radius of 25m. This behavior derives from the random distribution that was selected to be within 30m from the original point. Thus, we infer that using sophisticated queries can expose the user sensitive locations. In addition, increasing the distance for the perturbation of the points would create non plausible trajectories.

Figure 8 presents the average as well as the minimum and maximum entropy for the different techniques and the original user database. We use a radius of 50m for the clustering of the frequencies. As can be seen the entropy in the databases that use our technique is a lot higher than the other databases. Thus, we can conclude that the frequencies of the trajectories on the other techniques make it easy for an attacker to identify the user's sensitive locations.

Figure 9 shows the Area Coverage among techniques in km^2. This figure presents an interesting effect of our approach. As can be observed, our technique offers a much greater coverage to the users than the other techniques. This happens since each user usually moves within a small range. However, exchanging trajectories with other users enables them to preserve data from other areas that they might have never visited. This distribution enables us to query users about locations that they would not be aware of otherwise.

E. Overhead

In this set of experiments we estimate the overhead of our Exchange technique in terms of computational cost as well as the execution times on the Android devices. We present the overhead for the first 20 minutes of the experiments, extracted through profiling.

Figure 10 shows the execution times of our approach on the Android devices, which includes the estimation of the trajectories frequencies, the decision of the ones to exchange and the exchanges, which is denoted as an iteration. As can be observed, at the beginning the computational times are high (5-40 seconds). This happens due to the delays on the exchange of the trajectories. We have computed that each trajectory needs 0.8 seconds in average to be moved from the sender's database to the receiver's one because of the large amount of tuples in every trajectory. However, this happens only for the initial iterations where a lot of trajectories need to be exchanged. The computational time gradually decreases afterwards until the database entropy is stabilized, when our approach needs less than a second to execute. Figure 11 illustrates the CPU percentage of our participatory sensing system over time. As can be observed at the beginning of the experiment the large amount of exchanges was reflected with a cpu percentage of 26%. However, as the time goes by and the number of exchanges reduce, the CPU percentage is reduced to 9-10%. Thus, figures 10, 11 show that the cost for the exchanges depends on the amount of sensitive trajectories and not on the number of participants, since after the sensitive trajectories have been distributed to the system, the cost is minimal.

Figure 12 illustrates the query latency of our approach for different number of users, selected by LOCATE. For this experiment we used a number of Samsung Galaxy Tab 2 P3110 tablets, with Android 4.1. In this experiment a user instantiates a range $query_q$ that requests all the stopping points in the database produced in 2008, the query is forwarded to a number of devices that execute the query in parallel and return the results. We also present the corresponding processing and communication percentage cost of the query latency. We compare the latency to a centralized setting, where a central MySQL database contains the trajectories of the corresponding number of users (2-12). The database operates on an Intel Core i5 processor laptop with 4GB of RAM, to process the queries received from the mobile devices and return the results. As the figure illustrates, our approach almost always outperforms the centralized approach. The reason is that since our approach forwards the queries to multiple devices in parallel and receives the results, it manages to achieve smaller end-to-end latency, due to the parallel execution and the utilization of multiple communication paths. On the other hand, the centralized approach experiences larger latency because the data are transmitted through the same connection serially. This is primarily due to the TCP's standard receive window, that waits for an acknowledgement each time 64KB are transmitted, before transmitting more data, as opposed to the mobile setting where multiple buffers, one for each mobile

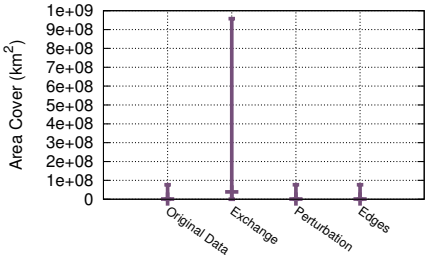

Figure 9. Area Coverage for the Techniques.

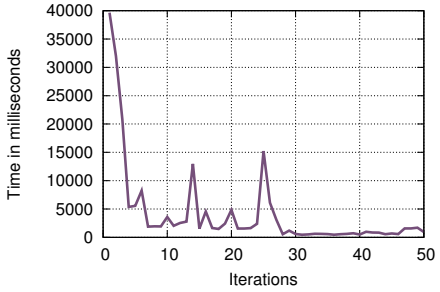

Figure 10. Execution Time.

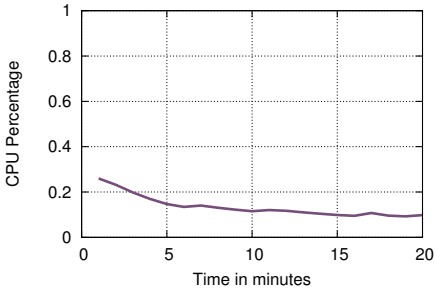

Figure 11. CPU Overhead.

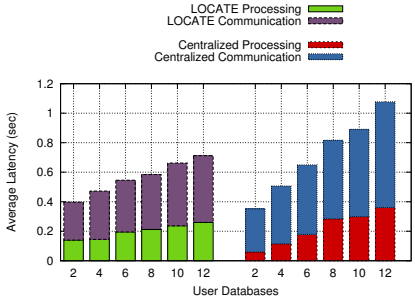

Figure 12. Latency.

device, are utilized, to receive data in parallel. Also the figure shows that the query latency in both cases increases as we employ additional devices, since it takes more time to retrieve (and merge) data from a larger number of users. Note that the centralized approach has lower computational latency for a small number of users, but as the amount of users grow, the distributed setting has the advantage of the parallel processing and receiving of the results.

VI. Related Work

Several papers have appeared in the literature that aim to preserve the privacy in databases that contain user trajectories. However in most of the cases they assume that a centralized database preserves the user data.

Pelekis *et al* in [5] examine the problems of user identification, sensitive location tracking and sequential tracking. They generate fake but realistic trajectories along with the real ones that preserve the trend of the original data. However in a local database this technique would not solve these problems since the user and his trajectories could be easily identified inside a range of values. Similarly in [6] they deal with the identification of user's trajectories by altering the paths. The key idea underlying the perturbation algorithm is to cross paths in areas where at least two users meet. Authors in [9] aim to preserve the k-anonymity using clustering so that sensitive information will not be leaked. Authors in [17] aim to preserve user's privacy by data suppression technique. They make a generalization to the spatiotemporal field of the data to avoid privacy breach. As mentioned above such a technique is not suitable for a local database that contains a lot of similar trajectories. In [18], they state that existing privacy algorithms are unable to meet accuracy requirements or fail to provide privacy guarantees in low density areas. Their technique develops an uncertainty aware privacy algorithm that uses a time-to-confusion metric to evaluate privacy in a set of location traces. This metric describes how long an individual vehicle can be tracked. Authors in [7] propose a perturbation method where the records have been perturbed so as the resulting records would look very different from the original ones and the distribution of data values is also very different from the original distribution. They also propose a reconstruction procedure to accurately estimate the distribution of the original data value. However the original data points cannot be reconstructed. In [19] they explore the use of random projection matrices for privacy preserving distributed data mining. They consider the problem of computing statistical aggregates like the inner product matrix, correlation coefficient matrix, and Euclidean distance matrix from distributed privacy sensitive data possibly owned by multiple parties.

Agrawal *et al* in [20] propose an algorithm that converges to the maximum-likelihood estimate of the original distribution and qualified the effectiveness of different perturbing distributions. In [21] entropy is used to quantify the degree of anonymity provided by schemes for anonymous connections. They consider attackers that obtain probabilistic information about users and the degree of anonymity is based on the probabilities that the attacker assigns, to the different users, as being the originators of a message.

In [22] they assume a participatory sensing system like ours and their goal is to preserve privacy. However they aim to preserve the user's current location private when a centralized server requests to assign the closest user to a number of points (Participatory Assignment problem), while our technique preserves privacy for all the user trajectories. Authors in [23] try to preserve privacy in participatory sensing systems by combining microaggregation with a tessellation model. However, they do not consider trajectories or data that have been stored but they only consider to avoid linking a user with his currently produced data. Delphine *et al* in [24] propose an anonymity-preserving reputation framework based on blind signatures, that uses periodic pseudonyms to prevent an adversary from compromising the user reports to extract private information. Finally, authors

in [25] apply negative surveys to protect the privacy of multidimensional data in participatory sensing applications with their proposed reconstruction algorithm.

Liu *et al* in [26] present Pickle, an approach for privacy-preserving collaborative learning. Pickle perturbs the training data on mobile devices with lightweight transformations to preserve the privacy of the individual training samples, but regresses these mathematical relationships between training samples, to preserve the accuracy of classification. Moreover, all the communication is between the user and the cloud with no coordination among users. On the contrary, we do not assume an infrastructure such as the cloud, and all the processing is done on the mobile devices. Furthermore, Pickle targets classification techniques for the data produced from participatory sensing rather than general queries.

VII. Conclusions

In this paper, we have presented LOCATE, a middleware that aims to provide privacy preservation for participatory sensing systems on Android smartphones. Detailed experimental results illustrate that using our technique maximizes the privacy levels on a local database by making all trajectories equiprobable, which is not possible with the existing techniques, is practical, efficient and depicts good performance. We have shown that our technique makes it difficult for an attacker to spot a user sensitive location and identify the user and we proved that our technique has a low overhead when the system is stabilized.

ACKNOWLEDGEMENT: This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the NSRF - Research Funding Program:Thalis-DISFER, Aristeia-MMD and the FP7 INSIGHT project. We are grateful to our shepherd, Baik Hoh, and the anonymous reviewers whose inputs greatly helped to improve the paper.

References

[1] S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, G. seop Ahn, and A. T. Campbell, "Metrosense project: People-centric sensing at scale," in *SenSys*, Boulder, Colorado, USA, Oct-Nov 2006.

[2] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *SenSys*, Berkeley, California, Nov 2009.

[3] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikinen, V. Tuulos, S. Foley, and C. Yu, "Data clustering on a network of mobile smartphones," in *SAINT*, Munich, Germany, July 2011.

[4] M. Olson, A. H. Liu, M. Faulkner, and K. M. Chandy, "Rapid detection of rare geospatial events: earthquake warning applications," in *DEBS*, New York, USA, July 2011, pp. 89–100.

[5] N. Pelekis, A. Gkoulalas-Divanis, M. Vodas, D. Kopanaki, and Y. Theodoridis, "Privacy-aware querying over sensitive trajectory data," in *CIKM*, Glasgow, Scotland, October 2011.

[6] B. Hoh and M. Gruteser, "Protecting location privacy through path confusion," in *SECURECOMM*, Athens, Greece, September 2005.

[7] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *SIGMOD*, Dallas, Texas, United States, May 2000.

[8] S. Datta, "On random additive perturbation for privacy preserving data mining," Master's thesis, UMBC, 2004.

[9] O. Abul, F. Bonchi, and M. Nanni, "Never walk alone: Uncertainty for anonymity in moving objects databases," in *ICDE*, Cancún, México, April 2008.

[10] A. K. Maji, F. A. Arshad, S. Bagchi, and J. S. Rellermeyer, "An empirical study of the robustness of inter-component communication in android," in *DSN*, Boston, USA, June 2012.

[11] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C. J. Scheiman, "Extending the operating system at the user level: the ufo global file system," in *ATEC*, Anaheim, CA, Jan 1997.

[12] T. Kakantousis, I. Boutsis, V. Kalogeraki, D. Gunopulos, G. Gasparis, and A. Dou, "Misco: A system for data analysis applications on networks of smartphones using mapreduce," in *MDM*, Bengaluru, India, July 2012.

[13] I. Boutsis and V. Kalogeraki, "Dynamic qos-aware event sampling for community-based participatory sensing systems," in *DEBS*, Berlin, Germany, July 2012, pp. 153–156.

[14] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Engineering Bulletin*, vol. 33, no. 2, 2010.

[15] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *WWW*, Madrid Spain, April 2009.

[16] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Y. Ma, "Understanding mobility based on gps data," in *UbiComp*, Seoul, Korea, September 2008.

[17] M. Terrovitis and N. Mamoulis, "Privacy preservation in the publication of trajectories," in *MDM*, Beijing, China, April 2008.

[18] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Preserving privacy in gps traces via uncertainty-aware path cloaking," in *CCS*, Alexandria, Virginia, USA, October 2007.

[19] K. Liu, H. Kargupta, and J. Ryan, "Random projection-based multiplicative data perturbation for privacy preserving distributed data mining," *IEEE Trans. on Knowl. and Data Eng.*, vol. 18, no. 1, pp. 92–106, Jan. 2006.

[20] D. Agrawal and C. C. Aggarwal, "On the design and quantification of privacy preserving data mining algorithms," in *PODS*, Santa Barbara, California, United States, May 2001.

[21] C. Díaz, S. Seys, J. Claessens, and B. Preneel, "Towards measuring anonymity," in *PET*, San Francisco, CA, USA, April 2002, pp. 54–68.

[22] L. Kazemi and C. Shahabi, "Towards preserving privacy in participatory sensing," in *PerCom Workshops*, Seattle, WA, USA, March 2011, pp. 328–331.

[23] K. L. Huang, S. S. Kanhere, and W. Hu, "Towards privacy-sensitive participatory sensing," in *PerCom*, Galveston, Texas, USA, March 2009.

[24] D. Christin, C. Rosskopf, M. Hollick, L. A. Martucci, and S. S. Kanhere, "Incognisense: An anonymity-preserving reputation framework for participatory sensing applications," in *PerCom*, Lugano, Switzerland, March 2012, pp. 135–143.

[25] M. M. Groat, B. Edwards, J. Horey, W. He, and S. Forrest, "Enhancing privacy in participatory sensing applications with multidimensional data," in *PerCom*, Lugano, Switzerland, March 2012, pp. 144–152.

[26] B. Liu, Y. Jiang, F. Sha, and R. Govindan, "Cloud-Enabled Privacy-Preserving Collaborative Learning for Mobile Sensing," in *SenSys*, Toronto, Ontario, Canada, Nov. 2012.