# PoolView: Stream Privacy for Grassroots Participatory Sensing

Raghu K. Ganti, Nam Pham, Yu-En Tsai, and Tarek F. Abdelzaher
Department of Computer Science, University of Illinois, Urbana-Champaign
rganti2,nampham2,ytsai20,zaher@illinois.edu

## ABSTRACT

This paper develops mathematical foundations and architectural components for providing privacy guarantees on stream data in grassroots participatory sensing applications, where groups of participants use privately-owned sensors to collectively measure aggregate phenomena of mutual interest. Grassroots applications refer to those initiated by members of the community themselves as opposed to by some governing or official entities. The potential lack of a hierarchical trust structure in such applications makes it harder to enforce privacy. To address this problem, we develop a privacy-preserving architecture, called *PoolView*, that relies on data perturbation on the client-side to ensure individuals' privacy and uses community-wide reconstruction techniques to compute the aggregate information of interest. PoolView allows arbitrary parties to start new services, called pools, to compute new types of aggregate information for their clients. Both the client-side and server-side components of PoolView are implemented and available for download, including the data perturbation and reconstruction components. Two simple sensing services are developed for illustration; one computes traffic statistics from subscriber GPS data and the other computes weight statistics for a particular diet. Evaluation, using actual data traces collected by the authors, demonstrates the privacy-preserving aggregation functionality in PoolView.

## Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics—*Time series analysis*; K.4.1 [**Computing Milieux**]: Computers and Society—*Privacy*

## General Terms

Algorithms, Design, Experimentation


*SenSys'08,* November 5–7, 2008, Raleigh, North Carolina, USA.


## Keywords

Privacy, Architecture, Data perturbation, Stream privacy, Grassroots participatory sensing

## 1. INTRODUCTION

Much of the past sensor networks research focused on *networking* issues; a scope naturally suggested by the name of the discipline. Another very important aspect of distributed sensing, however, is *data management*. In this paper, we focus on privacy as a category of data management concerns in emerging applications. Our work is motivated by the recent surge in distributed collection of data by self-selected participants for the purpose of characterizing aggregate real-world properties, computing community statistics, or mapping physical phenomena of mutual interest. This type of applications has recently been called *participatory sensing*, [5]. Examples of such applications include CarTel [22], BikeNet [10], MMM2 [8], and ImageScape [32].

This paper presents an architecture, mathematical foundations, and service implementation to enable *grassroots* participatory sensing applications. We consider communities of individuals with sensors collecting streams of private data for personal reasons. These data could also be of value if shared with the community for fusion purposes to compute aggregate metrics of mutual interest. One main problem in such applications is privacy. This problem motivates our work.

In this paper, we address privacy assurances in the absence of a trust hierarchy. We rely on data perturbation at the data source to empower clients to ensure privacy of their data themselves using tools that perturb such data prior to sharing for aggregation purposes. Privacy approaches, including data perturbation, are generally met with criticism for several good reasons. First, it has been repeatedly shown that adding random noise to data does not protect privacy [24, 21]. It is generally easy to reconstruct data from noisy measurements, unless noise is so large that utility cannot be attained from sharing the noisy data. Second, anonymity (another approach to privacy) does not help either. Anonymized GPS data still reveals the identity of the user. Withholding location data in a radius around home can be a solution, but opting to withhold, in itself may reveal information. Moreover, in a sparsely deployed network, the radius would have to be very large to truly anonymize the data. A third question is whether the assumption of lack of a centralized trusted entity is justified. After all, we already entrust our cell phone providers with a significant amount of information. It should not be difficult to

provide added-value services that benefit from the current (fairly extensive) trust model.

Below, we address the aforementioned questions prior to presenting our approach. The paper addresses the problem of privacy in time-series data[1]. The fundamental insight as to why perturbation techniques do not protect privacy is correlation among different pieces of data or between data and context (e.g., identity of owner). The authors take the small step of addressing correlations within a data stream. We show that with proper tools, non-expert users can generate appropriately-correlated application-specific noise in the absence of trust, such that data of these individuals cannot be reconstructed correctly, but community aggregates can still be computed with accuracy. We further explain how non-expert users might be able to generate the appropriate application-specific noise without trusting external parties related to that specific application. Observe that inability to reconstruct actual user data largely obviates the need for anonymity. We acknowledge that our solutions are not needed for scenarios where a hierarchy of trust exists. In contrast to such scenarios, in this paper, we are interested in providing a way for individuals in the community to collect information from their peers such as "how well does this or that diet or exercise routine work" or "what patterns of energy use at home really worked for you to reduce your energy bill"? Obviating the requirement to find a mutually trusted entity before data are collected is a way to encourage the proliferation of grassroots participatory sensing applications.

We adopt a client-server architecture, called *PoolView*, where clients share (perturbed) private sensory data and servers (called *pools*) aggregate such data into useful information made available to the community. PoolView presents a simple API for individuals to set up new pools the way they might set up a wiki or discussion group. Simple APIs are also provided for clients to subscribe to pools and export their data. Interactions between clients and servers rely on a common data-stream abstraction. A stream allows an individual to share a sequence of (perturbed) data measurements such as weight values or GPS coordinates (a logged trip). One main goal we address in this paper is to compute perturbation such that (i) it preserves the privacy of application-specific data streams against common reconstruction algorithms, (ii) it allows computation of community aggregates within proven accuracy bounds, and (iii) the perturbation (which may be application-specific) can be applied by non-expert users without having to trust the application. Hence, any person can propose a custom statistic and set up a pool to collect (perturbed) data from non-expert peers who can verify independently that they are applying the "right" (application-specific) perturbation to preserve their privacy before sharing their data.

As alluded to above, ensuring privacy of data streams via perturbation techniques is complicated by the existence of correlation among subsequent data values in time-series data. Such correlations can, in general, be leveraged to attack the privacy of the stream. For example, sharing a single data value representing one's weight perturbed by adding a random number between -2000 and 2000 pounds will usually not reveal much about the real weight. On the other hand, sharing the current weight value every day, perturbed by a different random number, makes it possible to guess the weight progressively more accurately simply by averaging the sequence to cancel out noise. Perturbing the sequence by adding the *same* random number every day does not work either because it will reveal the trend in weight measurements over time (e.g., how much weight the individual loses or gains every day). Our goal is to hide both the actual value and trend of a given individual's data series, while allowing such statistics to be computed over a community. Hence, for example, a community of weight watchers can record their weights as measured on a particular diet, allowing weight-loss statistics (such as average weight loss and standard deviation of loss) to be computed as a function of time on the diet.

To instantiate the architecture, we have implemented (code available at [30]) and deployed two PoolView services (pools), one for computing average weight of a self-selected community (e.g., all those on a particular diet), and another for computing traffic statistics in a privacy-preserving fashion. We present data from the above two case studies, collected by the authors.

The rest of this paper is organized as follows. Section 2 describes the perturbation techniques that we develop for sharing time-series data in a privacy-preserving manner. Section 3 describes PoolView, our privacy-centric architecture for participatory sensing. We discuss the results from the two case studies in Section 4. Related work is presented in Section 5. Finally, we conclude the paper in Section 6 and discuss directions for future research.

## 2. DATA PERTURBATION

Consider a participatory sensing application where users collect data that are then shared (in a perturbed form) to compute community statistics. The reader may assume an application where statistics are computed after the fact (such as average traffic or average energy consumption statistics), or where they evolve very slowly (such as weight statistics).

In this section, we provide the mathematical foundations needed for perturbing time-series data in grassroots participatory sensing applications. Our perturbation problem is defined as follows. Perturb a user's sequence of data values such that (i) the individual data items and their trend (i.e., their changes with time) cannot be estimated without large error, whereas (ii) the distribution of community data at any point in time, as well as the average community data trend can be estimated with high accuracy.

For instance, in the weight-watchers example, it may be desired to find the average weight loss trend as well as the distribution of weight loss as a function of time on the diet. This is to be accomplished without being able to reconstruct any individual's weight and weight trend. For another example, it may be desired to compute the average traffic speed on a given city street, as well as the speed variance (i.e., the degree to which traffic is "stop-and-go"), using speed data contributed by individuals without being able to reconstruct any individual's speed and acceleration curves.

Examples of data perturbation techniques can be found in [3, 2, 12]. The general idea is to add random noise with a known distribution to the user's data, after which a reconstruction algorithm is used to estimate the distribution of the original data. Early approaches relied on adding independent random noise, which were shown to be inadequate.

[1]We do not yet consider multimedia sensor data in this paper

For example, a special technique based on random matrix theory has been proposed in [24] to recover the user data with high accuracy. Later approaches considered hiding individual data values collected from different private parties, taking into account that data from different individuals may be correlated [21] [2]. However, they do not make assumptions on the model describing the *evolution* of data values from a given party over time, which can be used to jeopardize privacy of data streams. By developing a perturbation technique that specifically considers the data evolution model, we show that it is strong against attacks that extract regularities in correlated data such as spectral filtering [24] and Principal Component Analysis (PCA) [21].

## 2.1 A General Overview

We show in this paper that privacy of time-series data (measuring some phenomenon, $P$) can be preserved if the noise used to perturb the data is itself generated from a process that approximately models the phenomenon. For instance, in the weight watchers example, we may have an intuitive feel for the time scales and ranges of weight evolution when humans gain or lose weight. Hence, a noise model can be constructed that exports realistic-looking parameters for both the direction and time-constant of weight changes. We can think of this noise as the (possibly scaled) output of a *virtual user*. For now, let us not worry about who actually comes up with the noise model and what the trust implications are. Later, we shall revisit this issue in depth (Section 2.9).

Once the noise model is available (noise model generation is described at the end of this section), its structure and probability distributions of all parameters are shared with the community. By choosing random values for these parameters from the specified distribution, it is possible, for example, to generate arbitrary weight curves of virtual people showing weight gain or loss. A real user can then add their true weight curve to that of one or several locally generated virtual users obtained from the noise model. The actual model parameters used to generate the noise are kept private. The resulting perturbed stream is shared with the pool where it can be aggregated with that of others in the community. Since the distributions of noise model parameters are statistically known, it is possible to estimate the sum, average and distribution of added noise (of the entire community) as a function of time. Subtracting that known average noise time series from the sum of perturbed community curves will thus yield the true community trend. The distribution of community data at a given time can similarly be determined since the distribution of noise (i.e., data from virtual users) is known. The estimate improves with community size.

A useful refinement of the above technique is to separate in the virtual user model parameters that are *inputs* from those that express intrinsic properties of the model. For example, food intake may be an input parameter of a virtual user model. Inputs can be time-varying. Our perturbation algorithm allows changing the values of input model parameters with time. Since the input fed to the virtual users is not shared, it becomes very hard to extract real user data from added noise (i.e., virtual user) curves.

[2]Since the data correlation is across individuals of the population, we will not compare it with our work in the evaluation section

One last question relates to the issue of trust. Earlier, we motivated perturbation approaches in part by the lack of a central trusted party that would otherwise be able to privately collect real unperturbed data and compute the needed statistics. Given that non-experts cannot be asked to program noise models for each new application (or even be expected to know what these models are), and since they cannot trust the data collection party, where does the noise (i.e., the virtual user) model come from and how does a non-expert client know that the model is not fake? Obtaining the noise model from an untrusted party is risky. If the party is malicious, it could send a "bad" model that is, say, a constant, or a very fast-changing function (that can be easily separated from real data using a low-pass filter), or perhaps a function with a very small range that perturbs real data by only a negligible amount. Such noise models will not perturb data in a way that protects privacy.

The answer comes from the requirement, stated earlier, that the noise added be an approximation of the real phenomenon. Incidentally, observe that the above requirement does not mean that the noise curve be similar to the user data curve. It only means that both come from a model of the same structure but different parameters. Hence, in the weight example, it could be that the user is losing weight whereas the noise added is a curve that shows weight gain. Both curves come from the same model structure (e.g., a first order dynamic system that responds to food intake with a gain and time-constant). The models would have different parameters (a different gain, a different time-constant, and importantly a different input modeling the time-varying food intake).

With the above in mind, we allow the server (that is untrusted with our private data) to announce the used noise model structure and parameter distribution to the community of users. The model announced by the server can be trusted by a user only if that user's own data could have been generated from some parameter instantiation of that model with a non-trivial probability. This can be tested locally by a curve-fitting tool on the user's side the first time the user uses the pool. Such a general tool is a part of the client-side PoolView code distribution. Informally, a noise model structure and parameter distributions are accepted by a user only if (i) the curve fitting error for user's own data is not too large and (ii) the identified model parameter values for user's data (that result from curve fitting) are not too improbable (given the probability distributions of model parameters).

In the rest of this section, we formalize the notions of perturbation (Section 2.2), reconstruction (Sections 2.3, 2.4, 2.5, 2.6), and model validation (Section 2.7) discussed above. We prove properties of the approach such as the degree of privacy achieved and the community reconstruction error.

## 2.2 Data Perturbation Algorithm

Consider a particular application where a pool (an aggregation server) collects data from a community to perform statistics. To describe the perturbation algorithm, let $N$ be the number of users in the community. Let $M$ be the number of data points sent to the aggregation server by each user (we assume this to be the same across users for notational simplicity, but the algorithm does not depend on that). Let $x^i = (x_1^i, x_2^i, \ldots, x_M^i)$, $n^i = (n_1^i, n_2^i, \ldots, n_M^i)$, and $y^i = (y_1^i, y_2^i, \ldots, y_M^i)$ represent the data stream, noise

and perturbed data shared by user $i$, respectively. At time instant $k$, let $f_k(x)$ be the empirical community distribution, $f_k^e(x)$ be the exact community distribution, $f_k(n)$ be the noise distribution, and $f_k(y)$ be the perturbed community distribution. The exact community distribution is the distribution of a community with infinite population. In reality, this is not true. Therefore, we use the notion empirical community distribution to address the distribution of a true community with limited population. The notion of exact community distribution is useful when the reconstruction error of a small community is considered.

Most user data streams can be generated according to either linear or non-linear discrete models. In general, a model can be written as a discrete function of index $k$ (e.g. time, distance), application dependent parameters $\underline{\theta}$, inputs $\underline{\mathbf{u}}$, and is denoted as $g(k, \underline{\theta}, \underline{\mathbf{u}})$. Notice that $\underline{\theta}$ is a fixed length parameters vector characterizing the model while $\underline{\mathbf{u}}$ is a vector of length $M$ characterizing the input to the model at each instance.

For example, in the Diet Tracker application, according to one article [26], the weight of a dieting user over time [26] can be roughly approximated by three parameters: $\lambda_k$, $\beta$, and $W_0$. $\beta$ is the body metabolism coefficient, $W_0$ is the initial weight of the person right before dieting, and $\lambda_k$ is the average calorie intake of that person on day $k$. The weight $W(k)$ of a dieting user on day $k$ of the diet is characterized by a non-linear equation:

$$W(k) = W(k-1) + \lambda_k + \beta W(k-1)^{3/4} \quad (1)$$
$$W(0) = W_0 \quad (2)$$

While we do not assert the validity of the above diet model, we shall use it for illustration. In this example, the model parameter vector is $\underline{\theta} = (\beta, W_0)$ and the input to the model is $\underline{\mathbf{u}} = (\lambda_1, \lambda_2, \ldots, \lambda_M)$. From Equations (1, 2), given $\underline{\theta}$ and $\underline{\mathbf{u}}$, one can generate the weight of a user with high accuracy. While the model for a dieting person is not private and the probability distributions of weight parameters over a large population can be approximately hypothesized, it is desired to hide the parameters $\underline{\theta}$ and $\underline{\mathbf{u}}$ of any given user from being estimated. This protects an individual user's privacy.

Once the model for shared data is known, the entire data stream of user $i$ can be represented as a pair of parameter vectors $(\underline{\theta^{\mathbf{i}}}, \underline{\mathbf{u}^{\mathbf{i}}})$. We can assume that for the community, $\underline{\theta^{\mathbf{i}}}$ is drawn from a probability distribution $f_\theta(\underline{\theta})$ and $\underline{\mathbf{u}^{\mathbf{i}}}$ is drawn from another probability distribution $f_u(\underline{\mathbf{u}})$. Both the real distributions $f_\theta(\underline{\theta})$ and $f_u(\underline{\mathbf{u}})$ are unknown to the aggregation server.

The distributions $f_\theta(\underline{\theta})$ and $f_u(\underline{\mathbf{u}})$ are important since they characterize typical data streams of the users in a community. To generate noise with the same model as the data, the parameters $\underline{\theta}$ and $\underline{\mathbf{u}}$ are required. Because the real distributions $f_\theta(\underline{\theta})$ and $f_u(\underline{\mathbf{u}})$ are unknown, approximate distributions $f_\theta^n(\underline{\theta})$ and $f_u^n(\underline{\mathbf{u}})$, are used to generate $\underline{\theta}$ and $\underline{\mathbf{u}}$, respectively.

In short, given the data $x^i = (x_1^i, x_2^i, \ldots, x_M^i)$, the model $g(k, \underline{\theta}, \underline{\mathbf{u}})$, and the approximated distributions $f_\theta^n(\underline{\theta})$, $f_u^n(\underline{\mathbf{u}})$, the perturbed data for user $i$ is generated as follow:

- Generate samples $\underline{\theta_{\mathbf{n}}^{\mathbf{i}}}$ and $\underline{\mathbf{u}_{\mathbf{n}}^{\mathbf{i}}}$, from the distributions $f_\theta^n(\underline{\theta})$ and $f_u^n(\underline{\mathbf{u}})$, respectively.
- Generate noise stream $n^i = (n_1^i, n_2^i, \ldots, n_M^i)$, where $n_k^i = g(k, \underline{\theta_{\mathbf{n}}^{\mathbf{i}}}, \underline{\mathbf{u}_{\mathbf{n}}^{\mathbf{i}}})$
- Perturbed data is generated by adding the noise stream to the data stream $y^i = x^i + n^i$.

To achieve better privacy, a scaled version of the noise may be added to the data, thus the perturbed data will now be $y^i = x^i + An^i$, where $A$ is either a random variable chosen from a known distribution $f_A(A)$ or a constant. However, the choice of $f_A(A)$ (if $A$ is a random variable) or the value of $A$ (if $A$ is a constant) must be the same for all users in the community so that the aggregation server can be able to reconstruct the community distribution. In the situation that a scaling factor is used, the parameters associated with $A$ are provided by the aggregation server along with the model and the model's parameters.

## 2.3 Reconstruction of Community Average

In this section, we consider a simple case where the aggregation server is interested in estimating the community average at a certain time instance $k$. Since the parameter distributions ($f_\theta^n(\underline{\theta})$, $f_u^n(\underline{\mathbf{u}})$, $f_A(A)$) and the model $g(k, \underline{\theta}, \underline{\mathbf{u}})$ are known, the noise distribution at arbitrary time instance $k$ can be accurately calculated. All users use the same data model structure and parameter distributions to generate their noise streams. Therefore the noise distribution at any time instance $k$ is the same for all the users and is denoted as $f_k(n)$.

Upon receiving the perturbed data $y^i$ from all users, the aggregation server calculates the empirical average of the community data at time $k$ as $PA_k = \frac{1}{N}\sum_{i=1}^{N} y_k^i$. By the law of large numbers, if the number of users in the community ($N$) is large enough, the empirical value is equal to the expected value of the community perturbed data $E[y_k]$. We can write $E[y_k]$ as follows:

$$E[y_k] = E[x_k + An_k] \quad (3)$$
$$= E[x_k] + E[A]E[n_k] \quad (4)$$

Because the distribution of $A$ and $n_k$ are known to the aggregation server, $E[A]$ and $E[n_k]$ can be computed. Therefore the community average at time $k$ can be estimated as $E[x_k] = PA_k - E[A]E[n_k]$. Note that, Equation ( 4) follows from Equation ( 3) because $A$ is either a constant or a random variable that is independent of $n_k$.

In the special case that $A$ is chosen as a zero mean random variable, the estimated community average at time $k$ is also the average of the perturbed data at time $k$. In other words, the server simply averages the perturbed data to get (a good estimate of) the true community average.

## 2.4 Reconstruction of Community Distributions

We will now describe the more general problem of reconstructing the *distribution* (as opposed to the average) of community data at a given point in time. At time instance $k$, the perturbed data of each user is the sum of the actual data and the noise $y_k^i = x_k^i + n_k^i$. Thus the distribution of the perturbed data $f_k(y)$ is the *convolution* of the community distribution $f_k(x)$ and the noise distribution $f_k(n)$:

$$f_k(y) = f_k(n) * f_k(x) \quad (5)$$

All the distributions in Equation (5) can be discretized as:

$$f_k(n) = (fn(0), fn(1), \ldots, fn(L))$$
$$f_k(x) = (fx(0), fx(1), \ldots, fx(L))$$
$$f_k(y) = (fy(0), fy(1), \ldots, fy(2L))$$

And the Equation (5) can be rewritten as:

$$fy(m) = \sum_{k=-\infty}^{\infty} fx(k)fn(m-k) \quad (6)$$

Since convolution is a linear operator, Equation (6) can be written as

$$f_k(y) = Hf_k(x) \quad (7)$$

where H is a $L \times (2L+1)$ Toeplitz cyclic matrix, which is also called the blurring kernel, constructed from elements of the discrete distribution $f_k(n)$ as:

$$H = \begin{pmatrix} fn(0) & 0 & 0 & \dots & 0 \\ fn(1) & fn(0) & 0 & \dots & 0 \\ fn(2) & fn(1) & fn(0) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & fn(L) & fn(L-1) \\ 0 & 0 & 0 & \dots & fn(L) \end{pmatrix} \quad (8)$$

In Equation (7), $f_k(x)$ is the community distribution at time $k$ that needs to be estimated, $H$ is known and $f_k(y)$ is the empirical perturbed data distribution. This problem is well known in the literature as the *deconvolution problem*. Several algorithms have been developed to solve this problem and can be categorized into two classes. The first is a set of */iterative algorithms*, such as Richardson-Lucy algorithm, EM algorithm, and Poisson MAP method. The second class of algorithms are *non-iterative*, examples include Tikhonov-Miller restoration and SECB restoration. None of the iterative algorithms give bounds on the reconstruction error, while the non-iterative algorithms, supported by well defined mathematical optimization methods, give upper bounds on the reconstruction error. In this paper, the Tikhonov-Miller restoration method is employed to compute the community distribution.

The Tikhonov-Miller restoration [34] requires an apriori bound $\epsilon$ for the $L^2$ norm of the noise, together with an apriori bound $M$ for the $L^2$ norm of the community distribution:

$$||Hf_k^e(x) - f_k(y)||_2 \leq \epsilon \quad (9)$$

$$||(H^TH)^{-\nu} f_k^e(x)||_2 \leq M \quad (10)$$

Throughout this paper, $||\quad||_p$ denotes the $L_p(R)$ norm of a vector. The optimal solution $f_k(x)$ is chosen to minimize the regularized quadratic functional:

$$||Hf_k(x) - f_k(y)||_2^2 + \left(\frac{\epsilon}{M}\right)^2 ||(H^TH)^{-\nu} f_k(x)||_2^2 \quad (11)$$

The fraction $\lambda = \epsilon/M$ is called the *regularization coefficient* which governs the relative importance between the error and the regularized term [23].

By minimizing Equation (11), the exact expression for the optimal solution $f_k^*(x)$ can be found:

$$f_k^*(x) = Q_T^{-1} H^T f_k(y) \quad (12)$$

$$Q_T = H^TH + \left(\frac{\epsilon}{M}\right)^2 (H^TH)^{-2\nu} \quad (13)$$

Equation (12) and (13) are used in the aggregation server to reconstruct the community distribution. All the parameters $\epsilon$, $M$, and $\nu$ are empirically tuned to get a small reconstruction error. The optimal solution $f_k^*(x)$ may not form a probability distribution, thus normalization is needed to achieve a proper probability distribution. The relation between those parameters and reconstruction error is analyzed in the following section.

## 2.5 Error Bound on Community Distribution Reconstruction

If all the constraints in Equation (9) and Equation (10) are satisfied, the error bound of the reconstruction is given as:

$$||f_k^e(x) - f_k^*(x)||_2 \leq \sqrt{2}\{A(\nu)\}^{-1/2} M^{1/(1+2\nu)} \epsilon^{2\nu/(1+2\nu)} \quad (14)$$

$$A(\nu) = (2\nu)^{1/(1+2\nu)} + (2\nu)^{-2\nu/(1+2\nu)} \quad (15)$$

The reconstruction error bound in Equation (14) depends on $\epsilon$, $M$, and $\nu$. From Equation (14), we observe that the larger the $\epsilon$, the larger the reconstruction error's upper bound. We can rewrite Equation (9) as follows to observe the trade-off between the noise variance and the reconstruction error:

$$||Hf_k^e(x) - f_k(y)||_2 = ||f_k(n) * f_k^e(x) - f_k(n) * f_k(x)||_2 \quad (16)$$

$$= ||f_k(n) * (f_k^e(x) - f_k(x))||_2 \quad (17)$$

$$\leq ||f_k(n)||_2 ||f_k^e(x) - f_k(x)||_1 \quad (18)$$

Equation (18) is obtained from Equation (17) using Young's inequality. Note that $\epsilon$ is chosen so that Equation (9) is satisfied, therefore we can tighten the condition on $\epsilon$ such that:

$$||Hf_k^e(x) - f_k(y)||_2 \leq \epsilon \leq ||f_k(n)||_2 ||f_k^e(x) - f_k(x)||_1 \quad (19)$$

Then the error bound in (14) can be written as:

$$||f_k^e(x) - f_k^*(x)||_2 \leq \sqrt{2}\{A(\nu)\}^{-1/2} \times M^{1/(1+2\nu)} \times \times ||f_k(n)||_2^{2\nu/(1+2\nu)} \times ||f_k^e(x) - f_k(x)||_1^{2\nu/(1+2\nu)} \quad (20)$$

This equation gives us a good approximation of the community reconstruction error based on the noise distribution. The term $||f_k(n)||_2$ is the noise energy, this term is the sum of all the noise from all users at time instance $k$. The term $||f_k^e(x) - f_k(x)||_1$ is the sum of all the differences between the true community distribution at time $k$ and the empirical distribution constructed from all the community data points at time $k$, which is very small. We call this term *community sampling error*. The community sampling error depends on the number of users in the community, $N$. A larger $N$ implies a smaller community sampling error and vice versa. Thus, the larger the noise energy, the higher the reconstruction error and the larger the number of users in the community, the lower the reconstruction error. Hence, for a large enough community, a good compromise may be achieved between privacy (which we relate to noise energy) and reconstruction error.

In Section 2.3, we mentioned that privacy can be improved by multiplying the noise by a factor $A$. For simplicity, first consider the case when $A$ is a constant. Note that $||Af_k(n)||_2 = A||f_k(n)||_2$. Therefore the reconstruction error in Equation (20) is scaled by a factor of $A^{2\nu/(1+2\nu)}$. If A is a finite random variable then the bound becomes $||Af_k(n)||_2 = \delta(A)||f_k(n)||_2$, where $\delta(A)$ is a number whose value depends on the distribution of $A$. In both cases, the reconstruction error is scaled by a factor which can be estimated.

## 2.6 Privacy and User Data Reconstruction

In this section, we will analyze the level of user privacy achieved using our proposed perturbation algorithm. Many methods have been proposed to measure privacy. For example, in [3], privacy is measured in terms of confidence intervals. In [2], a measure of privacy using mutual information is suggested. However, the above methods do not take the data model as well as the exploitation method into account. In fact, privacy breaches are different depending on the exploitation method employed by the adversary. In this paper, we quantify privacy by analyzing the degree to which actual user data can be estimated from perturbed data using methods that take advantage of data correlations such as PCA and spectral filtering. Other possible estimation methods such as Maximum Mean Squared Estimation (MMSE) are also discussed.

First, consider traditional filtering methods such as PCA and spectral filtering. These methods work based on the assumptions that (i) additive noise is time independent and is independent of user data, and (ii) noise variance is small compared to the signal variance. With our proposed perturbation scheme, both assumptions are violated. The noise is generated from a known model but with unknown parameters, thus noise points are correlated. In addition, the noise and user data are generated from the same model. The noise variance is not necessarily small since it can be amplified by a factor $A$ as discussed above. The filtering techniques require prior knowledge about the noise in order to do accurate estimation. In our scheme, on the other hand, one does not know the noise distribution for any *single* user, since it is a function of the specific model parameters chosen, which remain private. We only know the distribution of such parameters over the community, but not their specific instances for any given user. Therefore, it is expected that the filtering techniques cannot reveal the user data with high accuracy. It is empirically shown in Section 4 that the PCA method is not successful in reconstructing user data. Other methods (not shown) present similarly poor reconstruction. Very little information is breached.

A better way to estimate user data is to estimate the user parameters only, since the model is known. MMSE is one of the most common methods to estimate parameters given the model. Assume that the user data is generated by $g(k, \theta_x, u_x)$, the noise by an approximated model $g_a(k, \theta_n, u_n)$, and the perturbed data $y$ is $g(k, \theta_x, u_x)+g_a(k, \theta_n, u_n)$. Then, the parameter estimation using MMSE is defined as follows:

$$(\theta^*, u^*) = \underset{(\theta,u)}{\operatorname{argmin}} ||y - g(k, \theta, u)||^2 \quad (21)$$

We consider the case when the noise is generated by a well approximated model. In this case, the perturbed data has the same dynamics as the user data and hence can be approximated by $\tilde{y} = g(k, \theta_y, u_y)$ with a very small error, $\delta = ||\tilde{y} - y||$. Then the optimal solution for Equation (21) is $(\theta^*, u^*) = (\theta_y, u_y)$. Thus, the error between the user parameters and the MMSE is $||(\theta^y, u_y) - (\theta_x, u_x)||$. In order to make it big, the set of possible noise streams must be large. If the above assumption holds and the data model is well approximated, then our approach is robust to MMSE attacks.

However, if an ill approximated data model (or a totally different model) is used to generate the noise stream, user data may be revealed. A malicious server may use this method to exploit the user parameters. In this case, instead of using the MMSE method defined above, the malicious server may use a slightly different estimation, such as:

$$(\theta^*, u^*, \tilde{\theta}^*, \tilde{u}^*) = \underset{\theta,u,\tilde{\theta},\tilde{u}}{\operatorname{argmin}} ||y - g(k, \theta, u) - g_a(k, \tilde{\theta}, \tilde{u})||^2 \quad (22)$$

In the estimation above, the malicious server tries to estimate both the parameters of the client and the parameters of the noise. Because there is a modeling mismatch, the above estimation may give a good approximation of the user data.

Similarly, a malicious server may send a good data model, but with a very small set of parameters. With this type of attack, using the parameter distributions sent by the server, the client can only generate a very small set of noise streams. Thus, the server can extract user data from perturbed data stream since the noise is predictable. We devised a method, which can be used on the user side, to effectively verify that the noise model announced at the server and its parameters are adequate. This is discussed in Section 2.7. Users may choose not to share their data if bad noise models or bad parameter distributions are detected as will be shown next.

## 2.7 Model and Parameter Verification

In Section 2.6, we have shown that a malicious server can deliberately announce a "bad" noise model in order to reveal user data by using special estimation techniques. In what follows, we will formalize the method to detect whether a model and its parameter distribution is malicious or not. The detection is based on the following observation: the user data should be a typical realization of the model which also means that the probability of the parameters of the user data, as sampled from the noise model parameter distribution, is high.

Given the model $g(k, \theta, u)$, the distribution $f^n(\theta), f^n(u)$ (both publicly announced to the community by the aggregation server) and user data $x^i$, we perform parameter verification as follows.

We estimate the user data parameters by minimizing the following quadratic function:

$$(\theta^i, u^i) = \underset{\theta,u}{\operatorname{argmin}} ||x^i - g(k, \theta, u)||^2 \quad (23)$$

The minimization problem in Equation (23) can be solved numerically by algorithms such as gradient descent or quasi Newton. A numeric solver with a simple (one-click) user API is included in the client-side PoolView code. Observe that if the given model is an ill approximation of the data model, then the error in this estimation is high. Thus to check the validity of the model it is required that the estimation error be less than a predefined threshold $p_1$, which depends on the mean and variance of user data, (i.e. $||x^i - g(k, \theta^i, u^i)|| \leq p_1$). Using triangular inequality, it can be easily shown that:

$$min(|||x_i|| - ||g(k)||_{min}|, |||x_i|| - ||g(k)||_{max}|) \leq ||x^i - g(k, \theta^i, u^i)|| \leq p_1 \quad (24)$$

In the Equation (24), $||g(k)||_{min}$ and $||g(k)||_{max}$ are the minimum and maximum norms of $g(k)$ over all possible noise curves. For each user, $||x_i||$, $||g(k)||_{min}$, and $||g(k)||_{max}$ are known. Hence this equation gives the user a lower bound for $p_1$. It is empirically shown that a good bound is usually within 10% of this estimated bound.

Finding noise model parameters that approximate the user data is not sufficient. The parameters that are obtained should not lie at the very tail of the parameter distribution sent by the server. If this happens, it would mean that either the user is anomalous (e.g., a very overweight person) or the server is not sending a representative distribution. The validity of the parameter distribution is checked by verifying if the probability of $(\theta^i, u^i)$ is larger than a threshold (i.e. $P(\theta^i, u^i) = f^n(\theta^i) f^n(u^i) \geq p_2$).

A disadvantage of this approach is that individuals whose data are anomalous will not know to trust the server even if the server was trusted. In a social setting, by contacting their peers, however, such individuals may be able to disambiguate the situation.

Further, the server can breach privacy by sequentially proposing different noise models to the user. In such a case, the user can use a simple technique of not accepting multiple noise models from a given server for a particular data stream.

Table 1 summarizes the various attacks (that breach privacy) and the countermeasures proposed in this paper.

| Attacks | Countermeasures |
|---|---|
| Reconstruction attacks using estimation techniques (e.g., PCA) | Use a noise model similar to data model |
| Malicious noise models from server | Reject models that do not fit data on client side |
| Sequential noise models from server | Limit noise model updates accepted by client |
| Malicious data from client | Reject outliers by server |

**Table 1: Summary of attacks and countermeasures**

## 2.8 Context Privacy

Thus far, we have developed a general data perturbation technique that preserves an individual user's privacy for time-series data. We observe that the privacy is preserved in the sense that the values and the trend of user data are not revealed (hard to infer). Another important aspect of privacy is *context privacy*. For example, data measurements are associated with a given time and place. Sharing data (e.g., on city traffic), even in perturbed form, still puts the user at a given time at a particular location. In this paper, we do not contribute to context privacy research. Traditional approaches to solve the problem typically rely on omitting some fields from the data shared or not answering queries for data (even in perturbed form) unless they are appropriately broad. For example, a user may reveal that they were on a particular city street at 11am on a Wednesday but not reveal which Wednesday it was. This could be enough to achieve a level of privacy and at the same time satisfy the statistical need of the aggregation server, say, if the statistic being computed is that of traffic density as a function of time of day and day of week. The policy used for blanking-out parts of the data fields shared to protect context is independent from our techniques to protect data. Context privacy policies are beyond the scope of this work.

## 2.9 Noise Model Generation

At the beginning of this section, we mentioned that the noise model must be similar to the data model to avoid reconstruction attacks. This seems problematic, since the whole purpose of measuring data is often to learn about the unknown, which seems inconsistent with having an a priori data model. There is a slight fallacy in the above argument. First, observe that in most scientific pursuits, there is a hypothesis that scientists try to prove or disprove. This means that some models and expectations already exist. These can be used to come up with a noise model. For example, there are well-known models for vehicular traffic [19], which can be used to generate the noise distribution ($f^n_\theta$). Methods to extract models from sensor network data, such as the one presented in [17], can also be used. Second, in most cases (even those not related to science), we need to have expectations on the shape of the data, at least so that we are able to tell if the sensor is working correctly or not. This means that we already have a mental model of what is probable and what is improbable. These expectations can be translated into a noise model. Further, a given model can evolve over time. As we learn more about the measured phenomenon, clients may accept a small number of noise model updates.

# 3. ARCHITECTURE AND IMPLEMENTATION

In this section, we will present the architecture that we develop for participatory sensing applications and how the theory we described in the previous section fits with our architecture. We then describe the implementation of our PoolView services.

## 3.1 General Architecture

The centerpiece of our architecture for participatory sensing is the *privacy firewall*, controlling the release of a user's private data.

A data owner will typically store their private data inside the firewall, which introduces a need for a *private storage layer*. The owner may be in possession of multiple (trusted) sensors that contribute data to her private storage. These devices collectively form the *sensing layer*. In the context of this paper, we assume that the sensors an individual owns generate time series data. Thus, for a user $i$, a given sensor generates $x^i = (x^i_1, x^i_2, \ldots, x^i_M)$. For example, a person on a diet might be using a Bluetooth scale that associates with their cell-phone to upload daily weight measurements that are then stored in the user's private storage (a prototype of this system was built by Motorola). Outside the privacy firewall are the aggregation services that collect perturbed data from multiple users to compute community information. These services form an *information distillation* layer.

The basic function of the privacy firewall is to screen or perturb user data in such a manner as to preserve the privacy of the data streams that the user owns. A *privacy table* is the central data structure of the firewall. It can be thought of as a two dimensional array whose dimensions are (i) aggregation services and (ii) data types. A cell corresponding to a given service and data type contains a pointer to the corresponding perturbation model.

To estimate the community distribution, $f_k(x)$, an aggregation server needs the perturbed distribution $f_k(y)$ and the blurring (noise) kernel, $H$. Since each client shares the per-

turbed time-series data (i.e., $y^i$), the server knows $f_k(y)$. Also, recall from Section 2 that the blurring kernel $H$ is obtained from the noise distribution, $f_k(n)$. Since the noise distribution is known to the server, the server has the requisite information to estimate the community distribution, which is done as described in Section 2.

Our service architecture is pictorially represented in Figure 1. The figure depicts the four layers of the architecture (namely, the sensor layer, the private storage layer, the privacy firewall layer, and the information distillation layer). It is useful to divide sensors in the sensor layer into ones that have Internet connectivity, such as sensors in 3G cellphones, and ones that don't. The latter would typically use proprietary or custom protocols to interface with a gateway that has Internet connectivity. It is possible that such protocols will support disruption-tolerant operation where data accumulates when the sensor is disconnected and is uploaded in bulk to the gateway when the sensor and gateway come into contact. This, for example, is the case with Smart Attire sensors described in [14], as well as the case with the vehicular study reported in the evaluation section.

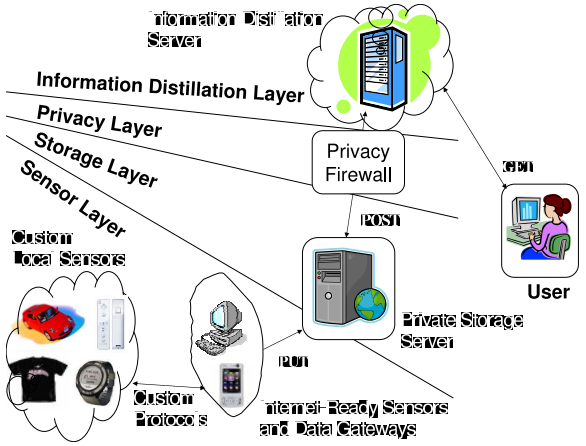


**Figure 1: PoolView architecture**

## 3.2 PoolView Implementation

In this section, we will describe the implementation details of our architecture. We implemented our architecture and instantiated it by deploying two applications, one that allows for the computation of traffic statistics, and the other that computes weight statistics.

The fundamental tie that links the layers of the information distillation architecture together is our *data stream* abstraction. A PoolView data stream is a generic time-indexed collection of *data items* from a given source. The stream is characterized by a data owner (the source), data object type, location, and (start and end) time of collection, as well as a sampling frequency if applicable. A stream may contain one or more data values. The above stream attributes are identified using XML tags.

The interfaces between layers in our architecture are implemented in HTTP. Namely, Internet-ready sensors and gateways in the sensor layer employ HTTP `PUT` commands to deposit data in storage servers. Note that, in this context, the storage server can be a simple laptop that the user owns. Users access information distillation servers using regular HTTP `GET` commands. In exchange for information, a distillation server requests relevant (perturbed) user data by issuing an HTTP `POST` query to the user's storage server. Users can also register with a distillation server to poll them for new data periodically, or have the distillation server poll them on demand (e.g., each time their storage server is updated). The `POST` query is essentially an SQL query expressed in XML to remove implementation dependencies between the client and server.

The HTTP `POST` query is intercepted by the privacy firewall, which perturbs the requested data before sharing it. The above arrangement allows us to use standard web servers to implement information distillation including perturbation and aggregation functions. A new HTTP data type is defined which characterizes a PoolView stream. When the web server receives a `GET`, `POST`, or `PUT` command on such a data type it invokes a (plug-in) handler for that data type. Hence, all is needed for implementing the scheme is to write the plug-ins. For example, in an Apache server, this corresponds to writing a module that handles the PoolView stream data type. In our current implementation, we use an Apache HTTP server with the `PUT` and `POST` handlers written as modules in Perl. The privacy firewall modules are also implemented as modules in Perl. The mathematically intensive operations on the client side, such as the ones that require curve fitting, probabilistic estimates are implemented in SciLab [33]. Java wrappers for SciLab are used to integrate the SciLab modules with Perl. Our implementation of the data storage server includes, in addition to the HTTP server, a MySQL database server for the storage and retrieval of personal data. A detailed document describing the protocol is available for reference purposes [31].

# 4. CASE STUDIES

In this evaluation section, we present two case studies with PoolView services, one is a *traffic analyzer* and the other is a *diet tracker*.

In both the case studies, when an individual user connects to the information distillation server of the corresponding service for community statistics, the server sends an HTTP `POST` request to the user's personal storage server asking for the requisite data. The request is intercepted by the user's privacy firewall. The request is validated by the user's privacy firewall by first authenticating it to ascertain if it is from the correct server and then if that server has valid access rights to the data that is requested. Data are then shared in a perturbed manner. We will first discuss the results from the traffic analyzer case study followed by the diet tracker.

## 4.1 Traffic Analyzer

The traffic analyzer case study is motivated by the growing deployment of GPS devices that provide location and speed information of the vehicles that they are deployed in. Such data can be used to analyze traffic patterns in a given community (e.g., average speed on a given street between 8am and 9am in the morning). Analysis of patterns such as rush hour traffic, off-peak traffic, average delays between different key points in the city as a function of time of day and day of the week, and average speeding statistics on selected streets can shed light on traffic safety and traffic congestion status both at a given point in time and historically over a large time interval.

With the above in mind, note that the aim of this evaluation section is to study the performance of the perturbation techniques. It is not the goal of this paper to actually study traffic comprehensively in a given city. Hence, we picked two main streets whose traffic characteristics we would like to study for illustration. To emulate a community of users, we drove on these streets multiple times (in our experiments, the authors took turns driving these streets at different times of day). We collected data for a community of 30 users. We used a Garmin Legend [15] GPS device to collect location data. The device returns a track of GPS coordinates. The sampling frequency used in our experiments was 1 sample every 15 seconds. Each trip represented a different user for our experimentation purposes. The stretch of each of the two roads driven was about 1.3 miles. Data was collected in the morning between 10 am and 12 noon as well as in the evening between 4 pm and 6 pm.

In a more densely deployed system, the assumption is that data will be naturally available from different users driving over the period of weeks on these city streets at different times of day. Such data may then be shared retroactively for different application purposes. For example, individuals interested in collecting data on traffic enforcement might collect and share speeding statistics on different city streets or freeways they travel (e.g., what percentage of time, where, and by how much does traffic speed exceed posted signage). Such statistics may come in handy when an individual travels to a new destination. Since speeding is a private matter, perturbation techniques will be applied prior to sharing.

For the purposes of this paper, we shall call the two streets we collected data from *Green Street* and *University Avenue*. The aggregation server divides city streets into small segments of equal length. The average speed on each segment is calculated from perturbed user data.

### 4.1.1 Generating the Noise Model

In order to employ our perturbation scheme, we need a noise model. Since the GPS data is collected with a very low frequency (1 sample every 15 seconds), speed may change dramatically on consecutive data points. Figures 2 shows the real speed curve of one user on Green street in the morning. We model the speed curve of each user as the sum of several sinusoidal signals (observe that any waveform can be expressed a sum of sinusoids by Fourier transform). For simplicity, we choose to use six sinusoids that represent the common harmonics present in natural speed variations of city traffic. The noise model is therefore as follows:

$$f(k) = a_0 + \sum_{i=1}^{6} a_i \sin(b_i * k + c_i) \qquad (25)$$

The speed model in Equation (25) is characterized by 19 parameters. Once the model for the speed is obtained, we need to model the distribution of all 19 parameters such that the speed stream generated by this model has the same dynamics as the real speed curves. The service developer will collect a few speed measurements empirically (which is what we did), take that small number of real speed curves, and use an MMSE curve fitting to find the range of each parameter. This approach is used by us to obtain the distribution of the parameters. The distribution of each parameter was then chosen to be a uniform within the range obtained. A sample of speed curve is shown in Figure 2.

Having produced an approximate noise model, the aggregation server announces the model information (structure and parameter distribution) to the users. Participating users use this information to choose their private noise parameters and generate their noise streams using client-side software (which includes a generic function generator in the privacy firewall). Each user's individual speed data is perturbed by the given noise and sent to the aggregation server when the user connects to the server. Typical perturbed data is shown in Figure 2.

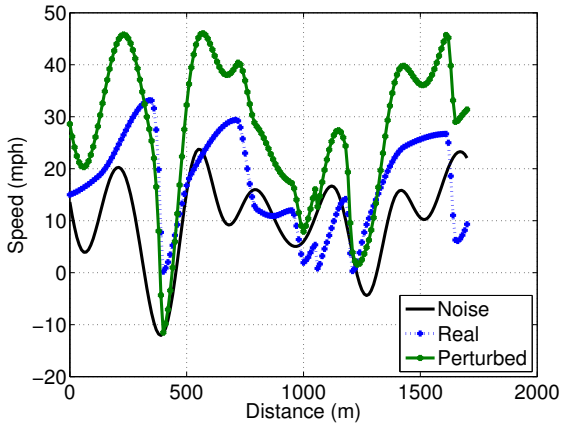


**Figure 2: Graph showing the real speed, noise, and perturbed speed curves for a single user**

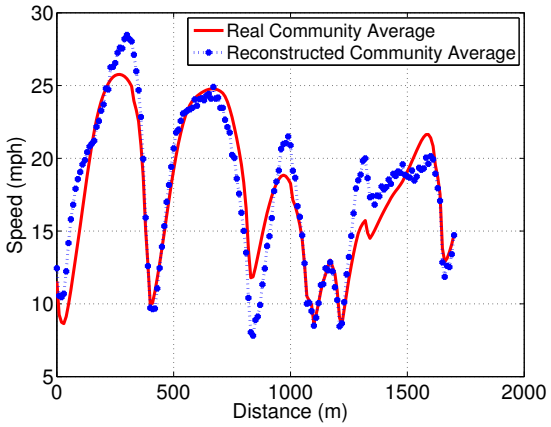


**Figure 3: Graph showing the reconstructed community average speed vs. distance for a population of 17 users**

### 4.1.2 Reconstruction Accuracy

In order to compute the community average, noise distributions at each time instance $k$ must be available for the aggregation server. Obtaining the exact noise distribution at each time k given the parameter $a_i$, $b_i$, and $c_i$ can be difficult. Therefore we approximate the noise distributions by generating a numbers of noise curves (10000 samples) following the model in Equation 25 and compute normalized histograms of noise values at each time instance. These histograms are approximations of noise distributions.

To show reconstruction accuracy using the community reconstruction method developed in Section 2.3, the computed

community average speed curve for each street is presented in Figure 3. Even with a very small community population (17 users), the community average reconstruction still provides a fairly accurate estimate (the average error at each point is 1.94 $mph$).

Next, we plot the community average reconstruction accuracy versus the scaling factor $A$ and community population $N$, which is shown in Figure 4(a). First, we examine the reconstruction error with respect to the scaling factor $A$ chosen from $\{1, 10, \ldots, 100\}$. It is theoretically shown in Section 2.3 that the reconstruction accuracy increases linearly with $A^{2\nu/(1+2\nu)}$. Thus, we should expect a linear error curve. This is verified in Figure 4(a). The errors computed in this paper are normalized by dividing the mean squared error by the number of data points. This can be interpreted as the average error for each reconstructed point. In this experiment, when $A = 80$, the normalized error is 8 $mph$, which is about one fourth of the average speed (30 $mph$). This might be unacceptable in some applications. Thus the scaling factor must be chosen with care.

Now, we compare the theoretical error bound devised in Section 2.5 and the empirical reconstruction error found above. Typical values for reconstruction parameters are $\nu = 2$, $M = 0.01$, and $||f_k^e(x) - f_k(x)||_1 = 0.5$. The normalized noise variance $||f_k(n)||$ can be bounded, $||f_k(n)||_{max} = 1.508$ (since we know the distribution of the noise for all time instances). The upper bound on the normalized error is also presented in Figure 4(a).

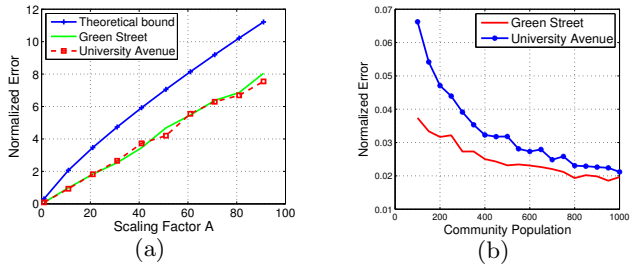


**Figure 4: Figures showing reconstruction error vs. scaling factor $A$ (a) and population of community (b)**

Next, we examine the reconstruction error versus the community population. Since our actual collected data was limited, we emulated additional user data by doing random linear combinations of data from real users. Figure 4(b) shows the normalized reconstruction error versus the community population. In this experiment, the scaling factor is fixed at $A = 1$. We observe that the error decreases exponentially with the number of users. In addition, the normalized error for a population $N = 100$ is about 0.05 which means the average error for each reconstruction point is 0.05 $mph$. This very small error suggests that our proposed reconstruction method can be used in a small community. In the above graphs, we plot the reconstruction errors for both Green Street and University Avenue.

In our next experiment, we compute the reconstructed community speed distribution at a given location on University Avenue. In order to estimate the distribution with high accuracy, it is required that the community population be large. Therefore, we emulated additional user data using the same method as described in our previous experiment. The real community speed distribution is shown in Figure 5(a). The reconstruction method discussed in Section 2.4 is used to estimate the community speed distribution from the perturbed community data, with the result being shown in Figure 5(b).

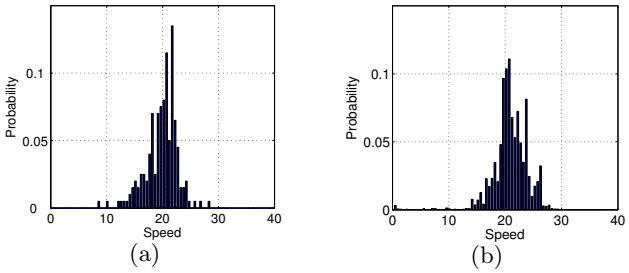


**Figure 5: Figures showing the real (a) and reconstructed (b) community speed distributions for a population of 200 users**

### 4.1.3 A Privacy Evaluation

In this section, we will analyze the degree to which an *individual* user data can be revealed in our scheme. Specifically, we choose the PCA method to obtain an estimate of the original user data from the perturbed data. The PCA method is usually very effective in reconstructing data from the perturbed data with additive noise. Figure 6 shows the real speed data of one user, the perturbed data, and the reconstructed data for the perturbation method that we developed in this paper. We observe that the reconstructed data is closer to the perturbed data and have very little correlation with the real data. We can conclude from this plot that PCA is not an effective exploitation method against our perturbation scheme. Figure 7 shows that what happens if white noise was used to perturb data, as opposed to the technique proposed in this paper. As the figure shows, in this case, PCA can reconstruct the original data curve accurately. This demonstrates the contribution of our proposed noise model generation.

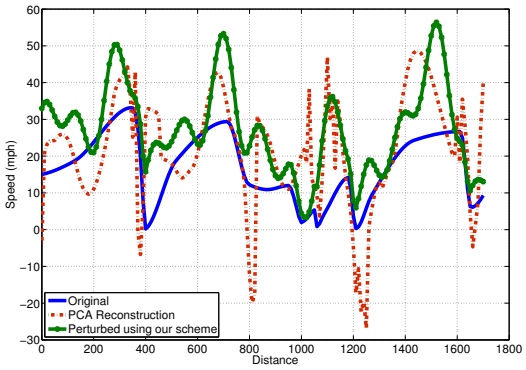


**Figure 6: PCA based reconstruction of average speed for one user when noise based on our method is used for perturbation**

For a more accurate evaluation, PCA is applied on the data of all users on both Green Street and University Avenue. Then the standard deviation of the errors, which are

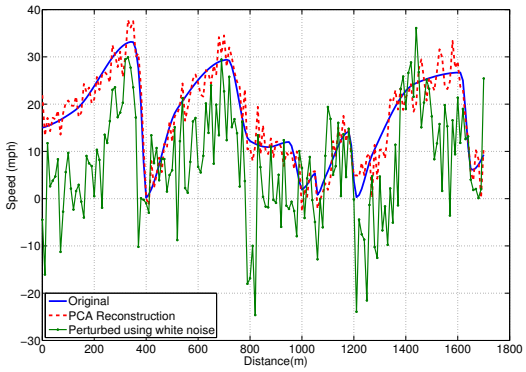


**Figure 7: PCA based reconstruction of average speed for one user when white noise is used for perturbation**

the difference between the reconstructed data using PCA and the real speed data, are calculated. The standard deviations for Green street in the morning and evening were 12.56 and 12.68 *mph*, respectively and those for the University avenue in the morning and evening were 15.56 and 10.47 *mph*, respectively. We notice that the average error in all the cases are more than 10 *mph* which is high in comparison with the average speed of 40 *mph*.

### 4.1.4 Coping with Malicious Servers

This section evaluates the techniques we developed in Section 2.7 to deal with malicious servers. A malicious server is one that "cheats" by announcing a poor noise model in an attempt to get poorly perturbed user data such that user privacy can be violated. Since the server shares both the noise model structure and parameter distributions, "cheating" can occur either by sending the wrong noise model (a model of an incompatible structure) or by sending a good model with bad parameter distributions (so that the noise curve can be easily estimated).

First, we consider an instance of a malicious server that sends a wrong noise model (for traffic data). Assume that the model sent by the server to users is a linear one, $y(k) = ak+b$, where $a$ and $b$ are two random variables uniformly distributed between -0.1 and 0.1. With this linear model, the server can easily compute an individual user's data trends. At the user side, the model is checked using the malicious server detection method discussed in Section 2.6. It is important for the user to choose the appropriate threshold $p_1$ (the acceptable fitting error). Too small a $p_1$ may cause the server to always be rejected (including good servers). Too large a $p_1$ may cause malicious server noise models to be accepted. In Figure 8(a), the acceptance rate of both malicious servers and good servers is plotted against the threshold $p_1$. We observe from the above figure that a safe threshold for $p_1$ in this case is $0.1 \leq p_1 \leq 0.3$. We can estimate $p_1$ using the method proposed in Section 2.7. For the "good" model, $||g(k)||_{min}$ and $||g(k)||_{max}$ can be computed since we know the range of all parameters, $||g(k)||_{min} = 0.616$ and $||g(k)||_{max} = 1.508$. The data of the user in this experiment has the norm of $||x|| = 1.269$. Thus, an estimate of $p_1 = 0.23$ is a good one.

Second, consider a malicious server that sends a noise model of acceptable structure but with a bad parameter

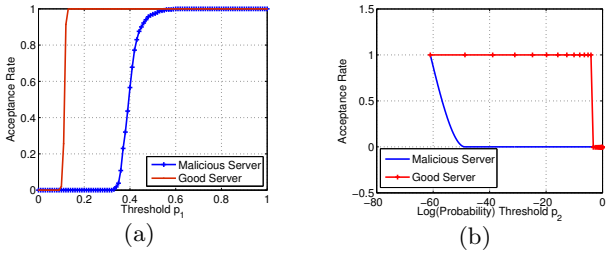


**Figure 8: Evaluation figures showing coping with malicious server, (a) shows acceptance rate vs. threshold $p_1$ and (b) shows acceptance rate vs. threshold $p_2$**

distribution. In this experiment, the distribution of all 19 parameters of our multi-sinusoid noise model is chosen to be Gaussian with the same means as a "good" model, but a very small variance $\sigma = 0.1$. Because $\sigma$ is very small, the parameters drawn from this distribution are almost the same as their means. Hence the noise curve can be easily predicted in most cases. Figure 8(b) shows the acceptance rate of the good and malicious server versus the value of threshold $p_2$ (on the probability that the users data may have come from the server supplied noise model). For computational convenience, the log of the actual probability is used as the threshold. A lower threshold is more permissive in that it accepts models that do not fit user data with high probability. From the figure, the safe range for the threshold is $-50 \leq p_2 \leq -5$ which is very wide. Thus choosing a good $p_2$ is easier than choosing a good $p_1$.

### 4.1.5 Coping with Malicious Users

Finally, we analyze the effect of malicious users on the accuracy of community average reconstruction. Observe that there is fundamentally no way to ascertain that the user-supplied sensory data is accurate. In the weight-watchers case, for instance, even if the scale could somehow authenticate the user and even if the system could authenticate the scale, there is nothing to prevent the user from climbing on the scale with a laptop or other materials, causing the reading to be incorrect. The system will work only if some motivation exists in the community to find out the real community data. We assume that for a group of self-selected participants genuinely interested in the overall statistic, such a motivation exists. The question is, how many malicious users (who purposely falsify their data) will the statistic withstand before becoming too inaccurate?

For this purpose, we generate a big community (N = 1000) and change the number of malicious users. Each malicious user generate their data according to an uniform distribution between 0 and *Range*. We are also interested in how the range of malicious data affects the reconstruction accuracy? Figure 9 plots the reconstruction error versus the percentage of malicious users for different ranges. The results show that the range of reconstruction error increases linearly but very slowly with the percentage of malicious users. In addition, the range of the malicious data has no effect on the reconstruction error. Thus, malicious users impose very little impact on the overall community reconstruction.

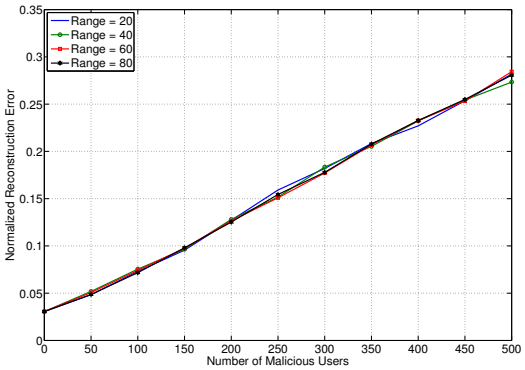


**Figure 9: Reconstruction error v.s. Number of malicious users**

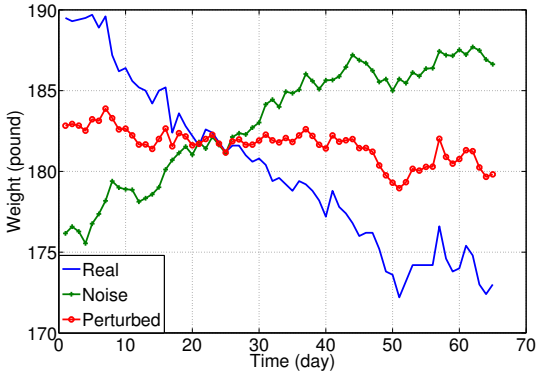


**Figure 10: Graph showing real weight, noise, and perturbed weight of a single user**

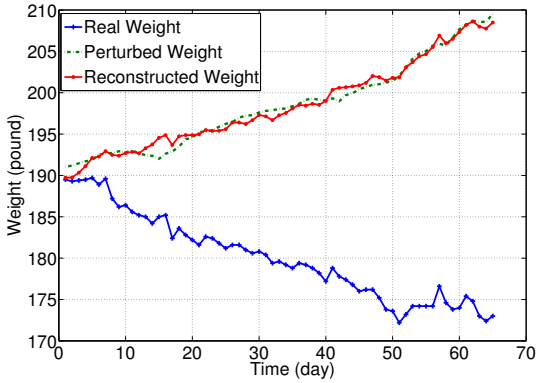


**Figure 11: Graph showing the results of PCA reconstruction scheme on a single user**

## 4.2 Diet Tracker

The diet tracker case study is motivated by the numerous weight watchers and diet communities that exist today. An individual on a particular diet monitors her weight on a periodic basis, perhaps by taking a weight measurement once a day. This individual would likely be interested in comparing her weight loss to that of other people on a diet in order to get a feedback regarding the effectiveness of the diet program she is following. Although, the person would like to do it in such a manner that her weight data remains private.

In the Traffic Analyzer application, to the extent of the authors' knowledge, there is no good speed model for a vehicle on a city road. Thus, the speed is modeled in a semi-empirical way. However, in many other applications, accurate data models are well known and hence can be used to provide more privacy. The Diet Tracker application is one such example. Several models for weight loss and dieting have been proposed in existing literature [26, 13, 4, 6]. We adopt the model proposed in [26], which is a non-linear model and is described by Equations (1) and (2). The above equations are used to generate the noise stream.

In our deployment, we recorded the weight of a single user over the course of sixty days, once each day. We generate the parameters for a typical user based on the data from our deployment and use these to emulate multiple users.

The parameters for this model include $\lambda_k$, $\beta$ and $W_0$. The range of $\lambda$ and $\beta$ can be found in [26]. The range of the initial weight $W_0$ can be taken as the weight of a normal adult which is from 80 pounds to 210 pounds. The simplest distribution for these parameters is uniform within their respective ranges. Samples of the real weight data, noise and the perturbed data are shown in Figure 10.

In this application, we demonstrate a different way of perturbing the user data, but use the same algorithm to reconstruct the community distribution. Given the generated noise $n$, and the data $x$, the perturbed data is generated as follows, $y = Ax+Bn+C$. In this type of perturbation, $A$, $B$ and $C$ are random variables whose distributions are known to the aggregation server and the users. The reconstruction of the community distribution can be done in a two-step process:

- Reconstruct the distribution of $Ax$ by considering $Bn+C$ as noise, then compute the distribution of $\log(Ax)$.
- Because $\log(Ax) = \log(A) + log(x)$, we could reconstruct the distribution of $\log(x)$ using the distribution of $\log(Ax)$ found above and the distribution of $\log(A)$. Finally, compute the distribution of $x$ from the distribution of $\log(x)$.

Note that the transformation of random variables by log and exp is trivial because both functions are monotonic. The reconstruction method used in each step is the same as the method discussed in Section 2.4. Figures 12(a) and 12(b) plot the original weight distribution and the reconstructed weight distribution using the above method, respectively. In this experiment, we use the same method described in the Traffic Analyzer application to generate a big community (500 users). For simplicity, we choose $C = 0$. $A$ and $B$ are drawn from uniform distribution between 0 and 10. We observe from the figures that the reconstructed community distribution is very close to the real distribution which suggests that the two-step reconstruction is a also good reconstruction method.

We observe from Figure 10 that the perturbed data contains a numbers of high frequency components, thus it is common to ask if the user data can be revealed using filtering techniques? We apply the PCA reconstruction method (same method used in the Traffic Analyzer application) to reconstruct an individual user's data. In order to employ PCA, we generated a virtual community containing 1000 users, where each user sends their perturbed data to the

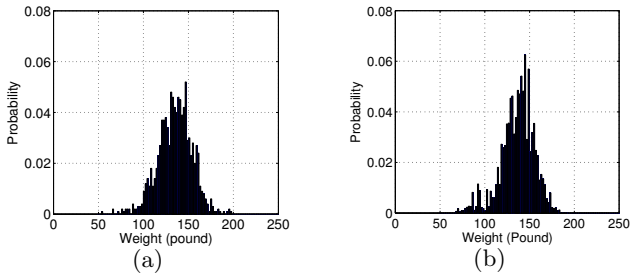


**Figure 12: Figures showing real (a) and reconstructed (b) community weight distributions for one user**

aggregation server. Figure 11 shows the real weight data, perturbed weight, and the reconstructed weight using PCA for a single user. The result shows that the reconstructed curve fits in the same direction as the perturbed data. Thus the filtering techniques again do not work with our perturbation scheme.

In conclusion, the empirical studies in this section confirm the robustness of the our perturbation technique. In the two applications, the server has successfully recovered the community information (the average and the distribution), and the user privacy is preserved against traditional attacks (filtering) and specialized attacks (MMSE). Our proposed techniques also provide means to detect malicious servers and give flexibilities by provisioning for multiple ways of perturbing user data.

# 5. RELATED WORK

Participatory sensing applications have recently been described as an important emerging category of future sensing systems [1]. Early applications have already been published including a participatory sensor network to search and rescue hikers in mountains [20], vehicular sensor networks (cars with sensor nodes) such as CarTel [22], that deployed sensor nodes in cars, and sensor networks embedded in client attire [14], cyclist networks (BikeNet) [10], cellphone camera networks for sharing diet related images (ImageScape) [32], and cellphone networks for media sharing (MMM2) [8]. The above applications, however, do not explicitly address the concerns of privacy.

An architecture for participatory sensing, called *Partisans*, has been proposed in [29]. In that paper, the main challenges addressed are those of data verifiability and privacy. In contrast to our work, the approach assumes a trusted third party. A similar trust model was assumed in [25].

Several privacy preserving data perturbation, analysis, and mining techniques have been developed in the past literature. We classify past work into three broad categories: (i) random perturbation (ii) randomized response, and (iii) secure multi-party computation. The techniques presented below can be leveraged in future incarnations of our architecture.

One of the first privacy preserving data perturbation techniques was proposed in [3]. In this technique, each client has a *single* numerical data item $x_i$, perturbed by adding a random number $r_i$ drawn independently from a known distribution. The distribution of the community can be reconstructed using the Bayes' rule to estimate a posterior distribution function. The work in [3] was extended by the authors' of [2], where a reconstruction algorithm was proposed that converges to the maximum likelihood estimate of the original distribution. Several papers [24, 21, 28], extended the technique presented in [3]. These papers show that privacy breaches occur under certain conditions, when the randomized perturbation approach is used. They then develop solutions to prevent such breaches. But, these methods do not address the problem of privacy in time-series data.

A technique for privacy preserving data clustering was developed in [27]. In the paper, a rotation-based data perturbation function is used to hide individual data. Due to the nature of the rotation matrix, the clusters are computed correctly despite randomly displacing the individual data points. It is also possible to do classification and association-rule mining[3] in a privacy-preserving manner. For example, privacy-preserving association rule mining algorithm is presented in [12]. A geometric rotation based approach for data classification is presented in [7]. This work does not address time-series data.

*Randomized response techniques* were first introduced by Warner [35] as early as 1965 to find an estimate of the percentage of people in a given population that have a sensitive attribute $X$. The idea behind the models is to ask questions that do not reveal any private information. This idea was extended in [11] to preserve privacy while mining categorical data (instead of numerical data) and to a multiple-attribute data set in [9]. The randomized response technique does not use perturbation for achieving privacy and the above work does not address privacy in time-series data.

*Secure multi-party computation* addresses computing functions of private variables where each member of the community knows (and must keep private) one of the variables. A comprehensive treatise on the basic results of secure multi-party computation is presented in [16]. The problem with this approach is its significant overhead that requires a large number of pairwise exchanges between users in the community. Such exchanges do not scale when users are not available simultaneously for purposes of completing the computation, or when the number of users involved changes dynamically.

# 6. CONCLUSIONS

In this paper, we presented an architecture and perturbation algorithms for stream privacy. It ensures the privacy of individual user data while allowing community statistics to be constructed. The architecture is geared for participatory sensing applications where community members may set up data aggregation services to compute statistics of interest. In such scenarios, the existence of mutual trust or a trust hierarchy cannot always be assumed. Hence, our data perturbation techniques allow users to perturb private measurements before sharing. The techniques address the special requirements of time series data; namely, the fact that data are correlated. Correlation makes it possible to attack privacy. A correlated noise model is proposed and implemented. It is shown that community data can be reconstructed with accuracy while individual user data cannot. In future research, we shall extend the architecture to

[3]We refer the reader to Chapters 5 and 6 of Han, [18] for the definitions of classification and association-rule mining

address privacy issues when multi-modal data are shared. In other words, multiple streams are shared by each client and such streams may be mutually correlated. Context privacy (not addressed in this work) will also be investigated. Further, we will address the problem of model checking for clients with data outliers.

## 7. ACKNOWLEDGMENTS

The authors thank the shepherd, Professor Nirupama Bulusu, and the anonymous reviewers for providing valuable feedback and improving the paper. The work reported in this paper is funded in part by Microsoft Research and NSF grants CNS 06-26342, CNS 06-15318 and DNS 05-54759.

## 8. REFERENCES

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

[2] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proc. of ACM Principles of Database Systems*, pages 247–255, 2001.

[3] R. Agrawal and R. Srikant. Privacy preserving data mining. In *Proc. of ACM Conf. on Management of Data*, pages 439–450, May 2000.

[4] S. S. Alpert. A two-reservoir energy model of the human body. *The American Journal of Clinical Nutrition*, 32(8):1710–1718, 1979.

[5] J. Burke et al. Participatory sensing. Workshop on World-Sensor-Web, co-located with ACM SenSys, 2006.

[6] C. Carson and H. Kevin. The dynamics of human body weight change. *PLOS Computational Biology*, 4(3):1000045, March 2008.

[7] K. Chen and L. Liu. Privacy preserving data classification with rotation perturbation. In *Proc. of IEEE International Conference on Data Mining*, pages 589–592, 2005.

[8] M. Davis et al. Mmm2: Mobile media metadata for media sharing. In *CHI Extended Abstracts on Human Factors in Computing Systems*, pages 1335–1338, 2005.

[9] W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *Proc. of ACM SIGKDD Conf.*, pages 505–510, 2003.

[10] S. B. Eisenman et al. The bikenet mobile sensing system for cyclist experience mapping. In *Proc. of SenSys*, November 2007.

[11] A. Evfimievski. Randomization in privacy preserving data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):43–48, December 2002.

[12] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the SIGMOD/PODS Conference*, pages 211–222, 2003.

[13] G. B. Forbes. Weight loss during fasting: Implications for the obese. *The American Journal of Clinical Nutrition*, 23(9):1212–1219, September 1970.

[14] R. K. Ganti, P. Jayachandran, T. F. Abdelzaher, and J. A. Stankovic. Satire: a software architecture for smart attire. In *Proc. of ACM MobiSys*, pages 110–123, 2006.

[15] Garmin eTrex Legend. www8.garmin.com/products/etrexlegend.

[16] O. Goldreich. Secure multi-party computation (draft). Technical report, Weizmann Institute of Science, 2002.

[17] C. Guestrin et al. Distributed regression: An efficient framework for modeling sensor network data. In *In Proc. of IPSN '04*, pages 1–10, April 2004.

[18] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, second edition, 2006.

[19] M. Herty, A. Klar, and A. K. Singh. An ode traffic network model. *J. Comput. Appl. Math.*, 203(2):419–436, 2007.

[20] J.-H. Huang, S. Amjad, and S. Mishra. Cenwits: a sensor-based loosely coupled search and rescue system using witnesses. In *Proc. of SenSys*, pages 180–191, 2005.

[21] Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *Proc. of ACM SIGMOD Conference*, pages 37–48, June 2005.

[22] B. Hull et al. Cartel: a distributed mobile sensor computing system. In *Proc. of SenSys*, pages 125–138, 2006.

[23] M. G. Kang and A. K. Katsaggelos. General choice of the regularization functional in regularized image restoration. *IEEE Transaction on Image Processing*, 4(5):594–602, May 1995.

[24] H. Kargutpa, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proc. of the IEEE ICDM*, pages 99–106, 2003.

[25] A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Toward community sensing. In *Proc. of IPSN*, 2008.

[26] R. E. Mickens, D. N. Brewley, and M. L. Russell. A model of dieting. *SIAM Review*, 40(3):667–672, September 1998.

[27] S. R. M. Oliveira and O. R. Zaiane. Privacy preservation when sharing data for clustering. In *Proc. of International Workshop on Secure Data Management in a Connected World*, pages 67–82, August 2004.

[28] S. Papadimitriou, F. Li, G. Kollios, and P. S. Yu. Time series compressibility and privacy. In *In Proc. of VLDB '07*, pages 459–470, September 2007.

[29] A. Parker et al. Network system challenges in selective sharing and verification for personal, social, and urban-scale sensing applications. In *Proceedings of HotNets-V*, pages 37–42, 2006.

[30] PoolView. http://smart-attire.cs.uiuc.edu/poolview/.

[31] PoolView Protocol Specifications. http://smart-attire.cs.uiuc.edu/poolview/files/fdtp.pdf.

[32] S. Reddy et al. Image browsing, processing, and clustering for participatory sensing: Lessons from a dietsense prototype. In *Proc of EmNets*, pages 13–17, 2007.

[33] SciLab. www.scilab.org.

[34] A. N. Tikhonov and V. Y. Arsenin. *Solution of Ill Posed Problems*. V. H. Winstons and Sons, 1977.

[35] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Jnl of the American Stat Association*, 60(309):63–69, March 1965.