PERTURBATION BASED PRIVACY FOR TIME-SERIES DATA

BY

NAM DUC PHAM

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Doctoral Committee:

Associate Professor Tarek F. Abdelzaher, Chair
Professor Richard E. Blahut
Assistant Professor Nikita Borisov
Professor Nitin H. Vaidya
ABSTRACT

This work is motivated by the emergence of participatory sensing applications, a new sensing paradigm that off-loads sensing responsibility from infrastructure sensors and professional sources to the crowd. This leads to unprecedented opportunities for sensory data collection and sharing. The privacy challenges in these applications arise naturally as personal data are shared among untrusted entities in the community. This dissertation develops mathematical foundations for optimal perturbation of both single-dimensional and multidimensional time-series data. The developed perturbation techniques allow users to effectively hide their original data while aggregated community statistics are still accurately reconstructed. Several real-world applications are also developed and successfully deployed that affirm the efficiency and accuracy of the perturbation and reconstruction techniques developed in this dissertation.
# TABLE OF CONTENTS

CHAPTER 1  INTRODUCTION ................................................. 1  
1.1  Introduction ......................................................... 1

CHAPTER 2  PERTURBATION OF SINGLE-DIMENSIONAL TIME SERIES ................. 8  
2.1  Introduction ......................................................... 9  
2.2  Perturbation and Reconstruction Algorithms ................................ 12  
2.3  Evaluation .......................................................... 30  
2.4  Conclusion .......................................................... 43

CHAPTER 3  OPTIMAL PERTURBATION OF SINGLE-DIMENSIONAL TIME SERIES ............. 44  
3.1  Optimal Perturbation of Time-Series Data .................................. 44  
3.2  Evaluation .......................................................... 55  
3.3  Conclusion .......................................................... 61

CHAPTER 4  PERTURBATION OF MULTIDIMENSIONAL TIME SERIES ......................... 62  
4.1  Joint Probability Density Function Reconstruction .......................... 62  
4.2  Perturbation of Location and Data ....................................... 71  
4.3  Simulation Results .................................................... 79  
4.4  Deployment Data ....................................................... 85  
4.5  Conclusion .......................................................... 90

CHAPTER 5  OPTIMAL PERTURBATION OF MULTIDIMENSIONAL TIME SERIES ................. 91  
5.1  Introduction ......................................................... 91  
5.2  Problem Formulation and Solution ....................................... 92  
5.3  Evaluation .......................................................... 98  
5.4  Conclusion .......................................................... 101

CHAPTER 6  RELATED WORK .................................................. 103  
6.1  Data Hiding Techniques ............................................... 103  
6.2  Privacy Measures ..................................................... 105  
6.3  Applications of Privacy-Preserving Techniques ................................ 106
CHAPTER 1

INTRODUCTION

1.1 Introduction

While classical embedded systems typically concerned themselves with interactions of computing artifacts with physical processes, future cyber-physical systems will live at the nexus of social systems, networks, and distributed physical environments. The explosive growth of mobile devices over the past decade has enabled the emergence of smart mobile devices that are capable of capturing, processing, and transmitting physical measurements of their users such as location, velocity and biometrics. Examples of such devices include cell phones with GPS capabilities, pedometers, accelerometers, vehicular with on-board diagnostics (OBD-II) readers, and watches with heart-rate monitoring sensors. These advances have encouraged the research community to explore new sensing paradigms that off-load sensing responsibility from infrastructure sensors and professional sources to the crowd. This paradigm, commonly known as crowdsourcing is exemplified in recent work on participatory sensing [1] and urban sensing [2]. Participatory sensing has tremendous potential because it harnesses the power of ordinary citizens to collect sensor data for applications spanning environmental monitoring, intelligent transportation, and public health, that are often not cost-viable using dedicated sensing infrastructure. Examples of such applications include BikeNet [3], DietSense [4] and CarTel [5].
This work focuses on the privacy problem of *time-series* data that arises when sensing is off-loaded to the crowd. Individuals who need to share data may be unwilling to reveal their private measurement. Nevertheless, the data are needed to support interesting new community applications. Data anonymization is often ineffective because the data values themselves may be highly correlated with the identity of the owner; for example, a GPS trajectory may clearly identify a person’s home. Our approach to resolve this problem is to perturb the data at the source prior to sharing, which allows users to control the amount of information being shared. The idea is to ensure that (i) original private data of an individual cannot be accurately reconstructed from shared perturbed data, yet (ii) the aggregate statistics of data collected from multiple sources can be accurately estimated despite the perturbation.

Below, we identify the main challenges in finding good data perturbation and reconstruction techniques and that satisfy the above two conflicting objectives.

- **Correlation within time-series sensor data**

Most data collected from sensors are correlated in time which might cause serious threats to user privacy. The dynamic correlations and auto-correlations, if not carefully considered, may allow for the reconstruction of the original data. From the signal processing perspective, correlated time series can be expressed in parametric models whose number of parameters is much smaller than the number of shared data points. Thus, instead of estimating all data points, adversaries can estimate the model parameters from which original data points or private information can be extracted.
It has been shown that perturbation techniques using independent noise [6, 7] are vulnerable to PCA based techniques [8, 9]. Kalman filter [10] is very effective in removing independent noise from perturbed data whose original data model is linear. In general, it is possible to filter independent noise from the shared data unless the noise is so large that no useful information can be extracted from data aggregation either.

The privacy problem for time-series data has caught the attention of the research community recently [11, 12]. No one, however, has been able to quantify the achieved privacy. Thus there is a desire for a perturbation scheme that is capable of providing quantifiable privacy for highly correlated time-series data.

- **Lack of good trade-off privacy measures**

A variety of privacy metrics have been proposed for different purposes. For example, a *data metric* [13, 14, 15, 16] measures the data quality in the entire anonymous table with respect to the data quality in the original table. A *search metric* [14, 17, 18] guides each step of an anonymization algorithm to identify an anonymous table with maximum information or minimum distortion.

In the context of perturbation techniques, it is natural to focus on *trade-off* metrics which measure the privacy and the corresponding utility achieved from the perturbed data. There have been some attempts to define good trade-off metrics [7, 19]. All of those, however, are designed for specific types of attacks and cannot be used to quantify the privacy in general. Another problem with previous privacy measures is that they require a broad knowledge of the data (e.g., distribution, model) which is hard to acquire in general. As a result, the lack of good
privacy quantification makes it hard to compare the privacy approaches objectively or choose the amount of noise to add in order to meet a given privacy objective. In this work, we aim to find a good trade-off privacy measure which requires little knowledge about the source data and is resilient to a wide range of attacks.

- **Multidimensionality of data**

In practice, the application might want to collect different kinds of information from users for the purpose of aggregating the real-world properties. For example, a traffic mapping application might want to collect both location and speed information to aggregate the speed distribution at a certain location in the city. However, a client might not want to admit to speeding, and might not want their location to be tracked. Multidimensional data exposes more security risks than single-dimensional data. Besides the auto-correlation of each stream, cross-correlation among data streams can also be used to breach privacy. In addition, multidimensionality of data introduces the scalability problem into the reconstruction algorithm. Thus we need to make sure that the reconstruction algorithm scales with the number of data streams. Therefore, it is desired to have both multidimensional perturbation and scalable reconstruction techniques that allow accurate aggregation of the community statistics in high dimension.

In this work, we contribute to three main aspects of the privacy problem. First, we present effective perturbation and reconstruction algorithms for both single-dimensional and multidimensional time-series data. Second, we present the fundamental bound on privacy and formulate the perturbation problem for time-series data as an optimization problem; hence an
optimal perturbation scheme can be found. Finally, we develop real-world applications that make use of the proposed perturbation and reconstruction techniques. The contributions of this work are described in detail below.

First, we solve the perturbation and reconstruction for one-dimensional time-series data. We show in this work that privacy of time-series data can be preserved if the noise used to perturb the data is itself generated from a process that approximately models the phenomenon. The intuition behind this idea is that if the noise and the signal have similar models, then the spectra of both overlap, making it impossible to separate the signal from the injected noise. Since the shared data is the sum of the real data and the noise, the distribution of the shared data is the convolution between the distribution of the real data and the distribution of the noise. Thus the problem of estimating the community distribution given the perturbed data and the noise distribution becomes a deconvolution problem. We solve the deconvolution problem by using the Tikhonov-Miller [20] technique with success. The advantage of applying statistical techniques to the privacy problem is that we are able to solve several associated privacy problems including detection of malicious users, and servers, and derive the relation between privacy and utility which is not possible with the policy-based privacy techniques.

We integrate the privacy techniques into Poolview [19], a participatory sensing application platform, and develop two sample applications: Traffic Analyzer and Diet Tracker. The Traffic Analyzer application allows users to share their speed data in a privacy preserving way for the purpose of aggregating traffic patterns such as rush hour traffic, off-peak traffic, average delay between different key points in the city as a function of time of day and day of the week, and the average speeding statistics on selected streets. The other application, Diet Tracker, 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. This individual would likely be interested in comparing her weight loss to that of other people on a diet in order to get feedback regarding the effectiveness of the diet program she is following. However, the person would like to do it in such a manner that her weight data remains private.

Second, to find the optimal trade-off between privacy and utility, we propose an information-theoretic privacy measure which quantifies the amount of information leak contained in the perturbed data. Using the proposed privacy measure, we are able to find a perturbation scheme which guarantees a tight upper bound on privacy. This is an important contribution since this bound offers a measure of “goodness” of time-series perturbation techniques in general. Then we derive an optimal perturbation in the sense of achieving the privacy bound for a given level of noise power.

Third, to aggregate the community statistics when multiple perturbed data streams are shared, we proposed a reconstruction technique for multidimensional data stream. Previous data perturbation techniques fail to ensure either privacy or correct reconstruction of community statistics in the case of correlated multidimensional time-series data. The algorithm proposed in this work allows participants to add noise to multiple correlated data streams prior to sharing in a privacy-preserved way while making sure that relevant community statistics are still reconstructible. We also developed a participatory sensing application for traffic monitoring which allows participants to “lie” about both their real location and speed, while letting the community estimate useful traffic statistics (e.g., speed map, percentage of speeding vehicle) with high accuracy.

Finally, we propose an optimal perturbation technique for multi-stream
data. We extend the optimal perturbation framework for single-dimensional data to the case of multidimensional data. We show that the information-theoretic privacy measure can be used as a privacy measure for multidimensional time series. We further propose an optimal perturbation scheme that allows users to perturb their multidimensional timeseries before sharing. The perturbation scheme not only provides privacy for individual streams but also protects data of one stream from being inferred from data of other streams.

The rest of this dissertation is organized as follows. Chapter 2 describes the algorithms for perturbing and reconstructing one-dimensional time-series data. Chapter 3 presents the fundamental bound on privacy of single-dimensional time series and an optimal perturbation scheme that achieves that bound. Next, we describe algorithms for perturbing multidimensional time-series data and reconstructing multidimensional community distribution in Chapter 4. Chapter 5 proposes an optimal perturbation scheme for multidimensional timeseries. Finally, related work and future developments are summarized in Chapters 6 and 7, respectively.
In this chapter, we develop a mathematical foundation for perturbing single-dimensional time series to provide privacy guarantees for stream data in participatory sensing applications. In participatory sensing, 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, making it possible to attack privacy. A correlated noise model is proposed and implemented. We show that community statistics can be reconstructed with accuracy while individual user data cannot be recovered. The algorithms are implemented on top of PoolView, a framework for developing participatory sensing applications. 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, demonstrates the functionality of the algorithms in real world applications.
2.1 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 work, 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 [1]. Examples of such applications include CarTel [5], BikeNet [3], and ImageScape [4].

In this chapter, we present mathematical foundations to enable grassroots participatory sensing applications. Unlike applications where data aggregation is performed by trusted entities such as phone service providers or city governments, grassroots applications may be initiated by ordinary individuals in the same manner as one might start a Wiki or blog on a topic of common interest. We consider communities of individuals with sensors collecting streams of private data for personal reasons, that could also be of value if shared with the community to compute aggregate metrics of mutual interest. One main problem in such application is privacy. This problem motivates our work.

In this work, 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 rea-
sons. First, it has been repeatedly shown that adding random noise to data does not protect privacy. 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 the 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 valid. 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.

This chapter addresses the above questions. The fundamental insight why perturbation techniques do not protect privacy is correlation among different pieces of data or between data and context (e.g., identity of owner). We 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. Also, note that our solutions are not needed for scenarios where a hierarchy of trust exists (e.g., applications driven by cell phone carriers). In contrast to such scenarios, in this work, we are interested in
providing a way for individuals in the community to collect information from their peers to answer questions 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?”

One main goal in this work 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.1 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

1We say usually because, for example, a shared value of 2300 pounds in the above example will still indicate a weight problem considering that at most 2000 pounds can be attributed to added noise. This issue is addressed later.
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.

The rest of this chapter is organized as follows. Section 2.2 describes the perturbation and reconstruction techniques that we develop for sharing time-series data in a privacy-preserving manner. We discuss the results from the two case studies in Section 2.3. Finally, we conclude the work in Section 2.4 and discuss directions for future research.

2.2 Perturbation and Reconstruction Algorithms

Consider a participatory sensing application where users collect data that are then shared (in a perturbed form) to compute community statistics. CarTel [5] describes the challenges and solutions for real-time data collection. We address the complementary problem of stream privacy. 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 are 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 [6, 7, 21]. 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. These approaches were shown to be inadequate. For example, a special technique based on random matrix theory has been proposed in [8] 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 [9]. 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 [8] and principal component analysis (PCA) [9].
2.2.1 A General Overview

We show in this work that privacy of time-series data (measuring some phenomenon) 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.

Once the noise model is (somehow) available, 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 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 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.

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 is 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. Informally, a noise model structure and parameter distributions are accepted by a user only if (i) the curve fitting error for the user’s own data is not too large and (ii) the identified model parameter values for the 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, reconstruction, and model validation discussed above. We prove properties of the approach such as the degree of privacy achieved and the community reconstruction error. The details on the data model and noise generation are discussed in detail in Section 2.2.2. The algorithm for reconstructing community statistics from perturbed data is discussed in Section 2.2.3 and Section 2.2.4, respectively. Furthermore, the analysis of error of the reconstruction algorithm is discussed in Section 2.2.5. Finally, the user privacy is presented in Section 2.2.6.
2.2.2 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 $x^i = (x^i_1, x^i_2, \ldots, x^i_M)$, $n^i = (n^i_1, n^i_2, \ldots, n^i_M)$, and $y^i = (y^i_1, y^i_2, \ldots, y^i_M)$ 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, $f^*_{k}(x)$ 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 the case; 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 $g(k, \theta, u)$, a discrete function of index $k$, which can be time, distance, etc., depending on the application, parameters $\theta$, and inputs $u$. Notice that $\theta$ is a vector of fixed length parameters characterizing the model, while $u$ is a vector of length $M$ characterizing the input to the model at each instance.

For example, in the Diet Tracker application, the weight of a dieting user over time [22] can be characterized by three parameters $\lambda_k$, $\beta$, and $W_0$, where $\beta$ 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:

$$W(k) = W(k-1) + \lambda_k - \beta W(k-1)^{3/4}$$  \hspace{1cm} (2.1)$$

$$W(0) = W_0$$  \hspace{1cm} (2.2)$$

In this example, the model parameter is $\theta = (\beta, W_0)$ and the input to the model is $u = (\lambda_1, \lambda_2, \ldots, \lambda_M)$. From Equations (2.1, 2.2), given $\theta$, and $u$, one can generate the weight of the 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 $\theta$ and $u$ of any given user to prevent their 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 $(\theta_i, u_i)$. We can assume that for the community, $\theta_i$ is drawn from a probability distribution $f_\theta(\theta)$ and $u_i$ is drawn from another probability distribution $f_u(u)$. Both the real distributions $f_\theta(\theta)$ and $f_u(u)$ are unknown to the aggregation server.

The distributions $f_\theta(\theta)$ and $f_u(u)$ 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 $\theta$ and $u$ are required. Because the real distributions $f_\theta(\theta)$ and $f_u(u)$ are unknown, approximate distributions $f_{\theta}^n(\theta)$ and $f_{u}^n(u)$ are used to generate $\theta$ and $u$, respectively.

In short, given the data $x^i = (x^i_1, x^i_2, \ldots, x^i_M)$, the model $g(k; \theta, u)$, and the approximated distributions $f_{\theta}^n(\theta)$, $f_{u}^n(u)$, the perturbed data for user $i$ is generated as follows:
• Generate samples $\theta_i^n$ and $\mathbf{u}_i^n$ from the distributions $f_{\theta}^n(\theta)$ and $f_u^n(\mathbf{u})$, respectively.

• Generate noise stream $n^i = (n_{i1}^i, n_{i2}^i, \ldots, n_{iM}^i)$, where $n_{ik}^i = g(k, \theta^n_i, \mathbf{u}_n^i)$.

• Generate perturbed data 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 where 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.

2.2.3 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 $(f_{\theta}^n(\theta), f_u^n(\mathbf{u}), f_A(A))$ and the model $g(k, \theta, \mathbf{u})$ 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 the perturbed data $y^i$ from all users, the aggregation server calculates the empirical average of the community data at time $k$ as
\[ PA_k = \frac{1}{N} \sum_{i=1}^{N} y^i_k. \] 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

\[
E[y_k] = E[x_k + A n_k] \tag{2.3}
\]

\[
= E[x_k] + E[A]E[n_k] \tag{2.4}
\]

Because the distributions 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 (2.4) follows from Equation (2.3) because \( A \) is either a constant or a random variable that is independent of \( n_k \).

In the special case where \( A \) 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.

### 2.2.4 Reconstruction of Community Distributions

We will now describe the more general problem of reconstructing the distribution (as opposed 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 \( y^i_k = x^i_k + n^i_k \). Thus the distribution of the perturbed data \( f_k(y) \) is the convolution of the community distribution \( f_k(x) \) and the noise distribution \( f_k(n) \):

\[
f_k(y) = f_k(n) * f_k(x) \tag{2.5}
\]
All the distributions in Equation (2.5) can be discretized as \( f_k(n) = (fn(0), fn(1), \ldots, fn(L)) \), \( f_k(x) = (fx(0), fx(1), \ldots, fx(L)) \), and \( f_k(y) = (fy(0), fy(1), \ldots, fy(2L)) \). And the Equation (2.5) can be rewritten as

\[
f_y(m) = \sum_{k=0}^{\infty} f_{k}(m - k) \quad (2.6)
\]

Since convolution is a linear operator, Equation (2.6) can be written as

\[
f_k(y) = H f_k(x) \quad (2.7)
\]

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

\[
H = \begin{pmatrix}
fn(0) & 0 & 0 & \ldots & 0 \\
fn(1) & fn(0) & 0 & \ldots & 0 \\
fn(2) & fn(1) & fn(0) & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & fn(L) & fn(L-1) \\
0 & 0 & 0 & \ldots & fn(L)
\end{pmatrix} \quad (2.8)
\]

In Equation (2.7), \( f_k(x) \) 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 the Richardson-Lucy algorithm, EM algorithm, and Poisson MAP method. The second class of algorithms is 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 the well defined mathematical optimization methods, give upper bounds on the reconstruction error. In this work, the Tikhonov-Miller restoration method is employed to compute the community distribution.

The Tikhonov-Miller restoration [20] 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:

\[
\|Hf^e_k(x) - f_k(y)\|_2 \leq \epsilon \tag{2.9}
\]
\[
\|(H^TH)^{-\nu}f^e_k(x)\|_2 \leq M \tag{2.10}
\]

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

\[
\|Hf_k(x) - f_k(y)\|_2^2 + \left(\frac{\epsilon}{M}\right)^2 \|(H^TH)^{-\nu}f_k(x)\|_2^2 \tag{2.11}
\]

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 (2.11), the exact expression for the optimal solution $f^*_k(x)$ can be found:

\[
f^*_k(x) = Q^{-1}_T H^T f_k(y) \tag{2.12}
\]
\[
Q_T = H^TH + \left(\frac{\epsilon}{M}\right)^2 (H^TH)^{-2\nu} \tag{2.13}
\]

Equations (2.12) and (2.13) are used in the aggregation server to recon-
struct 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.

2.2.5 Error Bound on Community Distribution Reconstruction

If all the constraints in Equations (2.9) and (2.10) are satisfied, the error bound of the reconstruction is given as

$$\| f^e_k(x) - f^*_k(x) \|_2 \leq \sqrt{2} \{ A(\nu) \}^{-1/2} M^{1/(1+2\nu)} \epsilon^{2\nu/(1+2\nu)} \quad (2.14)$$

$$A(\nu) = (2\nu)^{1/(1+2\nu)} + (2\nu)^{-2\nu/(1+2\nu)} \quad (2.15)$$

The reconstruction error bound in Equation (2.14) depends on $\epsilon$, $M$, and $\nu$. From Equation (2.14), we observe that the larger the $\epsilon$, the larger the reconstruction error’s upper bound. We can rewrite Equation (2.9) as follows to observe the trade-off between the noise variance and the reconstruction error:

$$\| H f^e_k(x) - f_k(y) \|_2 \leq \| f_k(n) \ast f^e_k(x) - f_k(n) \ast f_k(x) \|_2 \quad (2.16)$$

$$= \| f_k(n) \ast (f^e_k(x) - f_k(x)) \|_2 \quad (2.17)$$

$$\leq \| f_k(n) \|_2 \| f^e_k(x) - f_k(x) \|_1 \quad (2.18)$$

Equation (2.18) is obtained from Equation (2.17) using Young’s inequality. Note that $\epsilon$ is chosen so that Equation (2.9) is satisfied; therefore, we can
tighten the condition on $\epsilon$ such that

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

(2.19)

Then the error bound in (2.14) can be written as

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

$$
\times \|f_k(n)\|_2^{2\nu/(1+2\nu)} \times \|f_k^e(x) - f_k(x)\|_1^{2\nu/(1+2\nu)}
$$

(2.20)

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, and represents 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 the difference 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 the community sampling error. The community sampling error depends on the number of the 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 2.2.3, we mentioned improving the privacy 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 (2.20) 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
δ(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.

2.2.6 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 [6], privacy is measured in terms of confidence intervals. In [7], 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 work, 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 principal component analysis (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 following assumptions:

- Additive noise is time-independent and is independent of user data.

- 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 2.3 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. Assuming that the user data is generated by $g(k, \theta_x, u_x)$, the noise is generated by an approximated model $g_a(k, \theta_n, u_n)$ and the perturbed data is the sum of the $y = g(k, \theta_x, u_x) + g_a(k, \theta_n, u_n)$. The parameter estimation using MMSE is defined as follows:

$$
(\theta^*, u^*) = \arg\min_{(\theta, u)} ||y - g(k, \theta, u)||^2
$$

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 very small error $\delta = ||\tilde{y} - y||$. Then the optimal solution for Equation (2.21) 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 as:

\[
(\theta^*, u^*, \tilde{\theta}^*, \tilde{u}^*) = \arg\min_{\theta, u, \tilde{\theta}, \tilde{u}} \| y - g(k, \theta, u) - g_a(k, \tilde{\theta}, \tilde{u}) \|^2
\] (2.22)

In the estimation above, the malicious server tried to estimate both the parameters of the client and also 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 parameters distribution sent by the server, the client can only generate a very small set of noise streams. Thus, the server can extract user data from the 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 2.2.7. Users may choose not to share their data if bad noise models or bad parameter distributions are detected, as will be shown next.

2.2.7 Model and Parameter Verification

In Section 2.2.6, 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 this work, we proposed a method to detect if a model along with 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 propose a two-step model and parameter verification:

1. Estimate the user data parameters by minimizing the following quadratic function:

\[
(\theta^i, u^i) = \operatorname{argmin}_{\theta, u} \| x^i - g(k, \theta, u) \|^2
\]  

(2.23)

The minimization problem in Equation (2.23) can be solved numerically by algorithms such as gradient descent or quasi Newton. 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

\[
\min(|| x_i || - || g(k) ||_{\min}, || x_i || - || g(k) ||_{\max})
\]

\[
\leq || x^i - g(k, \theta^i, u^i) || \leq p_1
\]  

(2.24)

In Equation (2.24), \( || g(k) ||_{\min} \) 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.

### 2.2.8 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 trend of user data are not revealed (and are 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 work, 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 11:00 a.m. 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.

2.3 Evaluation

In this evaluation section, we present two case studies: a diet tracker and a traffic analyzer. Our privacy algorithms are implemented on PoolView, a framework for developing participatory sensing applications. 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, which validates the request 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 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.

2.3.1 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 8:00 a.m.
and 9:00 a.m.). 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 work 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, the author enlisted the help of colleagues in driving on these streets multiple times (in our experiments, we took turns driving these streets at different times of day). We collected data for a community of 30 users. We used a Garmin Legend [24] 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 10:00 a.m. and 12:00 p.m. as well as in the evening between 4:00 p.m. and 6:00 p.m.

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 be useful 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 work, 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.

2.3.1.1 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. Figure 2.1 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 chose to use six sinusoids that represent the common harmonics present in natural speed variations of city traffic. The noise model is therefore as follows:

\[
f(k) = a_0 + \sum_{i=1}^{6} a_i \sin(b_i \ast k + c_i)
\]

The speed model in Equation (2.25) 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 curves is shown in Figure 2.1.

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 2.1.

![Figure 2.1: Real speed, noise, and perturbed speed curves for a single user](image)

2.3.1.2 Reconstruction Accuracy

To show reconstruction accuracy using the community reconstruction method developed in Section 2.2.3, the computed community average speed curve for each street is presented in Figure 2.2. 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 2.3a.
Figure 2.2: Reconstructed community average speed vs. distance compared to the real community average for a population of 17 users.

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 2.2.3 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 2.3a. The errors computed in this work are normalized by dividing mean squared error by the number of data points; this can be seen as the average error for each reconstructed point. In this experiment, if $A = 80$ then the normalized error is 8, which is about one fourth of the average speed. This might be unacceptable in some applications, so the scaling factor must be used with care.

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 2.3b 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 error due to community population is small in comparison with the error due to the scaling factor $A$. This 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.
Figure 2.3: Reconstruction error vs. (a) scaling factor $A$ and (b) population of community

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 2.4a. The reconstruction method discussed in Section 2.2.4 is used to estimate the community speed distribution from the perturbed community data, with the result being shown in Figure 2.4b.

Figure 2.4: (a) Real and (b) reconstructed community speed distributions
2.3.1.3 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. In addition, filtering techniques such as spectral filtering [8] are a special case of PCA with different way of choosing principal dimensions. Figure 2.5 shows the real speed data of one user, the perturbed data, and the reconstructed data. We observe that the reconstructed data is closer to the perturbed data and has 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 2.5: PCA based reconstruction of average speed of a single user](image)

2.3.1.4 Coping with Malicious Servers

This section evaluates the techniques we developed in Section 2.2.7 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 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 the server sends to users is a linear one, \( y(k) = a \cdot k + b \), 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 2.2.6. 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 2.6a, 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 2.2.7. 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.

![Figure 2.6: Evaluations of coping with a malicious server](image-url)
Second, consider a malicious server that sends a noise model of acceptable structure but with a bad parameter 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 2.6b shows the acceptance rate of both good server and malicious server versus the value of threshold $p_2$ (on the probability that the user 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 $p_1$.

2.3.1.5 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 generates their data according to a uniform distribution between 0 and R. We are also interested in how the range of malicious data affects the reconstruction accuracy. Figure 2.7 plots the reconstruction error versus the percentage of malicious users for different ranges. The results show that the range 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 2.7: Reconstruction error vs. number of malicious users](image)

2.3.2 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 his/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 feedback regarding the effectiveness of the diet program she is following. However, the person would like to do it in such a manner that her weight data remains
In the Traffic Analyzer application, to the extent of the author’s 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. The weight of a person on a diet is a non-linear model [22] and is described by Equations (2.1) and (2.2). 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 60 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 [22]. 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 2.8.

![Figure 2.8: Real weight, noise, and perturbed weight of a single user](image)

In this application, we demonstrate a different way of perturbing the user data. 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 distribution is known to the aggregation server and the users. The advantage of using this type of perturbation is that it can effectively eliminate the edge effect which is discussed in Section 2.1. 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.
- Reconstruct the distribution of \( x \) using the distribution of \( Ax \) found above and the distribution of \( A \).

The reconstruction method used in each step is the same as discussed in Section 2.2.4. Figures 2.9a and 2.9b 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). In this experiment, we choose \( C = 0 \) for simplicity. \( A \) 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 method.

We observe from Figure 2.8 that the perturbed data contains lots of high frequency components, so it is natural to ask if the user data can be revealed using filtering techniques. We apply the PCA reconstruction method (same method used in 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 aggregation server. Figure 2.10 shows the real weight data, perturbed weight, and the reconstructed weight using PCA for a single user. The result
Figure 2.9: (a) Real and (b) reconstructed community weight distributions 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.

Figure 2.10: PCA based estimation of weight curve for a single user

In conclusion, the empirical studies in this section confirm the robustness of 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.
2.4 Conclusion

In this work, we presented the perturbation algorithms for stream privacy. The algorithms ensure the privacy of individual user data while allowing community statistics to be constructed. 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.

Although this work affirms the effectiveness of using correlated noise against personal data exploits, the optimality of this method still remains unknown. In the next chapter, we will discuss in detail the optimality of perturbation techniques for single-dimensional time series. In Chapters 4 and 5, we extend the algorithms to address privacy issues when multi-modality data are shared. In other words, multiple streams are shared by each client and such streams may be mutually correlated.
This chapter extends the perturbation based privacy technique proposed in the previous chapter by defining an information-theoretic privacy measure and finding the optimal perturbation noise that minimize this privacy measure. Subsequently, the optimal trade-off is explored between individual user privacy, achieved by perturbing data prior to sharing, and the corresponding accuracy of computed aggregate information. The new algorithm effectively hides individual user data by optimally perturbing the time series using knowledge of only the mean and the covariance of the original data. We evaluate it using both synthetic data and collected real application data. The results show that the method significantly improves the trade-off between privacy and the accuracy of reconstruction of aggregate information from shared perturbed data.

3.1 Optimal Perturbation of Time-Series Data

Providing privacy for time-series data is much harder than doing so for a single data point because the correlation among data points can be exploited to gain useful information from the noisy shared data. It has been shown in the previous chapter that correlated noise can be used to perturb time-series data with success. However, it only shows that the scheme works against some popular attacks (e.g., spectral filtering and Kalman filtering).
It remains unknown whether more clever attacks can reveal original data. In this section, we present a perturbation scheme in which the amount of information leak can be controlled. We first introduce a privacy measure that quantifies the amount of information leak contained in shared data. Then, a generic time-series perturbation scheme that minimizes the information leak, subject to a noise power constraint, is presented. Finally, we apply the proposed generic perturbation scheme to perturb an arbitrary stationary time-series signal such that individual data points, the average and the trend are hidden.

3.1.1 Privacy Measure

We seek a general privacy measure that can be applied to a wide variety of time-series signals. In this chapter, we assume that original time-series signals are stationary. As mentioned earlier, a good privacy measure must be independent of noise filtering methods. Another observation is that the privacy measure must be proportional to the power of the injected noise. We cannot perturb the data with an arbitrarily large noise that would render the shared data useless. Thus, limiting the noise power is a necessary constraint in defining a privacy measure. With those observations in mind, we propose the use of mutual information between the original data and the perturbed data as a measure for privacy.

Mathematically, let $X$ be the user data time series, $Z$ be the generated noise time series, and $Y = X + Z$ be the shared perturbed data time series. The information about the original data $X$ contained in the shared data $Y$ is the mutual information $I(X, Y)$ between $X$ and $Y$. The mutual information can be seen as the information leak about $X$ given the perturbed data.
Y. We have the following definition of the *minimal information leak* for a perturbation scheme:

**Definition 1.** The *minimal information leak* $P_X^o$ for a given data $X$ at a noise power level $P_0$ is defined as

$$P_X^o = \min_{Z} I(X, X + Z)$$

subject to $P_Z \leq P_0$

The unit of the information leak is *bits* or *nats*. Also, note that the lower the information leak, the better the achieved privacy. We call the noise $Z_X^o$ with power $P_0$ that achieves the minimal information leak the *optimal noise* for $X$. The chosen noise power $P_0$ is used to control the trade-off between privacy and utility. Ideally, we would like to find the value $P_0$ for a given utility value. This, however, is application specific and depends on the utility function of the application. In this work, we determine the value of $P_0$ based on empirical historical data at the server.

Theoretically, the optimal noise for a given data time-series $X$ can be obtained by solving the optimization problem in the Definition 1. Unfortunately, that problem is intractable because there is no general way to compute the mutual information $I(X, X + Z)$ for arbitrary probability distributions of $X$ and $Z$. Therefore, we relax the optimization problem by minimizing a tight upper bound on the information leak. To find an upper bound for the information leak, we use the following lemma [25]:

**Lemma 1** (Ihara, 78). Let $X$, $Y$ and $Z$ be $n$-dimensional random variables such that $X$ is independent of $Z$ and $Y = X + Z$. Let $(X_G, Y_G, Z_G)$ be a triplet of $n$-dimensional Gaussian random variables with the same covariance as the
triplet \((X, Y, Z)\). Then we have the following lemma:

\[
I(X, Y) \leq I(X_G, X_G + Z_G) + D(Z\|Z_G) \tag{3.1}
\]

where \(D(Z\|Z_G)\) is the Kullback-Leibler divergence of \(Z_G\) from \(Z\).

As \(D(Z\|Z_G) \geq 0\) with equality if and only if \(Z\) has the same distribution as \(Z_G\), the first observation from this lemma is that the noise that achieves the least upper bound on privacy is a Gaussian noise process. We also observe that the equality is achieved when \(X = X_G\) and \(Z = Z_G\). Thus, this upper bound on mutual information is tight. This result is very useful both in constructing the noise and in reconstructing the community distribution because we only need a very small number of parameters to characterize a Gaussian process. When the perturbation noise is a Gaussian process, the upper bound in Equation (3.1) can be computed as

\[
I(X, Y) \leq I(X_G, Y_G) = \frac{1}{2} \log \frac{\det(K_X + K_Z)}{\det(K_Z)} \tag{3.2}
\]

where \(K_X\) and \(K_Z\) are the covariance matrix of \(X\) and \(Z\), respectively. Since we assume that \(X\) and \(Z\) are both wide sense stationary (WSS) random processes, they are characterized by their covariance matrix. The sizes of the covariance matrices are chosen by the users such that they accurately represent the real signals. For stationary signals, the auto-covariance function decays geometrically. Thus, the sizes of the covariance matrices do not need to be very large. In real applications, \(n \leq 15\) is generally sufficient where \(n\) is the size of the covariance matrices.

The rest of the problem is to find the noise that achieves the least upper bound on information leak. First, we define the privacy achieved when
perturbing the time series \( X \) with covariance \( K_X \) with a noise \( Z \) with covariance matrix \( K_Z \) based on the upper bound on information leak derived in Equation (3.2).

**Definition 2.** The upper bound on the information leak \( \mathcal{P}_X \) for a given data \( X \) with covariance matrix \( K_X \) of size \( n \) perturbed by a Gaussian noise process \( Z \) with covariance matrix \( K_Z \) of size \( n \) is defined as

\[
\mathcal{P}_X = \frac{1}{2n} \log \frac{\det(K_X + K_Z)}{\det(K_Z)}
\]

In this definition, we divide the upper bound on information leak in Equation (3.2) by \( n \) to alleviate the effect of the size of the covariance matrix on the privacy definition. Also, there is no constraint on the noise power level in this definition, since it is already incorporated in \( K_Z \). In fact, the expected noise power can be computed as

\[
E[P_Z] = \frac{1}{n} \text{trace}(K_Z)
\]  

(3.3)

This definition on privacy gives us several insights on the performance of the optimal noise. In our previous work [19], we perturb user data with correlated noise, which has the same model as the original data. According to our definition of privacy, that approach is sub-optimal and achieves the privacy of \( \mathcal{P}_X = \frac{1}{2n} \log \frac{\det(2K_Z)}{\det(K_Z)} = 0.5 \). Furthermore, with the previous approach, the upper bound on information leak does not change with the noise power, making this method not good when high privacy guarantees are needed. Next, we will present an algorithm to find the noise which minimizes the upper bound on privacy.
3.1.2 Construction of the Best Noise

In this section, the noise that minimizes the upper bound on information leak is referred to as the best noise, and the upper bound on the information leak is referred to as information leak for simplicity. To construct the best noise, we first find the best noise’s covariance matrix $K^*_Z$ that minimizes the privacy measure defined in Definition 2. A method to generate the noise model from the covariance matrix will then be presented.

First, the optimization problem for finding the best noise subject to a power constraint is as follows:

$$
K^*_Z = \arg\min_{K_Z} \frac{1}{2n} \log \frac{\det(K_X + K_Z)}{\det(K_Z)}
$$

subject to

$$
\frac{1}{n} \text{trace}(K_Z) \leq P_0
$$
$$
K_Z \succ 0
$$
$$
K_Z \text{ is Symmetric Toeplitz}
$$

The constraint $K_Z \succ 0$ means that $K_Z$ is positive definite since it is the covariance matrix. In previous discussion, we assumed that the noise is a WSS process. Thus, the covariance of the generated noise is a symmetric Toeplitz matrix. The following lemma [26] shows that the objective function of this optimization problem is convex in $K_Z$.

**Lemma 2** (Diggavi and Cover (2001)). The function $f(K_Z) = \frac{1}{2} \log \frac{\det(K_X + K_Z)}{\det(K_Z)}$ is convex in $K_Z$ with strict convex if $K_X \succ 0$.

With the convexity of the objective function, this optimization problem becomes a semidefinite programming problem [27]. There are several off-the-shelf solvers [28, 29] available which can solve a medium scale semidefinite
programming problem effectively. However, the input objective function to those solvers must be an explicit convex function. Since our objective function is not an explicit convex function, an additional step has to be taken to transform our problem into a canonical semidefinite programming problem. The equivalent semidefinite programming problem is presented in the following theorem.

**Theorem 1.** The canonical semidefinite programming problem is given as

\[
K^*_Z = \arg\max_{K_Z} \log \det(t)
\]

subject to

\[
\begin{pmatrix}
K_Z - t & K_Z \\
K_Z & K_X + K_Z
\end{pmatrix} \succ 0
\]

\[
\frac{1}{n} \text{trace}(K_Z) \leq P_0
\]

\[
K_Z \succ 0
\]

\emph{K}_Z is Symmetric Toeplitz

**Proof.** First we transform the objective function:

\[
f(K_Z) = \frac{1}{2n} \log \frac{\det(K_X + K_Z)}{\det(K_Z)}
\]

\[
= \frac{1}{2n} \log \{\det(K_X) \det(K_X^{-1} + K_Z^{-1})\}
\]

\[
= \frac{1}{2n} \log \det(K_X) + \frac{1}{2n} \log \det(K_X^{-1} + K_Z^{-1})
\]

Since \(K_X\) is given, the term involved in \(K_X\) in the objective function can
be omitted. The objective function becomes

\[ f(K_Z) = \log\det(K_X^{-1} + K_Z^{-1}) \]
\[ = -\log\det((K_X^{-1} + K_Z^{-1})^{-1}) \]

Using matrix inversion lemma [30], we obtain

\[ f(K_Z) = -\log\det((K_X^{-1} + K_Z^{-1})^{-1}) \]
\[ = -\log\det(K_Z - K_Z(K_X + K_Z)^{-1}K_Z) \]

Now let \( t = K_Z - K_Z(K_X + K_Z)^{-1}K_Z \); the optimization problem becomes:

\[ K^*_Z = \arg\max_{K_Z} \log\det(t) \]
subject to

\[ t \prec K_Z - K_Z(K_X + K_Z)^{-1}K_Z \]
\[ K_Z \succ 0 \]
\[ \frac{1}{n} \text{trace}(K_Z) \leq P_0 \]
\[ K_Z \succ 0 \]
\[ K_Z \text{ is Symmetric Toeplitz} \]

Using the Schur’s complement theorem [31] yields the final optimization
The problem is:

\[
K^*_Z = \arg\max_{K_Z} \log \det(t)
\]

subject to

\[
\begin{pmatrix}
K_Z - t & K_Z \\
K_Z & K_X + K_Z
\end{pmatrix} \succeq 0
\]

\[
\frac{1}{n} \text{trace}(K_Z) \leq P_0
\]

\[
K_Z \succ 0
\]

The final form of the optimization problem is a canonical semidefinite programming problem since logdet is a concave function.

Having had the covariance matrix for the best noise, we will now construct the best noise model which is used to generate actual noise data point. It has been shown [32] that it is possible to construct a Gauss-Markov random process having a given covariance matrix. The model to generate the best noise is as follows:

\[
Z(k) = \sum_{i=1}^{n} \phi_i Z(k-i) + \epsilon_k
\]

where \( \epsilon_k \)'s are zero mean i.i.d. Gaussian random variables with variance \( \sigma^2 \). It is possible to determine \( \phi_i \) and \( \sigma^2 \) from \( K^*_Z \) by solving a Yule-Walker equation using the Levinson-Durbin iteration algorithm with the time complexity of \( O(n^2) \). The reader is referred to [32] for more details on constructing a Gauss-Markov random process given its covariance matrix.
3.1.3 Time-Series Perturbation

In a real application, users are usually concerned about privacy of real data values, the averages and sometimes the trends. For example, individuals sharing their bank account history usually do not want their transaction values and their average balance to be revealed. In the same manner, individuals on a diet might not want to reveal whether they are gaining weight or not. Thus, it is important for any perturbation algorithm to hide the individual data values, the trends and the average of the time series.

In the previous sections, we presented a noise generation method to optimally perturb an arbitrary time-series. However, applying the algorithm directly to the user data will only hide the data values effectively; the long-term trends and the average might still be breached. To solve this problem, we observe that the trend of a time-series can be estimated by re-sampling the time-series at an appropriately lower sampling frequency. Therefore, we decompose the time-series into the sum of multiple time-series components sampled from the original data at different frequencies. The decomposition can be done by various techniques, for example, by down-sampling the original data or by taking the discrete Fourier transform of the original data and using only part of the coefficients. In this chapter, we propose a simple decomposition method using moving average window. The decomposition is done by first constructing non-overlapping windows of samples with predetermined window size. The trend is formed taking the averages of each window. The residue is formed by subtracting the data points in each window by their average. This process can be done multiple times to decompose the original data into several components, each of which contains the trend on a different time scale. Figure 3.1 demonstrates a time-series being decomposed into the
trend and a high-frequency component.

![Graph showing original data, low frequency component, and high frequency component](image)

Figure 3.1: Time-series decomposition

The perturbation of the original time-series signal is done by perturbing the individual components using our proposed perturbation method explained in Section 3.1.2. The final perturbed signal is formed by combining all the perturbed components. It is not hard to see that this perturbation method is equivalent to perturbing the original signal at multiple frequencies. In addition, users can have different noise powers at different frequencies, making it possible to hide information at different time-scales to different degrees. The choices of the number of components and their associated noise power heavily depend on the type of shared data. In the evaluation section, we empirically choose the values for those parameters. The existence of the optimal values for those parameters is still an open question. Finally, the mean of the original signal can be hidden easily by adding a random value. The community distribution can be reconstructed from the perturbed data and the information about the generated noise models.
3.2 Evaluation

In this section, we discuss the evaluation of our proposed perturbation on both the ability to provide privacy and the performance of the reconstruction of the community distribution. The experiments are conducted on both simulated data and real deployment data. The results show that our proposed method provides the best trade-off between the privacy and the community reconstruction.

3.2.1 Simulated Data

The data used in the evaluation are generated according to an ARMA(1, 1) model described as follows:

\[ x_{k+1} = \phi x_k + \theta \epsilon_k + \epsilon_k \]  

(3.5)

where \( \phi = 0.95 \), \( \theta = 0.3 \) and \( \epsilon_k \)'s are i.i.d. Gaussian random variables with zero mean and covariance \( \sigma = 1 \). With the coefficients chosen above, the generated time series is stable (i.e., satisfies the stability condition of the perturbation algorithm). Using this model, it is possible to derive the exact form of the covariance matrix. To be general, however, in all the experiments, the covariance matrix will be estimated directly from the generated time-series. The size of the estimated covariance matrix is chosen to be 15, and the time step is 0.1 seconds in all the experiments.

In the first experiment, we generate the time series for one user according to Equation (3.5). The signal is then decomposed into the trend and a high frequency component. The trend and its perturbed data are shown...
in Figure 3.2a. The noise power $P_Z$ is chosen to be $1.5 \ P_X$, which clearly seen in the figure. Figure 3.2b shows the user data and the perturbed signal after combining both the perturbed trend and perturbed high frequency component.

![Figure 3.2](image)

(a) Low frequency components  
(b) Combined signals

Figure 3.2: (a) Real and (b) perturbed data

Figure 3.3 shows the power spectral densities of both original data and the generated noise. We can see that the noise power is only allocated at the frequencies where the data power is allocated. Thus, the spectra of both the noise and the original data overlap, making it impossible to separate them. The insight is that, when we generate the noise, we have to solve for its auto-covariance function. However, the power spectral density of a time series is the Fourier transform of its auto-covariance function. Thus, our proposed optimization framework actually solves the problem in the frequency domain. We can view the perturbation problem as an optimal power allocation problem with the utility function set to be the privacy.

Next, we evaluate the privacy guarantee by our proposed perturbation algorithm and compare the results with another perturbation scheme [19] called the Poolview scheme. Figure 3.4 shows the upper bound on information leak at different chosen noise power levels. We observe that the information leak decreases with noise power in our perturbation scheme while it is constant in the other scheme, which implies that our proposed perturbation scheme
Figure 3.3: Power spectral densities of real data and generated “noise” provides better privacy guarantee.

Figure 3.4: Information leak of different perturbation schemes

In the next experiment, we evaluate the real performance of three perturbation schemes: our previously proposed Poolview scheme, and a Gaussian independent noise perturbation scheme. We use the spectral filtering technique [8] to reconstruct the original data from the noisy data. We plot the individual reconstruction error versus normalized noise power in Figure 3.5. The result shows that the Gaussian noise perturbation scheme provides very little privacy. Most of the additive noise is filtered. The Poolview perturbation scheme provides better privacy and our proposed perturbation scheme provides the highest privacy.

To quantify the trade-off between the individual reconstruction error and the community reconstruction error, we further examine the community re-
construction error at different input noise powers in Figure 3.6. The results suggest that there is only a little difference in the community reconstruction error between two perturbation schemes. Thus, our proposed perturbation scheme provides a better trade-off, since it provides higher privacy at the same input noise power and the same reconstruction accuracy.

Figure 3.5: Individual reconstruction error at different noise powers

Since the information leak is measured in bits, it is desired to understand how this measure maps to the real individual reconstruction error. Figure 3.7 shows the individual reconstruction error versus the information leak. We can see that the reconstruction error decays almost exponentially with the information leak.

It is also important to understand the trade-off between the information leak and the accuracy of the community reconstruction error, which is pre-
sent in Figure 3.8. From the figure, it is easy to see that the best trade-off point is around 0.12 bits in the information leak.

3.2.2 Deployment Data

In this section, we perform experiments on real deployment data. For the purpose of understanding vehicular traffic patterns on campus, we collected vehicle speed data from 16 individuals over the course of three months. A range of vehicles were used in our experiments and a total of over 1000 miles were driven by our users. The speed data was collected using OBD-II devices [33] with the sampling intervals of 5 seconds.
In Figure 3.9, both the real speed and the perturbed speed of a user are shown for a short period of time. In this experiment, we add a random Gaussian variable to the perturbed data to hide the original average speed because it could tell people if the user was speeding or not, which is not desired. From the figure, we can see that both the dynamics and the average of the perturbed data are very different from the original data.

![Real Speed and Perturbed Speed](image)

**Figure 3.9:** Real speed and perturbed speed

The power spectral densities (PSD) of both original data and perturbed data for a single user are presented in Figure 3.10. From the figure, the PSD of the perturbed data overlaps the PSD of the original data making it hard to separate the two signals.

![PSD of Real Speed and Perturbed Speed](image)

**Figure 3.10:** PSD of real speed and perturbed speed

In the last experiment, we analyze the trade-off between privacy and community reconstruction accuracy in Figure 3.11. Observe that the reconstruc-
tion error is higher than that in the previous experiment with synthetic data. This is because the number of users in this experiment is lower than the number of users in the previous experiment. However, the shape of the trade-off curve remains similar. It is not hard to see that a good trade-off point would be at 0.2 bit of information leak with the reconstruction error of 0.05.

![Figure 3.11: Trade-off between privacy and community reconstruction accuracy](image)

3.3 Conclusion

In this chapter, theoretical foundations to optimally perturb time-series data are presented. For a given noise power, our proposed algorithm provides the highest privacy while keeping the error of the community reconstruction acceptable. Most of all, our algorithm provides a privacy guarantee (i.e., a bound) which, to the best of the author’s knowledge, is not possible with previous approaches.
Previous chapters presented different privacy techniques for single-dimensional time-series data. In this chapter, we present theoretical foundations, a system implementation, and an experimental evaluation of a perturbation-based mechanism for ensuring privacy of multidimensional time-series data. We focus on privacy for location-tagged participatory sensing data while allowing correct reconstruction of community statistics of interest (computed from shared perturbed data). The system is applied to construct accurate traffic speed maps in a small campus town from shared GPS data of participating vehicles, where the individual vehicles are allowed to “lie” about their actual location and speed at all times. An extensive evaluation demonstrates the efficacy of the approach in concealing multidimensional, correlated, time-series data while allowing for accurate reconstruction of spatial statistics.

4.1 Joint Probability Density Function Reconstruction

The main contribution of this work lies in the algorithm to accurately reconstruct the community joint density given the perturbed multidimensional stream data and the noise density information. Any statistical question about the community can be answered using the reconstructed joint density. In the traffic mapping application, for example, provided that the joint density of the location and speed of the city is known, one can answer questions such
as What is the average speed of a certain street? or What is the percentage of speeding vehicles in the city? There have been many efforts on the community distribution reconstruction. Agrawal and Srikant [6] proposed a Bayesian-based reconstruction of the probability distribution. In [7], the authors use the expectation maximization (EM) algorithm to estimate one-dimensional distribution from data perturbed with Gaussian noise. In [19], Ganti et al. employed the Tikhonov-Miller deconvolution technique to estimate the community distribution. However, all of these algorithms are developed to reconstruct a one-dimensional distribution. Hence, they do not scale to the problem of multidimensional distribution reconstruction. In this section, we present an iterative algorithm to estimate the discretized joint distribution of multidimensional data streams.

Let the number of data streams that each user wants to share be $M$. The shared data from each user are assumed to be drawn from a multivariate random variable $X = (X_1, X_2, \ldots, X_M)$; thus each data point is a length $M$ vector. The reconstruction algorithm does not distinguish which data points are from which user. Therefore, we can define the set of all data points from all users as $\bar{X} = \{x_1, x_2, \ldots, x_n\}$, where $x_i$ is a length $M$ data point and $n$ is the total number of data points from all users.

Each data point is perturbed by adding an $M$-dimensional noise data point generated from a known joint distribution $f_N(N_1, N_2, \ldots, N_M)$ which is known to all participating users (or rather to their client-side software). An aggregation server receives the set of $n$ perturbed data points from all users denoted as $\bar{Y} = \{y_1, y_2, \ldots, y_n\}$. We want to estimate the joint distribution of $X$ which is $f_X(X_1, X_2, \ldots, X_M)$ given the shared data $\bar{Y}$ and the knowledge of the noise distribution $f_N$.

Let us denote the sample space of $X_i$ as $\Omega_i$. Thus, the sample space of $X$
is $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_M$. In order to reconstruct the density of $X$, we first discretize the sample space $\Omega$. The sample space of $X_i$ is partitioned into $K_i$ bins (may not be uniform) denoted as $\{\Omega^1_i, \Omega^2_i, \ldots, \Omega^K_i\}$. Thus $\Omega$ contains $K = K_1 \times K_2 \times \ldots \times K_M$ $M$-dimensional bins in which the value of the density function is constant. The more bins, the better the discrete density approximates the continuous density. To simplify the notation, the following symbols are introduced:

- $\omega_I$: the $I^{th}$ bin of $\Omega$, thus $\Omega = \cup_{\omega_I} \omega_I$.
- $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$: where $\theta_i = f_X(X)$ with $X \in \omega_I$, is the set of all density parameters to be estimated.
- $m_{\omega_I}$: the volume of $\omega_I$, where a proper discrete density $\Theta$ should satisfy

$$\sum_{\omega_I} \theta_I m_{\omega_I} = 1 \quad (4.1)$$

To estimate $\Theta$, our approach is to employ the maximum likelihood framework. We need to find the density function parameters which maximize the log likelihood of the data $\bar{X}$ given the observations $\bar{Y}$

$$\hat{\Theta} = \arg\max_{\Theta} \log f_{X;\Theta}(\bar{X}|\bar{Y}) \quad (4.2)$$

The notation $f_{X;\Theta}$ means that the likelihood of $X$ is computed using the discrete distribution $\Theta$. Unfortunately, the likelihood cannot be computed directly at the aggregation server because only $\bar{Y}$ is known while $\bar{X}$ is missing. In this case, $\bar{X}$ is called the missing information, and $\bar{Y}$ is called the incomplete information. One possible solution to this problem is to search in the whole space of $\Theta$ for each possible value of $X$ to find the pair $X, \Theta$.
which maximizes the likelihood in Equation (4.2). This results in an undesirable algorithm which is exponential in the running time and is impossible to implement in most cases. A common procedure to solve the maximum likelihood estimation with incomplete information is the EM algorithm [34]. The EM algorithm effectively computes the expected value of the likelihood over all possible user data $X$ using an approximated density $\Theta^k$ computed in the previous step. Then the expected value of the likelihood is maximized to find the maximum likelihood solution. The advantages of using the EM algorithm are:

- The computational complexity of the algorithm is lower because it does not search the entire space of $\Theta$ and $X$ to maximize the likelihood in Equation (4.2).

- Under some mild conditions [35], the algorithm is guaranteed to converge to the maximum likelihood solution. Later in this section, we provide a proof that, in this problem, the EM algorithm actually converges to the true maximum likelihood solution.

To use the EM algorithm, the following auxiliary function $Q(\Theta|\hat{\Theta}^k)$ is defined:

$$Q(\Theta|\hat{\Theta}^k) = E_{X|Y} \left[ \log f_{X,\Theta}(\bar{X})|\bar{Y}, \hat{\Theta}^k \right]$$

(4.3)

The auxiliary function $Q$ is actually the expectation of the likelihood in (4.2) with respect to $X$ using the density of $X$ computed from the previous step which is $\hat{\Theta}^k$. The EM algorithm consists of two steps:

- E-step : Given the density computed from the $k^{th}$ step, compute the value of $Q(\Theta|\hat{\Theta}^k)$. 

65
• M-step: Compute $\hat{\Theta}^{k+1} = \arg\max_{\Theta} Q(\Theta, \hat{\Theta}^k)$.

Next, we will derive a closed form expression for $Q$ and the optimal solution which maximizes the likelihood function, and we will analyze the convergence of the algorithm.

**Theorem 2.** (E-step) The value of $Q(\Theta|\hat{\Theta}^k)$ is given by

$$Q(\Theta|\hat{\Theta}^k) = \sum_{\omega_i} \hat{\theta}_{\omega_i}^k \log(\theta_{\omega_i}) \phi_{\omega_i}^k$$  \hspace{1cm} (4.4)

$$\phi_{\omega_i}^k = \frac{1}{N} \sum_{j=1}^{N} f_N(y_j - \omega_I)$$ \hspace{1cm} (4.5)

$$f_{Y;\hat{\Theta}}(y_j) = \sum_{\omega_i} f_N(y_j - \omega_I)\hat{\theta}_{\omega_i}^k$$ \hspace{1cm} (4.6)

$$f_N(y_j - \omega_I) = \int_{\omega_I} f_N(y_j - \gamma) d\gamma$$ \hspace{1cm} (4.7)

**Proof.** We begin with the expansion of the auxiliary function $Q$ by noting that the data points are i.i.d.

$$Q(\Theta|\hat{\Theta}^k) = E_{X|Y} \left[ \log f_{X;\Theta}(|\bar{X})|\bar{Y}, \hat{\Theta}^k \right]$$

$$= E_{X|Y} \left[ \log \prod_{j=1}^{N} f_{X;\Theta}(x_j)|y_j, \hat{\Theta}^k \right]$$

$$= \sum_{j=1}^{N} E_{X|Y} \left[ \log f_{X;\Theta}(x_j)|y_j, \hat{\Theta}^k \right]$$

$$= \sum_{j=1}^{N} \int_{\Omega} \log f_{X;\Theta}(\gamma) f_{X|Y;\hat{\Theta}^k}(\gamma|y_j) d\gamma$$

In the last step, the expectation is taken over all possible values of $X$ given the observation $y_i$. We further expand the auxiliary function $Q$ using Bayes' formula and the fact that $f_{Y|X}(Y|X) = f_N(Y - X)$ because $N = Y - X$. 
\[ Q(\Theta|\hat{\Theta}^k) = \sum_{j=1}^{N} \int_{\Omega} \log f_{X;\Theta}(\gamma) \frac{f_{X,Y;\hat{\Theta}^k}(\gamma, y_j)}{f_{Y;\hat{\Theta}^k}(y_j)} d\gamma = \sum_{j=1}^{N} \frac{1}{f_{Y;\hat{\Theta}^k}(y_j)} \int_{\Omega} \log f_{X;\Theta}(\gamma)f_{X;\hat{\Theta}^k}(\gamma)f_{N}(y_j - \gamma) d\gamma \\
= \sum_{j=1}^{N} \frac{1}{f_{Y;\hat{\Theta}^k}(y_j)} \sum_{\omega_I} \int_{\omega_I} \log(\theta_{\omega_I}) \hat{\theta}_{\omega_I}^k f_{N}(y_j - \gamma) d\gamma \]

In the last equation, the integral over the \(\Omega\) is discretized and is computed as the sum of the integral over all subspaces \(\omega_I\) in which the value of the discrete density function is constant. Also, the value of \(f_{Y;\hat{\Theta}^k}(y_j)\) is computed as follows:

\[ f_{Y;\hat{\Theta}^k}(y_j) = \int_{\Omega} f_{Y}(y_j|x)f_{X;\hat{\Theta}^k}(x) dx = \sum_{\omega_I} \int_{\omega_I} f_{N}(y_j - x) \hat{\theta}_{\omega_I}^k dx = \sum_{\omega_I} f_{N}(y_j - \omega_I) \hat{\theta}_{\omega_I}^k \quad (4.8) \]

\[ Q(\Theta|\hat{\Theta}^k) = \sum_{j=1}^{N} \frac{1}{f_{Y;\hat{\Theta}^k}(y_j)} \sum_{\omega_I} \hat{\theta}_{\omega_I}^k \log(\theta_{\omega_I}) \int_{\omega_I} f_{N}(y_j - \gamma) d\gamma = \sum_{j=1}^{N} \frac{1}{f_{Y;\hat{\Theta}^k}(y_j)} \sum_{\omega_I} \hat{\theta}_{\omega_I}^k \log(\theta_{\omega_I}) f_{N}(y_j - \omega_I) = \sum_{\omega_I} \hat{\theta}_{\omega_I}^k \log(\theta_{\omega_I}) \phi_{\omega_I}^k \]

\(\square\)
Theorem 3. (M-step) The value of $\hat{\Theta}^{k+1}$ maximizing the auxiliary function $Q(\Theta|\hat{\Theta}^k)$ is given by

$$\hat{\theta}^{k+1}_{\omega_i} = \frac{\phi^k_{\omega_i}}{m_{\omega_i}} \hat{\theta}^k_{\omega_i} \quad (4.9)$$

Proof. This is an optimization problem with a constraint which ensures that $\Theta$ is a proper density function.

$$\hat{\Theta}^{k+1} = \text{argmax}_{\Theta} Q(\Theta|\hat{\Theta}^k)$$

$$\sum_{\omega_i} \theta_{\omega_i} m_{\omega_i} - 1 = 0$$

The Lagrangian of the optimization is given by

$$L(\theta_{\omega_i}, \lambda) = Q(\Theta|\hat{\Theta}^k) + \lambda (\sum_{\omega_i} \theta_{\omega_i} m_{\omega_i} - 1)$$

$$= \sum_{\omega_i} \hat{\theta}^k_{\omega_i} \log(\theta_{\omega_i}) \phi^k_{\omega_i} + \lambda (\sum_{\omega_i} \theta_{\omega_i} m_{\omega_i} - 1)$$

The optimized values $\hat{\theta}^{k+1}_{\omega_i}$ satisfy $\frac{\partial L}{\partial \theta_{\omega_i}}(\hat{\theta}^{k+1}_{\omega_i}) = 0$ and $\frac{\partial L}{\partial \lambda}(\hat{\theta}^{k+1}_{\omega_i}) = 0$:

$$\frac{\partial L}{\partial \theta_{\omega_i}}(\hat{\theta}^{k+1}_{\omega_i}) = \hat{\theta}^{k+1}_{\omega_i} \phi^k_{\omega_i} + \lambda m_{\omega_i} \quad (4.10)$$

$$\frac{\partial L}{\partial \lambda}(\hat{\theta}^{k+1}_{\omega_i}) = \sum_{\omega_i} \hat{\theta}^{k+1}_{\omega_i} m_{\omega_i} - 1 \quad (4.11)$$

Setting Equation (4.10) to zero yields

$$\hat{\theta}^{k+1}_{\omega_i} = -\frac{1}{\lambda m_{\omega_i}} \hat{\theta}^k_{\omega_i} \phi^k_{\omega_i} \quad (4.12)$$
Substituting (4.12) into (4.11) and setting it to zero yields

\[ \lambda = -\sum_{\omega_l} \hat{\theta}_{\omega_l}^k \phi_{\omega_l}^k \]

\[ = -\sum_{\omega_l} \hat{\theta}_{\omega_l}^k \frac{1}{N} \sum_{j=1}^{N} \frac{f_N(y_j - \omega_l)}{f_{Y;\hat{\Theta}^k}(y_j)} \]

\[ = -\frac{1}{N} \sum_{j=1}^{N} \frac{1}{f_{Y;\hat{\Theta}^k}(y_j)} \sum_{\omega_l} \hat{\theta}_{\omega_l}^k f_N(y_j - \omega_l) \] (4.13)

Since \( Y = X + N \), the density of \( Y \) is the convolution of the density of \( X \) and \( N \). Therefore,

\[ f_{Y;\hat{\Theta}^k}(y_j) = \int_{\Omega} f_{X;\hat{\Theta}^k}(x) f_N(y_j - x) dx \]

\[ = \sum_{\omega_l} f_{X;\hat{\Theta}^k}(x) f_N(y_j - x) \]

\[ = \sum_{\omega_l} \hat{\theta}_{\omega_l}^k \int_{\omega_l} f_N(y_j - x) dx \]

\[ = \sum_{\omega_l} \hat{\theta}_{\omega_l}^k f_N(y_j - \omega_l) \] (4.14)

Substituting (4.14) into (4.13) yields \( \lambda = -1 \). Therefore,

\[ \hat{\theta}_{\omega_l}^{k+1} = \frac{\phi_{\omega_l}^k}{m_{\omega_l}} \hat{\theta}_{\omega_l}^k \]

\( \square \)

In the next theorem, we show that the EM algorithm for this problem is guaranteed to converge to the maximum likelihood solution which is the solution for (4.2). Therefore the likelihood value increases slowly as it approaches the optimal solution. A stopping condition for the algorithm is when the likelihood difference between two consecutive steps is sufficiently small. The pseudo-code for the EM-based reconstruction algorithm is as
Algorithm 1 Multidimensional Density Reconstruction

Input: Perturbed data points $y_i$, Noise distribution $f_N$
Initialize $\Theta = \Theta^0$, $L^0 = 0$, $k = 0$, $\epsilon > 0$
repeat
  for all $\omega_I \in \Omega$ do
    $\phi_{\omega_I}^k = \frac{1}{N} \sum_{j=1}^N \frac{f_N(y_j - \omega_I)}{\sum_{\omega} f_N(y_j - \omega) \hat{\theta}_{\omega}^k}$
    $\theta_{\omega_I}^{k+1} = \frac{\phi_{\omega_I}^k \theta_{\omega_I}^k}{m_{\omega_I}}$
  end for
  $L^{k+1} = Q(\Theta^{k+1} | \Theta^k)$ \{Using Equation 4.4.\}
  $k = k + 1$
until $L^k - L^{k-1} < \epsilon$
return Estimated Density $\Theta^k$

Theorem 4. The estimated density function given by the algorithm converges to the maximum likelihood solution $\hat{\Theta}$ defined in the Equation (4.2).

Proof. We will first prove that $Q(\Theta | \hat{\Theta}^k)$ is concave in $\theta_{\omega_I}$. In Theorem 2, we prove that the value of the auxiliary function $Q(\Theta | \hat{\Theta}^k) = \sum_{\omega_I} \hat{\theta}_{\omega_I}^k \log(\theta_{\omega_I}) \phi_{\omega_I}^k$, which is the non-negative linear combination of $\log(\theta_{\omega_I})$. Since $\log(x)$ is concave in $x$, the non-negative linear combination of $\log(x)$ functions is also concave. Thus $Q$ is concave in $\theta_{\omega_I}$.

Wu [35] showed that the value of the likelihood increases after each iteration. Because $Q$ is concave, the iterative algorithm will finally converge to $\hat{\Theta}$ which maximizes the likelihood function defined in (4.2). \qed

From Theorem 4, it follows that the EM based estimation algorithm for this problem inherits all properties of the maximum likelihood estimation. Since the maximum likelihood estimation is asymptotically efficient [36], the proposed algorithm is suited for participatory sensing applications which usually involve large amounts of data.
4.2 Perturbation of Location and Data

Having presented a general algorithm for reconstruction of community statistics, it remains to decide on the perturbation function. This question is equivalent to choosing the noise probability density function, $f_N()$, from which noise samples are chosen. Perturbation is application specific, since it depends on what is being perturbed. We consider the class of applications where we perturb location-tagged data collected by vehicles. As a means to experiment with privacy issues involving location-tagged data, we developed a participatory sensing application for traffic mapping, which allows users to share their GPS data (longitude, latitude, and speed) in a privacy-preserving manner while ensuring that community statistics (e.g., average speed, traffic density maps) can be accurately computed. The traffic mapping application successfully employs the multi-stream perturbation and reconstruction framework developed in this chapter. Our application is made possible by the growing number of deployed GPS devices that provide location and speed information.

In our application, individuals collect GPS longitude, GPS latitude, speed and (coarsely discretized) time, using their own GPS devices. Once the aggregation server receives perturbed data from participants, the community joint density (i.e., the joint density of longitude, latitude and speed) is reconstructed using the above reconstruction algorithm. Speed-related statistics are then computed as a function of location on the map from the reconstructed joint density. In this chapter, we present useful community statistics that can be computed from the estimated multidimensional density such as community average speed, speed distribution, car density, and percentage of speeding vehicles on different streets.
The application was deployed on top of PoolView [19], an existing architecture for participatory sensing. PoolView is a generic client-server based architecture that enables individuals to collect, archive, and share sensor data with a community on the client side; PoolView provides software that collects sensor data from specific devices (e.g., Garmin GPS). We modified the PoolView client to use our new multidimensional data perturbation scheme. On the server side, we implemented the multidimensional density reconstruction algorithm and the algorithms used to estimate the aforementioned statistics.

4.2.1 The Perturbation Model

In this section, we propose an algorithm that generates fake (but realistic-looking) vehicle traces that perturb true user location and speed in a way that protects them from being estimated. The vehicle traces are recorded as displacements from an origin (of a coordinate framework) that lies at some agreed upon point in the city in question. These displacements, which we henceforth call *perturbation traces*, will then be added to real routes to generate perturbed routes. There have been many research efforts on generating vehicle traces [37, 38, 39, 40], and we can utilize one of those models to generate perturbation traces for our application. However, the vehicle traces used for perturbation do not need that level of accuracy. Thus, we develop a simplified model that generates perturbation traces using a minimal number of simple parameters.

It is key that the perturbation traces generated resemble real traces for the city in question. For example, in a city with a lot of curvy roads, generated perturbation traces containing only straight segments will not help conceal the identifying characteristics of the roads actually traveled. A ro-
bust perturbation trace generation algorithm must therefore incorporate as many features of the actual map as possible.

Our perturbation trace generation algorithm generates traffic routes made of sequences of straight line segments, each of a length drawn from the distribution of the lengths of city blocks. These segments are at angles generated from the distribution of city street intersection angles. This distribution heavily favors $0^\circ$ angles (continuing forward past an intersection) and $90^\circ$ turns. Other angles are generated with lower probability. We ignore U-turns because they occur with a very small probability. For speed, we use a sine curve for each road segment that peaks in the middle of the segment and slows down towards the beginning and end. The peak is drawn from the distribution of city street speed limits. The slowest point is a uniformly-distributed random fraction of the peak. These traces represent displacement to actual routes. This displacement can be scaled to control the noise variance.

![Figure 4.1: A perturbation trace generated by our algorithm](image)

In the following example, we demonstrate the generation of a vehicular perturbation trace and the scaling of the trace to achieve the desired variance.
Figure 4.1 shows a perturbation trace containing 40 points generated by the above algorithm. It is then added to a user data to form the perturbed path. The perturbed path thus generated is very similar to the path of a real vehicle, but is in fact fake. The real vehicle trace and perturbed vehicle trace are plotted in Figure 4.2a. Obviously, the path no longer conforms to a map, but the lack of individual street-identifying features in the perturbed trace makes it hard to infer the original trace without high uncertainty. Figure 4.2b shows the perturbed path when the perturbation trace is scaled by a factor of two.

Figure 4.2: Real path and perturbed path (a) before and (b) after scaling

Figure 4.3 plots the real speed, generated speed (the perturbation) and perturbed speed over time for a random vehicle. We can see that the generated speed curve has a similar rate of change as the real speed curve but higher values. Thus the spectrum of both curves overlaps, making it impossible to separate real speed from the perturbed speed.

Finally, for the purpose of reconstructing the community joint distribution, we need the joint distribution of the generated perturbation trace (the noise). Since it is hard to come up with an analytic solution for the joint distribution of the noise, we generate this distribution numerically. First, we generate a pool of noise data points from the model; then a non-parametric density
estimation with smoothing [41] is employed to estimate the joint distribution. In this application, 5000 vehicle traces, each of which contains 40 data points, are generated and used as input to the density estimation algorithm, which generates the joint distribution. The density estimation is a very popular technique in machine learning and can be used to estimate the density of any noise model in general.

4.2.2 Achieved Privacy

In this section, we analyze the extent of privacy offered to individual user data using our perturbation scheme. The information available to the aggregation server includes the perturbed data, the noise density function (known by the server) and the map on which the user traveled. First, note that the reconstruction algorithm proposed in this chapter cannot be used to reconstruct an individual’s real data from this information. Our proposed algorithm can only reconstruct community distribution from shared data of a reasonable number of participants. Using the available information, the malicious server can employ filtering techniques to remove additive noise from
the perturbed data. We call this kind of attack a *filtering attack*.

In this chapter, we analyze a filtering attack which applies a Wiener filter to remove additive noise from perturbed data. The Wiener filter uses the noise density information to filter the noise from perturbed data. One important assumption that the Wiener filter makes is that the noise samples are independent. However, this assumption fails because the noise samples generated by our algorithm are correlated, which makes the estimated data traces follow the perturbed path instead of the real path. For demonstration, we perturb a real user location trace with both correlated noise generated by our algorithm and independent Gaussian white noise and then perform the Wiener filter on both perturbed data set.

The result of the Wiener attack in the case of Gaussian white noise is shown in Figure 4.4a. The reconstructed path is very close to the real path and the reconstruction error is less than one block, which means that the attacker can easily figure out the place where the user has been. Figure 4.4b shows the real path, perturbed path and reconstructed path for the perturbation technique we developed in this chapter. We see that the reconstructed path follows the perturbed path. Therefore, the Wiener filter attack does not work as desired for the attacker. Users might want to increase the variance of the generated noise to get more privacy, but the reconstruction error might increase as well. Therefore, it is important to balance the trade-off between privacy and accuracy.

The second type of attack considered in this chapter is the range attack. It is possible to conduct the range attack in applications where the ranges of both the real data and the generated noise are finite. In this case, real data values can be inferred if boundary values of the perturbed data are observed. For example, suppose the real speed of a vehicle is in the range [0 to 50] and
the generated noise is also in the range \([0 \text{ to } 50]\). If the perturbed speed is 100, the attacker knows with certainty that the true speed is 50. In general, if the perturbed values are close to the boundary, privacy can be violated. In applications involving GPS location as a private variable, however, this attack is not effective. GPS location refers to a point of the globe. Perturbing that location by a few miles is sufficient for privacy, yet the perturbed location still refers to a point on the globe. In other words, the perturbed coordinates always refer to a valid data point. An exception is when map information is used to infer noise. For example, at coastal areas, one may safely assume that vehicles do not move on water, which generates a boundary on valid locations. The map-based attack will be discussed shortly. In general, the effect of range-based attacks can be mitigated if the noise distribution has a long tail such that arbitrarily large values are allowed with an arbitrarily low probability. (Many distributions, including Gaussian, have this property.) In this case, the range is infinite. There is no maximum value for the perturbed signal that can be used to breach privacy.

Another popular type of attack against additive-noise perturbation techniques is the *leak attack* [42]. In this type of attack, the attacker may be able
to estimate the seed of the pseudo random number generator which generates the noise curve if he can guess a few true data values. Then this seed can be used to generate the noise curve used by the user since the noise distribution is known. However, with our perturbation scheme, this attack is not possible because we only use the random number generator to generate the model parameters (e.g., number of turns, speed of each segment). The additive noise is then generated using those parameters and the model developed earlier in this section.

A vulnerability of our perturbation scheme is that it is possible to combine the real map with a clever estimation technique to estimate the most likely traveled path. We call this attack scheme a map-based attack. At this moment, it is still unknown if there exists a good map-based attack against our perturbation scheme. In this chapter, we argue that finding an efficient map-based attack is hard. One possible way to conduct the map-based attack is to look at the sequence of the turning angles in the GPS trajectory data. Since the probability that the noise angle and the real angle cancel out is pretty small, the turning angles from the perturbed data contain some information about the real turning angles. Combining with the map, it is possible to find the most probable traveled path. It is not easy, however, to find the likelihood of the real turning angle given the perturbed path. Because the perturbed path is created by adding the coordinates of the real path and the noise path, the angle in the perturbed path is dependent not only on the angle of both the real and noise paths but also on their magnitudes. A demonstration is given in Figure 4.5. In the figure, \(X_1, X_2, X_3\) represent the real path, \(Z_1, Z_2, Z_3\) represent the noise path and \(Y_1, Y_2, Y_3\) are the perturbed path. Although the real path is straight and the noise contains a right angle, the perturbed path contains an acute angle. It is not hard to
see that the resulting angle depends on both the angles and the magnitudes. In the upcoming sections, we only evaluate the immunity of our perturbation scheme against the filtering attacks.

![Figure 4.5: Addition of real path and noise path](image)

**Figure 4.5: Addition of real path and noise path**

### 4.3 Simulation Results

In this section, we evaluate the performance of the traffic mapping application with simulated data. The advantage of using simulated data is to give total control over traffic parameters (e.g., average community speed, speed map), which is hard to accurately measure in a real application. In addition, vehicular traces can be generated for a large number of “virtual” users, making it possible to evaluate the accuracy of the reconstruction algorithms. We also evaluate the accuracy computation of the community average speed using the reconstructed density in this section.

We use the ONE (Opportunistic Network Environment) [43] simulator to generate artificial traces of vehicle movements in a small city setup. The map used in this simulation is a part of Helsinki and is distributed with the ONE simulator. The simulator supports map based movement models that can import map data and constrain vehicle movement to the streets and roads.
of the imported map. Once the map data are loaded by the simulator, the mobility works as follows. Each vehicle starts at a random position on the map and picks a random destination to visit. The shortest path from the source to the destination on the map is computed, and the vehicle moves along that path with a speed (for each path segment) sampled from a pre-specified random distribution. After the destination has been reached, the vehicle waits for a random \textit{wait time} and then moves to another random destination.

Our goal is to make the data from the simulator as realistic as possible. The input map for the simulator is extracted from a real map and is shown in Figure 4.6 with the X and Y coordinates ranging from 0 to 4000 meters and 0 to 3600 meters respectively. Vehicle speeds are chosen to be Gaussian with mean 30 mph and standard deviation of 10 mph. The simulator generates at most 25 vehicles at a time, chooses the origin and destination, plans the shortest route and simulates the behavior of the vehicles along the route. When a vehicle reaches its destination, a new origin, destination pair for the vehicle is chosen. Trip data, including X and Y coordinates and vehicle speed, are sampled at a frequency of 1 Hz, and are stored in an external file for later use. The simulated data are then perturbed with perturbation traces generated by the algorithm discussed in Section 4.2.1. The perturbed data are then submitted to the aggregation server.

We collect data from 120 users, each of which contains 80 data points, from the simulation. In order to reconstruct the community joint distribution, we first have to specify the range of each dimension and the number of bins in each dimension. Those parameters are summarized in Table 4.1. In this simulation, we discretize the location in 100 m × 100 m bins which is fine enough to capture the street information. For more accurate reconstruction
of the joint density, more bins in each dimension might be needed but it would require more user data points and computational time. In this specific traffic application, we are only interested in the density values corresponding to the street locations. Our proposed algorithm allows us to do the reconstruction on those bins only, thus significantly reducing the time complexity of the algorithm.

Table 4.1: Parameters for the reconstruction

<table>
<thead>
<tr>
<th>Parameter Value</th>
<th>range of X (0 - 4000 m)</th>
<th>range of Y (0 - 3600 m)</th>
<th>range of V (0 - 60 mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Value</td>
<td>X bins</td>
<td>Y bins</td>
<td>V bins</td>
</tr>
<tr>
<td>Value</td>
<td>40</td>
<td>36</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 4.2: Noise variance in each data set

<table>
<thead>
<tr>
<th>Parameter</th>
<th>stddev of X (m)</th>
<th>stddev of Y (m)</th>
<th>stddev of V (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>100</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>500</td>
<td>500</td>
<td>36</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>900</td>
<td>900</td>
<td>60</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>1500</td>
<td>1500</td>
<td>76</td>
</tr>
<tr>
<td>Dataset 5</td>
<td>3000</td>
<td>3000</td>
<td>100</td>
</tr>
</tbody>
</table>

In the first experiment, we study the accuracy of the density reconstruction algorithm under various noise variance. The application must achieve high
reconstruction accuracy at a reasonably high noise variance level in order to provide sufficient privacy to users. To achieve this goal, we perturbed the simulation data using five different noise variances shown in Table 4.2.

We define the accuracy of the density reconstruction as a function of the average accuracy of all the bins:

$$r = \frac{1}{K} \sum_{i=1}^{K} \left( 1 - \frac{|\theta_i - \hat{\theta}_i|}{\theta_i} \right)$$

(4.15)

In Equation (4.15), $r$ is the computed accuracy, $\theta_i$ is the true discrete density parameter, $\hat{\theta}_i$ is the estimated density parameter. $\hat{\theta}_i$ is obtained by feeding the real density using real user data points to the density estimation algorithm.

The accuracies of the reconstructions as the function of the number of data points and noise variance are shown in Figure 4.7. The figure shows five different curves corresponding to the five datasets described above. The X axis is the number of data points, which varies from 120 points to 1200 points with 120-point increments. In the results, Dataset 1 achieves highest accuracy while Dataset 5 achieves lowest accuracy.

Next, we evaluate the achieved privacy for each dataset presented in Table 4.2. We assume that the attacker uses a Wiener filter to estimate vehicle traces of *individuals* from perturbed data and the noise distribution. Besides correlated noise, trip data are also perturbed with Gaussian noise with the same standard deviation for comparison purpose. We perform the estimation on the perturbed vehicle trace of all users and compute the average reconstruction error which is presented in Table 4.3 below.

From Table 4.3, the reconstruction error for the vehicle traces perturbed with correlated noise is very high as opposed to the Gaussian case in which
Figure 4.7: Percentage reconstruction accuracy as a function of number of data points and noise variance

### Table 4.3: Reconstruction error of individual data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Correlated Noise (m)</th>
<th>Gaussian Noise (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>334.5</td>
<td>145.0</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>1329.5</td>
<td>153.4</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>1942.4</td>
<td>189.8</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>3573.6</td>
<td>218.1</td>
</tr>
<tr>
<td>Dataset 5</td>
<td>4901.1</td>
<td>223.5</td>
</tr>
</tbody>
</table>

the error is small. With Dataset 1 (the noise covariance is small) the reconstruction of individual data is still high (about 3 blocks) which means good privacy is achieved. Also, with Dataset 5, although the reconstruction error of individual data is huge (about 40 blocks), the community distribution can still be accurately reconstructed (above 96%).

In the last experiment, we demonstrate the estimation of the community average speed using the joint distribution estimated in the first experiment. In addition, we also want to study the effect of the number of iterations on the accuracy of reconstruction. To compute the community average speed from the community joint distribution $f(X, Y, V)$, we first compute the speed
density $f(v)$

$$f(v) = \sum_{x=1}^{40} \sum_{y=1}^{36} f(x, y, v) \Delta_{XY}$$

Equation (4.16) is the marginalization of the discrete joint density over $X$ and $Y$ dimensions, where $\Delta_{XY} = (4000/40) \times (3600/36)$ is the area of a two dimensional bin $XY$. Then the average speed $\bar{v}$ is computed as $\bar{v} = \sum_{v=1}^{60} vf(v)$.

The result of the experiment is shown in Figure 4.8. Although Dataset 5 provides users with the highest privacy, the reconstructed average speed is still close to the true value. Another important observation from the graph is that the density reconstruction algorithm requires a very small number of iterations to converge. Results from 5 datasets show that 10 to 15 iterations are sufficient. The accuracy of the algorithm almost does not change after 20 iterations. In the next section, we evaluate the performance of the application using deployment data.

![Figure 4.8: Community average speed versus number of iterations](image)

Figure 4.8: Community average speed versus number of iterations
4.4 Deployment Data

In this section, we evaluate the traffic monitoring application with real deployment data. The data are collected by driving on all the streets within an area shown in Figure 4.9. There are a total of 15 users; each user drives the streets at will for 10 minutes. During the drive, we use a Garmin Legend [24] GPS device to record location and speed information. The sampling frequency of the device is 15 Hz which is enough to record changes in the location and speed since the speed limit in the area is 25 mph.

![Figure 4.9: Map used to collect data](image)

At the aggregation server side, to do the reconstruction, we need to specify the reconstructed region and the number of bins in each region. The reconstruction parameters are summarized in Table 4.4. For location, we divide each axis into 30 bins, each of width 0.01 mile, which is about the width of a street. This is important because we want to estimate the speed down to the resolution of a street. This can be done by looking at the specific bins corresponding to the target street.

In the first experiment, we study the density reconstruction accuracy as a
Table 4.4: Parameters for the reconstruction

<table>
<thead>
<tr>
<th>Parameter</th>
<th>range of X (1/100 mile)</th>
<th>range of Y (1/100 mile)</th>
<th>range of V (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0 - 300</td>
<td>0 - 300</td>
<td>0 - 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>X bins</th>
<th>Y bins</th>
<th>V bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

function of the number of data points used for reconstruction. We want to answer the question of how many data points we need to achieve a desired accuracy. Similar to the case of simulation data, we do the perturbation of the data with five different noise data sets, each of which has a different variance. The details of the noise datasets are presented in Table 4.5. The standard deviation of the noise specified in the table is comparable to multiples of the block length (about 75/100 mile), We run the density reconstruction algorithm multiple times, each time with a different number of data points. The data points are randomly picked from the total pool of data points contributed by all users. The number of data points taken for reconstruction is varied from 100 to 800.

Table 4.5: Noise standard deviation in each dataset

<table>
<thead>
<tr>
<th>Parameter</th>
<th>stddev of X (1/100 mile)</th>
<th>stddev of Y (1/100 mile)</th>
<th>stddev of V (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>45</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>75</td>
<td>75</td>
<td>10</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>100</td>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>150</td>
<td>150</td>
<td>20</td>
</tr>
<tr>
<td>Dataset 5</td>
<td>300</td>
<td>300</td>
<td>30</td>
</tr>
</tbody>
</table>

The results of the experiment are shown in Figure 4.10. From the result, the highest accuracy achieved is about 90% at about 800 datapoints while the lowest accuracy is about 83% at about 160 datapoints. The number of data points needed for a good estimate is thus surprisingly low. This can
be explained by the observation that since the data points are uniformly picked from the pool, there is a high chance that they scatter all over the map, thus capturing the speed information of the whole area. This makes the application practical in most city areas.

![Figure 4.10: Accuracy of the density reconstruction](image)

In the next experiment, we demonstrate the estimation of the community speed distribution. This community speed distribution can be useful in determining the average speed in the area or computing the percentage of speeding vehicles in that area. To compute the community speed distribution \( f(v) \), we marginalize the estimated discrete joint distribution \( f(x, y, v) \) as follows:

\[
f(v) = \sum_{x=1}^{30} \sum_{y=1}^{30} f(x, y, v) \Delta_{XY}
\]

(4.17)

where \( \Delta_{XY} = (300/30) \times (300/30) \) is the area of a two-dimensional bin in \( XY \) dimension. Figure 4.11a and 4.11b show the real community speed distribution and the estimated community speed distribution, respectively. We see that the two speed distributions are similar except for the first bin corresponding to zero speed. This can be explained because the density
estimation algorithm tends to produce a smooth distribution. Thus, the speed value of the bin is smoothed out. The percentage of speeding vehicles in the community can be computed as the sum of bins with greater than 25 miles/hr speed. In this case the real community percentage of speeding is about 7% while the estimated percentage of speeding is 10%, which is a good estimate.

![Real community speed distribution](a) \(\text{b) Reconstructed community speed distribution}\)

Figure 4.11: Real and reconstructed speed distribution

The advantage of our proposed reconstruction algorithm is that the information about the speed at any location or any street can be estimated from the joint distribution. In the next experiment, we show an example of reconstructing the speed distribution on two main streets, University Avenue and Washington Street. Also, the percentage of speeding vehicles on four main streets — University Avenue, Washington Street, Elm street and Neil street — will be computed from perturbed data. First, to compute the speed distribution on a certain street, we need to marginalize the estimated joint distribution along the street coordinates; then the resulting distribution is normalized to achieve a proper probability distribution. As an example, we compute the speed distribution of University Avenue. From the map, we know that University Avenue is on the X axis with \(0 \leq x \leq 300, y = 0\). In the discrete distribution, corresponding bins in the X dimension are \(1 \leq x_b \leq 30\)
and the bin corresponding to $y = 0$ is $y_b = 1$. Thus the speed distribution of University Avenue can be computed as

$$f(v) = \frac{1}{\sum_{x=1}^{30} \sum_{v=1}^{30} f(x, 1, v)} \sum_{x=1}^{30} f(x, 1, v)$$

(4.18)

The real speed distribution and estimated speed distribution of University Avenue and Washington Street are shown in Figure 4.12. We can see that the reconstructed speed distributions for both streets are very close to the real speed distributions.

![Figure 4.12: Real and reconstructed speed distribution](image)

Finally, we compute the percentage of speeding vehicles on all four main streets on the drive map. The percentage of speeding vehicles is computed as the tail probability of speed distribution of each street. Very accurate results are shown in Table 4.6. From the results, an interesting behavior of city traffic
is that the drivers tend to go over the speed limit more on large streets (Neil Street, University Ave) and less on small streets (Elm Street, Washington Street). The interesting part about this statistic is that it was generated entirely from perturbed data. Both the speed and location of individual drivers were perturbed. Yet, very accurate estimates were computed of the percentage of speeding vehicles at the actual locations.

Table 4.6: Percentage of speeding vehicles

<table>
<thead>
<tr>
<th>Street</th>
<th>Real %</th>
<th>Reconstructed %</th>
</tr>
</thead>
<tbody>
<tr>
<td>University Ave</td>
<td>15.60</td>
<td>17.89</td>
</tr>
<tr>
<td>Neil Street</td>
<td>21.43</td>
<td>23.67</td>
</tr>
<tr>
<td>Washington Street</td>
<td>0.5</td>
<td>0.15</td>
</tr>
<tr>
<td>Elm Street</td>
<td>6.95</td>
<td>8.6</td>
</tr>
</tbody>
</table>

4.5 Conclusion

In this chapter, we present theoretical foundations for perturbation based mechanisms for ensuring privacy while allowing correct reconstruction of community statistics of interest. Previous data perturbation techniques fail to ensure either privacy or correct reconstruction of community statistics in the case of correlated multidimensional time-series data. The algorithms proposed in this work allow participants to add noise to multiple correlated data streams prior to sharing in a privacy-preserved way while making sure that relevant community statistics are still reconstructible. A participatory sensing application for traffic monitoring is developed which allows participants to “lie” about their actual location and speed, while letting the community estimate useful traffic statistics (e.g., speed map, percentage of speeding vehicle, etc.) with high accuracy.
In Chapter 4, we show that correlated noise can be used to perturb multidimensional data and propose an iterative algorithm to reconstruct the joint distribution of the community from the perturbed data streams. In this chapter, we propose a solution to the problem of finding the optimal perturbation noise for multidimensional data as an extension to the problem proposed in Chapter 3. We use the same privacy metric described in Chapter 3 to measure the privacy for multidimensional data and derive a framework to find the optimal correlated noise that minimizes the privacy measure.

5.1 Introduction

As discussed in the previous chapter, the need of sharing multidimensional data streams arises in many applications. For example, in traffic analysis, people share their location and speed over time to compute the speed map of a particular area. It is essential to provide privacy for both speed and location while allowing the speed map to be accurately computed. In Chapter 4, we prove that multidimensional correlated noise can be used to perturb the data before sharing while allowing the community statistics to be reconstructed with accuracy. However, the level of privacy provided by the proposed algorithm is not discussed. In addition, to the extent of the author’s knowledge, there is no previous work that fully solves the problem of
privacy of multidimensional time-series data.

In this chapter, we extend the framework for finding optimal perturbation noise for single-dimensional time-series data to finding the optimal perturbation noise for multidimensional time series. We show that the mutual information between original data and perturbed data is still a good privacy measure for multidimensional time-series data. Furthermore, we extend the optimization framework in Chapter 3 to find the optimal noise that minimize the privacy measure.

One of the challenges of extending the framework for single-dimensional time series to multidimensional time series is how mutual information is computed between multidimensional streams. Many works [44, 45] have analyzed mutual information for multidimensional time series. The results, however, are still complicated and hard to apply in reality. In this work, we solve that problem by first proving that the optimal perturbation noise is an n-dimensional Gaussian process. Furthermore, the privacy for a multidimensional time-series is upper bounded by the mutual information of two multidimensional Gaussian random processes, which is trivial to compute.

The rest of this chapter is organized as follows. Section 5.2 introduces the problem and provides a framework for finding an optimal solution to the problem. Next, Section 5.3 evaluates the proposed algorithms on both synthetic data and real data. Finally, Section 5.4 concludes the chapter.

5.2 Problem Formulation and Solution

In this section, we will discuss in depth the problem of finding optimal perturbation noise for multidimensional time-series data and propose an optimization framework for finding the optimal noise.
5.2.1 Problem Formulation

Let $X_1, X_2, \ldots, X_n$ be $n$ original data streams from each user which they want to share. Our goal is to find $n$ optimal noise streams $Z_1, Z_2, \ldots, Z_n$ that will be used to perturb the original data before sharing. Let us denote $Y_1, Y_2, \ldots, Y_n$ be the perturbed data streams ($Y_i = X_i + Z_i$). The length of each data stream can be different in general. Without loss of generality, however, we can assume that they all have the same length since we can always choose the length of the shortest data stream as the representative length of all streams and truncate all data streams to this length.

Following our work in Chapter 3, we assume that all data streams can be modeled as wide-sense stationary (WSS) random processes. Furthermore, each data stream $X_i$ can be modeled (or approximated with high accuracy) as a stationary ARMA($p_i, q_i$) random process [46]. This means each data point in data stream $i$ is a linear combination of $p_i$ data points in the past plus a noise factor. This assumption is important in practice because it limits the size of the covariance matrix of each data stream. However, $p_i$ might not be known in advance in real applications. In such cases, we have to choose $p_i$ such that it is big enough to cover the correlation within the time series. $^1$

We define the vector random process $X = [X_1, X_2, \ldots, X_n]$. The covariance matrix $K_X$ of this vector random process is a block Toeplitz matrix as $^1$

$^1$In practice, $p_i$ is usually less than 10.
follows:

$$K_{\mathcal{X}} = \begin{pmatrix}
K_{X_1, X_1} & K_{X_1, X_2} & \ldots & K_{X_1, X_n} \\
K_{X_2, X_1} & K_{X_2, X_2} & \ldots & K_{X_2, X_n} \\
\vdots & \vdots & \ddots & \vdots \\
K_{X_n, X_1} & K_{X_n, X_2} & \ldots & K_{X_n, X_n}
\end{pmatrix} \tag{5.1}$$

In Equation (5.1), the matrix $K_{X_i, X_i}$ is the covariance matrix of $X_i$ of size $p_i \times p_i$ and $K_{X_i, X_j}$ is the cross covariance matrix of $X_i$ and $X_j$ of size $p_i \times p_j$. Note that $K_{X_i, X_j}$ is a symmetric Toeplitz matrix while $K_{X_i, X_j}$ is a Toeplitz matrix. Our goal is to find the optimal $n$ dimensional noise vector process $Z = [Z_1, Z_2, \ldots, Z_n]$ that provides the highest privacy for a given level of noise energy.

We solve this problem by finding the covariance matrix for the optimal perturbation noise first. Then we construct the model for the multidimensional noise by from the optimal covariance matrix found in the previous step. Finally, the data points for perturbing the data stream are generated using the constructed model. The details of those steps are discussed in detail in the following section.

### 5.2.2 Finding Optimal Perturbation Noise

Taking the same approach presented in Chapter 3, we define the privacy of the multidimensional time-series as the mutual information $I(\mathcal{X}, \mathcal{X} + Z)$ between the original data $\mathcal{X}$ and the perturbed data $\mathcal{Y} = \mathcal{X} + Z$. This approach has been proven to be a good privacy metric to measure the privacy of our proposed single-dimensional time-series perturbation technique. In this work, we will show that an information theoretic privacy measure can
also be used to measure privacy for multidimensional time series.

First, observe that computing the mutual information between original data and perturbed data is not easy. In fact, there is no general method for computing the mutual information between arbitrary random processes. Therefore, instead of computing the mutual information directly, we find a tight upper bound for the mutual information in general and try to minimize it in order to minimize the information leak.

In Chapter 3, we derive a good upper bound for the mutual information between two single-dimensional random processes. Here, notice that the derived upper bound on privacy is still applicable for multidimensional data because no assumptions about the type of data are made in Lemma 1. Thus, the Lemma can be rewritten as follows:

\[ I(\mathcal{X}, \mathcal{X} + \mathcal{Z}) \leq I(\mathcal{X}_G, \mathcal{X}_G + \mathcal{Z}_G) + D(\mathcal{Z}||\mathcal{Z}_G) \]  

(5.2)

In Equation (5.2), \( \mathcal{X} \) and \( \mathcal{Z} \) are the original data and the perturbed noise (both are multidimensional), respectively. Also, \( \mathcal{X}_G \) and \( \mathcal{Z}_G \), respectively, are Gaussian random processes having the same covariance matrices as \( \mathcal{X} \) and \( \mathcal{Z} \). The operator \( D(.,||.) \) represents the Kullback-Leibler distance between two random processes. Because the Kullback-Leibler distance is non-negative, thus the upper bound is minimize if and only if \( D(\mathcal{Z}||\mathcal{Z}_G) = 0 \). This means that the optimal perturbation noise must be a multidimensional Gaussian random process. In this case, the upper bound on mutual information becomes much simpler:

\[ I(\mathcal{X}_G, \mathcal{X}_G + \mathcal{Z}_G) + D(\mathcal{Z}||\mathcal{Z}_G) \geq I(\mathcal{X}_G, \mathcal{X}_G + \mathcal{Z}_G) \]

\[ = \frac{1}{2} \log \frac{\det(K_X + K_Z)}{\det(K_Z)} \]  

(5.3)
Finally, minimizing the upper bound on privacy is equivalent to minimizing the expression in the Equation (5.3). Observe that the upper bound on mutual information described in Equation (5.3) is a decreasing function with the power of the perturbation noise. The higher the noise power, the lower the information leak, hence the higher the privacy. However, the higher the noise power, the lower the utility acquired from the perturbed data. Therefore, we cannot perturb the original data with the noise with arbitrarily high power. Thus it is natural to impose the noise power constraint while minimizing the information leak.

The covariance matrix \( \mathcal{K}_Z^* \) of the optimal perturbation noise which minimizes \( I(\mathcal{X}, \mathcal{Y}) \) can be found by solving the following optimization:

\[
\begin{align*}
\mathcal{K}_Z^* &= \arg\min_{\mathcal{K}_Z} \frac{1}{2} \log \frac{\det(\mathcal{K}_X + \mathcal{K}_Z)}{\det(\mathcal{K}_Z)} \\
\text{subject to} & \\
\frac{1}{n} \trace(K_{Z_i, Z_i}) & \leq P_i \quad \forall i \\
K_{Z_i, Z_i} & \succ 0 \\
K_{Z_i, Z_i} \text{ is Symmetric Toeplitz} \\
K_{Z_i, Z_j} \text{ is Toeplitz}
\end{align*}
\] (5.4)

In the Equation (5.4), \( P_i \) is the power allocated to the noise stream \( Z_i \). This provides users the flexibility to control the privacy of individual data streams. By Lemma (1), the objective of the optimization shown in (5.4) is a convex function. Furthermore, the set of positive definite square matrix is also convex. Therefore, the optimization in (5.4) is a semi-definite optimization problem. However, this optimization has to be transformed into a canonical form of semi-definite optimization in order to be solved numerically. Using the same approach presented in Theorem (1), we obtain the
following equivalent optimization:

\[
K^*_Z = \arg\max_{K_Z} \log\det(t) \quad (5.5)
\]

subject to

\[
\begin{pmatrix}
K_Z - t & K_Z \\
K_Z & K_X + K_Z
\end{pmatrix} \succ 0
\]

\[
\frac{1}{n} \text{trace}(K_{Z_i,Z_i}) \leq P_i \quad \forall i
\]

\[
K_{Z_i,Z_i} \succ 0
\]

\[
K_{Z_i,Z_i} \text{ is Symmetric Toeplitz}
\]

\[
K_{Z_i,Z_j} \text{ is Toeplitz}
\]

Next, we need to construct the model for the multidimensional noise which
is used to generate actual noise data points. A common way to model mul-
tidimensional time series is to use the multidimensional Gaussian-Markov
model as follows:

\[
Z_k = \sum_{i=1}^{m} \Phi_i Z_{k-i} + \varepsilon_k \quad (5.6)
\]

In Equation (5.6), \(Z_k\) is an n-tuple containing the perturbation noise for
n data streams at time \(k\). The coefficients \(\Phi_i\) are \(n \times n\) matrices and \(\varepsilon_k\)
is a n-dimensional zero mean i.i.d. Gaussian random variable with covariance
matrix \(\Sigma\). The coefficients \(\Phi_i\) and \(\Sigma\) are computed such that the covari-
ance matrix of \(Z_k\) equals \(K^*_Z\). It is possible to determine those coefficients
by establishing a series of Yule-Walker equations and solve them using the
Levinson-Durbin algorithm [32].
5.3 Evaluation

In this section, we evaluate the performance of the developed perturbation algorithms with simulated data. In the first experiment, we want to show that the proposed perturbation technique effectively hides not only the correlation among data streams but also the correlation inside each stream. We generate two correlated data streams \([X_1, X_2]\) according to the following linear model:

\[
\begin{pmatrix}
    X_1 \\
    X_2
\end{pmatrix}_{k+1} =
\begin{pmatrix}
    0.9 & 0.1 \\
    0.2 & 0.1
\end{pmatrix}
\begin{pmatrix}
    X_1 \\
    X_2
\end{pmatrix}_k + \varepsilon_k
\]  

(5.7)

In the above equation, \(\varepsilon_k\) is the two-dimensional Gaussian noise with zero mean and covariance matrix

\[
\Sigma = \begin{pmatrix}
    1.0 & 0.3 \\
    0.3 & 0.5
\end{pmatrix}
\]

In the model, we choose the coefficients such that the power of \(X_1\) is about 10 times larger than the power of \(X_2\), which is useful when comparing the privacy achieved for each data stream. Using the model described in Equation (5.7), we generate 8192 data points for each stream and use them as the input to evaluate our proposed algorithms. As argued in the previous section, the power spectral density of the original data and the power spectral density of the perturbed noise have to overlap in order to provide privacy. Therefore, we first estimate the power spectral density of both data streams \(X_1\) and \(X_2\). To estimate the power spectral density of a random process given the real data points, we first fit the data points into a 6 tap auto-regressive (AR) model. Then the PSD of the random process is computed as the Fourier transform of the auto-correlation coefficients of the inferred AR model. The
power spectral density of both $X_1$ and $X_2$ is presented in Figure 5.1 where the $x$ axis presents the digital frequency and the $y$ axis present the power of the signal in dB.

![Figure 5.1: Power spectral densities of $X_1$ and $X_2$](image)

Next, we generate the noise stream $Z_1, Z_2$ using our proposed algorithm described in the previous section. The power constraints for $Z_1$ and $Z_2$ are the same as the empirical power of $X_1$ and $X_2$, respectively. The chosen size of the covariance matrix for both $Z_1$ and $Z_2$ is 6. The power spectral density of both the original data and the noise are plotted in Figure 5.2.

![Figure 5.2: Power spectral densities of original data and perturbed data](image)

Observe that the power spectral densities of $Z_1$ and $Z_2$ overlap with the power spectral densities of $X_1$ and $X_2$, making it hard to estimate the original
Next, we would like to compare the degree of privacy obtained by our proposed method in comparison with the perturbation by independent Gaussian noise. We perturb the original data with independent Gaussian noise and the optimal correlated noise at different signal-to-noise ratios. Because the original data is generated according to a linear system presented in Equation (5.7), the best estimator is a Kalman filter [10]. Therefore, we use the Kalman filter to try to estimate the original signal from the perturbed data. Figure 5.3 shows the error when estimating the original data from both data perturbed with Gaussian noise and data perturbed with optimal noise. In the figure, for the stream $X_1$, the error when estimating original data if it is perturbed with the optimal noise is about 20% larger than that perturbed with Gaussian noise with the same power. For stream $X_2$, however, the estimation errors are the same in both cases. This is because the power spectral density of $X_2$ in Figure 5.1 resembles the power spectral density of Gaussian noise. Thus, perturbing $X_2$ with Gaussian noise is almost optimal.

Figure 5.3: Estimation error when original signal is perturbed with Gaussian noise and optimal noise

Next, we will evaluate the community reconstruction error when our proposed optimal perturbation scheme is used. To do that, we simulate a large
community by generating 200 two-dimensional data streams using the model specified in the Equation 5.7. Then all of the data streams are perturbed using optimal correlated noise at different signal-to-noise ratios. Finally, the community distribution is reconstructed using the algorithm presented in the previous chapter. The error of the reconstruction is plotted in Figure 5.4. We observe that reconstruction error is very small even with a low signal-to-noise ratio.

![Figure 5.4: Reconstruction error versus signal-to-noise ratio](image)

Although Figure 5.4 provides a means to choose the noise power in order to achieve a certain level of utility (reconstruction error), it is still not clear about the level of privacy achieved at a fixed level of utility. Figure 5.5 plots the trade-off between privacy (in bits) and utility (in community reconstruction error) for both data streams. It is clear from the figure that the utility decreases when the privacy increases. This figure is useful in choosing the best trade-off between utility and privacy in designing the application.

5.4 Conclusion

In this work, we presented the algorithms to optimally perturb multidimensional time series. We show that the mutual information between the origi-
nal data and the perturbed data is a good measure of privacy. In addition, we present an optimization framework to find the optimal multidimensional correlated noise. Our data perturbation techniques allow users to perturb multiple private data streams before sharing while allowing useful aggregated community statistics to be reconstructed at the server side. With our proposed privacy measure, users can easily control the trade-off between the privacy and the utility, which plays an important role in designing participatory sensing applications.
CHAPTER 6
RELATED WORK

6.1 Data Hiding Techniques

In recent years, a number of techniques have been proposed for modifying or transforming data in such a way as to preserve privacy. Such methods can be classified into three main categories described in detail below.

First, randomization techniques that add noise to the original data points have been used to hide the real value of sensitive data and other attributes (e.g., the trend of the data over time) [47, 48]. They traditionally distort data for methods such as surveys which have an evasive answer bias because of privacy concerns [49, 50]. Fuller [51] and Kim and Winkler [52] show that some simple statistical information (e.g., means and correlations) can be preserved by adding random noise. In [6, 7], independent random noise (e.g., Gaussian) is used to perturb user data. Privacy can be achieved if the noise power is high enough. However, high noise power might decrease the utility of the shared data as well and the authors do not quantify this trade-off. Recently, Ganti et al. [19] proposed that correlated noise, which has the same distribution as real data, can be used to perturb time-series data. This perturbation method is resilient to traditional filtering techniques, such as Kalman filter [10], and spectral filtering [8]. However, it is not clear if other techniques can filter out noise from perturbed data more effectively. Our perturbation technique, proposed in this dissertation, not only outperforms
previous techniques in terms of privacy for a given noise level but also provides a guarantee on the achieved privacy, no matter what reconstruction technique is used.

Second, the \textit{k-anonymity model} \cite{15} was developed because of the possibility of indirect identification of records from public databases. For example, the identity of a patient can be inferred from their home address or cellphone number. In the k-anonymity method, the granularity of data is reduced using techniques such as generalization and suppression. This granularity is reduced such that any given record maps onto at least k other records in the data. The \textit{l-diversity model} \cite{53} was designed to handle some weaknesses in the k-anonymity model since protecting identities to the level of k-individuals is not the same as protecting the corresponding sensitive values, especially when there is homogeneity of the sensitive values within a group. Many variants of the above methods exist in current literature. A good survey of the corresponding algorithms may be found in \cite{54}

Finally, \textit{distributed privacy preservation} \cite{55, 56} is used to derive aggregate results from data sets which are partitioned across these entities. While the individual may not desire to share their entire data set, they may consent to limited information sharing with the use of a variety of protocols. The overall effect of such methods is to maintain privacy for each individual, while allowing the aggregate results to be correctly computed over an entire group. For this purpose, the data sets may either be \textit{horizontally partitioned} or \textit{vertically partitioned}. In horizontal partitioning \cite{57, 58, 59}, the individual records are spread out across multiple entities, each of which have the same set of attributes. In vertically partitioned data sets \cite{57, 60}, the individual entities have different attributes of the same set of records. Both techniques play equally important roles in a wide range of applications.
6.2 Privacy Measures

The problem of privacy quantification has been studied extensively and a variety of metrics have been proposed for different purposes. For example, the *minimal distortion metric* [13], which measures “similarity” between the original data and the anonymous data, has been used to measure data privacy. Another metric, called *distinctive attribute* [17], was used to guide the search for a minimally anonymous table in a full-domain generalization scheme.

The metric proposed in this work belongs to the category of *trade-off* metrics, which measure information leak and the utility achieved from the perturbed data. We note that the quantification of privacy alone is not sufficient without quantifying the utility of the data created by the randomization. In the early work [7], Agrawal and Aggarwal defined the information leak as revealing of specific data in a tuple. Often though, information can be leaked even if the adversary does not gain access to a specific data item. Such attacks usually rely on knowing aggregate information about the (perturbed) source data as well as the method of perturbation used when modifying the data. Also in [7], the authors proposed the use of mutual information to measure the leaked information. However, it is not used to evaluate the privacy in practice, because it needs knowledge of the user data distribution, which is not available in general, in order to estimate the achieved privacy. In [19], the authors justified the privacy as the error in the estimate of user data given the perturbed data. However, a limitation in those approaches is that they are only used to evaluate the effectiveness of a specific perturbation scheme. In recent work [61], the use of the rate distortion theory was proposed to analyze the trade-off region irrespective of the type of data source.
or the methods of providing privacy. However, this work does not address
the privacy problem for time-series data.

6.3 Applications of Privacy-Preserving Techniques

Privacy-preserving techniques have numerous applications in medical database
mining, homeland security and customer transaction analysis. In medical
data base applications, the Scrub system [62] employs privacy techniques
to de-identification of clinical notes and letters which typically occur in the
form of textual data. The Scrub system uses numerous detection algorithms
which compete in parallel to determine when a block of text corresponds to a
name, address or a phone number. It has been shown in [62] that the system
is able to remove up to 99% of the identifying information from the data.

The Datafly system [14] was designed to prevent identification of the sub-
jects of the medical records which may be stored in multidimensional format.
The multidimensional records may contain directly identifying information
such as social security number, or indirectly identifying information such
as age, zip-code or sex. This system was designed to address the concern
that removing only directly identifying information may not be sufficient to
guarantee privacy. The Datafly system is one of the earliest systems for
anonymization and it motivates a lot later works in the anonymity field,
including k-anonymity [15].

Several homeland security applications have been developed using privacy-
preserving techniques. In [63], a broad overview is provided on how privacy-
preserving techniques may be used in order to deploy these applications ef-
fectively without violating user privacy. An example of such systems is the
identity angel system [64], which crawls through cyberspace and determines
people who are at risk of identity theft. A web camera surveillance system [65] is used to monitor unusual activities in terms of facial count rather than using more specific information about particular individuals.
CHAPTER 7

CONCLUSION AND FUTURE WORK

The recent emergence of smart personal mobile devices allows ordinary users to collect, share and aggregate data without the need for dedicated infrastructure. This new sensing paradigm allows new applications to be deployed quickly and in a cost effective manner. The work in this dissertation addresses the privacy problem arising in those applications when private time-series data are shared across an untrusted entity in the community. We propose new perturbation techniques that allow users to protect their data by perturbing them before sharing. In addition, we derive algorithms to reconstruct useful community statistics from the perturbed data with high accuracy.

This work focuses on the privacy of time-series data. Time-series data is vulnerable to attacks because the correlation between data points in time can be easily used to reveal original data if it is not properly hidden. Cryptographic techniques can not be used in this type of application because no trust hierarchy exists between data providers and data consumers. Furthermore, identity anonymization techniques may breach information because of the strong correlation between the user’s identity and their private data. Previous research on perturbation based privacy for time-series data was mostly ad-hoc because there exists no universal privacy measure. As a result, different perturbation techniques cannot be compared to each other; each of them is usually proved to work in a very narrow range of applications. Furthermore, the lack of a good privacy measure makes it impossible to derive
optimal perturbation methods.

In this dissertation, we propose the use of mutual information between the original data and the perturbed data as a measure of privacy of time-series data. The advantage of this privacy metric is that it does not depend on the original data model or on the type of attack. This advantage makes it possible to compare the perturbation methods without making any assumption about the attack methods.

Using this privacy measure, we are able to derive the optimal perturbation methods for both single-dimensional and multidimensional time series. Details of the optimization framework to find the optimal perturbation noise were presented. The privacy achieved by using optimal perturbation methods is significantly higher than traditional perturbation approaches for the same noise power.

We also propose algorithms to reconstruct community distribution for both single-stream data and multi-stream data from perturbed data. A deconvolution technique was used to reconstruct distribution for single-stream data with success. For multi-stream data, we develop an EM-based iterative algorithm to accurately reconstruct the joint distribution of the multidimensional data.

7.1 Future Work

Since this work represents an early effort in this area, the emphasis was on developing general methodology and techniques that provide privacy for time-series data. The results presented here are very promising and suggest that continued work in this area is appropriate. In particular, additional work is needed to explore the relation between the perturbation noise power.
and the utility (i.e., the community density reconstruction error). Better knowledge about this relation will help in developing better perturbation techniques, making it easier for users to control the trade-off between utility and privacy.

As discussed in previous sections, the perturbation algorithms proposed in this dissertation work best if the random processes is stationary ARMA process (or can be approximated with high accuracy). In applications where the original data has different structure (e.g., distribution of the GPS data points described in the traffic application in Chapter 4), the perturbation proposed in this work cannot be applied. Thus there is the need to extend the optimization framework to those particular applications. Further work in this area can provide a way for the algorithm to adapt to changes in the data model without the need to regenerate the noise model.
REFERENCES


[23] M. G. Kang and A. K. Katsaggelos, “General choice of the regulariza-
tion functional in regularized image restoration,” IEEE Trans. Image

www8.garmin.com/products/etrexLegend


[26] S. N. Diggavi and T. M. Cover, “The worst additive noise under a co-
variance constraint,” IEEE Trans. on Information Theory, vol. 47, no. 7,

vol. 38, no. 1, pp. 49–95, 1996.

software package for semidefinite programming,” Optimization Methods

[29] J. Lofbert, “YALMIP : A toolbox for modeling and
http://control.ee.ethz.ch/ joloef/yalmip.php

bility identities,” 2000-2010. [Online]. Available:

[31] F. Zhang, The Schur Complement and Its Applications. New York,


[34] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood
from incomplete data via the EM algorithm,” J. of the Royal Statistical


