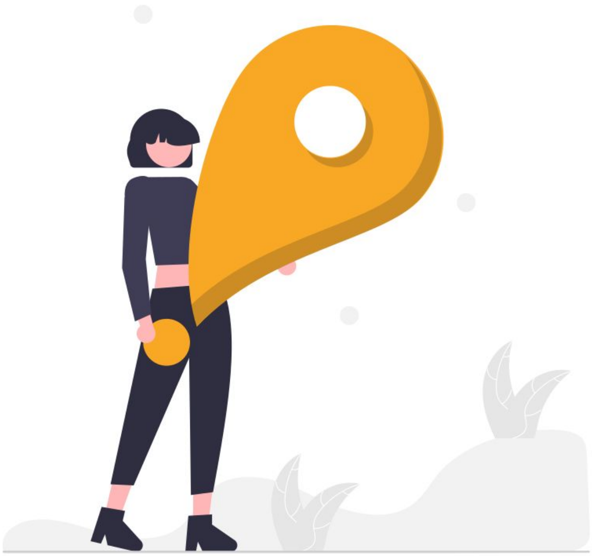
Privacy-Aware Crowdsensing Architecture

Context-Aware Systems
2020/21

Andrea Gurioli, Giovanni Pietrucci
Mario Sessa

1. Goals and Metrics

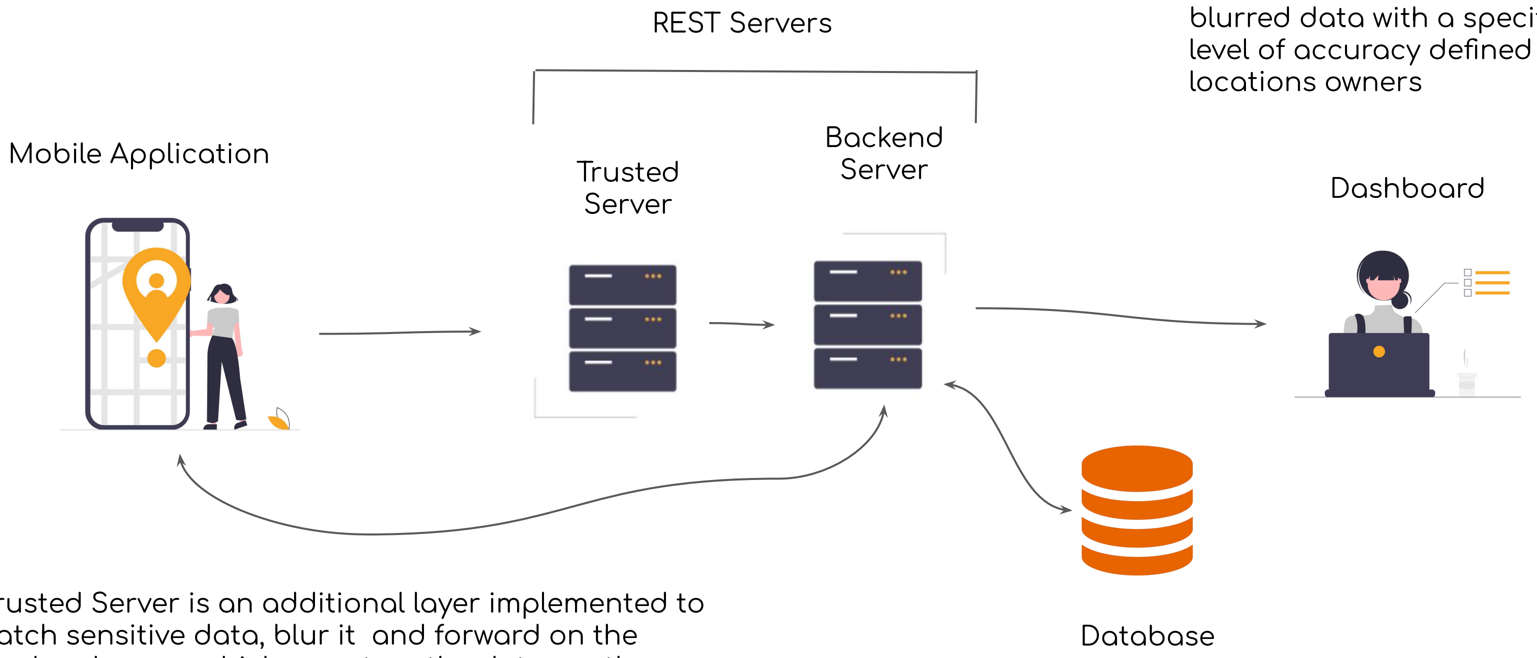


Goals: Build a context-aware systems for noise crowdsensing services with locations obfuscation settings in order to maintain privacy.

Metrics: Evaluation parameters consist of the optimization of the quality of service in regards to the privacy level. Obtaining a good trade-off between these two values is the main approach for a great model

2. Architectures Components

Dashboard can visualize only blurred data with a specific level of accuracy defined by the locations owners



Trusted Server is an additional layer implemented to catch sensitive data, blur it and forward on the backend server which can store the data on the database or send it to the dashboard.

3. Mobile Application

Mobile Application is implemented in **Java Android** using **Google Maps API** for locations surveys. A mobile user can also manage privacy settings about sent location management. Using smartphones microphones it can be detected the wave amplitude and used in the noise computation:

$$db = 10 \cdot \log_{10}\left(\frac{wa}{ref}\right)$$

Where *db* is the noise in decibel Watt, *wa* is the wave amplitude and *ref* is the reference mobile value equals to *1 Watt*.

It is also a **receiver endpoint** from the **backend server** which visualize the **noise mean** with a range of **3 kilometers** from the current blurred location.

4. Trusted Server

Trusted Server is a **express.js** security layer which can maintain app data in a temporal and secure list to apply aggregations based on the **spatial cloaking** algorithm. It returns in output an obfuscated location.

Trusted Server will use **privacy parameters** of the **location request** to manage the privacy level to apply for each survey.

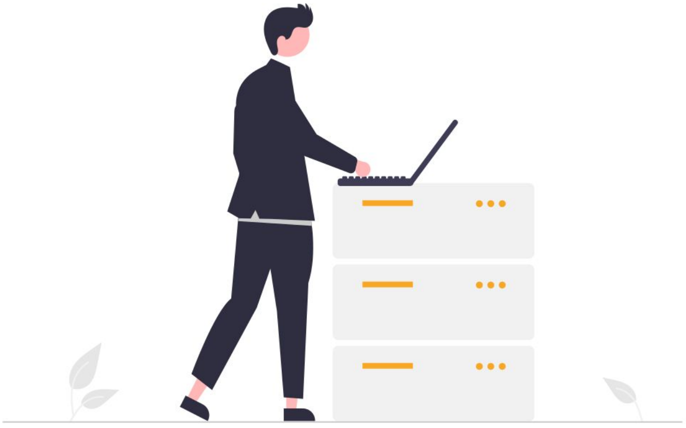
It is possible to avoid this procedure using a **no-privacy policy** on the privacy setting of the mobile application.

5. Backend Server

Backend Server is a **express.js application** that manages some **CRUD operations** between received data from the trusted server or persistence data from database.

It is able to:

- receive obfuscated geo-spatial locations from the trusted server and save them in the database
- take persistence data from the database, format it in **GeoJson** format and give it in response to the dashboard.
- calculate mean noise using persistence data given by **PostGIS** query related to a specific location.



6. Database

We use a simple Postgres database within **PostGIS** extension to manage geo-spatial data points.

The whole database has only one table which contains:

1. Row identification
2. Geo-spatial point made by geographical coordinates
3. Level of noise
4. Privacy evaluation
5. Quality of Service value
6. Alpha for the tradeoff

Persistence data is only obfuscated or no-privacy one to avoid possible data theft by malicious people.

7. Dashboard

Dashboard is a web application which uses a **Leaflet map** to visualize **GeoJson points** given by the backend server.

Using **leaflet layers**, we implemented **multiple views of data** for:

1. Markers and data visualization
2. Heatmap noise levels
3. Locations clustering
4. Interactive noise prediction

Clusters and predictions are made in backend using a script bridge with **machine learning models** written in Python.

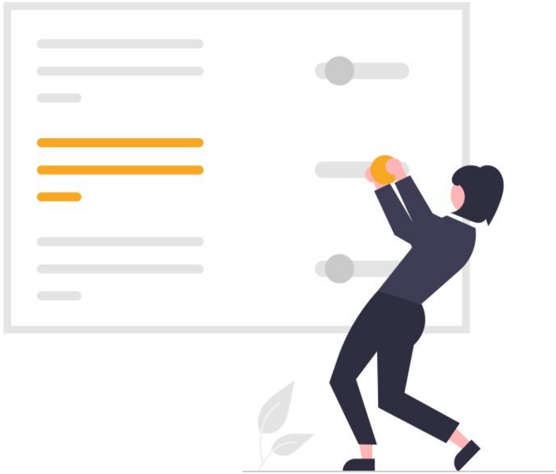
8. Data Management

Data sent in request or response between two components of the architecture are:

- **GeoJson** for geo-spatial locations
- **Json** for single attributes like noise means

Mobile Application and Trusted Server are able to manage sensitive data, meanwhile Backend Server, Dashboard and Database can see only blurred information. Furthermore, Trusted Server will delete sensitive data in the temporal pending list in case a customer time-to-live value, given in the request, expires.

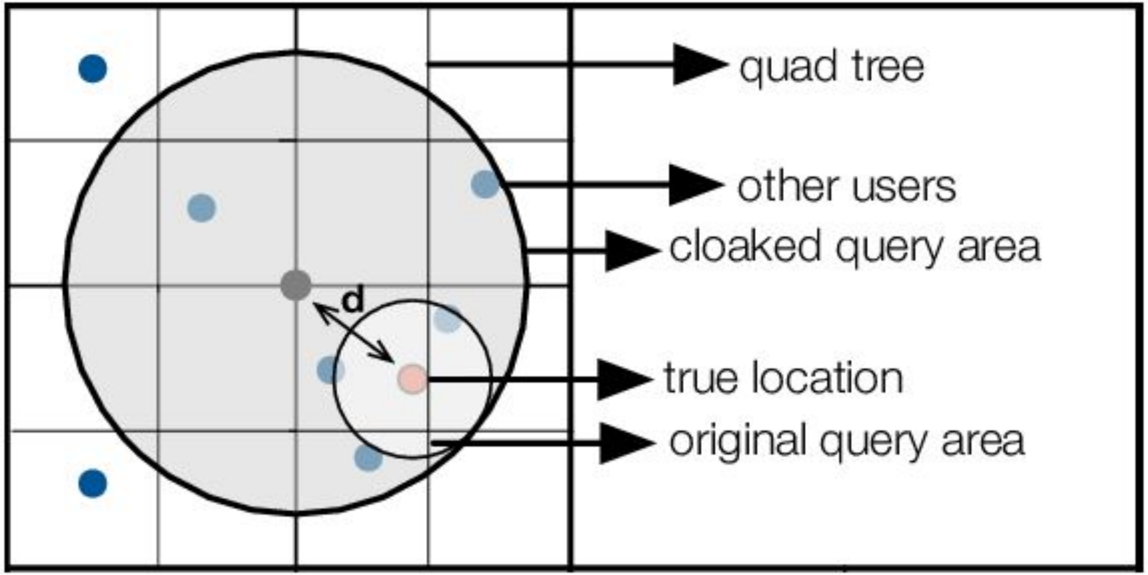
9. Data Privacy



Customers can decide directly their privacy metrics using **mobile application settings, switch off** these values to send a no-privacy survey or let the trusted server defines them using automatic configuration based on an alpha value to balance the **tradeoff between Privacy and Quality of Service**.

If a customer wants to obfuscate his points, Trusted Server can apply a **Spatial Cloaking technique** which returns, for a group of *k* surveys, an obfuscated location sent to backend server and stored in the database.

10. Locations estimation in spatial cloaking



11. Noise estimation in spatial cloaking

Given the spatial location, we estimate the point's noise using the **Inverse Square Law** for the **noise intensity propagation** of **k sources** on the generated one.

$$r_i = \frac{1}{d_i^2} \text{ for } i \text{ in} = 1, 2 \ldots k$$

$$L_i = \frac{db_i}{4\pi r_i^2}$$

Where d_i is the **geographical distance** between the source *i* and the spatial location. Finally, we calculate the mean noise using the logarithmic sum of propagated noises

$$L_\partial = 10 \cdot \log_{10}\left(\sum_{i=1}^{k} 10^{\frac{L_i}{10}}\right)$$

12. Privacy evaluation

$\forall k\left(x_k, y_k\right) \in P_{gen}$, given $\partial\left(x_\partial, y_\partial\right)$ the generated point.

Given $\sigma: R \rightarrow R$ a function where $\sigma(\lambda) = \dfrac{\lambda \times \pi}{180}$ is the radiant convertion function,

and $\varphi = x_k - x_\delta$ the differences between points longitudes

$$dist_{(k,\delta)} = \left(arc\sin\left(\sin\left(\sigma\left(y_k\right)\right) \cdot \sin\left(\sigma\left(y_\partial\right)\right)\right)\right) + \cos\left(\sigma\left(y_k\right)\right) \cdot \cos\left(\sigma\left(y_\partial\right)\right) \cdot \cos(\sigma(\varphi))\Big) \cdot R$$

$dist_{(k,\partial)} = dist_{(k,\partial)} \cdot \dfrac{180}{\pi}$ for the inverse radiant convertion

Finally, the privacy value is given by $p_\partial = \dfrac{\sum_{k \in P_{gen}} dist(k, \partial)}{n}$ with n the cardinality of P_{gen}

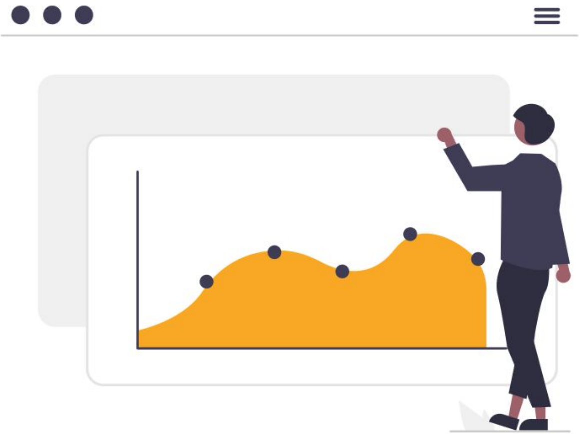
13. Quality of Service Evaluation

Metric privacy measures the differences in the mean geographical distance between generated points and generators. Meanwhile, **Quality of Service** measures the **mean square error** between estimated spatial location's noise, given by the **propagation** using **inverse square law**, and real locations noises.

$$q_{\partial} = \frac{\sum_{x \in P_{gen}} \left(L_x - L_{\partial} \right)^2}{n}$$

Where n is the number of generators points, L_x, L_{∂} are respectively noise intensity of generators and centroid (generated location).

14. Automatic Configuration



Mobile users can use an **automatic configuration** for **privacy settings** depending on an **alpha value** that automatically defines **k** and **range** for the spatial cloaking algorithm and decide the **best tradeoff** for the user surveys.

Alpha values are in a range between **0** (no-privacy) and **1** (privacy priority).

The alpha value is directly proportional to **k** and **range** parameters of the survey.

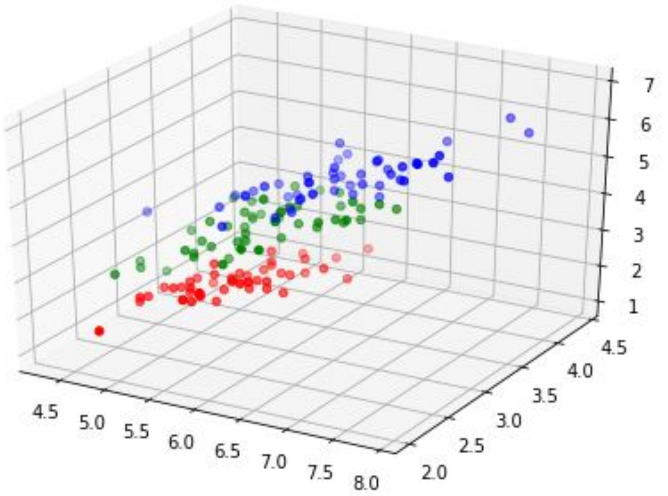
Runtime tradeoff is the result of:

$$\forall x \in P,\ tradeoff_x = \alpha \cdot p_x + (1 - \alpha) \cdot q_x$$

15. Clustering algorithm

We use a k-means algorithm to aggregate locations with similar properties:

1. **Distance** between points - **2d vectorial space**
2. **Noise** and **distance - 3d vectorial space**



In order to obtain a 3 dimensional clustering, it has been scaled the db values to the same '*space distance*' **magnitude**. The distance between points is computed in regards of the euclidean distance. The number of clusters is decided by the user and it's selected in the dashboard.

16. Noise Prediction

The **Noise Prediction** phase is splitted in three different **steps**:

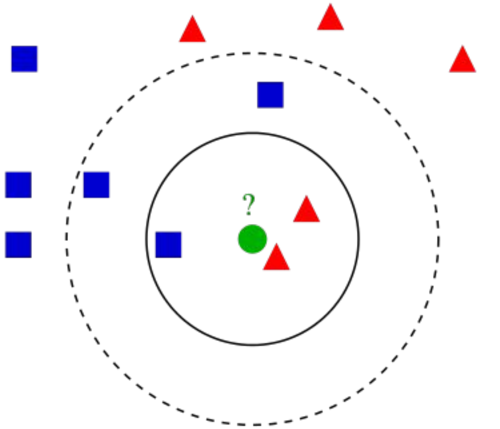
1. *'Within' query* to spatial point
2. Regressor *choice* and model *fitting*
3. *Prediction*

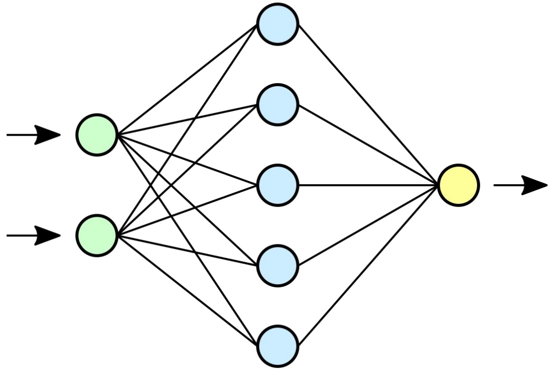
The regressor used due to the testing phase are:

1. **K-NN** if the query resulting points are more than 3 elements.
2. **NN** otherwise

Regression testing phase between **LR** and **Neural network:**

Regressor	LR	NN
MSE	298.7(±0.1)	287.2(±0.1)





15. Conclusions

Context-Aware systems can be very efficient in many real context environments to provide important **data surveys** using specific **MEMS sensors.**

One of the main problem with these approaches are the **vulnerabilities** of context-aware systems.

Nowadays, many online services **don't securely manage data,** and it is easy to **steal sensitive information** from an **unprotected provider**.

Adding **security protocols** and **horizontal security layers** should solve the problem. This approach has a **side-effect** of the **decrement of the quality of service metrics.** For that reason, we should consistently evaluate the **tradeoff** between **privacy and QoS** and choose the best balance.