Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. 3188

Optimized Acquisition of Spatially Distributed Phenomena in Public Sensing Systems

Jarosław Stachowiak

Course of Study:	Computer Science
Examiner:	Prof. Dr. Kurt Rothermel
Supervisor:	Dipl.-Inf. Damian Philipp
Commenced:	April 4, 2011
Completed:	October 25, 2011
CR-Classification:	C.2.1, C.2.2, C.2.3, C.2.4

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Jarosław Stachowiak)

Abstract

Nowadays, an increasing number of popular consumer electronics is shipped with a variety of sensors. The usage of these as a wireless sensing platform, where users are the key architectural component, and ubiquitous access to communication infrastructure has established a new application area called public sensing.

We present an opportunistic public sensing system that allows for a flexible and efficient acquisition of sensor readings. This work considers the usage of smartphones as a sensor network in a model-driven sensor data acquisition. We focus on efficiency of query dissemination to mobile nodes, while retaining high effectiveness regarding defined sensing quality of collected data. We adopted and extended an existing geographic routing protocol to design an efficient communication system that executes model-driven data acquisition and is robust to changing sensors availability. We use in-network processing paradigm to efficiently distribute queries to mobile nodes and to collect results afterwards.

The developed approach was simulated using OMNeT++ network simulator. To verify implemented algorithms and test the overall system performance, we run simulations in different scenarios and evaluate them using adequate coverage metrics. Moreover, we verify our intuitive extension to adopted routing protocol and show that it can have a strong impact on the efficiency of protocol in question.

Contents

1 Introduction 9

1.1 Motivation . . . 9
1.2 Problem Formulation . . . 10
1.3 Thesis Outline and Contributions . . . 10

2 Background 13

2.1 Public Sensing . . . 13
2.2 Model-driven Sensor Data Acquisition . . . 15
2.2.1 Introduction . . . 15
2.2.2 Prediction Model . . . 16
2.2.3 Modelling Sensor Data . . . 17
2.2.4 Quantifying Sensing . . . 18
2.2.5 Offline Sensor Selection . . . 19
2.2.6 Related work . . . 20
2.3 Positioning Systems . . . 21
2.4 Routing in MANETs . . . 23
2.4.1 Basic Principles of MANETs . . . 23
2.4.2 Topology-based Routing Techniques . . . 25
2.4.3 Position-based Routing . . . 26
2.4.4 Geographic Unicast Routing . . . 28
2.4.5 Greedy Perimeter Stateless Routing . . . 29

3 System Model 31

3.1 System Model. . . . 31
3.2 Challenges and Requirements . . . 33

4 System Design 35

4.1 Introduction . . . 35
4.2 Architecture . . . 36
4.3 Public Sensing . . . 38
4.3.1 Gateway . . . 38
4.3.2 Statistical Model. . . . 40
4.3.3 Queries . . . 40

4.3.4 Area Division into Regions . . . 42
4.3.5 Query Delivery Models . . . 43
4.3.6 In-Network Planning . . . 44
4.3.7 In-Network Aggregation . . . 46
4.3.8 Premature Routing Termination . . . 47
4.4 Location Information Management . . . 48
4.4.1 Division of Roles . . . 48
4.5 GPSR Protocol . . . 49
4.5.1 Delivery Area . . . 50
4.5.2 Burst Mode . . . 51
4.5.3 Planarization . . . 52

5 Implementation 53

5.1 Network Simulator . . . 53
5.1.1 Ad-hoc Routing in INETMANET . . . 54
5.1.2 IP Module Interface . . . 54
5.2 UMTS . . . 55
5.3 GPSR Protocol . . . 55
5.3.1 Beaconing . . . 55
5.3.2 Neighbour Table . . . 57
5.3.3 Greedy and Perimeter Mode . . . 58
5.4 GPSR Application Programming Interface . . . 61
5.4.1 API Interface and Messages . . . 61
5.4.2 Processing of API Messages . . . 63
5.5 GPSR Extensions . . . 63
5.5.1 Packet Formats . . . 63
5.5.2 Burst Mode . . . 64
5.5.3 Support for Mac-layer Failure Feedback . . . 67
5.5.4 Integration with OMNeT++ . . . 67
5.6 Public Sensing . . . 69
5.6.1 Packet Formats . . . 69
5.6.2 Gateway . . . 70
5.6.3 Node . . . 71
5.7 Location Information Management . . . 71
5.7.1 Custom Location Information . . . 71
5.7.2 Regions and Populated Regions . . . 73
5.7.3 Packet Formats . . . 73
5.8 Simulation Parameters . . . 75

6 Evaluation 77

6.1 Methodology . . . 77
6.2 Sensor Dataset . . . 77
6.3 System Setup . . . 78
6.3.1 Network . . . 78
6.3.2 Mobile Node . . . 80
6.3.3 Mobility . . . 81

6.3.4 NIC Settings 82
6.4 Metrics 82
6.5 Experiments 83
6.5.1 Validation of GPSR protocol 84
6.5.2 Burst Mode. 84
6.5.3 System Parameters 85

7 Summary **87**
7.1 Conclusion 87
7.2 Future Work 88

Bibliography **91**

Chapter 1

Introduction

1.1 Motivation

Technological evolution in networking, micro-fabrication, embedded microcontrollers, and integration of physical sensors in the past few decades has led to emerge of consumer electronics as wireless sensor platforms. Examples of such devices are smartphones (e.g., Android-based or Apple iPhone), PDAs (e.g., Nokia N810), and MP3 players (e.g., Nike + iPod). Today, sensors such as cameras, gyroscopes, and accelerometers, are becoming more and more prevalent in smartphones carried by billions of people. In parallel, other external sensors can be easily connected using for instance Bluetooth. As a consequence, user generated content has been extended to sensed data of the environment, e.g., humidity, temperature, pressure, pollution, and others. The wide adoption of smartphones with positioning systems, like widely-used GPS, enable a generation of massive-scale sensor networks formed from consumer devices. This creates a unique chance for widespread public participation in data collection that can be shared with everyone for the grater public good, and is refereed to as public sensing [14].

Traditionally, we used sensor networks composed of geographically dispersed customized sensor nodes that worked together to monitor physical or environmental conditions, such as air pressure, temperature, or pollution. Some drawbacks of sensor networks include the need to conserve resources at all costs, expensive maintenance, lack of robustness, difficulties to cover a larger area in static deployments and others. The idea of public sensing is to use a huge number of heterogeneous devices with sensing capabilities and exploit uncontrolled mobility of people carrying these devices. According to the ITU, by the end of 2010, there was an estimated 5.3 billion mobile cellular subscriptions worldwide. They report that access to mobile networks is now available to 90% of the world population and 80% of the population living in rural areas [44]. Therefore, if only a fraction of those users participate in public sensing, we have a huge number of available sensing devices.

Turning users into creators, custodians, actuators, and publishers of the data they collect, creates a new kind of interesting application areas. We are no longer required to deploy wireless networks of static sensors to monitor environmental conditions. Public sensing extends our possibilities by taking advantage of the large scale of sensors already existing in form of mobile devices. A large-scale weather forecasts can be formed with the aid of information collected through participatory public sensing [26]. Consider another example of monitoring noise pollution [34]. In urban areas, noise pollution is a major problem that affects human behaviour, well-being, productivity and health. In order to better understand the problem and develop counter measures, real world data needs to be gathered to assess the current noise climate in the city. In this example, mobile phones are used as noise sensors that actively involve users to provide additional qualitative input, e.g., annoyance rating. While these examples reveal some of the many benefits that public sensing offers, this is just the beginning to a new world of application possibilities.

1.2 Problem Formulation

In the area of public sensing, we study methods to acquire information from consumer sensing devices to provide valuable services. Such devices can be used for sensing environmental data such as temperature or air pollution. The potential of public sensing for these applications comes from the large number of people covering large public spaces while carrying their mobile phones. However, the use of these mobile sensors introduces two main challenges. Firstly, no guarantees about coverage of sensed area and data quality can be made, due to the uncontrolled mobility of people and lack of control over actions that influence this coverage. This can lead to sensing coverage gaps. Secondly, sensing should not interfere with normal usage of devices. In general, this requires efficiency in terms of energy consumption, i.e., sensing and communication of mobile nodes needs to be restricted. Therefore, we seek efficient methods for executing model-driven data acquisition, given the characteristics of a target phenomenon being sensed. The focus of this thesis is the algorithmic part of adapting optimized sensor data acquisition to public sensing systems.

1.3 Thesis Outline and Contributions

In detail, this thesis consists of the following tasks:

- Study of related work, particularly from the field of public sensing systems, model-driven sensor data acquisition and mobile ad-hoc networking.
- Design of an adaptive algorithm for selecting alternate sensor readings.
- Design of an efficient communication system for executing model-driven data acquisition.
- Preparation and configuration of a simulation environment.
- Analysis and evaluation of implemented approach.

In the following chapters, we present some background on public sensing systems, model-driven data acquisition, different positioning systems, and routing principles in mobile ad-hoc networks (Chapter 2). Then, we introduce our system model, and discuss some challenges and requirements (Chapter 3). Afterwards, we present the design specification of system in question, before moving on to the implementation details (Chapter 4). Then, we present the simulation setup, define different metrics suitable to our scenario, and discuss results (Chapter 5). Finally, we conclude this thesis and give a brief outlook on the future work (Chapter 6).

Chapter 2

Background

2.1 Public Sensing

Figure 2.1. Public sensing is one of the aspects of people-centric sensing [15].

Over the past decade, the focus of wireless sensor networking research has evolved from static networks of small resource-limited embedded devices to a people-centric approach, in which the focal point of sensing and the presentation of sensor-based information is for the benefit of the general public, rather than domain-specific scientists. As Liu et al. shows, such devices achieve a much higher coverage compared to the same number of static sensor nodes [33]. Areas that might never be covered in a stationary sensor network, can now be reached by moving sensors. The sensing capabilities found in modern consumer devices, combined with positioning systems, enabled a variety of sensing applications that can be supported. These applications are mainly driven by the needs to develop better social software to facilitate interaction and communication among groups of people, and predict the real-time change of real world to benefit human life [50]. There are many application areas, e.g., environmental monitoring, urban sensing, human health, social networks, and others.

Including human-carried devices as a fundamental block of a public sensing system raises the question to what extent people, as sensing device custodians, should be involved in sensing tasks. The community of researches has divided the spectrum of custodian awareness and involvement into *opportunistic* and *participatory* [32]. A participatory approach incorporates people into significant decision stages of the sensing system. Thus, the focus is put on tools and mechanisms that assist people in managing people-centric data. It also satisfies the requirements for privacy, and encourages involvement of the public, using social techniques. Participatory sensing puts the burden of supporting an application on the custodian, e.g., an application can request the user to rate taken picture. Since people have limited tolerance to endure interruptions on behalf of an application, this can limit the scale and diversity of applications that a purely participatory system could support. Opportunistic sensing, on the other hand, eliminates the requirement for active and conscious human participation. Instead, a sensing device is used whenever it matches the requirements of an application, e.g., a geographic location has been reached. The user is responsible only for configuring the privacy and transparency restrictions, i.e., the use of the device should not impact normal user experience. Regardless of the fact that participatory and opportunistic approaches are considered to be complementary, Lane et al. [32] believe that an opportunistic system design supports better large scale deployments and application diversity, since it does not place demands on user involvement.

Public sensing has been classified by Campbell et al. [15] as one of the three aspects of people-centric sensing (Figure 2.1). The first one, *public sensing*, covers the widest scope and focuses on gathering information that can be shared with everyone for the grater public good. The remaining two applications include: *personal sensing*, that focuses on personal monitoring (e.g., a physiological condition of the body), and *social sensing*, in which information about ourselves is shared within social and special interest groups. Every application reveals different challenges in terms of sampling data, understanding, presenting, and eventually sharing it with others. In this thesis, we focus on public sensing of environmental phenomena.

To give an intuition of what public sensing is about, consider the following examples. With wireless sensor platforms in the hands of masses, we can leverage community sensing to provide new opportunities for environmental monitoring and natural resource protection. Steed et al. [41] explored the area of monitoring carbon monoxide pollution. A set of tracked, mobile devices equipped with pollution sensors was used to collect data from pedestrians and cyclists. From analysis of raw GPS logs, they found some well-known spatial and temporal properties of CO. Further, they discovered that tracked mobile sensors allow for a fine-grained mapping of environmental sensors. Although, a study of CO was made, they further argued that these techniques can be applicable to other environmental properties such as temperature, humidity, noise and so on. The BikeNet application is another example of a project targeting environment pollution measurement [20]. Several metrics are measured to give a holistic picture of the cyclist experience, including the CO_2 levels. It facilitates public sensing by enabling multiple users merge their individual data, for example, to create pollution and allergen maps of their city.

2.2 Model-driven Sensor Data Acquisition

In this section, we provide background on model-driven sensor data acquisition, as well as basic terminology and concepts used in this thesis. We explain the idea of a prediction model and choose appropriate statistical model for the phenomena of interest. Afterwards, we specify the notion of sensing quality, to determine which set of sensors is preferable over another. We conclude with a description of an adopted sensor selection algorithm.

2.2.1 Introduction

In traditional static sensor networks it is difficult to gather all data of interest. The world consists of a set of observable phenomena that are continuous both in time and space. Sensing techniques acquire samples of physical phenomena at discrete points. Unfortunately, often they tend to acquire as much data as possible from the environment. This is a counter measure for incomplete query answers that are a result of faulty sensors, high packet loss rates, and others. Due to the energy constraints inherent to sensor networks, it is desirable to minimize the amount of sensed data. Querying all available sensors results in increased execution time and excessive consumption of resources, e.g., shorter battery life and increased network traffic.

For a given query, as more data is collected from participating devices, the total quality of query generally increases. However, querying all available nodes in the network leads to unnecessary resources usage that can result in, e.g., faster battery depletion or increased congestion in the communication medium. Since data from multiple sensors with overlapping and continuous sensing regions is usually correlated, we should try to remove redundant information by exploiting this property.

To make sensor systems more robust to these problems, many researches considered using a model for the phenomenon being sensed [19] [31] [18]. For example, a temperature of sensors geographically close to each other are likely to be correlated (*spatial correlation*), which can be captured by means of any non-trivial statistical model. Such statistical models can be used afterwards, to estimate the values of missing sensors from other correlated sensor readings, can account for biases in spatial sampling, or even identify readings that are returned by faulty sensors. Furthermore, models provide a way for optimizing the acquisition of sensor readings. A sensor should be tasked if the model itself is not sufficiently accurate to answer the query with acceptable confidence.

They key goal in public sensing is to continue to select the best subset of sensing devices to estimate a complex spatial phenomenon. Quantifying the usefulness of querying a sensor, e.g., by measuring how much a sensor reading is likely to improve the confidence of model-driven estimates, allows to find a “good” set of sensors to sample. We rely on existing information about abut the phenomenon to make sensor selections that promise to provide the maximum value of information. This minimizes the amount of sensed data, thereby reduces the power needed for consumer devices to make data measurements and transmit them through the network. Furthermore, resource limitations on consumer devices require that public sensing systems are able to adapt to changing resources availability. For example, a mobile phone might run out of memory or power before a required sensing task is completed. Furthermore, the consumer devices are not only heterogeneous in terms of resources (e.g. CPU power, memory, battery capacity), but also in terms of functionality that determines their sensing capabilities. This has a direct impact on the time and effort that must be taken to sense data with sufficient accuracy. Users of less capable devices might not meet the requirements of optimal sensing conditions.

In this thesis, we enrich interactive sensor querying with statistical modelling techniques. We consider temperature as the phenomenon of interest. A simple, yet often effective approach [19], is to assume that temperatures have a Gaussian multivariate distribution. A Gaussian model is used to capture spatial correlations between sensor readings and to derive requested readings. We use it to optimize the number of selected sensors but not the communication costs related to disseminating a query, since the topology is an unknown. With the aid of a model, we can decide when to query the sensor network to obtain new physical readings and refine estimates that yield high uncertainty. It is important to mention, however, that our approach is flexible enough with respect to the model and allows to incorporate other models with different characteristics. This forms the base of our model-driven approach for data acquisition.

2.2.2 Prediction Model

We introduce now a basic formalization of the world for model creation. The most common approach to formalize the problem of sensor selection is to treat sensor readings as *random variables* for which statistics such as mean, variance, and covariance are known or can be inferred. To map the raw sensor readings onto physical reality, a model of reality needs to be constructed. To simplify the following description, we assume that all sensors are of the same type and there is one attribute per sensor type, i.e., *temperature*.

Prediction can be performed on the basis of some predefined model, whose parameters can be learnt from historical data [40] or assigned by virtue of a priori knowledge. Consider a finite set $\mathcal{V}$ denoting all possible observations that we can make. Each $s \in \mathcal{V}$ corresponds to a position of a modelled sensor (hereafter, s is also identified with the underlying sensor). With each element $s \in \mathcal{V}$, a random variable X_s is associated. This random variable reflects a physical reading taken. For a subset $\mathcal{A} \subseteq \mathcal{V}$ with $|\mathcal{A}| = k$, a realization $x_\mathcal{A}$ is a vector of values $x_\mathcal{A} = (x_{s_1}, \ldots, x_{s_k})$, where x_{s_k} denotes a sensor reading taken at sensor s_k. A *prediction model* is a joint *probability distribution function* $P(X_1, \ldots, X_{|\mathcal{V}|})$ (a priori pdf) over all random variables $X_\mathcal{V}$. It assigns a probability $P(X_\mathcal{A} = x_\mathcal{A})$ for each joint realization of vector $x_\mathcal{A}$.

Suppose that we observe the value of attribute $X_\mathcal{A}$ to be $x_\mathcal{A}$, we can use Bayesian rule to condition our joint pdf $P(X_\mathcal{V})$ on this value, obtaining:

$$P(X_{\mathcal{V}\setminus\mathcal{A}} \mid X_\mathcal{A} = x_\mathcal{A}) = \frac{P(X_{\mathcal{V}\setminus\mathcal{A}})}{P(X_\mathcal{A} = x_\mathcal{A})}.$$

$P(X_{\mathcal{V}\setminus\mathcal{A}} \mid X_{\mathcal{A}} = x_{\mathcal{A}})$ is also refereed as the posterior density given the observation $x_{\mathcal{A}}$. This posterior distribution encodes the uncertainty in the prediction and can be used to make decisions, e.g, to interrogate the sensor network for updated readings. The process of computing the posterior distribution of variables given evidence (observed variables) is called probabilistic inference. In general, we make a vector of observations $x_{\mathcal{A}}$, and, after conditioning on these observations, obtain $P(X_{\mathcal{V}\setminus\mathcal{A}} \mid X_{\mathcal{A}} = x_{\mathcal{A}})$, the posterior probability of our set of attributes $X_{\mathcal{V}\setminus\mathcal{A}}$, given $x_{\mathcal{A}}$. Having $P(X_{\mathcal{V}\setminus\mathcal{A}} \mid X_{\mathcal{A}} = x_{\mathcal{A}})$, we can derive values of unobserved attributes $X_{\mathcal{V}\setminus\mathcal{A}}$ with some confidence.

2.2.3 Modelling Sensor Data

Many different mathematical approaches are used to model the phenomena of interest, which vary in accuracy and complexity. Yoon and Shahabi [49] mathematically model the property of spatial correlation using measured sensor data from different environments. According to their findings, linear correlation model and spherical model fit best their requirements to represent stochastic dependencies of temperature and light intensity phenomena. To give more intuition, we explain the modelling concepts on an example of simple linear correlation.

When we ask questions such as "*Is an attribute X is related to Y?*", or "*Does attribute X predict Y, and how well?*", we are interested in measuring and better understanding the relationship between two variables. Linear correlation is a straight-line relationship between two variables, which measures the extent to which values for one variable can be predicted from values of another variable. The most familiar measure of dependence between two quantities is the Pearson product-moment correlation coefficient. Given two variables X and Y with expected values μ_X and μ_Y and standard deviations σ_X and σ_Y, the Pearson coefficient $r_{X,Y}$ is defined as:

$$r_{X,Y} = \frac{E\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sigma_X \sigma_Y},\ r_{X,Y} \in \langle -1; 1 \rangle$$

where E is the expected value of a random variable. The closer the coefficient is to -1 or 1, the stronger is the correlation between the variables.

In the earlier considerations, we set the random variable X_s to model the temperature at some sensor $s \in \mathcal{V}$. Other possible probability distribution that has been considered to model a temperature as a spatial process is the multivariate Gaussian distribution [19]. A multivariate Gaussian distribution (hereafter, simply Gaussian) is the natural generalization of the unidimensional normal pdf, known as the "bell curve", to higher dimensions:

$$P\left(X_{\mathcal{V}} = x_{\mathcal{V}}\right) = \frac{1}{(2\pi)^{\frac{n}{2}} \left|\Sigma_{\mathcal{V}\mathcal{V}}\right|} e^{-\frac{1}{2}(x_{\mathcal{V}}-\mu_{\mathcal{V}})^T \sum_{\mathcal{V}\mathcal{V}}^{-1}(x_{\mathcal{V}}-\mu_{\mathcal{V}})},$$

where $\mu_{\mathcal{V}}$ is the mean vector and $\sum_{VV}$ is the covariance matrix. If we consider a subset, $\mathcal{A} \subseteq \mathcal{V}$, of the attributes, we simply select entries in μ and Σ corresponding to these attributes, and drop the other entries obtaining a lower dimensional mean vector $\mu_{\mathcal{A}}$ and covariance matrix $\Sigma_{\mathcal{A}\mathcal{A}}$. This is called marginalization, or projection, of given pdf $P(X_{\mathcal{V}})$

to a density over a subset of attributes $X_{\mathcal{V}}$. In the broader meaning, it is the process of deriving the joint distribution of a subset of $X_{\mathcal{V}}$. If we condition a Gaussian on the value of some attributes, the resulting pdf is also Gaussian. This makes marginalization pretty straightforward. The mean and covariance matrix of this new Gaussian can be determined by simple matrix operations.

Suppose that we observed a set of sensors measurements $X_{\mathcal{A}} = x_{\mathcal{A}}$. Then we can predict the value of defined attribute at any sensor $y \in \mathcal{V}$ conditioned on those measurements, $P(X_y \mid x_{\mathcal{A}})$. The distribution of X_y is a Gaussian whose mean $\mu_{y|\mathcal{A}}$ and variance $\sigma^2_{y|\mathcal{A}}$ are given by:

$$mean : \mu_{y|\mathcal{A}} = \mu_y + \Sigma_{y\mathcal{A}}\Sigma^{-1}_{\mathcal{A}\mathcal{A}}(x_{\mathcal{A}} - \mu_{\mathcal{A}}),$$

$$variance : \sigma^2_{y|\mathcal{A}} = \Sigma_{yy} - \Sigma_{y\mathcal{A}}\Sigma^{-1}_{\mathcal{A}\mathcal{A}}\Sigma_{\mathcal{A}y},$$

where $\Sigma_{y\mathcal{A}}$ is a covariance vector with one entry for each $u \in \mathcal{A}$ with value Σ_{yu}, and $\Sigma_{\mathcal{A}y} = \Sigma^T_{y\mathcal{A}}$.

This distribution represents spatial correlation of temperature in our sensor network. The advantage of the Gaussian is that it can be compactly represented (only the mean vector $\mu_{\mathcal{V}} \in \mathbb{R}^n$ and the covariance matrix $\sum_{VV} \in \mathbb{R}^{n \times n}$ have to specified or derived from data) and allows efficient inference, since we only have to perform basic matrix operations. Finally, it is more accurate than trivial models based on linear correlations.

2.2.4 Quantifying Sensing

To reduce the amount of transmitted data and conserve energy resources of participating nodes, we have to decide carefully which most informative sensors to query. The core problem of selecting a set of sensors is how to predict the information gain of sensors before obtaining readings. In practice, the prediction must be based on information currently available in the model. We define a notion of sensing quality to determine, which set of sensors is preferable over another. Consider a set function $F(\mathcal{A}) : 2^{\mathcal{V}} \to \mathbb{R}$, which assigns each subset $\mathcal{A}$ of sensors a real number value. Empty subset of sensors provides no information $F(\emptyset) = 0$. The goal is to select a subset $\mathcal{A}$ that maximizes the information utility function $F(\mathcal{A})$. Gathering more information should never reduce the sensing quality but at some point, adding a sensor to an existing set of most informative sensors is not as beneficial as if we would have chosen few sensors so far.

The information utility function can be defined in many different ways depending on the context, e.g., for spatial prediction, interesting metrics are the entropy criterion, mutual information and predictive variance, whereas for facilitating a decision making process, the decision theoretic value of information should be considered [30].

One sensing quality function that is commonly used in practice, is the *Shannon entropy* criterion, which measures the randomness of a given random variable. Mathematically, entropy can be defined as

$$H(X_{\mathcal{A}}) = -\int \left(p\left(x_{\mathcal{A}}\right)\right) logp(x_{\mathcal{A}}) dx_{\mathcal{A}},$$

where p is the mass probability function, $X_{\mathcal{A}}$ is the random vector of selected sensors, and $x_{\mathcal{A}}$ is its realization. Generally speaking, the smaller the entropy is, the more certainty we have about the value of the random variable. Therefore, the information utility measure based on entropy could be defined as

$$F(\mathcal{A}) = H(X_{\mathcal{A}}),$$

since the goal is to find sensors that reveal the highest degree of uncertainty. We are maximizing the entropy $H(X_{\mathcal{A}})$ of a subset of sensors. This selects the most uncertain sensors.

In practice, we are more often interested in selecting a subset of sensors that most effectively reduce the uncertainty about unavailable sensors. We thus can set our information utility function to maximize the information gain. This is known under the *mutual information* criterion and defined as

$$F\left(\mathcal{A}\right) = H\left(X_{\mathcal{B}}\right) - H\left(X_{\mathcal{B}} \mid X_{\mathcal{A}}\right) = I\left(X_{\mathcal{B}}; X_{\mathcal{A}}\right),$$

where $\mathcal{B} \subseteq \mathcal{V}$ is a subset of unobserved sensors whose uncertainty we want to reduce. This quantity measures the mutual dependence of random variables, i.e., it measures how much knowing some of these variables reduces uncertainty about the others. If all random variables are independent, then neither one of them contains any information about the others, therefore mutual information is zero.

If we are interesting in predicting temperature at all unavailable sensors, we can set $\mathcal{B} = \mathcal{V} \setminus \mathcal{A}$, i.e., the unavailable sensors depend on the observed sensors $\mathcal{A}$. Our goal is to maximize

$$F\left(\mathcal{A}\right) = H\left(X_{\mathcal{V}\setminus\mathcal{A}}\right) - H\left(X_{\mathcal{V}\setminus\mathcal{A}} \mid X_{\mathcal{A}}\right).$$

Mutual information requires an accurate estimate of the joint model $P\left(X_{\mathcal{V}}\right)$, while the entropy criterion only requires an accurate estimate at the selected sensors, $P\left(X_{\mathcal{A}}\right)$. Based on the findings of Guestrin et al. [22], it can be concluded that using the mutual information criterion is often a better approach for sensor selection in Gaussian models than entropy, both qualitatively and in prediction accuracy.

2.2.5 Offline Sensor Selection

The offline sensor selection task is about choosing the most informative sensors that have not been incorporated into the model yet. In the earlier considerations, we defined a notion of sensing quality and choose mutual information as our primary metric. Guestrin et al. have done an extended research on many interesting sensing problems that satisfy the diminishing returns law [22]. Adding a sensor to an existing set of sensors helps more, if we have chosen few sensors so far, and less if we selected already many. Mutual information satisfies this property. For such information utility functions, when attempting to select the best subset of sensors, the simple greedy algorithm (Algorithm 2.1) finds a solution which guarantees a constant-factor approximation of $(1-1/e)$ of the optimal sensing quality. We have to require only that the number of chosen sensors is small in comparison to the number of all modelled sensors.

Algorithm 2.1 Greedy algorithm for maximizing information utility function [22].

```
Input: Submodular function $F : 2^{\mathcal{V}} \to \mathbb{R}, k$
Output: Subset $A \subseteq \mathcal{V}$
1: $\mathcal{A} \leftarrow \emptyset$
2: for $(j = 1 \to k)$ do
3:     for $(y \in V \setminus A)$ do
4:         $\delta_s \leftarrow F(\mathcal{A} \cup \{s\}) - F(\mathcal{A})$
5:     end for
6:     $s* \leftarrow argmax_{s \in \mathcal{V} \setminus \mathcal{A}} \delta_s$
7:     $A \leftarrow A \cup \{s*\}$
8: end for
```

We adopt the mutual-information-based greedy sensor selection algorithm for our sensor selection task. The algorithm starts with the empty set $\mathcal{A} = \emptyset$, and adds iteratively a sensor that increases the sensing quality most. The information utility function F is given in form of a mutual information criterion and satisfies the diminishing returns property.

2.2.6 Related work

Krause [30] performs an extended research of observation selection problems. He presents a novel class of approaches for sensing and information gathering. Most importantly, he shows that many problems reveal structural properties such as intuitive law of diminishing returns, locality and conditional independence.

Deshpande et al. [19] proposes a model-based query prototype called BBQ, which uses a belief model based on time-varying multivariate Gaussians. Within proposed framework, the authors demonstrate that such models can help provide meaningful information and lead to efficient performance gains, in contrast to traditional data acquisition techniques.

In [22] Guestrin et al. considers near-optimal sensor placements modelled as Gaussian processes. The concept of mutual information is used to measure the effect of sensor placement on the posterior uncertainty of the model. Firstly, a proof that maximizing the expected quality is a NP-complete problem is given. Following that, the authors exploit submodularity of mutual information and design an approximation algorithm with a bounded error.

Yoon and Shahabi [49] investigates spatio-temporal correlations to improve existing in-network aggregation techniques. The Clustered AGgregation (CAG) forms clusters of nodes sensing similar values. The study confirmed that CAG can perform energy efficient in-network aggregation leveraging from spatio-temporal correlations, since the number of transmissions is significantly reduced, and the query uncertainty is bounded by a predefined threshold.

2.3 Positioning Systems

In mobile ad-hoc networks, position is of great importance in the context of geographic routing protocols [12]. The routing decision at each node is mostly based on the current positions of the forwarding node's neighbours and the geographic information of the destination. This localized decision scheme reduces the amount of traffic generated in the process of route discovery and sometimes significantly decreases the amount of state information that each participating node has to maintain. Before such decision can be made, a geographic position of a node must be somehow available.

The term "position" have many different meanings that vary across the applications. On an example of *Global Positioning System* (*GPS*), a physical location is provided in terms of latitude, longitude, and altitude. Modern positioning systems have pushed out many older terrestrial radio navigation systems, e.g., *LORAN* (*LOng RAnge Navigation*). In LORAN the locations are usually given with respect to fixed beacon locations [4]. The position of a mobile node can be determined in two ways: in a form of actual geographic coordinates, as obtained through the use of GPS, or virtual relative coordinates, which is the result of introducing a local coordinate system, e.g., with respect to the network topology [51].

There are many measurement techniques, which use known characteristics of signal propagation, to obtain a fine-grained location of an object. The most common measurement methods are the received signal strength, time-based, and directional methods [12]. The first method is used primarily to determine proximity. It makes use of signal attenuation, which can significantly vary in different environments, thus making it inappropriate for accurate distance measurements. Time-based methods measure distances by recording the time, which takes for a signal to be sent from the transmitter to the receiver. Directional methods use angle of arrival (AoA) or direction of arrival (DoA) to compute locations.

The GPS utilizes the concept of one-way time of arrival (TOA) ranging [27], and as a consequence it employs a time-base method. This is used as our primary positioning system, thus it is important to understand the innerworkings of GPS. The GPS satellite constellation nominally consists of 24 satellites arranged in 6 orbital planes with 4 satellites per plane. The TOA ranging concept is about measuring the time it takes for a signal transmitted by a satellite at a known location to reach a user receiver. This time interval, which is the signal propagation time, is used then to obtain the satellite-to-receiver distance. By measuring the propagation time of the signal broadcast from multiple satellites at known locations, the receiver can determine its position. This technique requires that the user receiver contains a clock that is synchronized with the satellite clocks. In reality, the system time is not accurately known, thus four measurements are required to determine user latitude, longitude, height, and receiver clock offset from internal system clock. GPS accuracy can be expected to be within 5 meters of true position in open sky settings and can decrease to 10 metres under canopies [46].

Other examples of positioning systems using time-based techniques are GALILEO [2], Russian GLONASS [7], and Chinese BeiDou [1]. The GALILEO satellite system is a project by the European Union, independent of GPS and dedicated specifically for civilian use worldwide. It is planned to be a 30-satellite constellation, fully compatible with the GPS. When completed, GALILEO is expected to provide multiple levels of service to users throughout the world. Among five defined services, the most appropriate one for public sensing is the *Open Service* (OS), since it is free of charge and meant for public use.

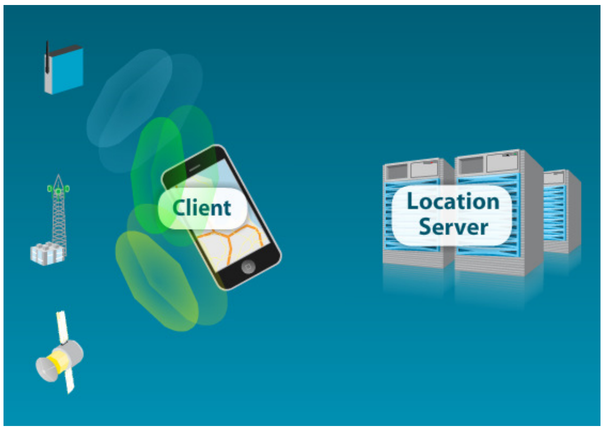


Figure 2.2. Determining location using Skyhook's system [8].

Many smartphones now include embedded GPS capabilities to locate a user in the event of an emergency or to support a wide variety of location-based services. These devices are used quite often in areas where GPS signals might be very weak (e.g., indoors or dense urban areas), making the demodulation of the GPS navigation data very difficult or even impossible. With mobile network assistance, however, it is possible to determine the location of a mobile phone under difficult circumstances. Assisted GPS (A-GPS) methods grew out of the need to simultaneously reduce the time to produce a first position fix (TTFF) and increase sensitivity further. A standalone GPS receiver needs orbital information of the satellites to calculate the current position. This information includes *ephemeris* data, used to calculate the position of each satellite in orbit, and information about the time and status of the entire satellite constellation, called the *almanac*. The network can accurately predict the GPS signal, that the handset receives, and convey that information to the mobile, greatly reducing search space size and shortening the TTFF from minutes to a second or less.

If a phone is not equipped with a GPS receiver or cannot obtain a fix for some reason, there are other advanced methods, including Wi-Fi positioning and a cell tower triangulation, that can help achieve the same goal. All Wi-Fi access points constantly broadcast signals that can travel hundreds of meters in all directions. Cellular towers signals, on the other hand, can travel thousands of meters. Skyhook [8] gathers these information with advanced hybrid positioning algorithms using basically two methods. Company employees drive the streets of medium-to-large cities around the world in vehicles that have sensitive GPS and Wi-Fi receivers. They also collect snapshots of Wi-Fi signals that their customers transmit in order to obtain a fix. When a user requests its position, Skyhook is able to quickly and accurately calculate location regardless of physical environment. It maintains a massive reference database of Wi-Fi access points and cell tower IDs. The lack of a GPS module in the first generation of iPhones was a surprise, given the fact that the device was integrated with Google Maps [6]. It turned out that Skyhook's system was used by the iPhone's Location Service (including the Maps software powered by Google Maps).

2.4 Routing in MANETs

In this chapter, we discuss some of the characteristic features of mobile ad-hoc networks (MANETs) that make them distinct and provide a brief overview of different routing aspects in such networks. We focus on position-based routing, and explain in details the functionality of adopted routing scheme.

2.4.1 Basic Principles of MANETs

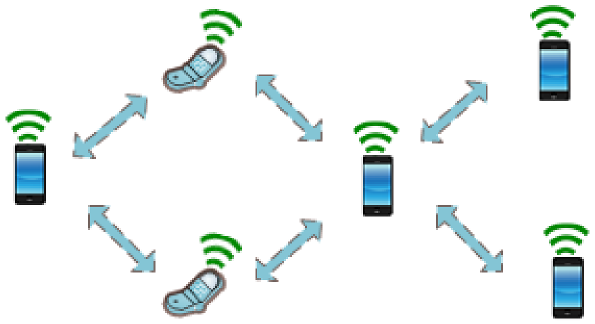


Figure 2.3. An example of a Mobile Ad-hoc Network (MANET).

Mobile ad-hoc networks are self-forming and self-organizing collections of nodes characterized by dynamic topologies with no fixed infrastructure [9]. Collaboration among nodes is fundamental to the function of a MANET. However, every mobile node is an autonomous and independent wireless device. A sample model of a mobile ad-hoc network is presented in Figure 2.3. Now, we discuss some properties of MANETs in the context of information exchange [13] [47].

Communication paradigm - a mobile ad-hoc network is built on the fly, where a number of nodes work in a cooperation without the engagement of any fixed infrastructure or base station. No node is given priority over another, and transmission can occur between any pair of nodes in range. Mobile nodes communicate in a bidirectional manner through multiple wireless links with transmission ranges of up to hundreds of meters. It is important the wireless communication medium efficiently is shared efficiently, since transmissions from different nodes that use the same communication channel at the same time may result in either a truncated message, or corrupted data received. Each node serves as a host, which generates user and application traffic, and a router, which carries out network control and routing protocols. A node communicates with other nodes that are outside its transmission range using a multi-hop routing strategy. All mobile nodes have globally unique IDs (e.g., MAC address or IP address), making the routing problems address-centric.

Resource constraints - participating entities are resource-constrained. Nodes are typically hand-held devices powered by rechargeable batteries. Power-aware and energy–efficient routing algorithms can improve the performance of the system and should be preferred. Nowadays, a typical smartphone has a strong processor, sufficient amount of memory to run more demanding applications and rechargeable battery that can last for several hours. Although the nodes are battery-powered, the energy consumption is of secondary importance (a difference to the wireless sensor networks), since each device could have its battery recharged or replace when needed. The most important aspect is to assure a certain quality of service (QoS) and scalability in context of a changing topology, limited bandwidth, and limited transmission power. A routing protocol, apart from providing a high delivery ratio of packets, should also be aware of the delay and throughput for the route of a source-destination pair, and be able to verify its longevity.

Dynamic network topology - the nodes in a MANET can move almost without any constraints, thus mobility becomes a great challenge. Wireless links are established in an arbitrary fashion. Network topology is more vulnerable to changes, making discovered routes prone to error. It is very difficult, if not impossible to use traditional wired network's routing mechanisms. In MANETs nodes can have different levels of scattering leading to various levels of density. The density is defined as the number of nodes in a predefined area. This has a major influence on how robustness is achieved. Generally, the data-link layer helps to ensure the reliability of one-hop transmissions. Thus, the correctness of routing protocol is an important part of the design.

We mentioned so far unique aspects of MANETs that make the task of routing complicated. The nodes move freely and randomly in the network. Every node can act as a host and a router at the same time. To communicate with each other, nodes use wireless communication without any fixed infrastructure, and are mainly battery-powered. Considering these special properties of MANETs, when thinking about frequent topology changes, we generally distinguish two different approaches for routing: *topology-based* and *position-based* routing [35].

Topology-based routing protocols exploit information about the links that exist in the network to establish and maintain source-destinations paths. This leads to a question whether the nodes should track information about routes to all possible destinations, or instead, track only destinations of immediate interest. The routing protocols are further divided into: *proactive*, *reactive* and *hybrid* approaches. Position-based (geographic) approaches build on the proactive and reactive techniques, eliminating some of the limitations by incorporating geographical information in routing.

2.4.2 Topology-based Routing Techniques

Proactive protocols employ classical routing strategies such as distance-vector routing (e.g., DSDV [38]) or link-state routing (e.g., OLSR [17]). They are often refereed to as table-driven, since they continuously exchange topological information among the network nodes. Thus, information about available routes is accessible immediately, even if they are currently not used. The main drawback of this approach is that the cost of maintaining routes can be very high, e.g., if the network topology changes too frequently.

Reactive protocols, on the other hand, establish new routes to destinations on demand basis and maintain only those, which are currently in use (e.g., DSR [25], AODV [37]). However, they still reveal some limitations. Firstly, before the packet can be transmitted, a route discovery has to be typically performed. As a consequence, the initial packet to be transmitted on the previously unknown route, is delayed. Secondly, significant amount of traffic can be still generated under frequent topology changes, even though the routing algorithm is restricted to the routes that are currently in use. Finally, there is a chance that the packet en route will get lost, if the current route breaks.

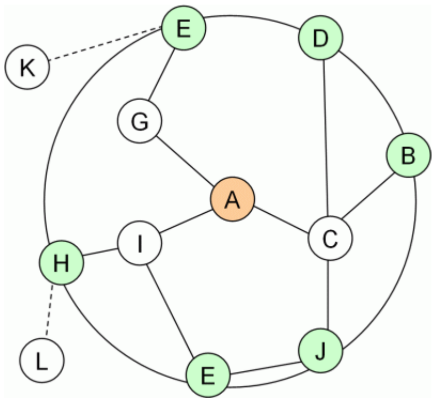


Figure 2.4. An example of a routing zone of node A with radius equal to 2 hops. E, D, B, J, E and H are the border nodes.

Hybrid routing protocols, such as *Zone Routing Protocol* (*ZRP*) [23], combine some characteristics of proactive and reactive protocols, since a mixture of both might yield a better solution. In ZRP, each node proactively maintains local routing information based on the periodic exchange of neighbour discovery messages. This local routing information is termed as routing zone and is not necessarily limited to a single-hop transmission range. The size of a zone is given by a routing zone radius. More precisely, a node's routing zone is composed of nodes whose minimum distance in hops from the node in question is not greater than the zone radius. An example of a routing zone is in Figure 2.4. Each node maintains its own routing zone, thus the routing zones of neighbouring nodes overlap. The ZRP's global route discovery mechanism is reactive. A route query is initiated, on demand, when no route is locally available. The query is then relayed to a border node where the zones overlap. Upon recipient of a route query packet, a node checks if the destination lies in its zone, or if a valid route to it, is available in its route cache. If the answer is positive, a route reply is sent back to the source via the reverse path. If not, the border node relays the query again.

A topology-routing protocol can also incorporate additional information about geographic positions of nodes, if available, to improve its efficiency. Instead of performing a network-wide search, it could employ directional flooding of control and data packets towards the final destination. A review of routing protocols for mobile ad-hoc networks can be found in [9]. In this thesis, we focus on the position-based routing because it is the base point for our public sensing system.

2.4.3 Position-based Routing

Position-based routing relies solely on geographic position information of nodes in the network. A dominant group of position-based routing techniques do not require establishing and maintaining end-to-end paths, as they mostly use positions of one-hop neighbours to forward packets. Topology changes affect only local knowledge of a sender, and therefore have a lesser impact on geographic routing than compared to topology-based routing. This makes geographic protocols highly scalable.

The main prerequisite for a geographic routing algorithm is that every node in the network knows its position. This can be achieved through different positioning techniques, e.g., GPS. Secondly, a sender must be able to figure out a geographic position information of the destination. Typically, a *location service* is responsible for maintaining up-to-date information about positions of nodes in the network [42]. The sender directly queries a location service for geographic position of the destination. This information is afterwards fixed in the header of a packet. It might be updated by an intermediate node though, if the node happens to know a more accurate position of the destination.

Geographic forwarding is a forwarding mechanism that moves a packet to gradually approach and eventually reach the intended destination. Semantics of a destination can vary among different approaches: it can be specified as a single position, list of positions or as a geographic region [36]. These semantics determine different types of delivery methods:

- *Geographic unicast routing* - uses geographic position information of nodes in the network to deliver data packets from a single source to a single destination. This is the most simple approach. It usually operates in one of the following two modes: *greedy forwarding* and *recovery mode* (void handling). These techniques are extracted from their respective geographic forwarding protocols and are discussed later in this chapter.
- *Geographic multicast* - is used to deliver packets from a single source to a list of destinations (Figure 2.5). Multicast routing could be trivially implemented using either

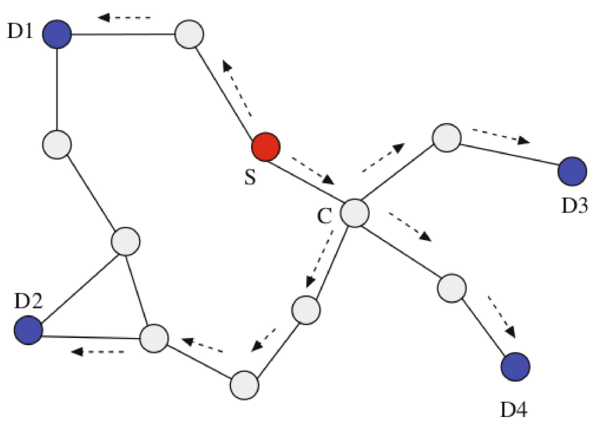


Figure 2.5. An example of geographic multicast.

network-range flooding, or unicast routing to each destination. Both approaches are correct but inefficient in terms of network resources, since paths from the source to each different destination may have many links in common. Thus, a multicast routing focuses strongly on minimizing the consumption of network resources by reducing redundant links.

- *Geocast* - enables the delivery of data packets to all nodes that are within a specified region. In a geocasting protocol, the members of a multicast group are determined by their physical locations. The source specifies a region, called *the geocast region*, and a geocasting protocol tries to deliver data packets only to nodes in this region. A geocasting protocol unusually operates in two major steps, as in Figure 2.6. Firstly, a data packet is forwarded from a source to one or more nodes in the geocast region. Secondly, a packet is distributed from one or more nodes in the geocast region to all nodes in this region.

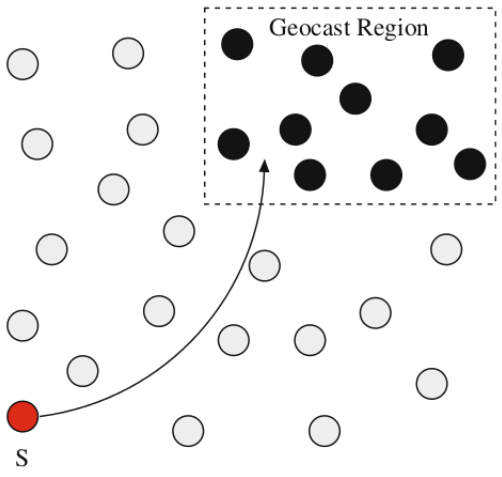


Figure 2.6. An example of geocasting.

2.4.4 Geographic Unicast Routing

Geographic unicast routing delivers data packets to a single destination and usually operates in one of the two following modes: *greedy forwarding* and *void handling*. In the greedy mode, a data packet is forwarded to a one-hop neighbour that is located closer to the intended destination than the forwarding node itself. The selection of the neighbour depends on the optimization criteria of the algorithm. If greedy forwarding fails to move the packet further due to the presence of communication voids, the recovery mode is triggered. Since a topologically valid path may still exist, the sender attempts to route the the data packet around the void.

Generally, there is a number of different greedy forwarding strategies a sender can use, e.g., a next-hop node is chosen as the nearest neighbour with positive progress. In order to apply a specific strategy, the sender must know the positions of its neighbours. This is typically achieved by employing a beaconing scheme. Each node periodically transmits a beacon, which includes its own position and possibly other information (e.g., nodes residual energy). The beaconing rate is closely related to the accuracy of positions. Even though the beaconing frequency can be adapted to the degree of mobility, there is always a fundamental problem of outdated position information. Inappropriately configured beaconing rate might lead to a significant decrease in the packet delivery ratio in dynamic topologies.

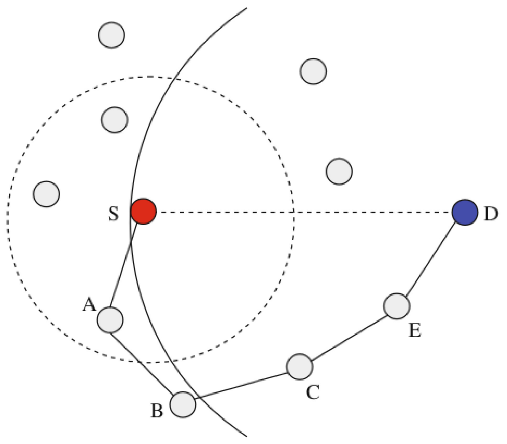


Figure 2.7. A communication void, with respect to the destination D, occurs at node S where greedy forwarding fails.

A greedy forwarding algorithm, that considers link reliability when sending packets to next-hop nodes, has to access the MAC layer to obtain information about possible packet errors. Thus, a typical geographic routing protocol that uses a greedy forwarding algorithm is actually a cross-layer protocol.

The recovery mode is triggered when the greedy forwarding reaches a local maximum from which it cannot recover (Figure 2.7). There are many different approaches to handle voids [16], including planar-graph-based, topology-based, link-reversal-based, geometric, heuristic, and hybrid. We focus on planar graphs, since adopted routing protocol uses them in void handling.

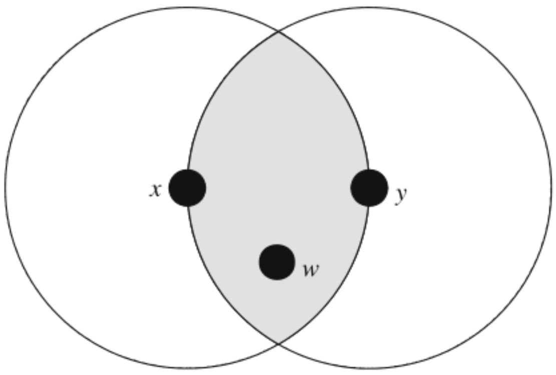


Figure 2.8. The RNG planarization algorithm.

In graph theory, a planar graph is a graph in which no two edges intersect. However, underlying graph of a mobile network is usually not planar. Additional algorithms are required to extract a planar subgraph from the original graph. Existing approaches include distributed planarization algorithms such as *Relative Neighbourhood Graph* (*RNG*) and *Gabriel Graph* (*GG*). In Figure 2.8, an example of RNG planarization algorithm is given, where an edge (x, y) remains in the planar subgraph if no witness node w is located within the shaded area. Planarization is essential for void handling. Otherwise, routing paths may contain loops.

2.4.5 Greedy Perimeter Stateless Routing

Greedy Perimeter Stateless Routing (*GPSR*) is a responsive and efficient routing protocol for datagram[1] mobile, wireless networks. It exploits the correspondence between geographic position and connectivity in a wireless network, and supports a variety of different network classes, e.g., rooftop, ad-hoc, sensor and vehicular networks. Unlike traditional Internet routing algorithms, e.g., *Distance Vector* or *Link State*, which use graph-theoretic notions of shortest paths and transitive reachability to find routes, GPSR maintains only states from immediate neighbours, which is sufficient to make correct forwarding decisions, without any additional topological information. We adapted this protocol to work with our public sensing system.

The state is described by a geographic position, which a mobile node can obtain through a positioning system (e.g., GPS). A simple proactive beaconing algorithm provides all nodes with their neighbour's positions: periodically, each node broadcasts a beacon, containing its own IP address and position. An intermediate node makes packet forwarding decisions based on its knowledge of the neighbours' positions and the destination's position, inserted in the packet header by the originator of the packet. By default, packets

[1]In other words, connectionless networks, where the communication between the two sites is on a one-off basis. The packet contains the full addressing information needed to transmit it.

are greedily forwarded to a neighbour that allows for the greatest progress to the destination, i.e., the distance to the destination's position is minimal from the chosen node. When no such neighbour exists, perimeter forwarding is used to recover from a local void. The packet traverses the face of the planarized local topology subgraph by applying the right-hand rule, until greedy forwarding can be resumed. The right-hand rule is guaranteed to find a path from the source to the destination, if there exists at least one such path in the original non-planar graph. However, the edge to the chosen next-hop must not intersect the line defined by the perimeter entry position and the final destination. In perimeter mode, if at any time node finds another candidate, from which the distance to the final destination is smaller than from the perimeter entry position, then the packet is returned to greedy mode again.

Perimeter routing is a complete void recovery solution proposed in GPSR, consisting of a distributed planarization algorithm that uses either RNG or GG, and a planar traversal algorithm. All nodes execute the distributed planarization algorithm periodically. It is important to notice that the decision, whether an edge is within the planar subgraph or not, can be made locally by each node, since each node knows the positions of all its neighbours. When a packet arrives at a local maximum, the planar traversal algorithm is used in GPSR to handle the void. The header of the packet usually carries additional information such as the position of the node where it entered the recovery mode, the first edge traversed on the current face, and the position of the last intersection, where a face change occurs. Therefore, each node can make all routing decisions in a highly localized manner.

Chapter 3

System Model

In this chapter, we present a system model and identify the assumptions made in this thesis. Afterwards, we discuss challenges and present requirements for our public sensing system.

3.1 System Model

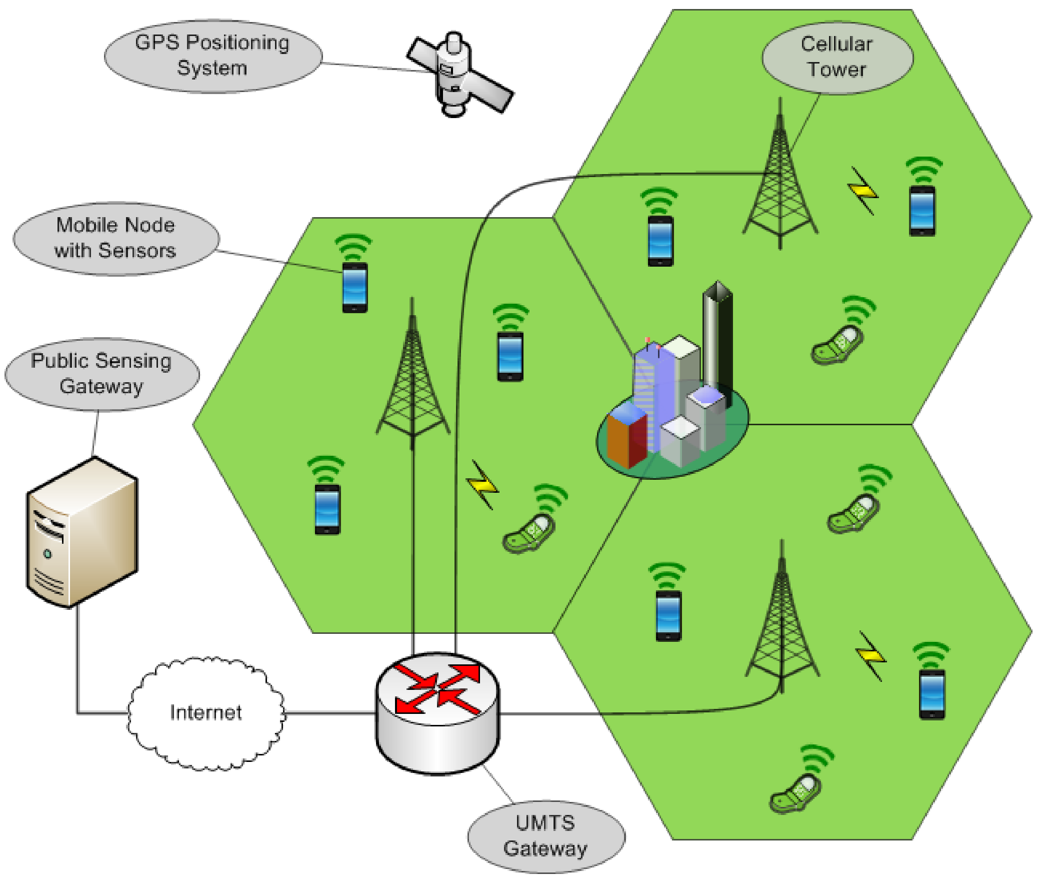


Figure 3.1. Overall system model.

We assume large-scale public sensing systems composed of possibly thousands of mobile nodes. An overview of our hybrid network is shown in Figure 3.1. The main elements of our Public Sensing System (PSS) are the Public Sensing Gateway (PSG), and a number of mobile nodes. The PSS requires that every node carries or has access to a set of environmental sensors. To extend sensing capabilities of a node (e.g., measure temperature), other external sensors can be easily connected using, e.g., a Bluetooth interface. For simplicity, we assume that each device carries required sensors. The PSG is an interface for users and a variety of applications to submit queries. Its main responsibility is to distribute queries in the system and report back sensor readings. It is connected to the Internet via a wired high-speed connection.

A mobile node supports two types of communication interfaces. A WiFi interface is used for a fast bidirectional communication between nodes over short distances, and an UMTS interface is used at any time for Internet access. We assume also that each node operates in an ad-hoc wireless mode and is aware of its communication range, thus can determine its neighbouring nodes. Furthermore, every node knows its current position through a built-in GPS receiver. The receiver itself does not provide ideal position fixes. We assume that an error for position fixes follows a two-dimensional normal distribution with a mean equal to zero and a standard deviation of σ. Moreover, every node runs our dedicated public sensing software, which comprises the sensing application and triggers adopted routing scheme. A node has also the ability to recharge itself. These characteristics form base requirements for a mobile node to be capable of executing a sensing task.

We use GPSR protocol to route packets between mobile nodes. In order for GPSR to work correctly, it is also required that a node can reach all other nodes within its radio range. The communication has to be bidirectional because IEEE 802.11 MAC protocol sends an acknowledgement for all unicast packets [43]. If a sender does not detect an ACK within given timeout, it tries to retransmit a packet reducing the link throughput and fairness, and increasing the number of collisions. Two-way communication over WiFi can be relaxed, what is beyond the scope of this thesis.

The whole playground is composed of smaller units called regions. Information about supported regions is fixed and stored in the PSG. We assume that every mobile node that executes sensing tasks must be associated with exactly one supported region. Otherwise, it is not considered as participant of our system and simply ignored. The playground can be divided in a variety of ways, from custom settings (e.g., grid-based), to infrastructure specific configurations (e.g., UMTS antenna coverage). A participating node has always information about its current region assignment. If a node moves out of a region, it requests a new assignment from the PSG. The concept of regions limits the scope of a query to only these sensors, which are bounded by a respective region.

When an application makes a request to the PSG, it needs to specify an area of interest. Philipp et al. [39] provided a flexible sensor abstraction in a form of *virtual sensor*. A virtual sensor v is described by its position, a sensing radius r, a type of reading (e.g., temperature) and a sensing period. A node n is said to cover v (fill the role of v), iff $\delta(n, v) \leq r$, where δ is used to denote the distance between two objects.

A node n takes a reading (on behalf of a virtual sensor), iff n can fill the role of specified virtual sensor. A sensing radius of a virtual sensor is assumed to be less than or equal to the communication range of a mobile node. A virtual sensor outputs a stream of virtual readings immediately upon accessing the sensor. An application defines an area of interest through a set of virtual sensors V. This approach provides a very flexible way to specify queries independently of node's mobility.

An application explicitly specifies a set of virtual sensors that form the area of interest. To get meaningful results, it is the application responsibility to define a "good" level of coverage, i.e., specify a sufficient number of virtual sensors. The PSG is responsible for distributing application query in the system. If a query spans more than one region, it is decomposed into many partial queries, each bounded by a respective region. Afterwards, queries are delivered to regions, which form entry points for queries. Mobile nodes are responsible for distributing locally a partial query using adopted routing scheme. We have no possibility of knowing in advance, which node can fill the role of specified virtual sensors, if at all. The nodes that can fill the role of virtual sensors are determined dynamically. After a reading is taken, it is sent back to the gateway via respective entry node. Finally, results are filtered and reported back to the application. We use WiFi communication wherever possible, since a study about energy consumption in mobile phones revealed that ad-hoc communication (e.g., WiFi) consumes less energy per packet compared to the cellular technologies (e.g., UMTS) [11]. To measure the amount of energy used to send packets over WiFi, an energy model presented by Xiao et al. is used [48].

Sensing resources form the foundation of the entire system. The phenomenon of interest monitored by the environmental sensors follows a known spatial distribution. In our considerations, we focus on monitoring temperature and model spatial correlations among sensor readings with the aid of a multivariate Gaussian distribution [19]. The model is learnt from historical data using standard algorithms [40].

In the following, we assume that our system is deployed in an urban setting. Moreover, the most local infrastructure components (e.g., cellular base stations) belong to different administrative domains, therefore we have no influence over their functionality, nor can we run our algorithms on these components. We also assume that the mobile network is formed by interconnected nodes, i.e., every node has several other nodes within its WiFi range.

3.2 Challenges and Requirements

The public sensing paradigm has several advantages. It enables large spatial coverage, which is not easy to achieve with a dedicated system because of infrastructural costs or jurisdictional issues. Resources can be shared opportunistically, hence the costs of sensing can be amortized over a large number of applications that run on the mobile devices. Furthermore, a shared infrastructure enables a community effect, which makes the development of new sensing applications feasible. For instance, a participant with a built-in camera can take pictures of damaged sidewalks that usually do not yield any significant findings, since the observation was taken in a very limited space and time window. Now, if multiple mobile phone owners share this type of information, the aggregation can be interesting enough to plan, e.g., to repair facilities. However, when many resources are shared across multiple applications that want to access sensor data, several challenges arise.

A shared system is more useful as the number of participants increases. This introduces significant challenges for scalability. To support system scalability in terms of query workload, a number of techniques for “scaling out” can be used. It is not feasible for a single gateway to handle user requests over a large area. One possible approach is to introduce loosely-coupled servers in role of proxy gateways, which are coordinated by a master gateway. Huge advances in high-speed communications, such as Gigabit Ethernet and Fibre Channel technology, have contributed greatly to the availability of ever larger, loosely-coupled server configurations. Network Attached Storage (NAS) can, e.g., be a very economical way of providing many gateways with rapid access to a common set of data.

Other practical considerations in enabling widespread adoption of a public sensing system arise in flexible and natural interfacing with the system itself, and efficient query dissemination that addresses power and bandwidth constraints. The user must be able to explicitly specify the points of interest, in which the reading should take place. After providing such specification, the system needs to be able to determine which mobile nodes are the most suitable to take an accurate reading. This is a complicated task because nodes are in constant movement and different positioning systems vary in accuracy. Furthermore, data collection from sensors should be minimized whenever possible. Instead of using information from all available sensors, approximate answers should be computed on carefully chosen subset of sensors. An efficient query distribution is also a non trivial task. Many reading requests can target sensors that are nearby each other. Such requests should be grouped and transported together. Accumulation of results should be performed before sending them back to the gateway. Since we have no control over the mobility of participating nodes, it can happen that a requested reading cannot be taken.

Chapter 4

System Design

This chapter presents a complete design specification of the system in question. Identified requirements are mapped to various components, and relationships between these components are defined. Following that, a detailed discussion on different aspects of the system is conducted.

4.1 Introduction

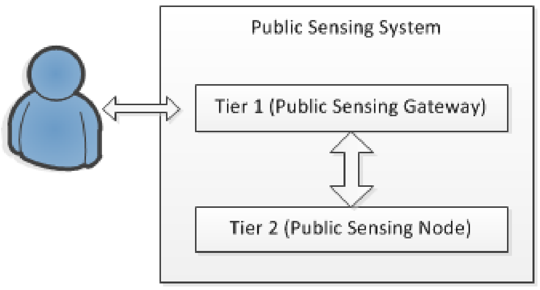


Figure 4.1. A general overview of system architecture.

The aim of this thesis is to implement an opportunistic public sensing system that enables efficient acquisition of point-wise readings from various kinds of sensors distributed over large areas. Our approach provides a flexible and intuitive way to execute various types of queries and acquire several types of sensor readings in parallel. The utility of applications that are envisioned to run in our system motivated us to design an architecture that meets identified requirements and takes advantage of various types of interactions between integral elements of the system. We specify a two-tier architecture and define a minimum set of required functionalities at each tier. The architecture is flexible enough not to impose any requirements on particular set of hardware platforms. An overview can be found in Figure 4.1.

The first tier (*Public Sensing Gateway*) is responsible for creating and distributing queries, and acts as a sink for data gathered in the second tier. Applications have the possibility to submit queries that are periodic and vary in execution time - so called periodic queries. Since the state of our public sensing system changes constantly, e.g., due to the uncontrolled mobility of nodes, periodic queries are implemented as one-shot queries submitted at the beginning of each period. This temporal decomposition enables to reconsider which nodes should be tasked each time and also counteracts situations, in which the results of a periodic query are stale. Afterwards, the gateway performs initial spatial decomposition, i.e., groups the targets of a query by region. Resulting partial queries are submitted to the second tier. The second tier (*Public Sensing Node*) is responsible for selecting nodes to fill the role of virtual sensors specified in partial queries, distribute these reading requests to selected nodes, and transport accumulated sensor readings back to the gateway.

4.2 Architecture

At a high level, the system can be viewed as composed of two tiers: the Public Sensing Gateway and the Public Sensing Node. Each tear is further composed of abstraction layers forming a communication framework. Control is passed from one layer to the next, starting at application layer in one tier, proceeding to the bottom, over the channel to the next tier and back up the hierarchy. Furthermore, the system can be viewed as comprising four classes of functional components: *temporal*, *spatial*, *sensor specific*, and others. A detailed system architecture is presented in Figure 4.2.

The first tier is responsible for issuing queries based on the application input. The most interesting part is the application layer that supports end-user processes. The remaining layers perform typical tasks as defined in TCP/IP model. In the Internet layer takes place device addressing, basic datagram communication and routing. The link layer is responsible for interfacing the suite to the physical hardware. The physical layer is not covered because the data link layer is considered the point which interfaces to the underlying networking architecture.

The application layer of the first tier is composed of the *Gateway Application* (*GA*), *Network Transformator* (*NT*) and supporting modules. The GA responsibility is to provide a flexible interface for user applications to specify a query, execute the query, filter the results and report them back to the application. Query execution is performed with the aid of *Statistical Model* (*SM*) and appropriate *Sensor Selection Algorithms* (*SSA*) components. The SM maintains information about correlations between modelled sensors. If a model does not contain sufficient information to evaluate a query, then an optimal set of virtual sensors is determined and a request is submitted to the system. This choice is made with the support of SSA component that uses the model to select the most informative set of sensors to query. In the present specification of our system, only a greedy selection scheme is supported. The *System Control* (*SC*) component contains the actual view of the world and is used to verify prediction accuracy of the model. The *Sensor Data Logging* (*SDL*) component provides a location for the communal amalgamation of collected sensor information. Details about modelled virtual sensors, per-query and per–target statistics are stored in this location. The NT component provides independence from differences in data representation by translating between the gateway application

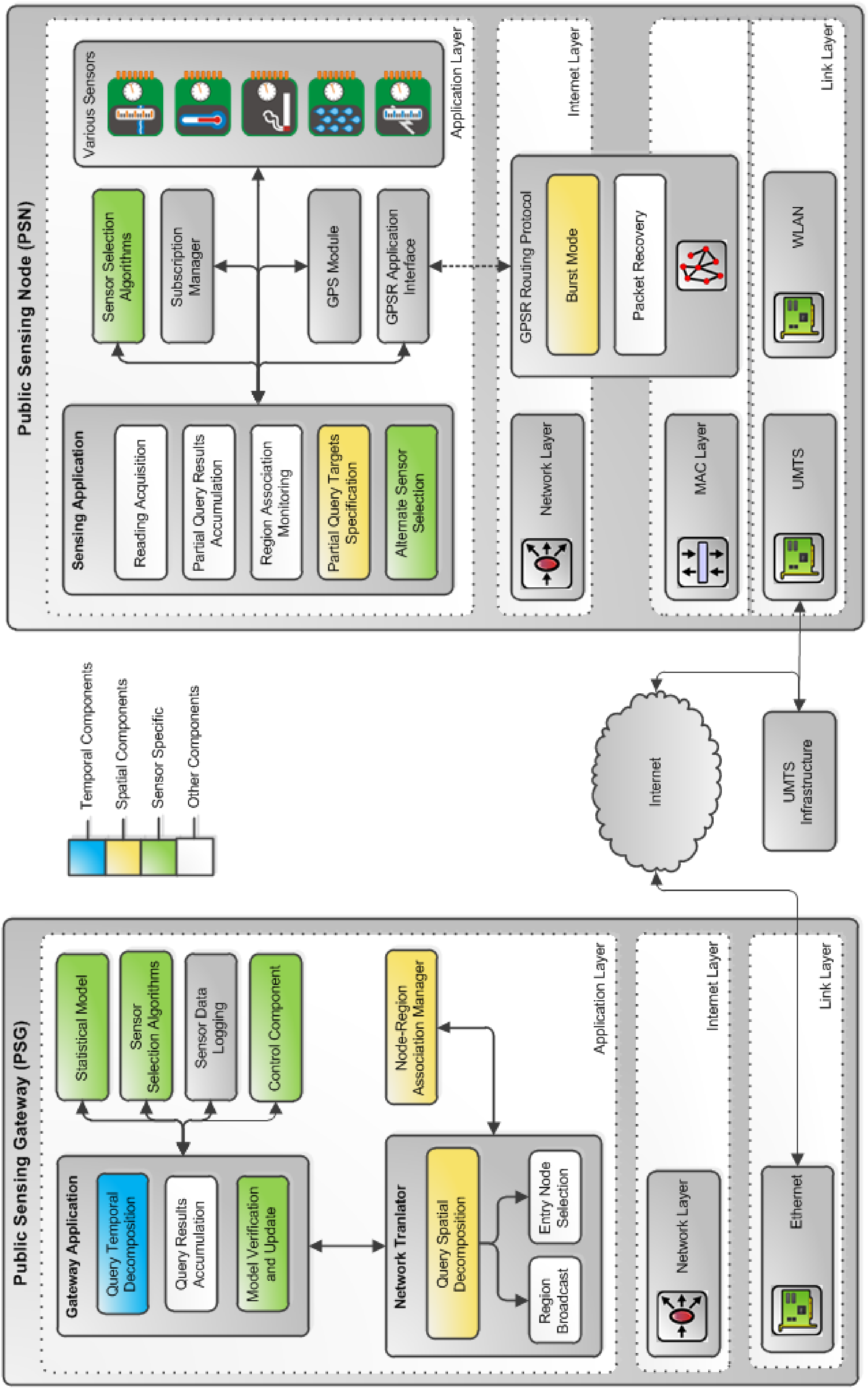


Figure 4.2. A detailed system architecture.

and network format, and vice versa. Furthermore, it performs the initial spatial decomposition of queries, since it groups query targets by region. The *Node-Region Association Manager* (*NRAM*) is responsible for employing the area strategy division and managing associations of participating nodes to corresponding regions.

The second tier performs the actual query execution, as it delivers reading requests to mobile nodes that can fill the role of specified virtual sensors. The application layer consists of modules that support the execution of sensing tasks. The *Sensing Application* queries the *Sensor* component once per reading request. The sensor is either built-in, or available through short range wireless connectivity technologies, e.g., Bluetooth. Taken readings are enriched with a position fix obtained from the *GPS Module*. A mobile node becomes a participant of the system, when it is associated by the gateway with a region of its residence. After breaking this association due to the mobility, the participant notifies the PSG immediately, by requesting a new assignment. The technical details of association and dissociation of a node are encapsulated in the *Subscription Manager* component. The Internet layer, on the other hand, is augmented with a geographical routing protocol to facilitate the delivery of requests. The sensing application is responsible for invoking adapted routing scheme upon receiving a partial query from the gateway. It achieves that through the *GPSR Application Interface* (*GPSR API*), which provides a clean and neat methods to commission a delivery of reading requests specified in the query. Grouping of requests is possible through the *Burst Mode* component incorporated in the routing module. If a reading request cannot be delivered, GPSR protocol notifies the application about that fact through its API. The application is given an opportunity to take appropriate action. *Alternate Sensor Selection* component is used to compensate the unavailability of sensors by determining other suitable sensors to query. This process relies on the knowledge contained in the statistical model stored on the PSG side, and requires additional information to be submitted with the query. This is be further explained in the following.

4.3 Public Sensing

This section presents major components forming the base of our system. We discuss different aspects of our design that contribute to public sensing. The concept of a gateway is covered in details. Afterwards, we introduce our interactive sensor querying enriched with statistical modelling techniques. Following that, the area division method and query distribution models are covered. Afterwards, we introduce the idea of in-network planning to select alternate sensors during query execution. Finally, a scheme is proposed for in-network aggregation of sensor readings.

4.3.1 Gateway

The *Public Sensing Gateway* (PSG) is one of the two major components forming our system. So far, overall architecture and information flow in the PSG were presented. Now, we focus on different aspects of its functionality that contributes to the overall system design.

Firstly, we are interested in computing approximate answers derived from the model. When monitoring spatial phenomena, such as the temperature, it is of fundamental importance to decide on the most informative sensors that should perform a reading. However, to find sensors which predict the phenomena best, one needs a model of the spatial phenomenon itself. Our model is initially learnt from historical data. Afterwards, we gain interactively information about taken sensor readings by examining user queries. This phase is called the *learning phase*. When the model is assumed to reach a state, in which predictions about unavailable sensor readings can be made with acceptable confidence, we move along to another phase, called the *prediction* phase. The statistical model is stored in a compact form and can be easily transported over the network. The main idea behind using a model is to improve the performance of the system by utilizing acquired, up to the present moment, data.

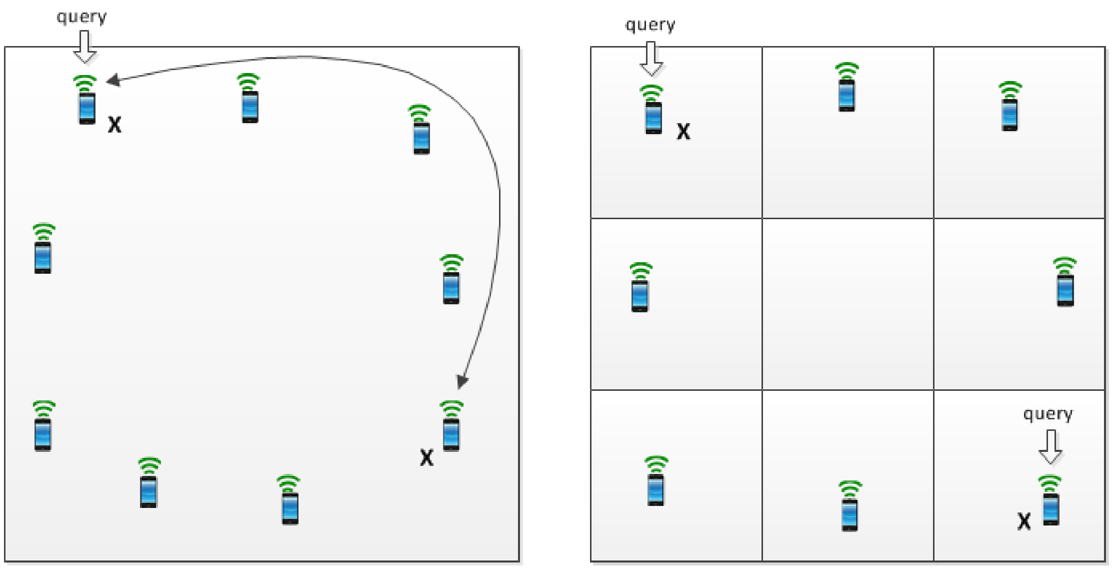


Figure 4.3. Exploiting locality results in decreased number of hops.

To limit the resources demand for a query execution over large regions, we introduced earlier the concept of regions. These logical entities are nothing more than a local group of nodes. Firstly, we want to reduce the number of hop counts to deliver reading requests. Exploiting locality through regions can have a favourable impact on routing between mobile nodes, since we use a position-based approach. Secondly, the Tobler's First Law of Geography states that: "nearby objects are more alike than are objects that are farther away". The degree of dependency among the observations of given phenomenon varies by absolute location across the space dimension, i.e., these relations expose local features. These local features motivate usage of local statistics to detect local patterns, which vary over distance and direction between points. Consider a situation presented in Figure 4.3. An application has submitted a query specifying two virtual sensors that belong to regions in the completely opposite parts of the playground. In the first example, the reading request for a second sensor has to travel all the way along the playground to reach a node supporting specified virtual sensor, and report back the reading via the same route.

The shortcoming of this approach is the communication overhead in terms of hop count and higher likelihood that the return route breaks, which forces the node at which this happens to transmit the result directly to the gateway. In the second example, readings request are sent to one of the nodes in the regions containing specified virtual sensor. To find an appropriate node fewer hops will be required, yielding a lower overall resource consumption and a lesser chance that the return route breaks.

4.3.2 Statistical Model

The aim of a model is to capture aspects of a phenomenon that are relevant to inquiry and to explain how the data could have come out as the realization of a random experiment. Important to understand is that a statistical model is a description of a sampling mechanism, not a description of specific data that it is applied to. For simplicity though, we think of a model in terms of data it describes. As mentioned before, its parameters are estimated using the data at hand. Our model-assisted inference is used to answer questions about the most valuable set of virtual sensors to query, when others are unavailable. In other words, the model becomes the lens through which we view the problem itself, in order to ask and answer questions of interest.

Model selection, diagnosis, and discrimination are important steps in the model-building process. This is typically an iterative process, starting with an initial model and refining it. In order to provide meaningful answers, the model must be correct to the extent that it sufficiently describes given phenomena. This state of art defines a *valid model*, in which predictions can be made. If at any time a model is considered to be *invalid*, no predictions are made and the model has to be updated. The learning phase starts. In the present design of system, we update the model by recomputing it from the scratch. To support this process, we defined the following disjoint sets of virtual sensors:

- *Non-Modelled.* These sensors are explicitly requested by the gateway. If a sensor is not available, no reading can be returned.
- *Modelled.* Correlations about readings obtained from these sensors are captured in a model. If a reading for a modelled virtual sensor is not available, a prediction attempt is made using other most informative virtual sensors. Those sensors are firstly queried for a reading.
- *Control.* Control readings are used to verify the validity of the model and are chosen at random. If the the control readings and corresponding predicted readings exceed a defined threshold, the model is considered invalid and learning phase is initiated. This determination is made on the basis of properties of the estimators. In our particular validation scheme we use an *absolute mean error* (EMA) and a *mean error quadratic root square* (RMSE) property.

4.3.3 Queries

As mentioned earlier, a query can be thought of as a collection of target points where sensing tasks should be executed. Since mobile nodes that can execute these sensing tasks are not known in advance, thus the query itself is specified by a set of virtual sensors.

A query distribution strategy is responsible for determining the most efficient method to reach the nodes capable of supporting defined virtual sensors.

The PSS can handle execution of one-shot and periodic queries. One-shot query means that the data processing is essentially done once, in response to the posted query, e.g., a one-shot SQL query over a large amount of data. Periodic queries are a natural projection of one-shot queries onto continuously changing volumes of data. Consider our wireless network of mobile sensors used for environmental monitoring. The key objective for such systems is to continuously monitor and correlate sensor measurements. Periodic queries are executed with the aid of one-shot queries. They are not directly implemented because we have to consider at the end of each period, which sensors should be selected for tasking. This is a direct consequence of continuous state changes in our system, e.g., uncontrolled mobility of participating nodes. The gateway performs temporal decomposition and submits a one-shot query to the public sensing system at the beginning of each period. Moreover, our system can handle multiple concurrent queries to run in parallel. Each query executed in the system has a globally unique ID (*Query ID*) assigned. This ID is used to track which readings responses are related to the posted query. It is important to mention that query IDs are globally unique in the system scope and do not require central coordination. If more than one public sensing gateway is to be used, a query ID incorporates as well the gateway ID.

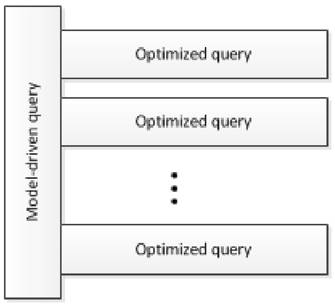


Figure 4.4. Model-driven execution scheme.

We defined two types of queries that our system supports: *unoptimized query* (*UQ*) and *model-driven query* (*MDQ*). An UQ simply queries all virtual sensors that are specified by the user. When a sensor is not available, no reading is taken. The quality of a query results depends directly on the availability of virtual sensors. A MDQ, on the other hand, uses a statistical model of real-world phenomenon and queries only these sensors, whose readings cannot be derived from the model with sufficient acceptable confidence. The MDQ execution comprises two phases: *learning* and *optimized.* In the *learning* phase, the system simply submits unoptimized one-shot queries. During this period we rely only on the physical availability of sensors and cannot provide predictions, since we have not feed sufficient information about sensor readings so far into the model. After query execution has been completed, we integrate collected readings into the model. The *prediction* phase starts when the model has collected sufficient number of samples. At every time when the application query is specified, the gateway tries to answer it at first using information contained in the model. When the answer cannot be provided with sufficient confidence, respective readings requests are sent to the network using optimized one-shot query. If at any point the model is considered to be invalid, the learning phase is initiated. Important to mention is that even though our system defines a learning period, it reports to the user application sensor readings immediately.

A model-driven query follows an execution scheme as presented in Figure 4.4. A multi round scheme is followed in order to attempt answering the specified query. The number of rounds is a system defined parameter. Firstly, we choose the most informative set of sensors to be tasked based on the knowledge contained in the model. In most cases this set is already smaller than the one specified within the application query. Since at the time of query submission we have no practical knowledge of whether all selected sensors can find their counterparts, we assume a more pessimistic scenario in which not all readings are taken. After query execution has been completed, we identify unavailable sensor readings if there were such, select alternate sensors to query and submit a new request again. We continue to do so until an application query can successfully be answered or we reach the maximum number of specified rounds, which means at this point that unavailable readings are assumed to be unresolvable in the current state of the system.

4.3.4 Area Division into Regions

In our public sensing system user applications gather data about phenomenon distributed over space. We operate in an area, in the frame of which user has the possibility to collect information by indirectly executing sensing tasks, thus we term it the *sensed area*. This area is further divided into equally-shaped regions. At present stage, our system supports two different structure schemes: *grid-based* and *honeycomb-based*. It internally keeps a list of regions it supports (e.g., defined by system administrator) and a dynamic list of respective resources (e.g., mobile nodes) that when tasked are most likely to return data consistent with the region.

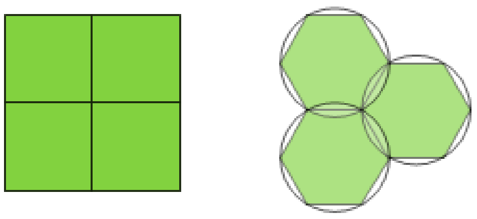


Figure 4.5. Grid-based vs. honeycomb-based structuring scheme.

Grid-based structure is a two dimensional structure made up of series of intersecting vertical and horizontal axes. It is the most intuitive structuring scheme. Honeycomb, on the other hand, enables modelling a mobile phone network, which is structured of cells. These cells are usually thought of as regular hexagons, making up a “honeycomb” structure. In reality these structural units are irregular due to site availability and topography. This is not an issue though in our case, since we are not interested in giving the exact reflection of network structure, but rather establishing a simple spatial structure model for our considerations. The difference between two defined schemes is that honeycombs overlap whereas cells in a grid adhere to each other as in Figure 4.5.

In some cases, when a region does not yield sufficient node density, some part of the query can leave the region of interest. This is a direct consequence of routing strategies that our geographic protocol employs. The concept of regions is suppose to limit the number of hops required to reach nodes supporting specified virtual sensors. However, this does not guarantee that participating nodes in a region can be reached from every other node in this region. If an orphaned entry node is chosen by the gateway to distribute the query, then the query may have to temporarily leave the region to route around a void to be successfully executed.

4.3.5 Query Delivery Models

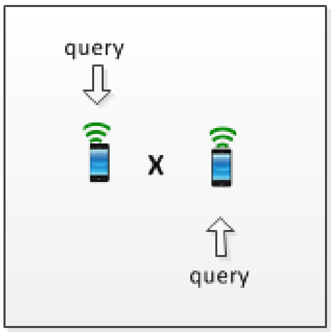


Figure 4.6. Overlapping readings in a region broadcast.

The gateway performs the first phase of spatial decomposition and assigns each partial query to a region. Nodes willing to participate in the sensing task register themselves with the system and are assigned a region. The gateway maintains information only about nodes' availability within a region, not their actual position. The density of nodes in a region determines the type of query delivery that we employ. We defined two different types of delivery methods: *region broadcast* and *position-based.*

In region broadcast we send our query directly to every node associated with a region. A node can take a reading on behalf of one virtual sensors and send it back to the gateway. If a node cannot fill the role of any virtual sensor, no response is sent. Occasionally (Figure 4.6) it can happen that returned readings will overlap, since we have no information about the proximity of nodes beforehand, and are sending blindly our query to the assigned region. In such case, the gateway incorporates more than one reading taken for a virtual sensors into the set of returned readings and passes them to the user application. To summarize, the following steps are taken:

1. Firstly, the most informative sensors are selected and an optimized query is specified at the gateway.
2. Query is decomposed into partial queries that correspond to supported regions.

3. A region broadcast is performed to distribute a partial query. Each node within a region receives the same partial query via UMTS.
4. Each node that can support specified virtual sensors takes a reading. Possibly more than one reading request can be satisfied by a node.
5. Nodes report the readings directly to the gateway via UMTS.

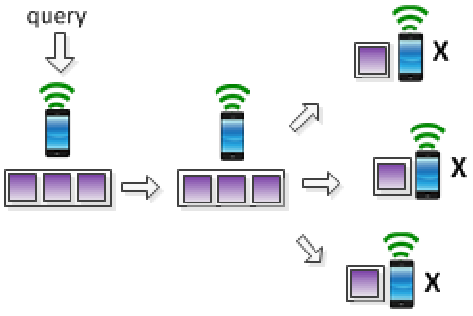


Figure 4.7. Grouping of reading requests sent in the same direction.

When the node density of a region allows for usage of position-based routing, we select a starting node for our query in the region of interest. This node is called the *entry node*. The gateway sends the query via UMTS to this particular node, which is made responsible for initiating position-based routing. Our adopted routing scheme handles the second phase of spatial decomposition. Readings requests for virtual sensors are sent towards their final destinations using intermediate mobile nodes. It can happen that more then one reading is sent in the same direction as in Figure 4.7. Those requests are always sent in a group and gradually distributed at each intermediate node to intended destinations. Selected nodes take a sensor reading and send it back via the same route to the entry node. Finally, the entry node sends accumulated query results via UMTS to the gateway. To summarize, the following steps are taken:

1. Firstly, the most informative sensors are selected and an optimized query is specified at the gateway.
2. Query is decomposed into partial queries that correspond to supported regions.
3. For each region that corresponds to a partial query an entry node is selected.
4. A partial query is sent to an entry node via UMTS. The entry node is an entry point for the query.
5. Entry nodes relay optimized query to other nodes via WiFi. Possibly more than one reading request can be satisfied by a node.
6. Nodes take a sensor reading and send results back to the entry node.
7. Every entry node passes collected results via UMTS to the gateway.

4.3.6 In-Network Planning

When a query is submitted to the system, we have no a priori knowledge about the availability of virtual sensors specified in the query. The uncontrolled mobility of nodes can lead to gaps making some readings unavailable. Large number of unavailable readings yields a rather low accuracy of query results. A model-driven query is executed in rounds to solve the problem of missing readings. In each round an optimized one-shot query is submitted with a set of virtual sensors determined by the sensor selection algorithm after incorporating information about the unavailability of virtual sensors from the previous round. Every time the query did not provide a sufficient coverage of virtual sensors, the gateway submitted a new one specifying virtual sensors with the highest information value in order to predict the unavailable ones. Additional necessity for communication yields more energy usage of mobile devices. Therefore, if the query is being already executed and requirements to obtain a reading cannot be satisfied, alternative sensors should be considered on-demand.

In our model-driven approach we address this problem by incorporating the model along with the submitted query. In a default setting, the statistical model is only available at the gateway, where the sensor selection scheme takes place. We shift the burden of sensor selection process to sensing applications running on mobile nodes, if a requested virtual sensor is not available. This process is called *alternate sensor selection*.

4.3.6.1 Alternate Sensor Selection

Alternate sensor selection provides means to refine the quality of query results during query execution. It is not applicable with region broadcast because nodes communicate only with the gateway, not with each other. In the position-based routing, on the other hand, a reading request can be as undeliverable leading to a reading drop. A node should be given a possibility to countermeasure this situation, since it knows already what virtual sensors are not reachable during the query execution. In a MDQ round-based approach, on the other hand, unavailability of virtual sensors can be detected at earliest at the end of each round, after the completion of optimized one-shot query. This is not very optimal, since the results have to be firstly reported back to the gateway.

Alternate sensor selection is a more efficient approach, since it does not involve the gateway in the process of sensor selection. All necessary readings can be taken in one round. Important to mention is that each node makes the decision about alternate sensors individually. In rare cases it can happen that two nodes in the same region select the same virtual sensor. As a consequence, a redundant reading is taken, which does not provide any additional information for the prediction model. Since this situation occurs rather infrequently, we simply ignore it.

The complete information that is required to complement the unavailable readings is stored at the gateway side. For mobile nodes to be able to execute alternate selection scheme, additional data should be carried along the query. This includes the model, which comprises virtual sensors and their roles, and status about availability of all modelled nodes. Sending the roles explicitly is required, as some of these sensors may have already been successfully queried (thus, a reading is stored at the gateway).

The algorithm for alternate sensor selection requires information about the region boundaries and three sets of sensors:

- *Available sensors.* This is a set of modelled sensors the algorithm can choose from. The data submitted along the query contains information about all available sensors in the system. When virtual sensors are detected as unavailable and alternates have to be selected, the set of all available sensors is firstly reduced by newly detected as being unavailable. This updated information is also passed along newly created requests. Afterwards, the set is limited to the current region and passed to the algorithm.
- *Taken readings* and *control readings.* These are the sets of modelled sensors for which normal readings and control readings either have been obtained or which have been requested and are not detected as unavailable.

The algorithm outputs two sets of sensors. The first one specifies additional virtual sensors that have to be queried, where as the second one defines a set of control readings to verify the validity of the model. The node where the alternate sensor selection process took place is responsible for distributing new requests to the network. These requests carry additional data to support alternate sensor selection algorithm at possibly some other node.

In some cases a part of the query can leave the region of interest and be dropped in a different one. The reason behind is that adopted routing scheme finds a route to a virtual sensor at all costs, disregarding the region assignment. If a part of the query is dropped at an intermediate node, then sensor selection algorithm is initiated. The current set of available sensors is reduced by the newly detected as being unavailable. Since the query execution time at the gateway is limited, the readings obtained as a consequence of alternate sensor selections are sent directly to the gateway. The gateway can figure out that different sensors are selected, since from its perspective these are the ones that it did not request. This process ends when all readings for selected alternates are taken or we ran out of available sensors.

4.3.7 In-Network Aggregation

A query specifying virtual sensors is injected by the gateway, also known as a *sink*, into the network through an entry node. Readings are taken by mobile nodes throughout some area, and then need to be reported back to the gateway to be further processed, and eventually forwarded to the user application. Since reducing resource consumption and increasing overall network efficiency is a major goal in distributed processing of data, we employ an in-network aggregation scheme to relay data back to the sink.

The gateway reduces already the amount of data transmitted over the network by requesting only the most informative set of virtual sensors, instead of all sensors defined in the user query. The mobile nodes supporting specified virtual sensors need to transport a sensor reading back to the sink, after taking one. The simplest and least optimal query plan would require each node to report its own readings via UMTS back to the gateway.

One problem is that a large number of packets must be sent to the gateway. The increased number of small packet transmissions necessary to propagate needed data is expensive. We can reduce the network overhead and resource consumption by locally processing raw data before it is transmitted.

In order to conserve both energy and bandwidth we aggregate sensor readings at specific intermediate nodes, termed the entry nodes. We do not perform any data processing but focus on merging data coming from different nodes into a single packet. When an entry node receives a query from the gateway, it initiates the query distribution process in the network. Every reading request is relayed via intermediary nodes until a reading can be taken. At each step the underlying routing algorithm tracks the route of the request in the network. The *reverse-route* consists of a list of intermediary IP addresses of nodes, including the entry node itself. It is a "reverse" source route indicating that the request was relayed through each node on the list (the last node in the list is the most recent relay). This list is used as a source route to send back a reading to the entry node. If at any time the reverse-route breaks, the result is sent directly to the gateway. The gateway can correlate received readings with the main query because each reading response carries a query ID.

In order to effectively aggregate data in the network, nodes must follow a coordinated communication scheme. For example, before forwarding data on the node, a parent should wait long enough until it has received readings from all of its child nodes. In our approach, the gateway is the root node of the gathering tree and dictates the maximum awaiting time for query results. The information about the query execution time is passed along the query. Other nodes need to apply a timing strategy in order to adapt. Determining how long to wait for readings from other participants is a difficult problem. If a node waits too long, the data may become stale and will not be useful to the gateway. We adjust the timeout of a node, after which it sends collected reading responses, depending on the node's position in the gathering tree. Each successor waits for a half time of its predecessor's awaiting time for query results. This means that nodes lower in the gathering tree should experience a timeout before the nodes that are closer to the gateway. After a timeout, the entry node will forward accumulated data to the gateway. Unfortunately, non-trivial delays in the underlying communication network can lead to late readings responses. In this case, late readings can be incorporated for user inquiry.

4.3.8 Premature Routing Termination

A region contains many mobile nodes but only few of them can fill the role of virtual sensors. When an entry node receives a query from the gateway, it is made responsible to initiate query distribution in a region of interest. Before reading requests reach their intended target nodes, they travel through intermediary nodes. Some of these nodes can be supporting other virtual sensors but were not initially selected. Independently, they are not as informative as targeted virtual sensor but as a collective, they can yield nearly the same information contribution. This forms the base for *premature routing termination.* In general, if we gather enough sensor readings from the intermediary nodes that can compensate for the targeted reading, we can terminate the routing and report the results back to entry node. This reduces the network traffic in terms of hops required to reach intended target, and as a consequence, decreases power consumption of nodes.

This requires for an intermediate node to have an insight into the currently relayed packet and a limited control over routing. We addressed these requirements by extending adopted routing protocol. Additionally, the gateway would have to be informed that different sensors were queried than initially intended. This information could be contained within the packet. This is merely a concept of premature routing termination and is not investigated in details as part of this thesis.

4.4 Location Information Management

In position-based routing protocols the main component is the geographic location information of nodes. In our system, we have assumed so far that such protocol is used in order to perform routing between participating nodes. The fact that each node is equipped with GPS, through which current position is obtained, can be further exploited as a means to achieve scalability in large networks by limiting the number of hops required for a routing scheme to deliver packets. We use a location management scheme that partitions the entire network area into regions, where each region contains nodes - the subscribers of the location management system. The gateway and mobile nodes are integral part of this system but play a different role in the location information management scheme.

4.4.1 Division of Roles

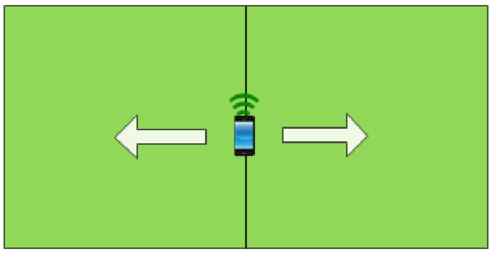


Figure 4.8. A node moves forth and back violating its region assignment.

The gateway is responsible for keeping track of the location information of the nodes. However, in regular location management schemes, when a node changes its position it must update its location information at the location server. This approach might be expensive in terms of packet updates that each nodes needs to perform for the purpose of maintaining fresh location information by the gateway. We reduce this cost by maintaining only region-wise information about mobile nodes instead of their current location information. We introduced earlier in this chapter two different area partition schemes: grid-based and honeycomb-based. The gateway enforces one of these partition schemes specified by the system administrator. The module responsible for managing nodes subscriptions is the *Node-Region Association Manager*. When a mobile node makes initially a subscription request to the gateway, it is logically associated with one of the supported regions and becomes an active participant of the system. The gateway in return sends a subscription response containing information about node's region assignment. If a node was already subscribed, thus was associated with a region, the gateway firstly removes previous assignment, then creates a new one, and afterwards notifies respective node about its subscription update. If a node is detected as being unreachable due to, e.g., ICMP errors, it is simply removed from the association manager.

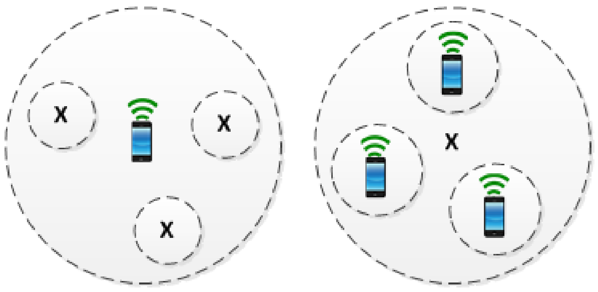
The nodes forming an ad-hoc mobile network, on the other hand, are responsible for informing the gateway about any changes violating their present region assignment. Upon making an initial subscription request, a node is associated with a region and informed about the borderlines of aforementioned region in the response. Since the nodes forming the network move in an unconstrained manner, at some point this association can be violated. If that is the case, the node is made responsible to request a new region assignment from the gateway. However, such action should not be taken immediately upon a violation detection. This can result in a burst of packets being sent to the gateway and increase overall network overhead. In Figure 4.8 a node is moving back and forth between adjacent regions. To address this issue, we define a *relaxation period*. A node can only request a new region assignment, after a number of position updates of a node have been confirmed to violate the present region assignment.

4.5 GPSR Protocol

To reduce the network overhead caused by frequent location information updates from the mobile nodes, we maintain only region-wise information about the nodes at the gateway. In consequence, the gateway can not know in advance which node can fill the role of virtual sensors specified in the query. This imposes a requirement for on-demand selection schemes of nodes to obtain sensor readings. Since each virtual sensor is defined by its position and a sensing radius, position-based schemes are perfectly suited to fulfil this task. We adopted GPSR scheme as our default routing protocol to transport packets between mobile nodes and perform the discovery of nodes that can support specified virtual sensors. Moreover, it has been shown in the literature that WiFi communication is more energy-efficient when compared to UMTS cellular technology [11].

We have extended the default implementation of GPSR protocol to support directional grouping of packets (*burst mode*), which enables each intermediate node to forward a group of packets as a single one, to possibly more than one node lying in the general direction of the destination. This supports resource-efficient query dissemination, since we are no longer required to send packets one by one in the same direction. Moreover, adopted routing protocol assumes the use of exact location information about the destination node. A node defines a notion of coverage of a virtual sensor in order to perform a reading. Thus, using exact location information about the destination is not sufficient to find a suitable virtual node. Instead of routing to a specific destination at a known position (x, y), we are interested in routing to a specific destination whose position is somewhere in a circle C with a known centre and $radius$. Finally, to make premature routing termination possible, each sensing application running on an intermediate node must able to access the currently forwarded packet. We introduced a notion of *tight coupling* between the application and underlying routing protocol, which will be discussed as part of *GPSR Application Programming Interface.*

4.5.1 Delivery Area



(a) $radius \leq \frac{1}{2}TxR$ **(b)** $radius > \frac{1}{2}TxR$

Figure 4.9. Transmission range (TxR) of a node vs. delivery radius.

When a node wishes to send a geographic packet, it specifies the exact location information about the destination D and the surrounding delivery radius of the packet. These two parameters form a *circular delivery area.* The packet shall reach at most all nodes n, for which distance to the destination is less than or equal to the specified radius ($n.distanceTo(D) \leq radius$, where the *distanceTo* method returns a real valued number representing the distance between two points). A node can determine whether its position falls within the delivery radius because the required information is contained within the packet itself. A reading request always corresponds to only one reading response, thus only one node must perform a reading, if at all. Since we are using a distance metric to define a circular delivery area, it can happen that more than one node is in the area, which would violate our assumption about message exchange pattern. If a node falls within the radius of delivery, it is asked to propagate the packet to a node that distance to the destination is smallest among others. In this case, two possible scenarios apply (Figure 4.9).

In the first example (Figure 4.9a), a node that received a reading request can easily forward the packet to its intended recipient. This decision can be made immediately based on the local knowledge of a node. A fundamental requirement for position-based routing protocols is that each node maintains updated information about neighbouring nodes with their current position, i.e., all nodes within transmission range. In the second example, on the other hand, the node does not have information about all nodes in the delivery area. This would require a distributed decision making algorithm to agree upon a node in the delivery region to be the closest to the destination, which is beyond the scope of this thesis. A sensing radius of a virtual sensor is assumed to be less than or equal to the WiFi range of a mobile node.

4.5.2 Burst Mode

GPSR protocol is enhanced with the ability to handle multiple application requests at once. In order to give some intuition, consider a burst mode in automatic firearms, where a predetermined number of rounds can be fired with a single pull of a trigger. By analogy, we adapt the concept of burst mode to the process of sending packets. Instead of sending packets one by one, we transport them in a burst of packets in one shot over the channel. This can reduce delivery costs, without trading off against delivery ratio at all. The grouping criterion defines nodes affiliation to different bursts. The nodes affiliation is indicated by specific fields in the GPSR packet:

- *Next-hop.* This is the IP address of the next node along the path to the final destination.
- *Packet mode* (GPSR). This is a flag indicating whether a packet is in greedy mode or perimeter mode.
- *Lf* (GPSR). This field is a point on a line connecting sending node's and final destination's position, shared between the previous and new face of a planar graph.

We identified these fields after careful analysis of original routing algorithms from the work of Karp et al. on the GPSR protocol [28]. The next-hop address is returned upon successful completion of GPSR algorithms, and the two remaining fields are GPSR specific and can be modified internally by these algorithms. All aforementioned fields form a *burst key* that can uniquely identify GPSR packets belonging to the same burst.

Example 4.1. Structure denoting a single target.

```
struct TargetInfo {
  Coord destPos;
  double radius;
  cPacketPtr appPacket;
}
```

To support burst mode without the necessity of modifying the original algorithms, we extended the regular GPSR packet to handle multiple targets (*GPSR burst packet*). A target is specified by a circular delivery area, i.e., a destination position and delivery radius, and an application specific packet to be transported (Listing 4.1).

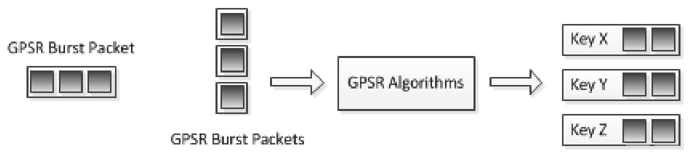


Figure 4.10. Processing of burst packets.

A GPSR burst packet is essentially a GPSR packet with a collection of targets. GPSR internal algorithms cannot handle burst packets directly. Every time when these algorithms need to be invoked, original burst packet is broken into GPSR packets containing only a single target, passed as the input parameters to the underlying algorithms, and the output is grouped into bursts using a burst key. The overall process is presented in Figure 4.10.

To give an overview of how the concept of burst packets is correlated with virtual sensors consider the following example. A user application is interested in measuring a temperature at certain points of the sensed area. The gateway submits a query to the network as a single reading request. The query is a collection of different virtual sensors specified by the user application. When a query arrives at the entry node, it is firstly decomposed into smaller reading requests, each corresponding to one virtual sensor (*indirect spatial decomposition*). It immediately follows that only one reading can be taken at a time. These reading requests are application specific. Afterwards, the sensing application delegates creation of protocol specific packets to the API and supplies additional information required to successfully deliver the packet, i.e., information about the targets. A target destination position and delivery radius are set accordingly to the characteristics of a virtual sensor contained within the decomposed request. From this point forward, it is the GPSR protocol responsibility to deliver the packets to the respective mobile nodes.

4.5.3 Planarization

GPSR protocol uses faces in the planar subgraph to perform perimeter routing, which guarantees packet delivery. The main objective of a network planarization is to get a connected planar subgraph of a network. Following planarization schemes have been considered: Gabriel Graph and Relative Neighbour Graph. They depend on having current position information for a node's set of neighbours. As nodes move, a planarization becomes stale, and less useful for accurate perimeter forwarding. Therefore, we replanarize the graph upon acquisition of a new neighbour, loss of a former neighbour, by a neighbour's beacon timeout, and a MAC transmit failure indication. According to Karp this is not sufficient if nodes only move within a node's radio range, but no nodes move into or out of it [29]. In order to keep the planarized graph maximally up to date, we update planarization upon receipt of every beacon that affected the neighbour table, e.g., a beacon carries a new position of a node that figures already in a set of known neighbours.

Chapter 5

Implementation

5.1 Network Simulator

OMNeT++ is an extensible, modular, component-based, C++ simulation library and framework which also includes an integrated development and a graphical runtime [45].

OMNeT++ provides a generic *component architecture*. The model designer has the flexibility to map concepts such as network devices, protocols or the wireless channel into the model. Model components are termed *modules*, which primarily communicate with each other via message passing either directly, or via predefined conditions. Simple module can be grouped into *compound modules*; the number of hierarchy levels is unlimited. Messages may represent events, packets, commands, jobs or other entities depending on the model domain.

Modules communicate by exchanging messages that may contain any type of data, in addition to regular attributes such as a timestamp. Simple modules can send messages either directly to their destination, or along a predefined path, through gates and connections. Compound modules transparently relay messages between their inner realm and the outside world. The message can arrive from another module or from the same module (*self-messages* are used to implement timers). *Gates* are the input and output interfaces of modules. Messages are sent out through output gates and arrive through input gates. An input gate and output gate can be linked by a *connection*. Connections are created within a single level of module hierarchy: within a compound module, corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module can be connected.

OMNeT++ has the basic machinery and tools to write simulations, but itself it does not provide any components specifically for computer networks or any other domain. Instead, these elements are provided by different simulation models and frameworks such as the INET Framework and INTEMANET. INET Framework contains models for several Internet protocols: UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, IEEE 802.11, MPLS, OSPF, and others. INETMANET is a fork of the INET Framework, and extends INET with support for mobile ad-hoc networks. INETMANET supports AODV, DSR, OLSR, DYMO and other ad-hoc routing protocols [5].

5.1.1 Ad-hoc Routing in INETMANET

Ad-hoc routing schemes incorporated in OMNeT++ are mostly existing public versions of protocols [10], which are available in different implementations. The best candidates to be migrated to OMNeT++ were considered those, which were realised to work with the Linux kernel. Unfortunately, at the time of writing this thesis, there was no mature implementation of GPSR that has been validated in a realistic application scenario. Thus, we had to provide a custom solution, which notably increased the implementation effort and time. Moreover, the documentation of INETMANET on how to provide custom extensions to the framework, was incomplete.

5.1.2 IP Module Interface

IP module allows very flexible interfacing with higher-layer protocols. Before sending a packet over IP, the sending module must fill in an *IPControlInfo* object and attach it to the packet with C++ method *setControlInfo()* method. IPControlInfo carries control information for sending/receiving packets over IP. When sending to a higher-layer protocol, the following fields are required: destination IP address and encapsulated protocol. INETMANET defines several higher-layer protocols (*name* (*number*) : *output gate*):

- TCP (6) : 0
- UDP (17) :1
- ICMP (1) : 2
- IGMP (2) : 3
- RSVP (46) : 4
- OSPF (89) : 5
- SCTP (132) : 6
- MANET (254) : 7
- MANET (135) : 7

When delivering a packet to a higher-layer protocol, the output gate is determined from the protocol field in the IP header. A mapping from a protocol number to an output gate can be modified in the configuration files. For more details, please refer to the documentation.

When a packet is sent to the network, the sending module must fill the following fields in an *IPControlInfo*: destination IP address, source IP address and protocol. If we want to avoid using IP routing, a next-hop address and an output interface must be additionally specified. IP module controls also the lifetime of a packet using a hop counter field in an *IPControlInfo* object. Before sending a datagram, a hop counter check is performed and, if its zero then an ICMP error message is sent to the sender.

IP module allows two different modules on the same host to communicate with each other. This is accomplished by setting the destination address in the IP header to the loopback address. Sending and receiving modules must be connected to the appropriate gates defined in the network layer. These packets are not transmitted outside of a host, and are discarded if the gate is disconnected. This mechanism is used to implement two-way communication between GPSR module and its API.

5.2 UMTS

Mobile phones are equipped with a number of communication channels. In our scenario we are using a hybrid communication setting, i.e., Wireless Local Area (Wireless LAN, or WLAN) and Universal Mobile Telecommunication System (UMTS). UMTS is a high-tier system that is currently a standard for 3G mobile networks. It addresses the growing demands of the mobile and Internet applications. This component is used by the public sensing system, running on a stationary server in the Internet, to communicate with mobile nodes and perform queries in a distributed fashion.

Institute of Parallel and Distributed Systems at the University of Stuttgart provided an UMTS module [21], which is a suitable abstraction of real UMTS communication system based on empirical studies. This model was implemented in OMNeT++ simulation network and integrated into the INTEMANET framework.

The architecture of UMTS is based on the GSM/EDGE standard. This allows a simple migration for existing GSM operators. It specifies a complete network system, covering the radio access network (UMTS Terrestrial Radio Access Network, or UTRAN), the core network (Mobile Application Part, or MAP) and the authentication of users via SIM cards.

5.3 GPSR Protocol

Our protocol is a complete implementation of Greedy Perimeter Stateless Routing scheme [28]. We further extended the protocol to handle multiple targets. Implemented module can be easily integrated with INETMANET framework. Detailed information on the GPSR specific packet formats can be found in the original work.

5.3.1 Beaconing

Position information available at each node is a key element in geographical routing. *GPSRBeacon* module implements a simple proactive beaconing algorithm that provides nodes with positions of their neighbours: each node periodically broadcasts a hello message (called a *beacon,* Listing 5.1), which contains an IP address of a sending node (*srcAddr*) and its position (*srcPos*). A *nextHopAddr* field represents a broadcast address, and *timeToLive* is always set to 1 because INETMANET framework supports only a single-hop broadcast scheme. A beacon from a neighbouring node is processed as follows:

1. If a sender of a beacon was unknown, it is added to a neighbouring table with status *recent* and planarization is triggered, to reflect possible topology changes. Since the

receiving node was not aware of the sending node, it can make an assumption that the sending node does not have any knowledge whatsoever about the receiver. In this case, a beacon can be immediately scheduled but this could trigger another immediate beacon broadcast from the sender side, since the sender was before not aware about the receiver and makes an identical assumption. This can result in increased network traffic and therefore, it is the system administrator responsibility to make a trade off between accuracy of information in a neighbouring table and network traffic. An optional parameter *immediateBeacon* is defined to control this behaviour.

2. If a sender was already in the neighbouring table, its status is reset to *recent* and its position is possibly updated when it differs form the formerly known. When a position update occurs, planarization must be triggered again.

Example 5.1. GPSR beacon packet format.

```
packet GPSRBeaconPacket {
IPAddress srcAddr;
Coord srcPos;
IPAddress nextHopAddr;

short timeToLive;
}
```

The interval B between two consecutive beacons can be specified. To avoid synchronization of neighbour's beacons we jitter each transmission by 50% of the interval B [28]. The mean inter-transmission interval between beacons is uniformly distributed in $[0,5B; 1,5B]$. When a neighbour does not send a beacon for a longer than specified period T, a node assumes that the neighbour is gone out of range and deletes the neighbour from its table. This is a two-step process: after the first timeout T, a node is unmarked as being *recent,* and after the second timeout it is finally removed from the neighbour table. In this thesis, we use $T = 4.5B$, three times the maximum jittered beacon interval. The following parameters control timing behaviour of *GPSRBeacon* module:

- *beaconJitter* - jitter in beaconing interval expressed in percentages.
- *beaconInterval* - beaconing interval (B).
- *beaconExpiry* - timeout period of a beacon ($T = 4.5B$).

To minimize the cost of beaconing, GPSR makes the most of the sent data packets, by adding a position of local sending node to the GPSR packet. Piggybacking, at a small cost in bytes, allows all GPSR packets to serve as beacons. When a node sends a data packet, it can afterwards reset its inter-beacon timer. This optimization reduces beacon traffic in regions of the network that actively forward data packets.

Optionally information such as addition, deletion, and update of a neighbour are forwarded to the application using GPSR API.

5.3.2 Neighbour Table

GPSR protocol relies on geographic location information to make correct forwarding decisions. It is nearly stateless and requires propagation of topology for only a single-hop. The neighbour table is used to maintain information about present state of immediate neighbours and performs the following:

- it adds and removes neighbours,
- it updates and looks up status of neighbours,
- it finds next-hop node for greedy forwarding,
- it finds clockwise node for perimeter forwarding,
- it creates a planarized subgraph.

The aforementioned functionalities are implemented in the *GPSRNeighTable* module.

Geographic location information is used to make forwarding decision but we use Internet Protocol for communication to relay packets across the network. Thus, each entry in a neighbour table must contain information about node's poisition and IP address. The basic building block used by a neighbour table is *HostEntry*, a subclass of *NodeEntry* as in Listing 5.2. NodeEntry contains information about node's IP address and position. HostInfo adds additional information about node's status. Current implementation defines two different statuses: *recent* and *active*. It is assumed that the IP address of a mobile node is unique among all nodes participating in routing.

Example 5.2. Basic information block in a neighbour table.

```
class NodeEntry: public cPolymorphic {
protected:
  IPAddress nodeID;
  Coord pos;
}

class HostEntry: public NodeEntry {
protected:
  uint status;
}
```

Forwarding an IP datagram generally requires the node to choose the address and relevant interface of the next-hop node or (for the final hop) the destination host. A neighbour table can be seen as a limited routing table.

When a node is going to forward a packet, it must determine whether it can send it directly to its destination, or whether id needs to pass it through another node. If the latter applies, it needs to determine which node to use. This determination is solely done based on geographical locations of immediate neighbours. Thus, the neighbour table needs to store additional information along the IP address.

- *NEIGH_STATUS_RECENT* - used when a node is added or updated in a neighbour table. It is primarily used to keep the table up to date.
- *NEIGH_STATUS_ACTIVE* - determines which node participates in routing and is used in planarization.

Apart from basic node management, the neighbour table is responsible for triggering planarization, finding the greedy node and applying the right-hand rule for traversing a graph in the perimeter mode.

5.3.3 Greedy and Perimeter Mode

Algorithm 5.1 Forwarding a GPSR greedy packet.

```
Input: GPSR packet ($gpsrp \neq \emptyset$)
Output: forward packet or enter perimeter mode
1: $destPos \leftarrow$ get destination position from $gpsrp$
2: $nextHop \leftarrow$ find greedy node to $destPos$
3: if ($nextHop = \emptyset$) then
4:     enter perimeter mode
5: else
6:     forward packet to $nextHop$
7: end if
```

When a GPSR data packet is created, it is initially in greedy mode. The originator also sets the geographic location of the destination. This field is left unchanged as the packet is forwarded through the network.

GPSRGreedy module is responsible for finding a neighbour that is geographically closest to the packet's destination. Internally, the *GPSRGreedy* module delegates this task to the *GPSRNeighbourTable*, since it maintains all information about neighbours. In Algorithm 5.2, a detailed explanation of greedy scheme is presented. There are situations in which the only route to a destination requires a packet to move temporarily farther in geometric distance from the destination.

GPSRPerimeter module provides a means to handle such situations by forwarding a packet around voids. Some fields of a GPSR packet can be changed, such as the position where the packet entered perimeter mode, the first edge traversed on the current face, and the position of the last intersection, where a face change occurs. The detailed explanation on how perimeter forwarding scheme is implemented and how face changing is achieved, is presented respectively in Algorithm 5.2 and Algorithm 5.3. Every module, after finishing processing of a packet, can modify the packet and either produce a routing decision, or drop it.

Algorithm 5.2 Forwarding a GPSR perimeter packet.

```
Input: GPSR packet ($gpsrp \neq \emptyset$)
Input: information whether packet enters perimeter mode ($init$)
Output: forward packet or drop packet
 1: $nextHop \leftarrow \emptyset$
 2: $destPos \leftarrow$ get destination position from $gpsrp$
 3: $node \leftarrow$ get current node IP address and position
 4: if ($init$ = true) then
 5:    set packet mode to perimeter
 6:    $nextHop \leftarrow$ find clockwise node about itself from $destPos$
 7:    if ($nextHop = \emptyset$) then
 8:       drop packet // no path
 9:    end if
10:    $gpsrp.Lp \leftarrow node.pos$
11:    $gpsrp.Lf \leftarrow node.pos$
12:    $gpsrp.Le \leftarrow (node, nextHop)$
13: else
14:    $nextHop \leftarrow$ find greedy node to $destPos$
15:    if ($nextHop \neq \emptyset$) then
16:       if ($gpsrp.Lp \neq nextHop.pos$) then
17:          set packet mode to greedy
18:          forward packet to $nextHop$
19:       end if
20:    end if
21:    $nextHop \leftarrow$ find clockwise node about itself from $gpsrp.srcPos$
22:    if ($nextHop = \emptyset$) then
23:       drop packet // no path
24:    end if
25:    $tailNode \leftarrow gpsrp.Le.tail$
26:    $headNode \leftarrow gpsrp.Le.head$
27:    if (($nodePos = tailNode.pos$) and ($nextHop.pos = headNode.pos$)) then
28:       drop packet // loop on perimeter
29:    end if
30:    if ($faceChange(gpsrp,$ inout $nextHop) =$ false) then
31:       drop packet // no path
32:    end if
33:    forward packet to $nextHop$
34: end if
```

Algorithm 5.3 Changing face in a planar graph.

```
Input: GPSR packet ($gpsrp \neq \emptyset$)
Input: next-hop node (inout $nextHop \neq \emptyset$, can be modified)
Output: true upon successful determination of $nextHop$ on a new face, false otherwise
 1: $destPos \leftarrow$ destination position from $gpsrp$
 2: $node \leftarrow$ current node IP address and position
 3: loop
 4:    $line1 \leftarrow$ find equation of a line given $gpsrp.Lp$ and $destPos$
 5:    $line2 \leftarrow$ find equation of a line given $node.pos$ and $nextHop.pos$
 6:    $intersectionPoint \leftarrow$ find intersection point of $line1$ and $line2$
 7:    if ($intersectionPoint = \emptyset$) then
 8:       return true
 9:    end if
10:    // this node borders an edge where the inersection point lies
11:    $distToLf \leftarrow$ find distance from $destPos$ to $gpsrp.Lf$
12:    $distToIntersectionPoint \leftarrow$ find distance from $destPos$ to $intersectionPoint$
13:    if ($distToLf \leq distToIntersectionPoint$) then
14:       return true // no progress to the destination
15:    end if
16:    $gpsrp.Lf \leftarrow intersectionPoint$
17:    $nextHop \leftarrow$ find clockwise node about itself from $gpsrp.Lf$
18:    if ($nextHop = \emptyset$) then
19:       return false
20:    end if
21: end loop
```

5.4 GPSR Application Programming Interface

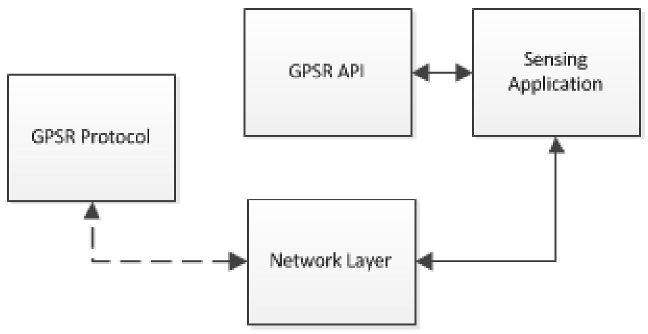


Figure 5.1. Overview of GPSR API communication scheme.

An important aspect of GPSR implementation was to make the application logic independent from the underlying routing protocol. We defined a set of particular rules and messages that modules exchange to communicate with each other. GPSR API describes the expected behaviour while the GPSR protocol provides an actual implementation of this set of rules. The API defines a two-way communication scheme that exploits an existing infrastructure of interconnected modules as in Figure 5.1.

5.4.1 API Interface and Messages

GPSR API uses gates of a *Sensing Application* module to exchange messages along a predefined path with the routing protocol. The message is firstly sent through the output gate of an application, then forwarded through the network layer, and eventually arrives at the input gate of a routing module in question. The routing module uses the same path to send messages back to the application, which then calls *handleMessage()* method of the API instance. This clean design concept avoids using C++ interfaces between the calling and the called module, thus breaking a notion of loose coupling to implement two-way communication.

To exchange messages, a *GPSR API packet* is defined (Listing 5.3). The IP module imposes a requirement for all messages that are passed through the module to represent packets. However, within a GPSR API packet, arbitrarily complex data structures can be carried. For simplicity, we refer later to API packet as *API message*. API message specifies information about the action (*actionType*) that should be taken upon the object that is being carried (*obj*). Moreover, the message itself has also limited capabilities to control routing by cancelling currently relayed packet (*cancelRouting*).

Sensing Application has no knowledge of how the routing protocol operates internally. The creation of protocol specific packets is delegated to the API, and additional information required to successfully deliver the packet are supplied, i.e., information about targets. After specifying targets, *sendGPSRPacket()* is called. Targets can be either supplied as a list, or one by one. This determines whether burst mode is active or not.

Example 5.3. GPSR API packet.

```
packet GPSRAPIPacket {
  short actionType enum(ActionType) = 0;
  cOwnedObjectPtr obj = NULL;
  bool cancelRouting = false;
}
```

Action type determines which participant of the communication is responsible for handling the object and how he should act upon it. GPSR API defines following action types:

- *SEND* actions are intended for the routing protocol to signal that a packet is newly created and should be processed accordingly. The API generates a SEND message, when an application requests to send a GPSR packet by passing information about targets.
- *RECV* actions are intended for the API and inform that a packet is delivered to its destination. Application must consume the packet specified in API message.
- *DROP* actions are intended for the API. An application must consume the packet. This enables an application to act upon packets that fail to reach their destination. A recovery mechanism can be implemented in case of such drops.
- *RELAY* actions are intended for the API and the GPSR protocol. When a packet is en route from its source node to the destination, it can be passed on, at an intermediate node, to the application. When the API receives a RELAY message, it passes it to the application. The application must not consume the packet contained in API message, otherwise an error occurs. However, it has the possibility to cancel routing by setting an appropriate flag in API message. The API is made responsible to guarantee that this packet is not eventually forwarded to the network.

Example 5.4. GPSR API interface.

```
class IGPSRApiInfo {
public:
  virtual ~IGPSRApiInfo() { }
  virtual void receiveApiInfo(GPSRApiInfo *apiInfo) = 0;
};
```

Application that wants to receive notifications from GPSR API, needs to conform to a contract that specifies what operations an API supports. We defined an interface *IGPSRApiInfo* that contains one single method *receiveApiInfo()* (Listing 5.4).

5.4.2 Processing of API Messages

To provide more insight into how API messages are processed consider an example of API message arriving from the routing protocol. A detailed processing scheme of the message is presented in Algorithm 5.4.

Algorithm 5.4 Processing of API message.

```
Input: API message ($apim \neq \emptyset$)
Output: Consume or relay carried object ($apim.object$)
1: $action \leftarrow apim.actionType$
2: if ($action$ is SEND) then
3:     error // GPSR module is responsible for originating packets
4: end if
5: if ($action$ is RECV) then
6:     pass $apim$ to application
7:     // application processing
8:     confirm that application consumed carried object ($apim.object$)
9: end if
10: if ($action$ is RELAY) then
11:     pass $apim$ to application
12:     // application processing
13:     $cancelRouting \leftarrow$ check if cancel routing flag is set in $apim$
14:     if ($cancelRouting =$ true) then
15:         delete $apim.object$
16:     else
17:         relay $apim.object$ to GPSR module
18:         // indirect send through network layer
19:     end if
20: end if
21: if ($action$ is DROP) then
22:     confirm that application consumed carried object ($apim.object$)
23: end if
24: delete $apim$
```

5.5 GPSR Extensions

5.5.1 Packet Formats

One of the main contributions to the overall GPSR functionality is the burst mode. In Listing 5.5 a detailed definition of *GPSRBurstBasePacket* can be found. The packets is capable of specifying a number of targets that are in the general direction of the destination. *copyTargets* flag is used when a packet is duplicated and determines whether information about targets must be copied to a newly created packet or not. Since one target is a special case of a number of targets, GPSR burst packet is a super class of

Example 5.5. GPSR burst packet definition.

```
packet GPSRBurstBasePacket extends GPSRBasePacket {
  bool copyTargets = true;
  abstract TargetInfo targets[];
}

// query targets
virtual void addTarget(const TargetInfo& target,
  bool updateLength = true);
virtual TargetInfo& getTarget(unsigned int k);
virtual TargetInfoVector getTargets();
virtual unsigned int getTargetsCount() const;
virtual bool hasSingleTarget() const;

// burst control unit
virtual BurstControlUnit::Ptr getBurstControlUnit() const;
virtual void setBurstControlUnit(BurstControlUnit::Ptr ptr);
virtual bool isBurstPacket() const;
virtual bool isPartOfBurstPacket() const;
```

GPSR packet. For convenience, *hasSingleTarget()* method is specified to determine if the packet is in a state that allows for a packet to be processed by GPSR algorithms.

5.5.2 Burst Mode

Algorithm 5.5 Decomposing a GPSR burst packet for processing.

```
Input: GPSR burst packet ($gpsrburstp \neq \emptyset$)
Output: GPSR packet(s)
 1: if ($gpsrburstp$ target count $< 1$) then
 2:    error
 3: else if ($gpsrburstp$ target count $> 1$) then
 4:    // packet contains many targets
 5:    $gpsrburstp.copyTargets \leftarrow$ false
 6:    $bcu \leftarrow$ create BCU for $gpsrburstp$
 7:    $bcu.collectedPackets \leftarrow \emptyset$
 8:    while ($gpsrburstp$ has more targets) do
 9:       $t \leftarrow$ remove first target from $gpsrburstp$
10:       $gpsrp \leftarrow$ duplicate $gpsrburstp$ fields (ignores remaining targets)
11:       $gpsrp.target \leftarrow t$
12:       $gpsrp.bcu \leftarrow$ attach $bcu$
13:       process $gpsrp$ // contains a single target
14:    end while
15: else
16:    // packet contains a single target
17:    process $gpsrburstp$
18: end if
```

A *GPSR packet* is considered to be a *GPSR burst packet* if it specifies more than one target. Initially, all packets are assumed to specify multiple targets. Implemented routing algorithms can handle only one target at a time, thus additional processing is required. *Burst Control Unit* (*BCU*) class is used to decompose a GPSR packet with multiple targets before calling GPSR algorithms, and after their completion is responsible for collecting decomposed GPSR packets along with their routing decision. One BCU is bonded to exactly one GPSR burst packet. In Algorithm 5.5, a process of decomposing a GPSR burst packet into many single-target GPSR packets is presented. If only one target is specified, a GPSR burst packet is handled directly.

Algorithm 5.6 Collecting GPSR packets after processing is completed.

```
Input: GPSR packet ($gpsrp \neq \emptyset$)
Input: Routing decision
Output: Collected GPSR packets
 1: if ($gpsrp$ has attached BCU) then
 2:     $bcu \leftarrow$ get from $gpsrp$
 3:     $key \leftarrow$ generate key from routing decision and $gpsrp$ specific fields
 4:     $bci.collectedPackets \leftarrow (key, gpsrp)$
 5:     if ($bci$ has all parts) then
 6:         aggregate and forward packets
 7:     end if
 8: else
 9:     forward packet
10: end if
```

The *GPSRBurstKey* class implements a burst key concept. The burst key is a unique combination of packet specific fields and a routing decision. We firstly convert these elements to an appropriate string representation and then combine them by appending one string to another. When the processing of a GPSR packet is completed, every routable packet is collected using a burst key in its BCU. After the last packet of a burst is collected, routable GSPR packets are further aggregated and send to the network. An overview of this process is presented in Algorithm 5.6.

The GPSR algorithms can either produce a routing decision, or determine that a packet is unroutable. This yields two different directions of forwarding depending on the outcome: a packet can be either passed to an upper layer (application), or a lower layer (network). Moreover, each direction of forwarding specifies further an aggregation strategy. The application is interested in the type of action it should take upon receiving a packet, therefore all packets with the same action type are aggregated accordingly. The network, on the other hand, requires a packet with filled IP control information set accordingly to the respective routing decision. In these two distinct cases different keys are defined: for application, packets are grouped based on the action type, and for the network, using a burst key. Aggregation of collected GPSR packets is presented in Algorithm 5.7.

Algorithm 5.7 Aggregating collected GPSR packets.

```
Input: Map of collected GPSR packets ($collectedPackets$)
Output: Map of aggregated GPSR burst packets ($aggregatedPackets$)
 1: $aggregatedPackets \leftarrow \emptyset$
 2: if (all packets of a burst are collected) then
 3:    for all ($cEntry$ in $collectedPackets$) do
 4:       $key \leftarrow cEntry.key$
 5:       $gpsrp \leftarrow cEntry.value$
 6:       $gpsrburstp \leftarrow \emptyset$
 7:       for all ($aEntry$ in $aggregatedPackets$) do
 8:          if ($key = aEntry.key$) then
 9:             $gpsrburstp \leftarrow aEntry.value$
10:             break
11:          end if
12:       end for
13:       if ($gpsrburstp = \emptyset$) then
14:          $gpsrburstp \leftarrow$ duplicate $gpsrp$ fields
15:          $gpsrburstp.bcu \leftarrow \emptyset$ // detach BCU
16:          $gpsrburstp.copyTargets \leftarrow$ true
17:       end if
18:       add $gpsrp.target$ to $gpsrburstp.targets$
19:       delete $gpsrp$
20:    end for
21:    return $aggregatedPackets$
22: else
23:    error
24: end if
```

5.5.3 Support for Mac-layer Failure Feedback

In order to make wireless communication more robust in mobile scenarios, we react to link layer notifications, e.g., when a packet exceeds its maximum number of retransmissions to the next-hop[1]. This can be an indication that a neighbour node has failed or gone out of range, or that a network is in in congestion collapse state, in which little or no useful communication can happen due to congestion. Consider a network where round-trip time for a packet exceeds the maximum retransmission interval for any node. That node begins to introduce more and more copies of the same datagrams into the network. Eventually, all available buffers in the node are full and packets must be dropped. This does not happen in our scenario, since we are not flooding the network with packets. We interpret indications of link-level retransmission failures as not receiving a beacon from a neighbour for a longer than specified timeout (the neighbour has gone out of range). To enable link layer notifications, a routing module must call during initialization phase *linkLayerFeeback()* method and override virtual *processLinkBreak()* method.

A problem with unavailability of a neighbour node can also happen during resolution of network layer addresses into link layer addresses. *ARP module* retries a predefined number of times to send an *ARP Request*, and if no reply is returned, the packet in question is dropped. We extended default ARP module functionality to provide custom notifications about dropped packets. However, in case of ARP failures no actual countermeasure can be taken, since if ARP requests are not resolved, no data packets can be sent either. We avoid this situation by not introducing excessive network congestion.

5.5.4 Integration with OMNeT++

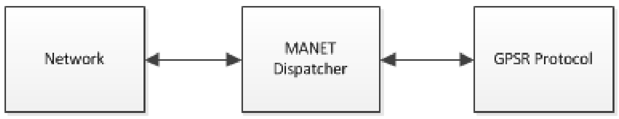


Figure 5.2. Connection of an ad-hoc routing module to the network layer.

Ad-hoc routing protocols included in the INETMANET framework are managed through a MANET manager [10]. Existence of this module is fundamental to trigger the ad-hoc routing procedures in the IP network layer. For this reason, we had to reproduce a minimum set of functionalities from the MANET manager to integrate GPSR scheme.

MANET dispatcher is a managing module that instantiates dynamically at runtime GPSR protocol, *manetroutingprotocol,* and connects it to the appropriate IP interface of the MANET dispatcher (Figure 5.2). It is connected to the IP module and manages message dispatching from one module to another. The GPSR protocol module is a child module of MANET dispatcher.

[1]The standard defines seven retransmissions in IEEE 802.11.

Algorithm 5.8 Processing of packets in MANET dispatcher.

```
Input: Network packet $p$
Output: Forward or drop packet
 1: if (MANET routing is not active) then
 2:    delete $p$
 3: else
 4:    if ($p$ arrived from IP module) then
 5:       // arrived on "from_ip" gate
 6:       if (chosen protocol is supported) then
 7:          if ($p$ is type of ControlManetRouting) then
 8:             if ($p.optionCode$ = MANET_ROUTE_NOROUTE) then
 9:                error
10:             end if
11:             delete $p$ // ControlManetRouting packets are not forwarded
12:          else
13:             send $p$ directly to routing module
14:             // send directly to "from_ip" gate of routing module
15:          end if
16:       else
17:          delete $p$ // unsupported protocol
18:       end if
19:    else
20:       if (chosen protocol is supported) then
21:          if ($p$ did not arrive from routing module) then
22:             delete $p$ // unknown gate, should arrive on "from_manet" gate
23:          else
24:             send $p$ to routing module
25:             // send to "to_ip" gate of IP module
26:          end if
27:       else
28:          delete $p$ // unsupported protocol
29:       end if
30:    end if
31: end if
```

In addition to triggering ad-hoc routing procedures, it is partially in charge of controlling the IP table. At the beginning of simulation, all entries in the IP table are erased. The IP module is forced to transfer any forwarded packets to MANET dispatcher, which relays them further to a routing module. The GPSR protocol is responsible for augmenting a packet with an IP control information required by the IP module to forward packets. Moreover, MANET dispatcher interprets *ControlManetRouting* messages from the IP module and throws an error if GPSR protocol did not specify enough routing information for the IP module. Detailed processing of packets in MANET dispatcher can be found in Algorithm 5.8.

5.6 Public Sensing

5.6.1 Packet Formats

Example 5.6. *GatewayApplication* packets.

```
packet DataRequest {
  int queryID;
  double timestamp;
  double queryPeriod;

  VirtualSensor::Ptr virtualSensors[];
  SensorRole sensorRole[];
}

packet INPDataRequest extends DataRequest {
  VirtualSensor::Ptr allVirtualSensors[];
  SensorRole allSensorRoles[];
  SelectSensorsIntMGMIntelLab::CoordSet virtualSensorAvailability;
  SelectSensorsIntMGMIntelLab::Ptr sensorSelector;
}

packet DataResponse {
  int queryID;
  double requestTimestamp; // timestamp copied from the request
  double readingTimestamp; // timestamp when the reading was taken
  double reading;

  VirtualSensor::Ptr virtualSensor;
  SensorRole sensorRole;
  Coord actualPosition;
}
```

Two different classes of packets are specified. The first class carries domain specific knowledge (Listing 5.6). The *GatewayApplication* creates a query with a globally unique ID (*queryID*) that is completed after a specified period of time (*queryPeriod*). A corresponding *DataRequest* packet is created to request readings from the system and is consumed by *Sensing Application*. *virtualSensors* array contains, specified in the query virtual sensors, with each having an explicit role assigned. The readings request is forwarded to the *NetworkTransformator*. For simplicity, we consider *DataRequest* to be equivalent to a query because they mutually describe themselves. *INPDataRequest* packet extends ability of *DataRequest* to carry information about the model and currently employed sensor selection algorithm. These type of packets are used only when in-network planning is active. *DataResponse* packet is consumed by the *GatewayApplication* and stores information about taken reading, timestamps, and a position of a node that took the reading.

Example 5.7. *NetworkTransformator* packets.

```
packet PSGatewayReq {
  IPAddress destAddr;
  IPAddress srcAddr;
  bool useInNetworkProcessing = false;
  bool regionBroadcast = false;
}

packet PSGatewayResp {
  IPAddress destAddr;
  IPAddress srcAddr;
  cArray readings;
}
```

The second class of packets (Listing 5.7) travels over a network and encapsulates packets supplied from higher layers. *PSGatewayReq* carries a packet passed from the *GatewayApplication* and provides additional information on how the packet must be further processed in a *Public Sensing Node*. *PSGatewayResp* packet is sent from *Public Sensing Node* and carries at least one *DataResponse* packet. *destAddr* and *srcAddr* fields are used to address appropriate sides of communication, e.g., if a *Public Sensing Node* sends *PSGatewayResp* then *destAddr* points to the IP address of the *Public Sensing Gateway*, and *srcAddr* is set to the IP address of the sending node.

5.6.2 Gateway

Public Sensing Gateway (*PSG*) is modelled using the NED language. *IPSGateway* interface is specified to define gates used exchange packets with network layer, and parameters to control behaviour of the gateway (See Section 5.8). *PSGateway* module implements *IPSGateway* interface and is composed of two modules:

- *GatewayApplication* is responsible for providing a flexible user interface for applications to specify a query, and afterwards communicate query results back to the application. It triggers in-network planning by sending *INPDataRequest* packets instead of regular *DataRequest*. The module is provided by the IPVS of University in Stuttgart, and as such is not discussed in details.
- *NetworkTransformator* serves as an intermediate layer between the *GatewayApplication* and the network, and as such provides independence from differences in data representation. Furthermore, it performs spatial decomposition of a query in question. If a query spans more than one region, then it is decomposed into many partial

queries, each corresponding to a region of interest. A collection of supported regions is always available from the *Node-Region Association Manager*. A detailed information on how this is achieved is presented in Algorithm 5.9.

5.6.3 Node

Public Sensing Node (*PSN*) runs *Sensing Application* that interprets requests from the gateway. PSN specifies configuration parameters that are described in details in Section 5.8. If a request is made from the gateway to PSN, then a node in question is called the entry node. Upon receiving *PSGatewayReq*, the entry node determines the delivery method and acts accordingly. Processing of a gateway request is presented in Algorithm 5.10. If a region broadcast is not specified, the entry node starts a local query and delegates creation of protocol specific packets to GPSR API. Afterwards, it awaits reading responses for a specified period of time. The process of collecting reading responses is presented in Algorithm 5.11. In in-network planning, *Sensing Application* can counteract dropped GPSR packets carrying reading requests by selecting alternate sensors to query (Algorithm 5.12). In order for this to happen, current region boundaries are made available from *Subscription Manager*. If a node is an active participant of the system, SM can provide such information at any time.

5.7 Location Information Management

5.7.1 Custom Location Information

Example 5.8. Extending GPSRProtocol class to use a custom location information.

```
class GPSRProtocolExtended: public GPSRProtocol {
protected:
  psf::IGPSChip *gpsChip;

  virtual double getXPos();
  virtual double getYPos();
  virtual double getSpeed();
  virtual double getDirection();
};
```

In a default configuration, a position of a node is obtained using methods specified in *ManetRoutingBase* class. To enable position updates, *ManetRoutingBase* has to be instructed to subscribe for host position updates category. During the initialization phase of *GPSRProtocol* module, *registerPosition()* function must be called. If we are interested in using a custom location information without interfering with current implementation, *GPSRProtocol* class should be subclassed and appropriate methods ought to be redefined as in Listing 5.8. In the version of system, a GPS chip module from IPVS is used to provide uncertain location updates.

Algorithm 5.9 Sending partial queries to regions.

```
Input: A query (query), Node-Region Association Manager (manager)
Output: Send partial queries, each to a respective region
Output: Drop a partial query, if no nodes are associated with the region
 1: $sensorsAssigned \leftarrow$ false
 2: $regionPartialQueryMap \leftarrow \emptyset$ // maps a region to a partial query
 3: $supportedRegions \leftarrow$ get supported regions from $manager$
 4: for all $region$ in $supportedRegions$ do
 5:   if ($sensorsAssigned = query.count$) then
 6:     break
 7:   end if
 8:   $partialQuery \leftarrow$ get partial query for $region$ from $regionSensorListMap$
 9:   for all (virtual sensor $vs$ in $query$) do
10:     if ($vs.pos$ is contained within $region$) then
11:       $partialQuery \leftarrow$ add $vs$ to the list
12:       $sensorsAssigned \leftarrow sensorsAssigned + 1$
13:     end if
14:     if ($sensorsAssigned = query.count$) then
15:       break
16:     end if
17:   end for
18: end for
19: for all ($entry$ in $regionSensorListMap$) do
20:   $region \leftarrow entry.key$
21:   $partialQuery \leftarrow entry.value$
22:   if (no nodes in $region$) then
23:     drop $query$
24:   else
25:     if (node density in $region$ is less than specified threshold) then
26:       // use region broadcast
27:       for all ($node$ in $region.registeredNodes$) do
28:         send partial query to $node$
29:       end for
30:     else
31:       // use geographic routing
32:       $node \leftarrow$ select one node from $region.registeredNodes$
33:       $send$ partial query to $node$
34:     end if
35:   end if
36: end for
```

5.7.2 Regions and Populated Regions

To represent regions in a flexible manner, *AbstractRegion* and *AbstractPopulatedRegion* are specified. *AbstractRegion* interface defines methods for testing whether a specified position is within boundaries of a region in question, and calculating its area. *Abstract-Populated* interface defines methods to register, unregister, and test whether a node is already registered. *PouplatedRegion* class implements these interfaces and forms a base for the location management system.

5.7.3 Packet Formats

Algorithm 5.10 Processing of a gateway request in the entry node.

```
– Main routine
Input: Gateway request ($req \neq \emptyset$)
Output: Start a local query or take readings and send them back to gateway
 1: if ($req.isRegionBroadcast$ = true) then
 2:    $nodePos \leftarrow$ get current node position
 3:    $collectedReadings \leftarrow \emptyset$
 4:    for all (virtual sensor $vs$ in $req.partialQuery$) do
 5:       $dist \leftarrow$ distance from $nodePos$ to $vs.pos$
 6:       if ($dist \leq vs.radius$) then
 7:          $reading \leftarrow$ take reading
 8:          $collectedReadings \leftarrow$ collect $reading$
 9:       end if
10:    end for
11:    if ($collectedReadings$ is not empty) then
12:       send readings back to gateway
13:    end if
14:    delete $req$
15: else
16:    $queryID \leftarrow req.partialQuery.queryID$
17:    $queryPeriod \leftarrow req.partialQuery.queryPeriod/2$
18:    start a local query with $queryID$ and $queryPeriod$
19: end if
– Local query routine
Input: Gateway request ($req \neq \emptyset$)
Output: Distribute reading requests
 1: create a list of reading requests from $req.partialQuery$
 2: assign to each reading request local $queryID$
 3: // each reading request corresponds to a virtual sensor specified in $req.partialQuery$
 4: if ($req.inNetworkProcessing$ = true) then
 5:    pass the list of reading requests to GPSR API
 6: else
 7:    pass reading requests one by one to GPSR API
 8: end if
 9: // GPSR API is responsible for wrapping reading requests into protocol-specific pack-
    ets and sending them to routing module
```

Algorithm 5.11 Collecting reading responses in the entry node.

```
Input: Reading response ($resp \neq \emptyset$)
Input: Reverse route of the reading response ($reverseRoute$)
Output: collect reading or send back to gateway
 1: if ($reverseRoute$ is empty) then
 2:     // response arrived at the entry node
 3:     $queryID$ ← get local query ID from $resp.queryID$
 4:     $query$ ← get query with $queryID$ from query manager
 5:     if ($query = \emptyset$) then
 6:         // response is late
 7:         send reading back to gateway
 8:     else
 9:         add reading to $query$ results
10:     end if
11: else
12:     // route is broken at intermediate node
13:     send reading back to gateway
14: end if
```

Algorithm 5.12 Selecting alternate sensors in a node.

```
Input: INPDataRequest ($inp$), Region boundaries ($region$)
Output: A list of readings and control readings to request
 1: // sensors known to be available at the time of submitting the query
 2: $availableSensors$ ← get available sensors from $inp$
 3: // restrict available sensors to respective region
 4: for all ($vs$ in $availableSensors$) do
 5:     if ($vs$ is not contained within $region$) then
 6:         remove $vs$ from $availableSensors$
 7:     end if
 8: end for
 9: // an assumption is made that all readings have been taken and returned to the
    gateway except newly detected as being unavailable
10: $takenReadings$ ← $inp.allVirtualSensors$
11: $takenControlReadings$ ← $inp.allVirtualSensors$
12: for all ($vs$ in $inp.query$) do
13:     remove $vs$ from $availableSensors$
14:     remove $vs$ from $takenReadings$
15:     remove $vs$ from $takenControlReadings$
16: end for
17: // select sensors from $availableSensors$ knowing that some readings have been taken
18: $result$ ← run greedy selection algorithm with $availableSensors$, $takenReadings$ and
    $takenControlReadings$ as parameters
19: $readingsToRequest$ ← extract readings to request from $result$
20: $controlReadingsToRequest$ ← extract control readings to request from $result$
```

Example 5.9. Packets exchanged as part of location information management.

```
packet SubscriptionReq {
  IPAddress destAddr;
  IPAddress srcAddr;
  Coord srcPos;
}

packet SubscriptionResp {
  IPAddress destAddr;
  IPAddress srcAddr;
  RegionPtr region;
}
```

To support location information management, a number of packets is defined to implement two-way communication over the channel (Figure 5.9). *SubscriptionReq* packet is sent from the node to subscribe to the gateway or carry out a subscription update. *de*stAddr and *srcAddr* fields contain respectively the IP address of the server and the sending node. Position obtained from GPS module is stored in *srcPos*. *SubscriptionResp* packet, on the other hand, is sent from the gateway to the mobile node and contains information about the assigned region. In the following, underlying details of the structure being passed are explained.

5.8 Simulation Parameters

Parameters used in the simulation can be assigned in either the NED files or the configuration files, e.g., *omnetpp.ini*. A default value can also be given, which is used if the parameter is not assigned otherwise. A value assigned in NED cannot be overwritten form ini files. We "hardcode" some parameters in NED files, since they are considered to be part of the model. All other parameters that are considered to change during experimentation are put into ini files. Finally, all parameters are evaluated at the start of the simulation and cannot be change later.

Parameters are grouped by module name. For convenience, wildcard matching pattern of modules is provided in the brackets . We defined the following parameters with their default values:

1. *Net80211* module
   - *numManetHosts*
   - *pgs_x*
2. *SimulationControl* module (*.simulationControl)
   - *useInNetworkPlanning*
   - *useOptimizedOperation*
   - *warmup*

3. *GPSRRouting* module (**.manetroutingprotocol)
   - *timeToLive* (64)
   - *immediateBeacon* (false)
   - *beaconJitter* (0.5)
   - *beaconInterval* (1s)
   - *beaconExpiry* (3 * *beaconInterval* * (1 + *beaconJitter*))
   - *numOfBeacons* (0)
   - *planarization* (1)
   - *eraseTimedOutFragmentsAfter* (20s)
   - *tightlyCoupled* (false)
4. EnhancedManetHost module (*.mobileStation[*])
   - *mobilityType*
   - *gpsChipType* ("ExactGPSChip", hardcoded in parent)
   - *psCpuType* ("PSCpu", hardcoded in parent)
5. PSGateway interface (**.psGateway)
   - *useInNetworkProcessing* (true)
   - *nodeDensityThreshold* (0.5)
   - *regionPartition* (0)
   - *numColu*mns (1)
   - *numRows* (1)
6. PSCpu module (*.mobileStation[*].psCpu)
   - *sendPosAfterXUpdates* (5)
   - *psCpuLog*
   - *wiredServer*

Chapter 6
Evaluation

To check the effectiveness and efficiency of our algorithms, we evaluated them using a dataset representing environmental readings gathered over time. This chapter presents the simulation setup and performance evaluation results.

6.1 Methodology

Our public sensing system is implemented using OMNeT++ 4 network simulator and INETMANET framework. A node uses 802.11 implementation from INETMANET for ad-hoc WiFi communication. To improve the runtime of our simulations, the effective communication distance is restricted to 125m. For the mobile Internet access, we use a simple model of a UMTS channel shared amongst all nodes [21]. Node mobility is modelled using the Random Waypoint mobility model. Nodes are initially randomly distributed over a simulation area.

We run our simulations for one simulated day each. Adopted GPSR protocol requires initial convergence time. Nodes need to have sufficient time to gather information about neighbours, before making any forwarding decisions. A warmup period of $10s$ is introduced to delay the first query submission. We adjust the timespan between consecutive queries, so that at most one query is active in the system.

To run the actual simulations with real sensor data, we firstly run preliminary simulations. The main objective of preliminary simulations is to determine the optimal system parameters. We test the fraction of sensors for which a reading is obtained in simulations with a varying number of nodes, size of simulation area and sensing radius of a virtual sensor. After finding a “sweet spot”, we verify the effectiveness and efficiency of our algorithms with real sensor data.

6.2 Sensor Dataset

In the network research community many datasets are publicly available, but only few are suitable for public sensing. We run our simulations with the Intel Lab data [3], which is the sensor data collected in a specific sensor deployment in the *Intel Berkley Research* lab.

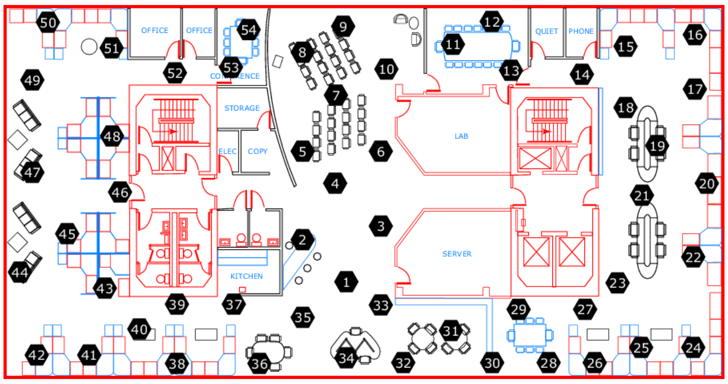


Figure 6.1. Sensor's arrangement in the *Intel Berkley Research* lab.

The dataset contains continuous time series of 54 sensors, monitoring different environmental parameters such as temperature, humidity and illuminance. These sensors reflect spatial correlations of the phenomena being monitored. A diagram representing sensor arrangement of sensors is shown in Figure 6.1. Using the original Intel Lab dataset as the input, a sensor reading at a specific point in space and time is made available to a node supporting specified virtual sensor.

Example 6.1. Intel Lab data provider module.

```
simple IntelLabDataProvider like BasicDataProvider {
  double warmup = default(0);
  string filename;
  double stretchFactor_x = default(1.0);
  double stretchFactor_y = default(1.0);
  double stretch_time = default(1.0);
  bool keepNAN = default(true);
}
```

IntelLabDataProvider models the sensor data used in our simulation. If a reading is not available in the original dataset, a NaN value (Not a Number) is stored to indicate this fact. Otherwise, the provider reports the last seen value. A stretch factor is defined in the *IntelLabDataProvider* module to extrapolate the original dataset over specified playground. To avoid excessive network traffic, we also manually adjust the timespan between consecutive queries. The module definition can be found in Listing 6.1.

The original Intel Lab data provider module is supplied by the IPVS. We had to adapt it, in order to integrate it with our public sensing system.

6.3 System Setup

6.3.1 Network

We defined a network that primarily consists of mobile nodes and a wired host. The wired host plays the role of public sensing gateway. Every mobile node has a sensing application

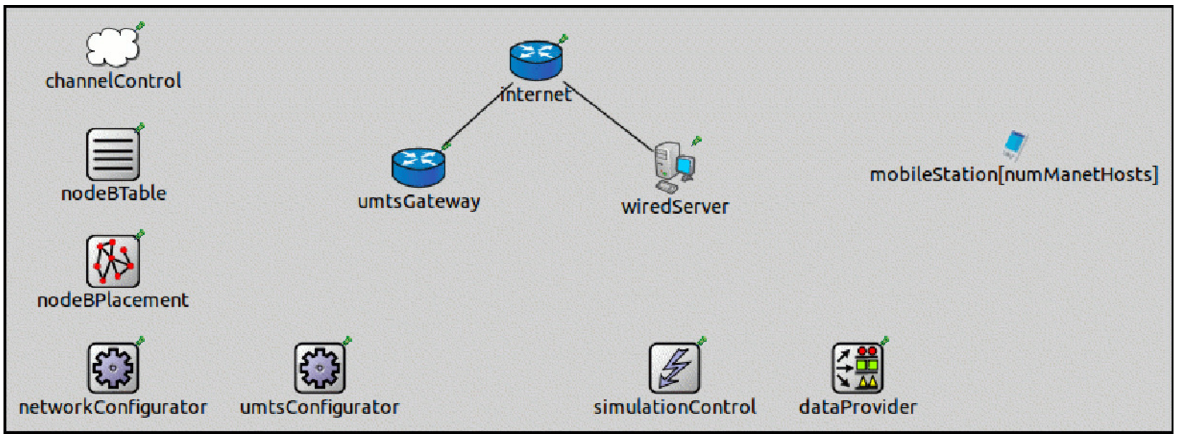


Figure 6.2. Overall network setup for Intel Lab data.

running that performs basic sensing tasks. The nodes act also as routers, since they forward data packets to other nodes. The overall network setup is presented in Figure 6.2. A network requires additional modules to make a simulation runnable. The following named modules are defined:

- *networkConfigurator* performs auto configuration of a network. It configures IP addresses and routing tables of all modules which have the *@node* property. The module only runs once, at the beginning of the simulation. All hosts and routers are in the same network and only differ in the host part. For simplicity, the module assigns one IP address to all interfaces of a node (mobile nodes are equipped with two interfaces: UMTS and WiFi). It is essential that IP addresses are unique throughout the network because current implementation of GPSR protocol uses them as unique IDs to store neighbours.
- *channelControl* is responsible for housekeeping associated with radio channels in wireless simulations. This module gets informed about the location of nodes, and determines which nodes are within communication or interference range. The packets are sent directly to the recipient and there is no dynamic connections creation.
- *nodeBTable* is a part of UMTS system and maintains information about all Node B's in the network. This includes their position, maximum transmission range and the provider ID.
- *nodeBPlacement* is a part of UMTS system. Node B placement module determines how Node B's are laid out in the network. The module enables more realistic simulations of mobile networks, e.g., where the maximum range of different node B's varies.
- *umtsConfigurator* maintains routing tables of all Node B's and mobile nodes to allow data forwarding based on routing tables. The module only runs once, at the beginning of the simulation.
- *simulationControl* loads the configuration parameters of specified model (e.g., Intel Lab model), creates continuously different types of queries, records physical and virtual readings, validates and updates the model if necessary.

- *dataProvider* serves as a bridge between an application and the actual data source. Information about sensor readings is stored in a file in form of a matrix. The data provider is used to retrieve sensor data from the file, provide readings upon request and determine the query periods.

Example 6.2. Network parameters setup.

```
network Net80211_intelLab {
  parameters:
    @display("i=block/network");
    int numManetHosts;
    int num_Provider;
    double pgs_x;
    double pgs_y = psg_x;
    string placementType;
}
```

The simulated network requires a basic parameters setup as in Listing 6.2. The number of nodes used in a simulation is determined by *numManetHosts*. *num_Provider* specifies a number of UMTS providers. Nodes are randomly deployed in a square playground of side-length *pgs_x*. *placementType* determines the lay out of Node B's in the network.

6.3.2 Mobile Node

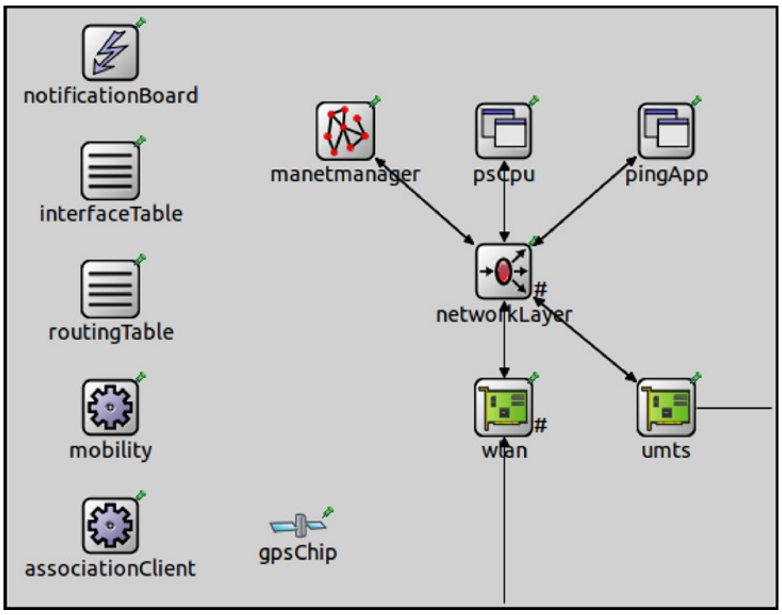


Figure 6.3. Extended mobile node module.

We defined an internal structure of a mobile node as in Figure 6.3. The host is equipped with the new IEEE 802.11g implementation. The mobility model is dynamically specified with the *mobilityType* parameter. The remaining significant modules are:

- *interfaceTable* contains a table of network interfaces (eth0, wlan0, etc.) in the host. Interfaces are registered dynamically during the initialization phase of NICs modules.
- *routingTable* maintains an IPv4 routing table and is directly accessed from other modules via C++ interface.
- *notificationBoard* makes possible for several modules to communicate in a publish-subscribe manner.
- *mobility* is responsible for moving around the node in the simulated playground.
- *wlan* is a network interface card that implements a 802.11g communication in ad-hoc mode. A wireless energy model can be defined.
- *umts* is a network interface card that encapsulates elements required for UMTS communication.
- *networkLayer* represents protocols of the network layer: *IP*, *ARP* and *ICMP*. The *ErrorHandling* module receives and logs ICMP error replies. The IP module performs IP encapsulation/decapsulation and routing of datagrams. The ARP module performs address resolution for interfaces and acts as a bridge between the IP module and the NICs. ICMP deals only with sending and receiving ICMP error packets.
- *associationClient* is a part of UMTS system.
- *manetmanager* dynamically creates GPSR protocol module and maintains two-way communication between the module and *networkLayer*.
- *gpsChip* adds uncertainty to the specified mobility model.
- *psCpu* performs basic public sensing tasks.

6.3.3 Mobility

Example 6.3. Mobility settings.

```
*.mobileStation[*].mobilityType = "RandomWPMobility"
*.mobileStation[*].mobility.x = -1
*.mobileStation[*].mobility.y = -1
*.mobileStation[*].mobility.updateInterval = 1s
*.mobileStation[*].mobility.speed = uniform(1mps,3mps)
*.mobileStation[*].mobility.waitTime = 1s
```

A mobility module plays an important role in the simulation, since it provides location information of a node and handles its movement in an unobstructed rectangular plane. A mobility module is needed even if the node is stationary, because it stores the location of a node, needed to compute wireless transmission range. After consideration, we have chosen the *Random Waypoint mobility model*, which configuration is given in Listing 6.3. This model assumes that nodes move in line segments. A random destination position is uniformly chosen for each line segment over the whole playground. The speed is defined also as a variate *uniform(minSpeed,maxSpeed)*. When a node reaches a target position, it waits for a specified amount of time (*waitTime*), calculates a new random position and the node moves on. The initial *x, y* coordinates of nodes are randomly distributed over the playground.

6.3.4 NIC Settings

Two interesting things to be mentioned about the 802.11g radio module are *sensitivity* and the *bit error rate table* parameter. The sensitivity determines which received signals with power below should be ignored. In other words, it limits the range of communication between two nodes. Firstly, we calculate the minimum receive power of an incoming signal (Equation 6.1). Then, the effective coverage area of a transmitter is given by Equation 6.2. This radio wave propagation model is implemented in the current version from INETMANET 802.11g radio module.

$$minReceivePower = 10^{\frac{sensitivity}{10}} \tag{6.1}$$

$$communicationDistance = \left(\frac{\left(\frac{lightSpeed}{carrierFrequency}\right)^2 \times transmitterPower}{16 \times \Pi^2 \times minReceivePower} \right)^{\frac{1}{alpha}} \tag{6.2}$$

The bit error rate table parameter allows to define a file with precomputed values of packet error rates (PERs). The file contains entries in which for different speeds of wireless medium, packet sizes and signal-to-noise (SNR) ratios, a bit error rate is defined. This stipulates a more realistic simulation. The file was already included in INETMANET framework.

6.4 Metrics

Accurate evaluation of our public sensing system requires two elements: a representative workload and a set of suitable metrics. Queries are our primary mechanism for retrieving sensor readings from a network and are represented by a set of virtual sensors. In the following considerations, we define quality of query results and efficiency of query execution.

Quality of a query is measured by *coverage*, which is defined as the ratio of number of obtained sensor readings to the total number of all sensors specified in the query. This definition is intuitive enough for a user, since it can be easily determined from the coverage, whether a query captured complete picture of the real world. From a user perspective, an increased coverage leads to more meaningful results, which is the main goal in terms of effectiveness. Furthermore, we examine the *prediction accuracy* of predicted readings that is assessed by root mean square deviation. This is a frequently used measure, used to measure differences between values predicted from a model and the values actually observed.

To measure the degree of optimization in terms of efficiency, we define the following metrics:

- *Network load* is defined as the total number of transmitted packets in the network (WiFi and UMTS).
- *Number of saved sensor readings.* Since we enrich our interactive sensor querying with statistical modelling techniques, this number represents the difference in the number of sensors initially specified in a query and the number of sensors required to answer the query with sufficient confidence. A large number of saved sensor readings has a positive impact on the network load.
- *Number of redundant sensor readings.* In proposed model-driven execution scheme we submit optimized queries in rounds. If a query from the first round does not provide sufficient coverage, another attempt is made to take sensor readings. This increases the energy consumption of nodes and the overall network load, therefore each additional reading taken in next rounds, is considered to be redundant.

6.5 Experiments

Table 6.1. Different simulation scenarios.

Scenario name	Use MDQE	Query delivery method	Use INP
Naive	no	region broadcast	no
MDQE	yes	region broadcast	no
Optimized	no	region broadcast + position-based	no
Optimized MDQE	yes	region broadcast + position-based	no
Adaptive MDQE	yes	region broadcast + position-based	yes

To verify the performance of our system, we specified five different simulation scenarios as shown in Table 6.1. In the model-driven query execution (MDQE) scheme, a statistical model is used to query these sensors, whose readings cannot be derived from the model itself with sufficient confidence. This contributes to the number of saved sensor readings. The region broadcast is the most naive query delivery method, as it blindly sends a query via UMTS to each node contained within a region. In position-based approach, a query is delivered to an entry node in a region of interest and disseminated via WiFi. However, position-based approach is used only if the node density in a region is above a predefined threshold. This threshold is determined in preliminary simulations. Lastly, in-network planning (INP) triggers alternate sensor selection, if a sensor happens to be unreachable. This is performed during query execution and can have a positive impact on the number of redundant sensor readings.

In the following, we firstly validate the correctness of GPSR protocol implementation. This is a necessary requirement to make meaningful conclusions in determining optimal system parameters, and running the actual simulations. Secondly, we confirm our intuitive concept of burst mode and show that results have an impact on GPSR efficiency. Finally, we determine optimal system parameters and perform the actual simulations.

6.5.1 Validation of GPSR protocol

The effects of incorrect protocol implementation can have a significant impact on performance, not mentioning bringing down an entire network. However, network protocols are difficult to test due to the exponential size of the state space they define. We are in no position to perform tests for all possible combinations of events (packet arrivals, packet losses, timeouts, etc.) and protocol states.

We took an advantage of the fact that GPSR protocol is implemented in OMNeT++, and propose the following simplified validation scheme adapted to our system:

1. Get from the simulator information about mobile nodes.
2. Specify a query in the following way: for each mobile node define a corresponding virtual sensor, whose position is equal to the position of a node in question, and sensing radius is adjusted to the node's mobility.
3. Choose an entry node.
4. Start a query.
5. Run Dijkstra algorithm to compute shortest paths to all other nodes. Await readings from reachable nodes. If a path exists, GPSR protocol guarantees packet delivery.
6. Collect query results.
7. If the query timeouts, check whether all results are collected.
8. Select a different entry node and go back to point 4. If all nodes have been selected, finish the simulation.

Assumptions made and things worth to keep in mind:

- Queries must not overlap. The timespan between consecutive queries must be experimentally adjusted.
- Positions of nodes do not overlap and, as such, neither corresponding virtual sensors.
- TTL of GPSR packet should be specified accordingly to the number of nodes.
- ARP timeouts can lead to missing sensor readings.

Proposed testing scheme generates continuously queries under nodes mobility. If a new query is started before the previous one is completed, this can be an indication of a routing problem (assuming that everything else is set appropriately). Proposed testing scheme can be further extended to support overlapping queries by exploiting the query ID.

6.5.2 Burst Mode

The burst mode is an extension to the adopted routing protocol, responsible for grouping targets in the general direction of the destination. To get a more accurate impression of how well this extension performs, we compared it to the naive approach, in which all requests are sent one by one. The parameters chosen for our simulation are presented in Table 6.2. We ran the simulation for every possible combination of parameter values. The

Table 6.2. Values for the parameters in burst mode simulation.

Parameter name	Value
numManetHosts	20..120 step 5
pgs__x	300, 500, 700
inNetworkProcessing	false, true

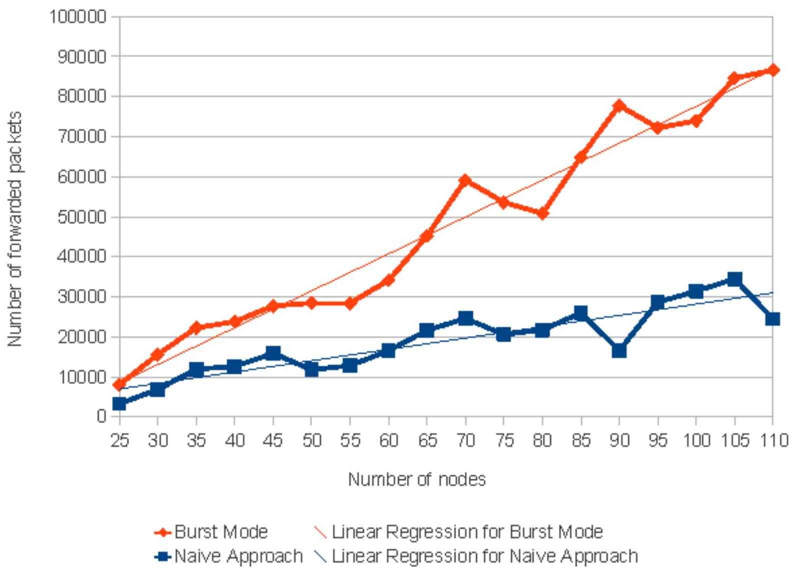


Figure 6.4. Burst Mode vs. Naive Approach.

testing methodology is similar to the proposed validation scheme of GPSR protocol. When the last chosen node finished executing a query specified by virtual sensors corresponding to all other nodes, the simulation was completed. In this case we had do adjust the size of data queue in NIC, since the number of messages sent in the naive approach exceeded the default queue size of a node. Our simulations confirm initial expectations towards this approach (Figure 6.4). Firstly, the number of forwarded packets in the network, required to reach every destination, is significantly smaller than in the naive approach. As a consequence, the amount of energy required for WiFi transmission is reduced. Secondly, the performance gain of burst mode decreases with the number of targets specified in one shot. In a special case, if only one target is specified, the performance of burst mode and the naive approach are considered to be equal.

6.5.3 System Parameters

Table 6.3. Values of parameters in specified scenarios.

Parameter name	Value
numManetHosts	40, 50, 60, 70
pgs__x	100,120,140,160

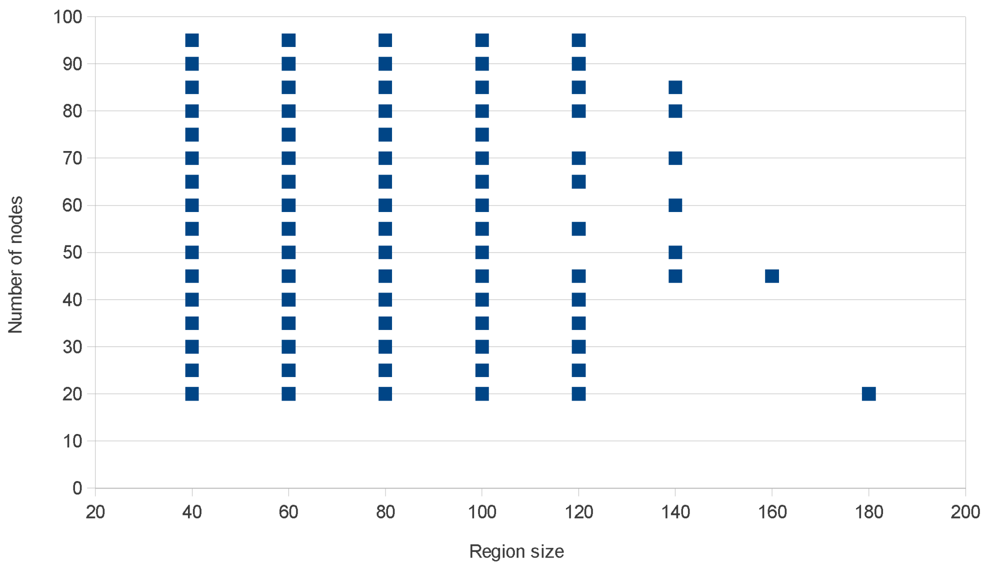


Figure 6.5. Scenarios indicating fully connected networks of mobile nodes.

To determine the optimal system parameters, we firstly compare two specified scenarios: region broadcast and position-based, with disabled model-driven query execution. Intuition dictates that the former can yield worse results, if the network is partitioned. Recall that in NIC settings, the transmission range of WiFi is limited to 125m. To avoid such situations, we experimentally determine a minimum number of nodes for a given region size required to form a network of interconnected nodes. In Figure 6.5, we present different scenarios from which information about fully connected networks can be inferred. This does not show, however, that all nodes remain reachable at any time, but is rather a clue what can be expected during simulations. Based on our preliminary research, we specified parameters for the actual simulations as in Table 6.3. Unfortunately, the time frame planned for this thesis did not allow us to investigate further, in order to test the system with a representable workload.

Chapter 7

Summary

7.1 Conclusion

In this thesis, we presented an opportunistic public sensing system which takes an advantage of user smartphones to perform a flexible and efficient acquisition of sensor readings. The great potential of public sensing comes from the large number of people covering large public spaces. However, the usage of smartphones as mobile wireless sensing platforms introduces two main challenges. Due to the uncontrolled mobility of people, no guarantees about coverage of sensed area and data quality can be made. This can lead to sensing coverage gaps. Furthermore, sensing should not interfere with normal usage of smartphones, which requires in general, efficiency in terms of energy consumption. We use statistical modelling techniques to enrich interactive sensor querying. To design an efficient communication system for query dissemination, we adopted and successfully extended GPSR protocol.

We firstly specified a conceptual model that describes and represents our system. In this model, a node is capable of WiFi communication over short distances. For the mobile Internet access, a simple model of a UMTS channel is used. For the purpose of public sensing, every node has access to an identical set of environmental sensors and runs our sensing software. A GPS module is used to obtain a position fix. As part of the system model, we defined also an interface for applications to submit queries, i.e., the gateway. An application specifies a query through a set of virtual sensors. This approach provides a flexible way to specify requests independently of node's mobility. To reflect spatial properties of phenomena of interest, we use a probabilistic Gaussian model.

In the system design, detailed architecture of our two-tier public sensing system was presented. We provided multiple views on different aspects of implemented approach, such as the gateway, model-driven execution scheme and different types of queries. Our system supports unoptimized and model-driven queries. The former requests all virtual sensors specified by the application. Model-driven query, on the other hand, requests only these sensors, whose readings cannot be derived from the model with sufficient acceptable confidence. A multi round scheme is implemented in order to attempt answering the specified query. We use in-network processing paradigm to efficiently distribute queries to mobile nodes, and collect results afterwards. To reduce the number of hops required for adopted routing scheme to successfully deliver a request, we introduced the concept of area division into regions. Different query delivery models were defined as well. To refine the quality of query results during query execution, we use in-network planning to employ alternate alternate sensor selection. Finally, we extended adopted routing protocol to carry multiple targets in the general direction of the destination.

The system was simulated using OMNeT++ 4 and INETMANET extension. A dataset reflecting spatial correlations of environmental phenomena, such as temperature, was used to provide real sensor readings. To evaluate our public system, we defined metrics fitting our scenarios. To measure the quality of query results, we investigated the fraction of obtained sensor readings to all sensors specified in the query. Furthermore, we examined the prediction accuracy of readings to determine how far predicted values varied from the values actually observed. To perform actual simulations, adopted routing protocol had to be verified to strengthen final conclusions. Our empirical studies confirmed the correctness of implementation, as far as our system is concerned. Moreover, we showed that intuitive concept of burst mode have a promising impact on GPSR efficiency. However, the preliminary simulations results revealed that routing-base approach is suitable in rather dense scenarios.

7.2 Future Work

The periodical broadcasting in GPSR of hello messages has several drawbacks such as unnecessary utilization of network resources, interferences with regular data packets, and consumption of scarce battery power. It is a time-based strategy that is applied by most of other position-based routing protocols proposed in the literature [24]. Other beaconing strategies based on the distance that node has travelled since last beacon and speed, should be considered. Velocity and direction of nodes should be used to predict the time when two nodes are not within transmission range.

In more mobile scenarios, the number of unreachable nodes in the neighbour table of GPSR can increase due to the too greedy selection strategy employed in the original design. Moreover, in reality transmission ranges may be strongly irregular due to obstacles and interferences. Further research should be done to restrain too greedy selection and introduce some notion of a safety margin.

The multivariate Gaussian distribution was used to model the temperature phenomena. Although it performs well, it does not directly provide any information on temperature at locations where no sensors were placed. In such cases, regression techniques could be used to perform prediction [30]. Unfortunately, they provide no notion of uncertainty about these predictions. Gaussian processes should be considered in the future work as an extension of multivariate Gaussian distribution to infinite-sized collections of real-valued variables.

List of Figures

2.1 Public sensing is one of the aspects of people-centric sensing [15]. 13
2.2 Determining location using Skyhook's system [8]. 22
2.3 An example of a Mobile Ad-hoc Network (MANET). 23
2.4 An example of a routing zone of node A with radius equal to 2 hops. E, D, B, J, E and H are the border nodes. 25
2.5 An example of geographic multicast. 27
2.6 An example of geocasting. 27
2.7 A communication void, with respect to the destination D, occurs at node S where greedy forwarding fails. 28
2.8 The RNG planarization algorithm. 29

3.1 Overall system model. 31

4.1 A general overview of system architecture. 35
4.2 A detailed system architecture. 37
4.3 Exploiting locality results in decreased number of hops. 39
4.4 Model-driven execution scheme. 41
4.5 Grid-based vs. honeycomb-based structuring scheme. 42
4.6 Overlapping readings in a region broadcast. 43
4.7 Grouping of reading requests sent in the same direction. 44
4.8 A node moves forth and back violating its region assignment. 48
4.9 Transmission range (TxR) of a node vs. delivery radius. 50
4.10 Processing of burst packets. 51

5.1 Overview of GPSR API communication scheme. 61
5.2 Connection of an ad-hoc routing module to the network layer. 67

6.1 Sensor's arrangement in the *Intel Berkley Research* lab. 78
6.2 Overall network setup for Intel Lab data. 79
6.3 Extended mobile node module. 80
6.4 Burst Mode vs. Naive Approach. 85
6.5 Scenarios indicating fully connected networks of mobile nodes. 86

Bibliography

[1] BeiDou Navigation System, Wikipedia.

[2] GALILEO Satellite System, Wikipedia.

[3] Intel Lab data, http://db.csail.mit.edu/labdata/labdata.html.

[4] LORAN General Information, http://www.navcen.uscg.gov/?pageName=loranMain.

[5] OMNeT++ Network Simulation Framework, http://www.omnetpp.org/.

[6] Researchers Crack IPhone's Wi-Fi Positioning System.

[7] Russian GLONASS System, Wikipedia.

[8] Skyhook, http://www.skyhookwireless.com/howitworks.

[9] Mehran Abolhasan, Tadeusz Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1 – 22, 2004.

[10] A. Ariza-Quintana, E. Casilari, and A. Triviño Cabrera. Implementation of manet routing protocols on omnet++. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 80:1–80:4, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[11] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[12] S. Basagni, M. Conti, S. Giordano, and I. Stojmenović. *Mobile ad hoc networking.* IEEE Press, 2004.

[13] Doina Bein. Self-configuring, self-organizing, and self-healing schemes in mobile ad hoc networks. In *Guide to Wireless Ad Hoc Networks*, Computer Communications and Networks, pages 27–41. Springer London, 2009. 10.1007/978-1-84800-328-6

[14] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, and Ronald A. Peterson. People-centric urban sensing. In *Proceedings of the 2nd annual international workshop on Wireless internet*, WICON '06, New York, NY, USA, 2006. ACM.

[15] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, Ronald A. Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristóf Fodor, and Gahng-Seop Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, 12:12–21, July 2008.

[16] D. Chen and P.K. Varshney. A survey of void handling techniques for geographic routing in wireless networks. *Communications Surveys Tutorials, IEEE*, 9(1):50 –67, quarter 2007.

[17] Thomas Clausen and Philippe Jacquet. Optimized link state routing protocol (olsr). RFC 3626, Internet Engineering Task Force, October 2003.

[18] Abhimanyu Das and David Kempe. Sensor selection for minimizing worst-case prediction error. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 97–108, Washington, DC, USA, 2008. IEEE Computer Society.

[19] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 588–599. VLDB Endowment, 2004.

[20] Shane B. Eisenman, Emiliano Miluzzo, Nicholas D. Lane, Ronald A. Peterson, Gahng-Seop Ahn, and Andrew T. Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Trans. Sen. Netw.*, 6:6:1–6:39, January 2010.

[21] Thorsten Frosch. Umts-implementierung für omnet++, 2011.

[22] Carlos Guestrin, Andreas Krause, and Ajit Paul Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 265–272, New York, NY, USA, 2005. ACM.

[23] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The zone routing protocol (zrp) for ad hoc networks. Internet-draft, IETF MANET Working Group, July 2002. Expiration: January, 2003.

[24] Marc Heissenbüttel and Torsten Braun. Optimizing neighbor table accuracy of position-based routing algorithms, 2005.

[25] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, volume 353 of *The Kluwer International Series in Engineering and Computer Science*, pages 153–181. Springer US, 1996. 10.1007/978-0-585-29603-6

[26] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia*, 14:8–13, October 2007.

[27] E.D. Kaplan and C.J. Hegarty. *Understanding GPS: principles and applications.* Artech House mobile communications series. Artech House, 2006.

[28] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM.

[29] Brad Nelson Karp. *Geographic routing for wireless networks.* PhD thesis, Cambridge, MA, USA, 2000. AAI9988566.

[30] Andreas Krause. *Optimizing sensing: Theory and applications.* Dissertation, Carnegie Mellon University, United States, Pennsylvania, 2008.

[31] Andreas Krause, Eric Horvitz, Aman Kansal, and Feng Zhao. Toward community sensing. In *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, pages 481–492, Washington, DC, USA, 2008. IEEE Computer Society.

[32] Nicholas D. Lane, Shane B. Eisenman, Mirco Musolesi, Emiliano Miluzzo, and Andrew T. Campbell. Urban sensing systems: opportunistic or participatory? In *Proceedings of the 9th workshop on Mobile computing systems and applications*, HotMobile '08, pages 11–16, New York, NY, USA, 2008. ACM.

[33] Benyuan Liu, Peter Brass, Olivier Dousse, Philippe Nain, and Don Towsley. Mobility improves coverage of sensor networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pages 300–308, New York, NY, USA, 2005. ACM.

[34] Nicolas Maisonneuve, Matthias Stevens, Maria E. Niessen, Peter Hanappe, and Luc Steels. Citizen noise pollution monitoring. In *Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government*, dg.o '09, pages 96–103. Digital Government Society of North America, 2009.

[35] M. Mauve, A. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *Network, IEEE*, 15(6):30 –39, nov/dec 2001.

[36] S. Misra, I. Woungang, and S.C. Misra. *Guide to Wireless Ad Hoc Networks.* Computer communications and networks. Springer, 2009.

[37] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90 –100, feb 1999.

[38] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24:234–244, October 1994.

[39] Damian Philipp, Frank Dürr, and Kurt Rothermel. A sensor network abstraction for flexible public sensing systems. In *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, October 2011.

[40] Anton Schwaighofer, Volker Tresp, and Kai Yu. Learning gaussian process kernels via hierarchical bayes. In *In Advances in Neural Information Processing Systems (NIPS*, pages 1209–1216. MIT Press, 2004.

[41] Anthony Steed and Richard Milton. Using tracked mobile sensors to make maps of environmental effects. *Personal Ubiquitous Comput.*, 12:331–342, February 2008.

[42] Ivan Stojmenović. *Location Updates for Efficient Routing in Ad Hoc Networks*, pages 451–471. John Wiley & Sons, Inc., 2002.

[43] Y.C. Tay and K.C. Chua. A capacity analysis for the ieee 802.11 mac protocol. *Wireless Networks*, 7:159–171, 2001. 10.1023/A:1016637622896.

[44] International Telecommunication Union. The World in 2010: ICT Facts and Figures, 2010.

[45] Andras Varga. Omnet++. In Klaus Wehrle, Mesut Güneş, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 35–59. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-12331-3

[46] Michael G. Wing, Aaron Eklund, and Loren D. Kellogg. Consumer-grade global positioning system (gps) accuracy and reliability. *Journal of Forestry*, 103(4):169–173, 2005.

[47] S.L. Wu and Y.C. Tseng. *Wireless ad hoc networking: personal-area, local-area, and the sensory-area networks.* Wireless networks and mobile communications series. Auerbach Pub., 2007.

[48] Yu Xiao, Petri Savolainen, Arto Karppanen, Matti Siekkinen, and Antti Ylä-Jääski. Practical power modeling of data transmission over 802.11g for wireless applications. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 75–84, New York, NY, USA, 2010. ACM.

[49] Sunhee Yoon and Cyrus Shahabi. The clustered aggregation (cag) technique leveraging spatial and temporal correlations in wireless sensor networks. *ACM Trans. Sen. Netw.*, 3, March 2007.

[50] Daqing Zhang, Bin Guo, Bin Li, and Zhiwen Yu. Extracting social and community intelligence from digital footprints: An emerging research area. In Zhiwen Yu, Ramiro Liscano, Guanling Chen, Daqing Zhang, and Xingshe Zhou, editors, *Ubiquitous Intelligence and Computing*, volume 6406 of *Lecture Notes in Computer Science*, pages 4–18. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-16355-5_4.

[51] Srdjan Čapkun, Maher Hamdi, and Jean-Pierre Hubaux. Gps-free positioning in mobile ad hoc networks. *Cluster Computing*, 5:157–167, 2002. 10.1023/A:1013933626682.