Automatic Management of Bluetooth Networks for Indoor Localization

Thesis for the degree of Master of Science in Software Engineering
Chair: Design and Analysis of Communication Systems (DACS)
Faculty: Electrical Engineering, Mathematics and Computer Science
University of Twente

Author:

Markus Jevring, s0154377
2008-08-21

Supervising committee:

Dr. Ir. Cristian Hesselman (Telematica Instituut)
Dr. Ir. Aiko Pras (University of Twente)
Dr. Ir. Geert Heijenk (University of Twente)

Automatic Management of Bluetooth Networks for Indoor Localization

Markus Jevring
markus@jevring.net
University of Twente

Abstract

This work explores automatic management of Bluetooth-based sensor networks for indoor localization. In particular, we will discuss algorithms that can reduce the number of active Bluetooth sensors needed in such a network, while maintaining comparable localization performance to that of the un-optimized, or full network. The main advantage of such optimization is that it reduces need for human effort in planning the Bluetooth network, which in turn reduces the costs of managing these systems. This is particularly important in ubiquitous computing systems in general, which typically contain many (embedded) sensors, possible of different types.

Our contributions are the algorithms themselves and the experimental evaluation of the algorithms and the networks that the algorithms create. Our algorithms are based on the individual values of the contributing devices to the network. These algorithms use an architecture we created called AMBIENT to affect the changes to the network after the algorithms have performed the necessary calculations. The AMBIENT architecture is built upon an existing distributed application called the Context Management Framework. We will discuss in detail the inner working of the algorithms, the AMBIENT infrastructure used, and the performance of the various algorithms. Our results show that the optimizers can produce networks of approximately half the size of the original network while maintaining similar prediction performance.

Table of Content

1 Introduction

In this thesis, we will look at how to optimize a network of Bluetooth devices that is being used to determine the locations of other Bluetooth devices in an indoor setting. Our goal is to have as few devices in the network active as possible in order to, for instance, reduce power consumption. We accomplish this by the use of algorithms that create smaller networks by calculating which Bluetooth devices should be available. The devices that should not be available are made unavailable remotely via a remote management framework that we have constructed. These algorithms determine which Bluetooth devices should be available based on the devices' relative value in the network. We then evaluate the performance of these networks in a real-world experimental setting. The reduced network should be able to offer a localization performance comparable to the "full" network in order to be effective. Our work contributes to the automatic configuration and optimization of ubiquitous computing environments.

This chapter starts with an introduction of the fields that are most relevant for our work: ubiquitous and context-aware computing (Section 1.1), location-based services (Section 1.2), and automatic management (Section 1.3). We also include a primer on fingerprinting and fingerprint-based localization systems (Section 1.4). Next, we discuss the goals, approach and contributions of this work (Section 1.5), which are part of the intersection of the three research areas outlined before. We conclude this chapter with an overview of the rest of this thesis (Section 1.6).

1.1 Ubiquitous and Context-aware Computing

The term ubiquitous computing was coined by Mark Weiser in an article published in 1991 in Scientific American [1]. In this article, Weiser describes his vision for something he calls Ubiquitous Computing. Weiser believed that, for computing to benefit society, it must integrate into, and become part of, the environment in which we live our normal lives. Utilities like electricity, plumbing or the telephone system are all incredibly complex, yet people use them daily without ever having to care about their inner workings. Weiser writes "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". Ubiquitous computing devices could be sensors in clothing that monitor health, or sensors built in to office buildings that sense people's location and offers services based on where a user is. An example could be that a user might want to print a document, and, instead of knowing which printer is available at his current location, he would simply tell the system to print the document and the system finds the printer nearest to him, prints the job, and then tells him where he can pick up the printouts.

An important enabler for ubiquitous computing is the ability of a system to adapt its service offering to the particular situation of a particular "entity" (e.g. a user, a device, or a location) which is called context aware computing [36][43]. In this paradigm, tasks are carried out based on, or to determine, contextual information about some entity. Imagine a user is running late for a meeting. The other meeting participants can see that he is in his office, because the location determination system in the building places his phone in that location, and in addition to this, the user is typing on his keyboard. This lets the other participants decide if they should wait for the user or not. When the user gets up and goes to the meeting, he doesn't have to switch his telephone to silent mode, because not only has the system detected that he is in a meeting room, but his calendar also indicates that he is in a meeting. On the way home from work later that day, his phone chimes as he approaches the grocery store close to his house. His refrigerator noticed that he is out of cheese, and that the milk will expire the following morning, so it instructs his phone to tell the user to purchase more cheese and milk at a convenient location. As the user is known to walk home, the system determines that the most convenient location is the grocery store closest to the user's house, on his way home.

This is a series of telling examples of what context aware computing is capable of. Context aware computing fuses computation with knowledge of the real world. This knowledge is primarily gained by sensors but can also, as in the example with the calendar above, be derived from input from the user. The marriage of ubiquitous and context aware computing is an advantageous one, as ubiquitous

computing strives to populate the world with sensors, and context aware computing strives to use the kind of information that such sensors would provide.

1.2 Location-based Services

One popular branch of context aware computing is location-based services. Location-based services make use of location information about an entity to offer services based on this information. For example, services can use either the location of the user and offer certain services to the user based on the user's location, or they can use the location of one entity and offer services to another entity. An example of the first case could be a Personal Digital Assistant (PDA)-based tour guide that gives information about the sights in the user's current location. An example of the other case could be a system that keeps track of where your co-workers are, so that they can be easily located, should you need to meet them. Other prime examples include the office scenario mentioned above (indoor location of user), as well as the printing scenario in the previous section (location of the printer).

A familiar scenario is the use of Global Positioning System (GPS) navigation [38] in modern cars. Knowing your current location and your intended destination, along with a set of maps, lets you plot an optimal course to get to your intended destination. While GPS is often an excellent tool for location determination outdoors, it is less suited for indoor location determination as the signals have trouble penetrating structures such as buildings [2]. GPS also has problems in urban areas where tall buildings can create urban canyons where a GPS receiver would have trouble getting a clear signal from enough satellites to pinpoint its location [3].

The problem of determining indoor location can be solved in different ways. There are several ways to determine the location of an entity without using GPS. One option is to use range-combining techniques. These techniques use signal characteristics information like time-difference-of-arrival (TDOA) [17][18][19][20][21] or received-signal-strength-indicator (RSSI) [14][22] to perform calculations, such as multilateration or trilateration, to determine the location of an entity in relation to some other infrastructure devices, for which locations are known a priori. There are several products and projects that use these approaches. These will be discussed in more detail in the chapter on related work.

One way to determine location is to use fingerprinting [33]. A fingerprint is a collection of signal property readings for a particular device, for instance in terms of response rate or signal strength. A system that uses this technique would work by creating fingerprints for each known location, such as an office or a conference room, and then compares them with the current fingerprint of the device we want to locate. The "best" match is the most likely location of the device we want to locate (and of the person carrying this device). A fingerprint for a particular location contains information about which sensors in a network can see the device that represents the location in question, as we will see later in this work.

In our work, we create fingerprints with the use of a network of inquiring Bluetooth devices. This system is that described in [37]. We chose this approach due to the low cost and pervasiveness of Bluetooth devices, and because it does not require the user to wear a custom device, or run any special software. The only requirement is that the user carries a discoverable Bluetooth device. As most mobile phones today are equipped with Bluetooth, this requirement is easily satisfied.

1.3 Automatic Management

We define automatic management as to the ability of a system to carry out management [44] tasks with little or no human intervention. Examples of management tasks include configuration, fault detection, error handling, monitoring, etc. Automated management is sometimes referred to as autonomic management [4]. The word "autonomic", in this case, refers to the autonomic nervous system in humans, and its ability to manage the running machine that is the human body. Autonomic management has four key points, called the self-* properties [4]. These are:

- Self-Healing: the ability to detect when errors occur and fix them

- Self-Configuration: the ability to automatically determine the configuration of the system, which components should have which settings, etc.
- Self-Optimization: the ability to choose the best configuration for the different components to maximize quality according to a certain metric. Examples of metrics are performance, speed, memory usage or power usage.
- Self-Protection: the ability to protect itself from attacks.

We believe that automatic management is particularly important for ubiquitous computing as they typically contain many different types of networked sensors that are embedded in the environment ("disappear from sight" [1]). Our expectation is that this explosion of complexity and heterogeneity will result in a steep increase in the costs of managing ubiquitous computing systems, for instance in terms of equipment, software and people. One potential solution to help combat this increasing complexity is to employ automatic management solutions This is especially true in the case where ubiquitous computing systems make their way in to the lives of non-technical people. For instance, the average consumer cannot be expected to know how to manage their home health monitoring network or automated lighting system, and will therefore require automatic management solutions to do it for them.

As location-based systems rely on a potentially large number of infrastructure devices being deployed in the service area, the task of managing them quickly becomes very resource intensive. Thus location-based systems are prime candidates for automatic management. Management in this particular case involves operations like changing the settings on each device, or turning the devices on or off to suit the currently desired environment.

Automatic management of systems like these allow some of the self-* properties to be fulfilled. The system can be self-configured based on the current needs of the environment. It can also be self-optimized based on some criteria set either by humans or some derived criteria from the environment. For example, a criterion could be to reduce the power consumption of the network, which could be done by reducing the number of sensors active in the network, or telling the devices to scan less often. There could also be financial reasons. One might be interested in deploying as few devices as possible to reduce cost, or one might be interested in finding the most valuable (with regard to contribution) devices and spend extra resources on them by, for example, adding redundancy. It could also be self-healing by determining the most suitable replacement for devices that that are lost due to failure or due to being turned off.

1.4 Fingerprinting

The system that we will be building upon, described in [37], uses a technique called fingerprinting to determine location. When the system wants to determine the position of an unknown device, it looks at the fingerprint for that device from the last N minutes (at the time of this writing, N is three). It then compares this fingerprint with the recorded fingerprints of all known locations. These comparisons are done by calculating the divergence (explained below) between the *observed* fingerprint (the fingerprint of the unknown device) and the recorded fingerprints (of the known locations). The locations are then ranked, in ascending order, by these divergences. As a low divergence measure indicates a high similarity between two fingerprints, the first location is the most likely one, and so on. The concepts introduced here will be explained in detail below. We will also briefly mention fingerprint normalization. The advantages of fingerprint normalization will be discussed in Section 7.5.

Unknown device – An *unknown device* is a device whose position we want to determine. This will typically be a mobile phone or PDA with Bluetooth on and discoverable. A fingerprint for an unknown device is referred to as an *observed* fingerprint. An unknown device is sometimes referred to as a *mobile device*.

Fingerprint – A fingerprint is a set of <inquiring device, response rate> pairs. Fingerprints are linked to devices. In the case of known locations (see below), the fingerprint identifies a room. In the case of unknown devices, the fingerprint indicates the location of a device, which in turn is connected to an entity, such as a person. Fingerprints for a given device are created by looking at all the inquiry results over the last N minutes (N is three for unknown devices and ten for known locations), and seeing how often the device that is being fingerprinted is seen by each of the inquiring devices. These inquiries are performed at fixed intervals, and each inquiry has a fixed duration. The *response rate* is defined as the ratio between how many times the fingerprinted device is seen by the inquiring device and how many inquiries the inquiring device sent during the time period. Each fingerprinted device can be seen by zero or more inquiring devices. A set of pairs of inquiring devices and their respective response rate for the device being fingerprinted is put into the fingerprint of that device. The resulting fingerprint can be visualized as a histogram, where each bar indicates an inquiring device, and the bar height indicates the response rate for that inquiring device. Each bar is called a *contribution*.

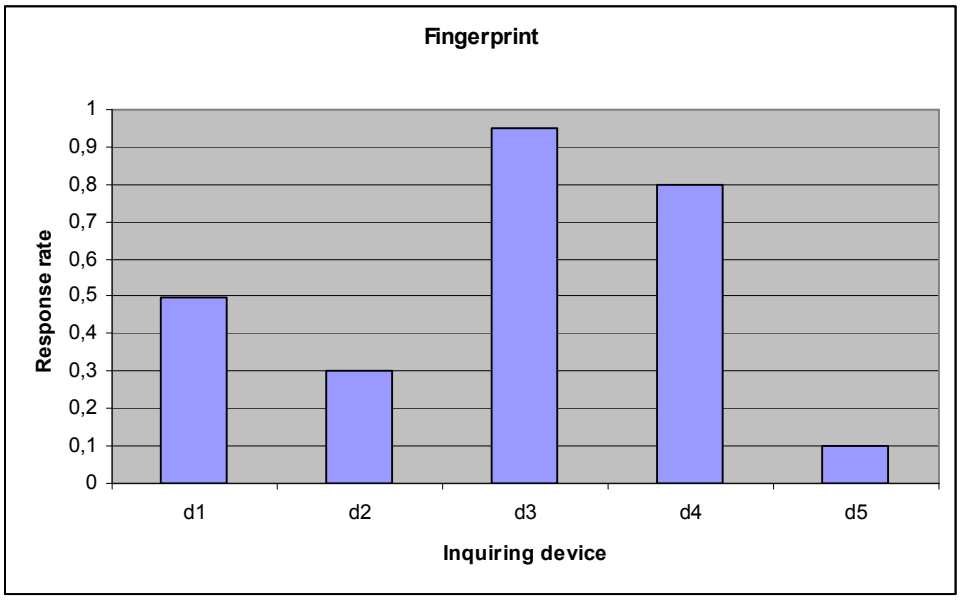

Figure 1 – A sample fingerprint made up of response rates for different inquiring devices.

Fingerprints for known locations are created by positioning a Bluetooth device at the location and instructing the fingerprinted device to become discoverable (enter the *inquiry scan* substate). As most known locations are also locations that contain the infrastructure devices, however, those devices are already in place. The fingerprinted device will stay in this mode for a pre-determined amount of time. The inquiries that discovered the fingerprinted device during this time are used to create the fingerprint of that device. Fingerprints for unknown devices are taken continuously, as unknown devices are always discoverable. This continuous data is fed into a sliding window that is N minutes long (N is currently three minutes). The data in the sliding window is the data used to create the fingerprint of the unknown device.

Known location – A *known location* is a location in which an unknown device can be determined to be located by the localization algorithm. Examples of known locations include offices, meeting rooms and break areas.

Divergence – A divergence measure is a measure of how much one vector of values differs from another. There are both symmetrical and asymmetrical divergence measures. An example of a symmetric divergence measure is the Jensen-Shannon distance [10]. The reason that the Jensen-Shannon measure is called a distance, rather than a divergence, is that it is symmetrical. An example of an asymmetric divergence measure is the Kullback-Leibler divergence [7][8][9]. A divergence calculation, in the context of the Bluetooth positioning system that we employ, is the divergence between one fingerprint and another. These divergence calculations are used to determine how well

the fingerprint of an unknown device matches the fingerprint of a known location. A smaller divergence indicates a better match. The divergence calculation currently in use in the system is the Jensen-Shannon distance.

Fingerprint normalization – Divergence calculations can choose to operate on normalized fingerprints or un-normalized fingerprints. Fingerprints that are normalized will have had their contributions normalized so that they all add up to 1.0. As we will see later, the difference of performing divergence calculations on normalized and un-normalized fingerprints can yield significantly different results.

1.5 Goal, Approach and Contributions

This work has two primary goals. Our first goal is to create algorithms that can perform a particular kind of optimization on a particular kind of network under a particular condition. The optimization is the reduction of the number of sensors needed in the network. The network is a Bluetooth-based network for indoor localization that uses fingerprinting. The condition is that the optimized networks should maintain prediction performance comparable to that of the un-optimized or "full" network. By "full" we mean a network where all sensors are available. We shall then implement these algorithms, and construct a framework in which they can be used, to facilitate experiments to determine the performance of the algorithms. This is an example of the self-optimization mentioned in Section 1.3. The advantages of such algorithms are:

- Reduced need for planning. Because the algorithms can find the most valuable infrastructure devices in a network, given the task of the system (localization), we can remove or reduce the need of human interaction in the planning stage.
- Reduced power consumption. Systems that rely on information from a multitude of devices consume power in relation to how many devices it uses, and the settings of these devices. Reducing the amount of power that the network consumes has both financial and ecological benefits. By reducing the number of devices that are active at any given time, you reduce the money you would have to spend on electricity, and you also reduce the carbon footprint by virtue of using less energy. This could also be beneficial in resource constrained environments. A reduction in the number of sensors also means that mobile devices in the system are being scanned less often. This could contribute to a power reduction on the mobile devices as well; however we have not conducted any such tests.

Our second goal is to evaluate the performance of the optimization algorithms, in particular in terms of the size of the networks they create, the time it takes the algorithms to create these networks, and the resulting prediction performance of these optimized networks. Knowing the characteristics and performance of each of the algorithms, we gain information about which algorithms are suitable for which situations, or if some algorithms are in fact not suitable for use at all.

Our approach to reach these goals consists of four steps:

1. Conduct a literature study of other work in the field to explore directions already taken regarding automatic management and find suitable algorithms to reduce the number of sensors.
2. Develop optimization algorithms.
3. Design and implement a prototype system that can dynamically control the settings and availability of Bluetooth devices in a network, including optimization algorithms that determine which devices should be available, and which should not be.
4. Conduct experiments by applying the optimization algorithms using the prototype to determine to what degree the size of the Bluetooth network can be reduced, while maintaining performance comparable to that of the "full" network.

We call our system AMBIENT (Automatic Management of Bluetooth-based Indoor localization NETworks). It makes use of an existing infrastructure for indoor location determination using

Bluetooth, which is part of a larger system called the Context Management Framework [5]. An important requirement for the system is that it should not require custom hardware for the persons being tracked or custom software on the devices they carry.

Our contributions are:

- Optimization algorithms that can create smaller networks whose prediction performance is comparable to that of the full network.
- The evaluation of these algorithms using experiments on a “live” system.

1.6 Thesis Outline

After this introduction, we will first analyze related work and discuss where our work differs from that of others (Chapter 2). After this we will describe the optimization algorithms we used to approach the problem (Chapter 3). To test these algorithms, we implemented them and inserted them into an existing distributed software framework containing a Bluetooth indoor positioning. This is framework is described in Chapter 4. This is followed by a description of the management system we built (Chapter 5). This description includes the component roles in the architecture, and how they relate to each other. After this we will describe the experiment used to test the system (Chapter 6), followed by a chapter detailing the results of the experiments (Chapter 7). This is followed by the conclusions (Chapter 8) and finally, a chapter on future work (Chapter 9).

2 Related Work

This chapter contains a look at what others in the field are doing. We will look at related work regarding different aspects of our research, specifically indoor location determination (Section 2.1), automatic management (Section 2.2), and optimization algorithms (Section 2.3). Each of these aspects will be presented in one of the following sections. As our work mostly involves innovation in the latter two, these will carry the most emphasis. In the section on indoor location determination, we will focus on systems that employ fingerprinting techniques, as this is the type of mechanism that the indoor positioning system that we use employs.

2.1 Indoor Location Determination

There exists a large body of work on indoor positioning systems. These systems employ different mechanisms of determining location, such as ultrasound [17][18][20][21], WiFi [14][22], Bluetooth [3][30], RFID [15][16][23] and GSM [33]. These systems use different methods of analyzing this data to determine location, such as time-difference-of-arrival [17][18][20][21], received-signal-strength-indicator (RSSI) [14][22], signal-to-noise-ratio (SNR) [14], and fingerprinting [14][30][33][37][39]. The ultrasound systems provide result with 3cm accuracy, but are very expensive. Fingerprinting systems, on the other hand, are comparatively cheap, but provide worse accuracy.

We will now take a more detailed look at systems that use fingerprinting, or methods similar to fingerprinting, to determine location. One such system is Nibble [14]. Nibble is a positioning system that uses a WiFi infrastructure, combined with signal characteristics analysis to determine location. This is possible because WiFi provides a usable Received Signal Strength Indicator (RSSI). Another signal characteristic that can be used to the same effect is the signal-to-noise ratio (SNR). Nibble combines these signal characteristics readings of the device of the person being tracked (such as a laptop or a PDA), along with readings taken for the locations in different locations with Bayesian network calculations to determine the most likely location of the user. Another system that uses WiFi is the RADAR system [22]. RADAR uses the RSSI readings from each WiFi access point in its surroundings, combined with a radio map of the building to determine location. The radio map has information about the measured RSSI values for all access points at all locations. This map is created statically, presumably at installation time. This system uses technology similar to the fingerprinting that our system uses, as will be described in the following chapters. Both Nibble and RADAR are similar to our system, in that they collect signal characteristics readings beforehand, and use them to determine the location of an entity. They differ from our work, in that they use WiFi, and the signal characteristics that WiFi provides, whereas we use Bluetooth.

Genco [30] proposes a system that uses Bluetooth devices for positioning. The system uses the link quality measure of a Bluetooth connection to determine location. It uses a three-step process, link quality sampling (analogous to fingerprinting), Bluetooth base-station deployment, and finally real time positioning using the sampled link quality data. The link quality sampling is used to determine where the base stations should be located, and is also later used in location determination. Genco discusses different methods used for localization, and concludes that neural networks perform well, and have a low run-time complexity. The system uses genetic algorithms to determine the best location for the Bluetooth infrastructure devices. Genco also concludes that, due to the dissimilarity of installation sites, link quality sampling needs to be performed at each individual site. Genco's system is similar to ours in that it uses measurements taken of locations to aid in location determination, as opposed to relying in these measurements to estimate distance. While Genco uses these measurements to determine the location of the infrastructure devices, our infrastructure devices are fixed.

Otsason et al. [33] describe a system that uses fingerprinting techniques combined with GSM signal strength to provide accurate indoor localization information to an accuracy of 5 meters. They do this by using a device (a GSM modem with a richer-than-normal API) that can capture signal strength information from more than the normal 1 or 6 best cells. This solution is called wide fingerprinting, as they rely on as many as 29 sources of signal strength information. They get this many sources by

including signal strength information from cells that are detected, but where the signal strength is too low to be able to communicate securely. The locations that they fingerprint are approximately 1.5 meters apart. This gives them higher localization accuracy, but also requires many more fingerprints to be created. However, since they use GSM signals, as opposed to 2.4 GHz unlicensed signals, as in the case of Bluetooth and WiFi, they experience less interference, and as a result, they do not need to recreate their fingerprints, once originally created. Otsason et al. uses Euclidean distance as a measure when comparing fingerprints during the location determination phase. Because they use a special device to read the GSM signal strength, their approach is somewhat limited. They do state, however, that it should be simple to modify other GSM hardware to deliver the same kind of data. The fingerprints described by Otsason et al. are similar to those that we use. However, since Otsason et al. use GSM signals for fingerprinting, their fingerprints are stable over time, and do not need to be recreated. Our fingerprints vary over time, and as such must be recreated with regularity. We will discuss this fact, and the implications, in Chapter 9.

2.2 Automatic Management

There is more work done on solutions that perform dynamic reconfiguration on a network, rather than on an application level. We have chosen to include examples of both to show how our work is similar to the application-level reconfiguration work being done, and to show how our work is different from the network-level work. Our work is strictly application-level.

Jadwiga, Henricksen & Hu [28] propose an application-independent model for adaptation in context management systems. This model would allow the context management system to replace or alter context sources based on the context that it finds itself in. They provide an example of a context-aware surveillance system that suffers a breakdown. The sensors lost are replaced by others deployed from a mobile emergency response vehicle. This is done (presumably except the deployment of the vehicle) in an automatic fashion. The replacements that need to take place are based on a context model of what context information should be available. If, for instance, there should be facial recognition in the area of the breakdown, then the emergency response vehicle would deploy extra cameras to replace the ones lost. This is similar to our approach in that it optimizes for the (context aware) application that uses the information that the system delivers. While we optimize for a target network quality, their system adapts to fit a prescribed context model that defines which context should be available. This means that we aim to reduce something full to something smaller, whereas they try to repair a partial network so that it can provide the same functionality as the full network.

The Lira [29] infrastructure is designed to enable a *manager* to change the settings of *reconfiguration agents*. The managers communicate with the reconfiguration agents via SNMP. Lira was inspired by network management, such as management of routers and switches etc. Even so, the framework is generic enough to be applicable to any situation. The remote management framework in Lira is similar to the event emission mechanism that we employ. In AMBIENT, we continue to specify a particular application where our system is used, whereas in Lira, they only present a general architecture. While we have specified a particular application in which we use AMBIENT, AMBIENT is designed in such a way that a number of tasks could be carried out using the infrastructure.

SNOWMAN [24] is a system that aims to reduce the energy cost of communication in a Wireless Sensor Network (WSN), by employing a hierarchical communication structure. Nodes are divided into clusters based on their proximity and power levels. The head of each cluster is the one that communicates to the base station. Cluster heads are chosen based on residual energy, i.e. who has the most battery left. This information is kept in a continuously updated energy map. The management goals in SNOWMAN differ from our goals in AMBIENT, as the goals in SNOWMAN are focused on routing-issues in the network, and ensuring that data gets delivered with as little energy expenditure as possible. AMBIENT can also be used to lower the energy expenditure by reducing the number of devices needed in the network. Unlike SNOWMAN, however, our device removal selection is based on the value to the application running on the network, as opposed to the network itself.

Fransiscani et al. [26] propose techniques for handling the reconfiguration scenario in Mobile Ad-Hoc Networks. The aim of their work is to optimize the performance of the Ad-hoc On Demand Distance Vector Routing (AODV) algorithm [25]. They have created a set of algorithms. Three algorithms were created for the case where devices are identical, and one algorithm that is able to deal with a set of heterogeneous devices. They do this by taking concepts from the peer-to-peer (p2p) world and applying them to MANET routing. The difference between the work done by Fransiscani *et al.* differ from our work in aim, similar to that of SNOWMAN. The aim is the network itself, rather than the application running on the network.

Burgess and Canright [27] write about configuration management in ad-hoc networks using techniques designed for management of normal networks. They explore some models to combat the problems in using techniques designed for normal networks. They conclude that centralized approaches to management of ad-hoc networks suffer from scalability issues as the network grows. Burgess and Canright provide an overview of different methods that can be used for policy communication and enforcement, and the advantages and disadvantages of such methods.

2.3 Optimization Algorithms

Our aim is to create sensor networks that contain fewer devices than the full networks. To do this, we must find a way to select only the devices that will best serve our needs, and include them in the network. There are a variety of ways of doing this. We will examine some of these methods in this section.

One approach is to use dimensionality reduction algorithms [40]. Dimensionality reduction algorithms work by analyzing the values of each variable and retain those that say the most about the data set. Each data set contains a number of vectors. The variables in these vectors are also called dimensions. We can apply this to our problem by mapping fingerprints to vectors, and devices in the fingerprints to dimensions in the vectors. Therefore, finding a way to reduce the number of dimensions in a vector translates to reducing the number of devices in a fingerprint, and by extension, reducing the number of dimensions in all the vectors in the data-set lets us reduce the number of devices in the whole network. One dimensionality reduction algorithm is principal component analysis [34] (PCA). PCA will create a new coordinate system based on an existing set of data. Each axis in the new coordinate system will be created along the most valuable dimension, after the last one created. For instance, imagine a set of data points in 3D shaped like a loaf of bread. The first axis would be the length of the loaf (because it has the widest range of data points, the highest variance). The second axis would be the width of the loaf (the second highest variance), and so on. This can be applied from any number of dimensions to any other number of dimensions, creating new axes along the dimensions that have the highest variance. The algorithms we chose are dimensionality reduction algorithms. One algorithm uses entropy as a measure to determine which dimensions most uniquely identify the data, and those dimensions are kept. The entropy is calculated based on the *a posteriori* probabilities of being in a room, as indicated by each device. The other algorithm does not use entropy, but rather looks at the loss in estimated prediction performance as an indicator for which dimensions to keep. These algorithms are more suitable than PCA for the problem, as we cannot change the possible dimensions of the coordinate system, only chose which dimensions to keep.

Another way to solve a related set of problems is by using decision trees [41]. Decision trees classify objects based on their attributes. The root node of a (sub)-tree is chosen based on the decisiveness of an attribute to classify the object. One such algorithm is ID3 [35]. Created by Ross Quinlan, the Iterative Dichotomizer (ID3) works by calculating the entropy for each of the attribute, and selecting the one with the best entropy to form the node of the current (sub)-tree. This works in a recursive manner until the attributes have been exhausted. This creates a tree where the most valuable attribute is at the top. This property would allow us to select the root node of a tree in an iterative manner, or select a series of root nodes in a pre-order traversal manner to select the set of most valuable devices.

Unfortunately, ID3 requires that the variables have discreet values, so that branches can be formed. This requirement also makes it unsuitable for our use, as our fingerprints deal with continuous values.

Yet another way is to use graph-based methods. Cărbunar *et al.* [31] suggests using Voronoi diagrams to reduce the number of sensors required in a network to improve energy efficiency. They use Voronoi diagrams to determine if the sensing area of one sensor is completely covered by a set of other sensors. If so, then that device can be removed. They present both centralized and distributed ways of achieving this. They assume that sensors are unimpeded by objects in the environment, and perform their calculations with perfect circular coverage areas for each sensor. This might be a problem for indoor systems, as radio propagation due to walls and furniture and people do not lead to perfectly circular (or spherical in 3D) coverage areas [33].

3 Optimizers

When we set out to develop algorithms to solve the problem of reducing the number of devices needed in an indoor localization network using Bluetooth devices, we had two goals in mind:

1. The algorithms should deliver significantly smaller networks than the "full" network. For instance, an optimizer that created a network of 50 devices from a full network of 100 devices is better than one that creates a network of 95 devices.
2. The algorithms should not require any knowledge of the physical world. That is, they should not have to take information about the building layout, or other interfering devices, for instance, into account when they run. The only input data should be information that the system already has. In this case, the fingerprints and information about which devices are available

We developed two algorithms who fit these characteristics; the entropy sort optimizer (Section 3.2) and the worst contributor removal optimizer (Section 3.3). Both these algorithms operate solely on data that the system already has (the fingerprints, and the set of available devices), and they deliver significantly smaller networks than the "full" network. A discussion on the relative performance of the optimizers can be found in Section 7.4.

We also developed a quality metric that the algorithms use to determine if the network they have generated is good enough or if they should continue optimizing. The optimizers are given a target quality level to optimize for, and this quality metric is what the optimizers use to verify that they have reached the specified target. As the network optimization algorithms rely on this quality metric, we will start by describing it in Section 3.1. Next, we explain the entropy sort optimizer (Section 3.2) and the worst contributor removal optimizer (Section 3.3) in detail. At the end of the chapter, we will briefly discuss other optimization algorithms that we developed, but did not use (Section 3.4).

3.1 Quality Metric

The quality metric described below measures the "estimated quality" of a network. This estimated quality of the network is a measure of how unique the fingerprints in a network are, or what the average probability of mistaking them for each other is. Let us, for the sake of this metric, define a fingerprint as a point in Euclidean space, where each dimension is represented by the each of the devices that contribute to the fingerprint. The position of the fingerprint along each of these dimensions is then the response rate, or contribution, of that device to the fingerprint. For example, the fingerprint in Figure 1 translates to the point (0.5, 0.3, 0.95, 0.8, 0.1). To estimate the quality of a given network, we created the Euclidean distance clustering metric. It works as follows; it defines the quality of a network as 1 minus the average error rate of all fingerprints. The error rate for a fingerprint is the probability of mistaking the fingerprint for any of the other fingerprints. A fingerprint is mistaken for another if the Euclidean distance (cf. Equation 2) between them is less than a specified clustering radius R. The error rate of a fingerprint is *(n – 1) / n*, where *n* is the number of fingerprints within the clustering radius R of the fingerprint in question. The clustering radius R is chosen by experimentation, which is discussed in Section 7.1. Equation 1 shows the formula used for these calculations. Figure 2 shows pseudo-code for the Euclidean distance clustering metric.

$$p = 1 - \sum_{i=0}^{n} \frac{e_i}{n}$$

$$e_i = \frac{(n_i - 1)}{n_i}$$

Equation 1 - Equation for the Euclidean distance clustering metric, where n = the number of devices in the same cluster

$$ned = \frac{\sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}}{|x| + |y|}$$

$$|x| = \sqrt{\sum_{i=0}^{n} x^2}$$

Equation 2 - Normalized Euclidean distance, where x and y are contributions from each fingerprint for the same device.

This normalization is due to the triangle inequality [42]. The triangle inequality states that:

- Each side in a triangle is *less than* or equal to the *sum* of the other sides.
- Each side in a triangle is *greater than* or equal to the *difference* between the other sides.

Using the second property of the triangle equality, we know that the sum of the sides will always be greater than the hypotenuse, which means that, if we divide the length of the hypotenuse with the sum of the lengths of the sides, we will always get a number below 1. As this is true for all triangles, we use this property to normalize our distances between 0 and 1. The edge case is where the triangle has one angle of 180^o and two angles of 0^o, in which case the length of the hypotenuse is strictly equal to the sum of the lengths of the sides.

```
for (f in fingerprints) {
       numberOfDevicesInMyCluster = 0
       for (f' in fingerprints) {
              d = normalizedEuclideanDistance (f, f')
              if (d <= radius) {
                     numberOfDevicesInMyCluster++
              }
       }
       errorRate(f) = (numberOfDevicesInMyCluster - 1) / numberOfDevicesInMyCluster
       errorRateSum += errorRate(f)
}
quality = 1 - (errorRateSum / sizeof(fingerprints))
```

Figure 2 - Pseudo-code for the Euclidean distance clustering metric

Figure 3 shows an example of what happens to fingerprint uniqueness with the removal of different dimensions. The leftmost picture shows two fingerprints with two contributing devices (dimensions). In the center picture, the horizontal dimension has been removed, and as a result the fingerprints are still distinguishable from each other, which is good. In the rightmost picture the vertical dimension has been removed, causing the fingerprints to indicate the same point on the remaining dimension, which makes them indistinguishable from each other, which is bad. The rightmost picture has a higher error rate than the center picture, as we are more likely to mistake the fingerprints in the rightmost picture for each other compared than the fingerprints in the center picture. Optimization is about removing dimensions, and the quality metric indicates whether the removal of a particular dimension yields a high or low level of fingerprint uniqueness. Thus, the quality metric can indicate how unique the fingerprints in a network are, for a certain value of R.

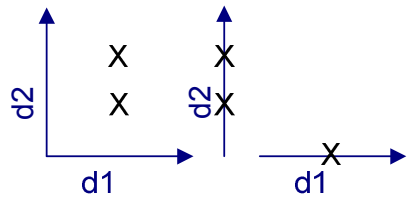

Figure 3 – What happens to fingerprint uniqueness with the removal of different dimensions

3.2 Entropy Sort Optimizer

The entropy sort optimizer determines which the most valuable devices are, and adds them to the network in order. This value, and subsequently this order, is based on entropy [32]. Entropy is a measure of disorder or randomness of a random variable. A high entropy indicates that there is high uncertainty between the values the variables can take, i.e. that they are more equally likely to be chosen. For instance, a dice that always returned six when rolled would have low entropy because the outcome has a low uncertainty, whereas a fair die that is equally likely to return any of the numbers would have high entropy because the outcome has high uncertainty. The highest possible value entropy can have is the logarithm of the number of possible values the variable can have. We are interested in devices with low entropy, because these devices will more distinctly indicate each location that they contribute to. A device with high entropy would contribute equally to all fingerprints that it contributes to, and therefore be less valuable.

The entropy sort algorithm works as follows; we start with an empty *current* network, and a set of *remaining devices*. We iterate over the set of remaining devices and, for each device in this set; we calculate the entropy given the devices already in the current network. The device that yields the network with the lowest entropy is moved from the set of remaining devices to the current network. Each time a device is moved to the current network, we calculate the quality of the network, using the metric described in Section 3.1. If the quality did not increase after the addition of this latest device, the device is discarded and the algorithm continues with the next remaining device. If the quality *did* increase, the quality of the current network is compared to the target quality, and if the quality of the current network is larger than or equal to the target quality, the optimizer returns the current network. Figure 4 shows the pseudo-code for this algorithm. The entropy sort optimizer was chosen because of its reliance on an external factor, namely entropy, to determine the value of a device.

```
network = emptyset
while(!remainingDevices.isEmpty()) {
       for(device : remainingDevices) {
              entropy = calculateEntropyForDevice(device)
              if (entropy <= bestDeviceEntropy) {
                     bestDevice = device
                     bestDeviceEntropy = entropy
              }
       }
       remainingDevices.remote(bestDevice)
       network.add(bestDevice)
       quality = calculateQuality(network)
       if (quality <= currentQuality) {
              network.remove(bestDevice)
              continue;
       } else {
              currentQuality = quality
       }
       if (currentQuality >= targetQuality) {
              return network
       }
}
```

Figure 4 - Pseudo-code for the entropy sort optimizer algorithm. Entropy is calculated as defined below.

We will now go in to more detail about how the entropy sort optimizer works, including explanations of the entropy calculations, and the data we put in to them. As the entropy sort optimizer is quite complex, we will start with a simple example to aid us. Assume that we have three known locations and only one inquiring device. The probabilities of being seen by this device, while being in these locations, are as in Figure 5. The probability of being seen by a device is identical to the *response rate* that we define in Section 4.2. We use *seen by* instead of response rate here, as we want to abstract away from the fact that a location is actually represented by a device. The probability of one device "being seen by" another device is the same as one the response rate of the first device in the fingerprint of the second.

$$p(D \mid Loc = A) = 0.3$$
$$p(D \mid Loc = B) = 0.9$$
$$p(D \mid Loc = C) = 0.2$$

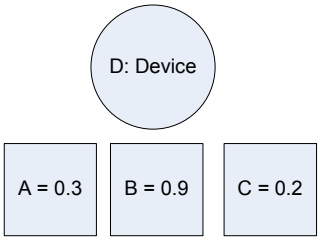

Figure 5 - Probabilities of being seen by the device, while being in the locations

$$p(A \mid B) = \frac{p(B \mid A) * p(A)}{p(B)}$$

Equation 3 - Bayes theorem

$$H = -\sum c * \log(c)$$

Equation 4 - Entropy

Using Bayes' theorem from Equation 3, we can now calculate the *a posteriori* probabilities of being in the respective locations, while being seen by this device, as described in Equation 6. *p(Loc = L)* is the *a priori* probability of being in any location, which is 1 divided by the number of locations. The probability of being seen by a device D is defined as the sum of probabilities of being seen by device D while being in location L, multiplied by the probability location L, i.e. the average probability of being seen by device D. This is defined in Equation 5.

$$p(D) = \sum_{L \in Locations} p(D \mid Loc = L) * p(Loc = L)$$

Equation 5 - Probability of being seen by a device D

$$p(Loc = L \mid D) = \frac{p(D \mid Loc = L) * p(Loc = L)}{p(D)} = \frac{p(D \mid Loc = L) * p(Loc = L)}{\sum_{X \in Locations} p(D \mid Loc = X) * p(Loc = X)}$$

Equation 6 - Probability of being in location L while being seen by device D

This gives us the following values:

$$p(Loc = A \mid D) = 0.214$$
$$p(Loc = B \mid D) = 0.643$$
$$p(Loc = C \mid D) = 0.143$$

Figure 6 - Probabilities of being in the respective locations, given that we are seen by the device

This probability distribution shows us how good this device is at determining the location of some entity. To get a single measure of this, we use entropy. If we calculate the entropy for the probability distribution that we got from the device above (using Equation 4), we get the following. This is the entropy of location probabilities, given that we are seen by device D.

$$H(D) = -\sum_{l} p(Loc = l \mid D) * \log(p(Loc = l \mid D)) = 0.387$$

Equation 7 – Entropy of location probabilities, given that we are seen by device D

As there are three possible locations that can be predicted, the maximum entropy is $H_{max} = \log_{10}(3) = 0.4777$. This entropy is fairly high, as the probabilities of being in locations A and C while being seen by this device are fairly similar. A device with probabilities *{0.1, 0.5, 0.9}* for the locations above would have a lower entropy than a device with probabilities *{0.6, 0.61, 0.62}*, and is therefore a more valuable device.

We now expand our example to apply to a network of devices. We define the probability of being seen by this network of devices while being in a location is the product of the probabilities of being seen by each device respectively, while in that location. Thus we have (N is the network of devices):

$$p(N \mid Loc = L) = \prod_{D \in N} p(D \mid Loc = L)$$

Equation 8 – Probability of being seen by any device D in network N, while in location L

$$p(N) = \prod_{D \in N} p(D)$$

Equation 9 - Probability of being seen by any device D in network N

Using Bayes' theorem in a similar manner to how we used it with one device (in Equation 6), we can now get the probability of being in a location while being scanned by *any* device in the network, by using *p(N | Loc = L)* from Equation 8 in place of *p(D | Loc = l)* in Equation 6 as well as replacing *p(D)* in Equation 6 by *p(N)* defined in Equation 9, resulting in Equation 10 With this knowledge, we can now calculate the entropy for each device in a network. Entropy for each device is calculated as the entropy of that device, given that we already have some network N. The entropy of this new network is the entropy of the existing network, with the device in question added. We use this method to iteratively build a network starting with an empty set of devices. Entropy is calculated the same way as in Equation 7, except with the *a posteriori* location probability distribution of the *network N* defined in Equation 10, instead of the *a posteriori* location probability distribution of a *device D* defined in Equation 6 . In other words; to calculate the entropy of the network N, *c* from Equation 4 is defined as *p(Loc =L | N)* from Equation 10.

$$p(Loc = L \mid N) = \frac{P(N \mid Loc = L) * p(Loc = L)}{p(N)}$$

Equation 10 - Probability of being in location L while being seen by any device in network N

3.3 Worst Contributor Removal Optimizer

The worst contributor removal optimizer starts with a full network and continues by removing devices from it until the target quality is reached. It determines which device to remove by calculating which device is the worst contributor in the current network. It does this by calculating the difference in network quality the removal of each device would yield. The device that yields the lowest decrease in quality is the worst contributor, and as such is removed. It does this iteratively until the quality of the network drops below the target quality. When this happens, the last device to be removed is re-added to the network, and that network is returned from the optimizer. The worst contributor removal optimizer was chosen because of its simplicity. It relies on the quality metric calculation discussed in Section 3.1 to do the heavy work. It offers a very intuitive view of relative device values; much more so than the entropy sort optimizer. Our experiments have shown that the worst contributor removal optimizer will generally create smaller networks than the entropy sort optimizer, given the same target quality. This will be discussed further in Chapter 7.

```
while (!currentNetwork.isEmpty()) {
	worstContributor = findWorstContributor(currentNetwork)
	currentNetwork.remove(worstContributor)
	if(calculateQuality(currentNetwork) < targetQuality) {
		currentNetwork.add(worstContributor)
		return currentNetwork
	}
}

findWorstContributor(network) {
	currentQuality = calculateQuality(network)
	lowestQualityLoss = 1.0
	for (device : network) {
		network.remove(device)
		qualityLoss = currentQuality - calculateQuality(network)
		if (qualityLoss < lowestQualityLoss) {
			lowestQualityLoss = qualityLoss
			lowestQualityLossDevice = device
		}
		network.add(device)
	}
	return lowestQualityLossDevice
}
```

Figure 7 – Pseudo-code for worst contributor removal optimizer

3.4 Other optimizers

We also created a set of other optimizers, but for various reasons, they were not included in the experiments. One optimizer, called the powerset optimizer, would create the powerset of networks and calculate the quality for each, returning the network with the highest quality. While this optimizer would guarantee to find the optimal network, a running time of O(2^n) made it prohibitively slow to run. Another optimizer we created was the random device removal optimizer. This optimizer would remove devices at random from the network until it hit the provided target quality, and then stop. While the random device removal optimizer showed similar characteristics as the entropy sort optimizer and worst contributor removal optimizer, when plotting quality against the number of devices in the network, it is unlikely that the networks created by this optimizer would have good real-life performance, as no deliberation regarding which device to remove is done.

4 Existing Infrastructure

This chapter will describe the existing infrastructure that forms the foundation upon which AMBIENT is created. In Section 4.1, we will describe the Context Management Framework (CMF), which is the framework that we use to extract context information and do Remote Procedure Calls (RPC). Section 4.2 will describe the Bluetooth indoor positioning system that is part of the CMF. We will discuss AMBIENT itself in Chapter 5.

4.1 Context Management Framework

The Context Management Framework (CMF) is a distributed infrastructure that allows a multitude of context sources to run on different machines such as desktop PCs, PDAs and smart phones [4]. The CMF enables context aware applications to discover and obtain information from networked sensors. This information described the context of entities, such as users, devices and places, and is therefore referred to as context information. This context information is provided by context sources, which make use of the aforementioned sensors in the network to deliver this context information. These context sources can either be physical sensors such as keyless entry systems, pressure mats or temperature sensors, but can also be software solutions providing information such as what music a user is currently listening to, or where the user is currently located. The CMF also offers facilities to enrich context, for example by employing reasoners that use multiple context sources to produce higher level context information, but these facilities are outside the scope of this thesis.

The CMF consists of many components, but only the following are related to our work. Everything else is outside the scope of this thesis:

- Context source – This is a sensor, hardware or software that provides context information, as described above. Hardware sensors have software wrappers that enable the CMF to use the information they provide.
- Context broker – A context broker does life-cycle management of context sources. A context broker is typically responsible for a number of context sources. A context broker is also referred to as a container, as it contains the context sources.
- Discovery Mechanism / Service Registry – This component acts as a simple registry where components can register themselves and the services they provide and look for other RPC endpoints using a set of name-value pairs. The discovery mechanism can work in both centralized and distributed modes.
- Context consumer – This is any CMF application that requests context information from one or more context sources. An example could be an employee tracking application that would combine context information from an indoor localization system to determine where in the building a user is, and GPS information when the user is outside the office.

All communication between components is done via RPC. The RPC mechanism allows components to export their own interfaces for use by others, and to import other the interfaces of other components to invoke methods on them. References to these interfaces can either be passed as method arguments via the RPC mechanism, or they can be discovered using the discovery mechanism. The RPC mechanism supports both synchronous and asynchronous communication.

Context can be acquired from context sources either by asking them explicitly, or via a subscription. The subscription follows the publish-subscribe pattern. Context can either be requested “raw”, or can be filtered by providing the context source with SPARQL [6] queries. All context information is provided in RDF format.

4.2 Bluetooth Indoor Positioning System

The CMF contains a Bluetooth-based indoor positioning subsystem. This system uses fingerprinting, as described in Section 1.4 to determine location. This system works by aggregating Bluetooth scan information from nodes in the network equipped with Bluetooth devices and running a particular

Bluetooth context source. This context source periodically scans for other discoverable Bluetooth devices in its proximity. A (possibly empty) list of discovered devices is delivered by these context sources, via a publish-subscribe mechanism, to a central point where the data is collected and used to determine the location of the discoverable Bluetooth devices scattered around the building. These devices can for example be Bluetooth enabled mobile phones, PDAs or special-purpose Bluetooth devices. We have created a context consumer that subscribes to this information and puts all the inquiry results in to a database. This database can then be used for analysis of results over time. This database is used for fingerprint creation for both known and unknown devices, as we will describe below.

The Bluetooth indoor positioning system has several characteristics, but we will mention the ones that we believe set our system aside from many others in the field:

- The system does not force the user to carry any dedicated or extra hardware. Any mobile phone or similar device with Bluetooth enabled and discoverable will work.
- The system does not require the user to install any software on the device being located. This lets us use normal mobile phones, as opposed to only smart phones.
- The system provides room level accuracy.
- The system is cheap to deploy, giving it a low entry threshold. This is due to the fact that the infrastructure software is deployed on the desktop PCs of normal people around the building. There is no dedicated infrastructure for the system. The choice of not using dedicated infrastructure, as we will see later, can have adverse effects. These adverse effects have also been observed elsewhere [3].

5 AMBIENT

The AMBIENT system is responsible for managing the availability of devices in a Bluetooth network. Its goal is to minimize the number of Bluetooth devices involved in localizing a mobile device while maintaining performance comparable to that of the "full" network. AMBIENT is the main vehicle for answering the research question of Section 1.4. In this chapter, we will discuss the AMBIENT system in detail. We will start by describing the requirements of AMBIENT (Section 5.1). We will then discuss the component roles in AMBIENT (Section 5.2), as well as the operation of the system (Section 5.2). The optimization algorithms used in AMBIENT are described in Chapter 3.

5.1 Requirements

In this section we will describe the requirements for AMBIENT. The requirements here are numbered for reference only, without implied order or priority. AMBIENT should be able to:

1. Detect changes to the state of the network. The system needs to be able to determine the state of the network at any given time to be able to have some information upon which to base its decisions. If the system has this information, it becomes easy for system administrators to add or remove devices to the network, as the network will be able to perceive the change and adapt accordingly. This applies to all context sources, but Bluetooth devices in particular.
2. Use of different optimization algorithms. Making the optimization algorithms modular lets us change them depending on the situation at hand. It also allows for easy creation of new optimization algorithms without changes to the infrastructure or the software running on the clients.
3. Automatically create fingerprints. As the system relies on fingerprints, and is also able to dynamically adapt to changes in the network, it also needs to be able to create new fingerprints automatically to take the changes of the network into account. New devices that get added to the system need to both be fingerprinted and need to be incorporated into the fingerprints of the other devices. In the extreme case, where all devices have been removed and replaced with new ones, the old fingerprints would be utterly useless, and would need to be recreated.
4. Remotely control the availability and configuration of Bluetooth devices. The optimization algorithms allow the system to create new networks based on the existing networks. To affect the changes required to put the optimized network in place, we need to be able to remotely control the availability of the Bluetooth devices. Another reason we need to be able to do this is because we need to be able to automatically create fingerprints for the network, and this also requires changing the configuration and availability of the devices in the network.
5. Cope with temporary failure in the various architecture components. This is needed to provide a robust platform. This not only gives us smooth testing for free, as we can alter one or more parts of the architecture during development without the need to restart each of them at the same time, but also provides robust operation. It also rids us of hierarchical dependencies, and lets us make assumptions that, even if a component that we need currently is not available, we are guaranteed to be able to handle it when that component does become available.

5.2 Components

The architecture of AMBIENT consists of four main component roles. We have *Event Emitters* that emit events about the state of context sources; we have the *Configuration Consumers* that accept instructions; and finally we have the *Reconfiguration Decision Point*s that accept emitted events and send out configuration instructions. We also have the *Service Registry*, which is a simple lookup service. In this chapter, we will discuss these components, their responsibilities, and their relationship to each other. Figure 8 shows how these components are related to each other, and what messages they pass.

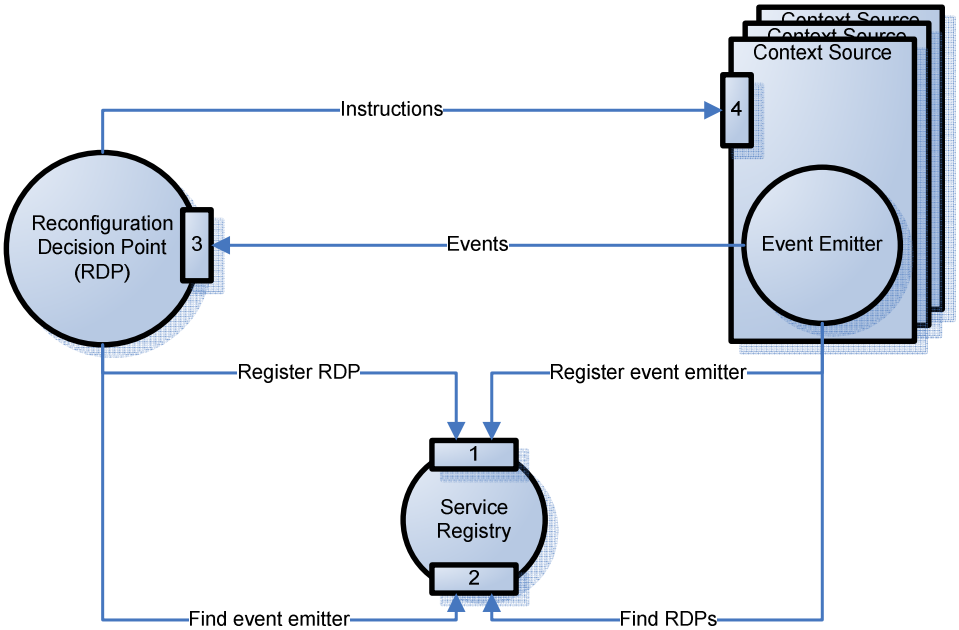

Figure 8 – The component roles of AMBIENT and their relationships

5.2.1 Reconfiguration Decision Point

The reconfiguration decision points (RDP) are the decision makers in AMBIENT. They decide which settings a context source should have, and if it should be available or not. Some RDPs are simple and, for example, only instruct the context sources in the network to go to some default state. Other RDPs are more complex. For example, the optimization algorithms we will discuss in Chapter 3 operate as RDPs. As does the automatic fingerprinting mechanism, described in Section 5.3.2.

Most RDPs will maintain state information about the network in memory and act on this local representation of network state when making decisions. This is done to reduce the amount of communication needed, and also to speed up the decision making process. An RDP performing calculations on a state representation in memory is bound to be faster than if that state had to be collected from the context sources in the network each time a deliberation was to be made. The RDPs acquire this state by querying all the context sources in the network that provide the type of context that the RDP can handle. The RDP finds these context sources by querying the service registry for context sources that provide context that matches the type that the RDP can control. For example, if an RDP knows how to control context sources that provide Bluetooth scans, it would inquire with the service registry for all context sources that provide the Bluetooth scan context type. Once it has a list of these context sources, it can ask them in turn emit an event to the RDP representing the current state of the context sources.

The context sources periodically send state updates to the RDPs on the network, using the latest emitted event. This is used as a keep-alive to let the RDPs know that the context source is still alive and in a particular state. This is also helpful if a context source happens to be unreachable, for example due to network issues, when the RDP requests its state.

5.2.2 Event Emitter

An Event Emitter emits events on behalf of context sources running on a particular host. Each context source has its own event emitter. The event emitter emits events containing information about the current state of a context source and its settings. These settings are stored as a string and are interpreted by the recipient in a proprietary manner. An RDP that is responsible for controlling a context source of a certain type is expected to know how to serialize and de-serialize the settings of

that type of context source, and to know the syntax and semantics of these settings. The syntax and semantics of these serialized settings can vary between different types of context sources. This allows us to have a generic event format in which we can keep a representation of any settings.

5.2.3 Configuration Consumer

The configuration consumer interface enables context sources to be remotely controlled. This interface is used to pass configuration parameters to the context sources, and to instruct context sources to be available or unavailable. As event emitters and configuration consumers are both linked to the same context source, the event emitters will always include a reference to their corresponding configuration consumer interface in all the events they emit. Configuration consumers work as a wrapper between the instructions that the RDPs send and the methods that need to be invoked on the context source in question. As such they are merely a thin layer, but they are required to expose this functionality as a remote interface.

5.2.4 Service Registry

The Service Registry is an integral part of the CMF. While it is not specially designed for the AMBIENT architecture, it plays such a crucial role that it warrants extra mention. The Service Registry is basically a discovery mechanism. It can run in either a centralized or a decentralized fashion. It allows services to register its RPC endpoint with it, along with a set of name-value pairs that define the parameters with which the service has been registered. The discovery mechanism can later be queried, using a combination of name-value pairs to locate the service endpoints that some part of the CMF might need. These service endpoints can then be used to invoke remote commands. The Service Registry is an existing component in the CMF that AMBIENT uses.

Both RDPs and Event Emitters use the Service Registry. RDPs register themselves with the service registry based on the types of context sources they can manage. This is later used by the event emitters when they want to emit events. They ask the service registry about recipients for the type of event they want to emit.

5.3 Operation

This section will describe the operation of the various components in AMBIENT. In Section 5.3.3 we will discuss how the robustness requirement is fulfilled. We will discuss how the components handle the breakage scenario of the others. In Section 5.3.1 we will discuss how an event emission takes place, as this is crucial to how AMBIENT operates. In Section 5.3.2 we will discuss the different phases of operation. Those phases are: the fingerprinting phase, the optimization phase, and the localization phase.

5.3.1 Event emission

The image below shows how a normal event emission takes place. The state changes in the context source; the event emitter for the context source asks the service registry for references to entities that wish to receive events about its context type, and emits the event to the event recipient indicated by the service registry. This lookup is done for each event emission, as RDPs can come and go during the life-time of a context source. RDP receives this event, updates its local view of the state of the network, and then acts on the event. It is important to note that not all events trigger an immediate reaction. An RDP will only issue instructions if the state of the context source is an undesired one. If this is the case, instructions are sent in order to get the context source to change its state.

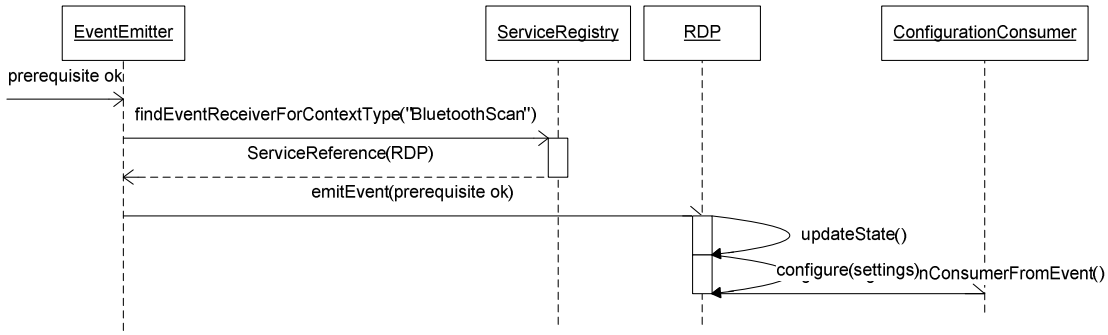

Figure 9 – Sequence diagram for regular event emission.

The events that an event emitter emits pertain to the life-cycle of a context source. Figure 10 and Figure 11 display this life cycle. Figure 10 shows a complete view with all possible transitions, while Figure 11 shows a simplified view without any error transitions.

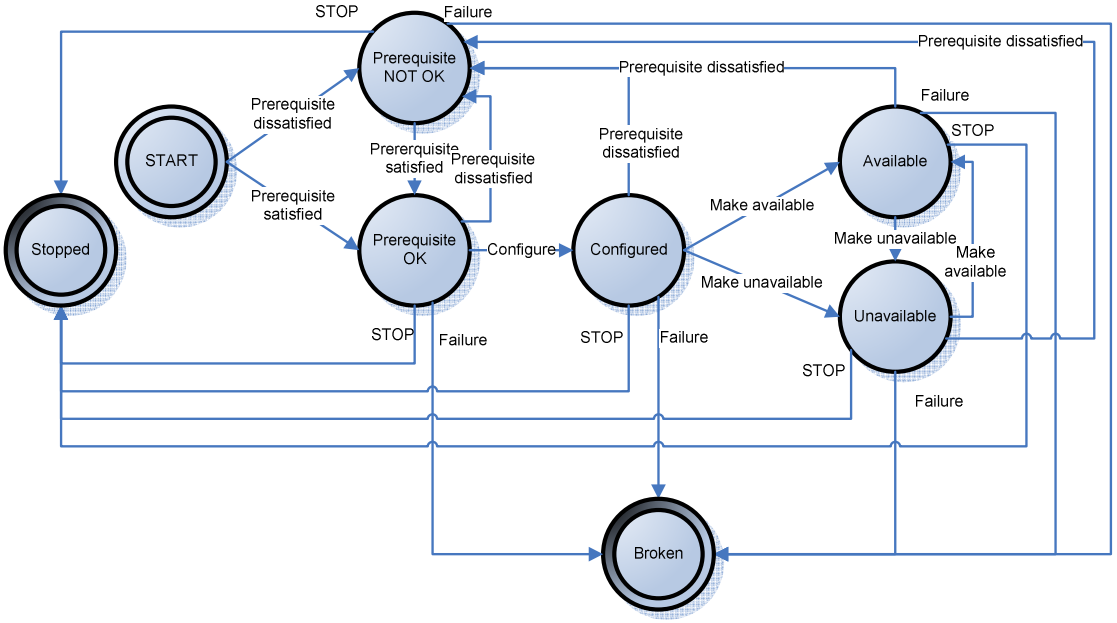

Figure 10 – The complete life-cycle of a context source

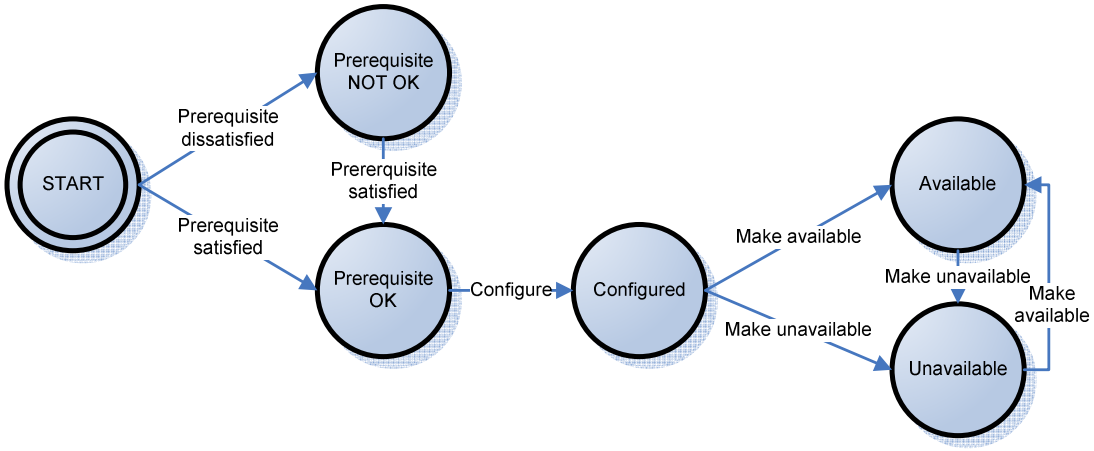

Figure 11 – A simplified image of the context source life-cycle

The transitions in the figures indicate the external events that take it to this state. These events can be instructions sent by the RDP, changes by the user via a GUI, or external events like prerequisites being met or seizing to be met. It starts by determining if the prerequisites for starting the context source are met. If so, the event emitter informs the RDP that this is the case. This causes the RDP to send configuration instructions regarding which settings the context source should have. When the context source has applied those settings it informs the RDP about this again. Once the RDP is content that the context source has the settings it wants, it decides whether to make the context source available or not, and sends instructions to the context source to that effect. If at any time the prerequisites seize to be met (for instance if required hardware is removed), or the context source is stopped, the sequence starts again. Upon entering each state, the corresponding event is sent to an RDP.

A context source enters the “stopped” end-state briefly before being shut down. This is to inform the RDP that it has performed a shutdown for other reasons than being broken, such as the host computer shutting down. As such, there is no way out of this state, and there should not be, since the context source is not on after having entered this state. This allows RDPs to clear out the entry for that context source from their state representation.

A context source enters the “broken” state when an unrecoverable failure has occurred. Some context sources can sense this themselves, and others need to be told by external observers. There is no way out of this state, as it indicates an unrecoverable error.

When in an available or unavailable state and receiving instructions with new settings, we do not transition to the configured state, but rather do a special reflexive transition that emits an “available” or “unavailable” event, updated with the new settings, to indicate the new settings of the context source. This is seen by the RDP as a simple state refresh. This is done to avoid leaving the “available” or “unavailable” states, which could trigger the RDP to perform reconfiguration calculations to cope with the state change of the context source in question. This state refresh is further discussed in Section 5.3.3.

The events that an Event Emitter can generate are the following:

- Prerequisite ok – This means that the prerequisites needed to start the context source are satisfied. An example of this would be when a Bluetooth device that the CMF supports is connected to the computer on which the framework runs.
- Prerequisites not ok – This is the opposite of the ”prerequisite ok” event. An event emitter will emit an event carrying this state if the prerequisites suddenly stopped being satisfied, such as if the Bluetooth device in the example above is removed from the system.

- Stopped – This event is emitted when the context source has been stopped for some reason, such as being shut down. This allows the RDP a simple way to clear out stopped context sources from its internal state representation.
- Configured – This event is emitted when a configuration change has happened.
- Available – This event is emitted when the context source has been instructed to become available, and has done so.
- Unavailable – This event is emitted when the context source has been instructed to become unavailable, and has done so.
- Broken – This event is emitted when the context source breaks for some reason. Some context sources are able to detect this themselves, whereas some other context sources might need external monitoring to determine if this has happened.

The event emitter asks the service registry for references to all event receivers who have registered to receive events regarding the context type of the event emitter. The event emitter then sends off the event to each of the event receivers in the list. These event receivers are primarily RDPs, but can be any component that implements and exports a certain remote interface. This is akin to a centralized publish-subscribe mechanism, where the subscribers (event recipients) register their wish to receive updates about context in the service registry, and the publishers (event emitters) query this service registry each time they want to emit an event to find the appropriate subscribers.

The event emitters can be configured to emit any type of event. While currently only the events mentioned above exist, it would be possible to extend this mechanism so that it could emit events as a result of some aggregate analysis, or the monitoring of one or more other components. Event emitters do not detect the events themselves, but rather are informed that the events have happened by some other entity. As a result, it is possible to add any kind of detection mechanism behind an event emitter and ask the event emitter to emit events corresponding to those observed events.

5.3.2 Operating phases

This section will describe the three different operation phases in the AMBIENT architecture. These phases are:

- Fingerprinting phase
- Optimization phase
- Localization phase

During the fingerprinting phase, fingerprints for the known locations are created. During the optimization fingerprints, AMBIENT will determine which devices should be available and which ones should not. During the localization phase, the CMF uses the fingerprints created, together with the optimized network to determine the location of various entities. The first two phases will be discussed in this chapter, as they are new to the system. The localization phase was already present, and as such has already been described in Section 4.2. The optimization phase can deal with changes to the network, such as the addition or removal of devices. Depending on the configuration and desired behavior of the network, the optimization phase can either be run at certain times, or can be run continuously. In that sense the phases are not sequential, but rather different tasks that AMBIENT carries out. The localization phase is obviously always active,

Fingerprinting phase – The fingerprinting process is run as an RDP. This is done because the fingerprinting process requires the use of the remote control facilities for context sources that AMBIENT was produced to provide. It uses these facilities to alter the settings and availability of the context sources so as to aid in the fingerprinting of the devices in the network. This RDP registers with the service registry to receive events for the BluetoothScan context type. The fingerprinting RDP starts by gathering a representation of the state of the network as described in Section 5.3.3. Once it has gathered this information, it instructs all context sources to use a set of default settings. These default settings make the context sources available, and make them regularly inquire for discoverable

devices. Once this has been done, the fingerprinting RDP instructs each context source, in a round-robin manner, to change its setting to become discoverable and stop inquiring for other devices. The context source that is currently discoverable is the device that is currently being fingerprinted. Each device remains discoverable for a pre-set amount of time. While one device is discoverable, all the other devices continue to inquire. Any successful discoveries get recorded in a database and these discoveries are then used to create the fingerprint. After one device has been fingerprinted it is instructed to revert to the default settings, after which the next device in the list is fingerprinted. This is described in Figure 12. Once fingerprinting is complete, the fingerprinting RDP will default to a behavior where it tells all devices to be available with default settings. Context sources that come online after this process has been started report their state to the RDP, which tells them to become available. They are then added to the end of the list of devices to be fingerprinted.

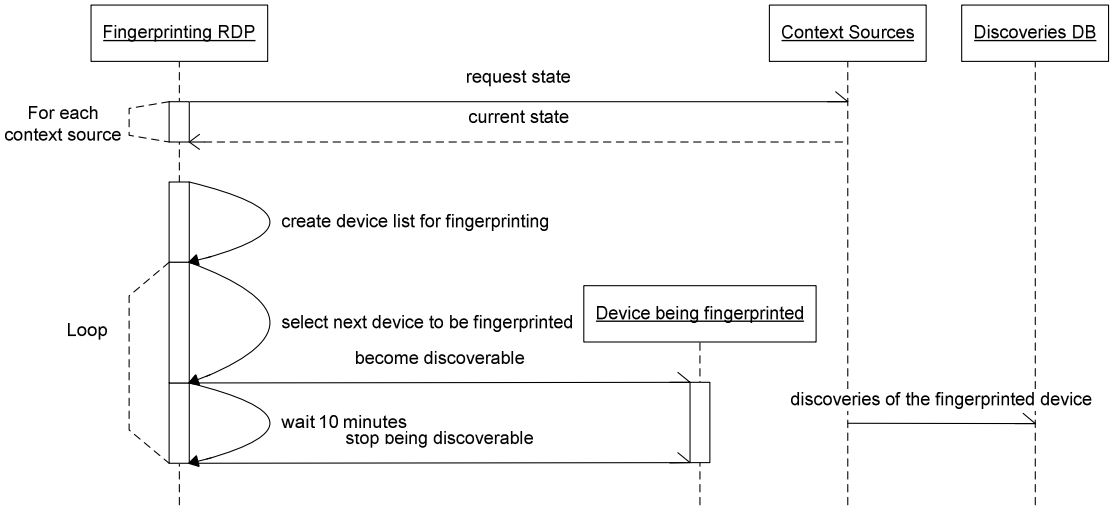

Figure 12 – Sequence diagram of the fingerprinting process

In the current configuration, creating a fingerprint for a known location takes 10 minutes. This is due to the fact that enough samples need to be collected for there to be a meaningful response rate. This also reduces to impact of temporarily changing radio conditions in the environment. As a result, fingerprinting the whole network takes a considerable amount of time. A network with 48 devices would take a full working day to fingerprint. This means that, to have recently updated fingerprints, solutions must be put in to place that deal with this. Such solutions are discussed in Chapter 9.

Optimization phase – In the optimization phase, an RDP is started and loaded with an optimization algorithm. This is currently done "by hand", as to selection of optimization algorithm, but could possibly be done automatically. This RDP will first assemble a complete view of the network as previously described in Section 5.3.3. This view comprises the state information of all the context sources on the network. This information is used by the optimizer to determine which devices should be available and which should not. It then creates a smaller optimized network using the state information gathered about the network and the fingerprints that were gathered earlier. Once it has calculated which devices should be available and which should not, it uses the Configuration Consumer interfaces of the context sources that are responsible for the Bluetooth devices and instruct each one to be available or unavailable, as the algorithm will have determined. This sequence is described in Figure 13. Note that, in this figure, the context sources have had their "configuration consumer" interface and "event emitter" role merged, for simplicity.

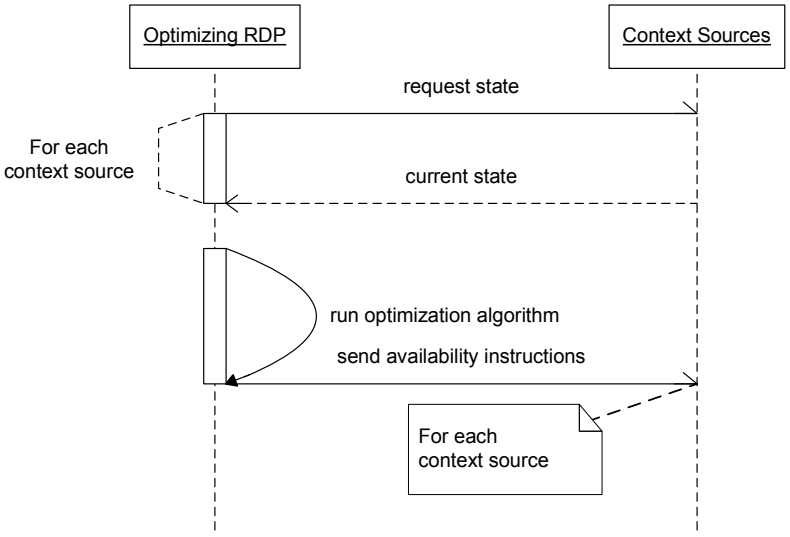

Figure 13 – Sequence diagram showing the optimization procedure

When told to run continuously, the optimization phase can dynamically handle the addition and removal of devices to and from the network. As removal and addition constitute state changes (addition going from no state at all to the starting state, and removal going from some state to no state), these are taken in to account when the optimizers runs the next time. The optimizer could either be triggered by this change itself, or it could be scheduled to run every so often to take changes like these in to account. The method chosen could depend on the network characteristics. For example, a highly dynamic network would benefit from an optimizer that runs often, whereas a network with low dynamicity would not need to optimize often, and could choose to only do so on certain events, or a certain amount of events. This also aligns well with the spirit of reducing power consumption, as less power would be spent running the optimizers if they are run more seldom.

5.3.3 Filling the robustness requirement

As per requirement 5, we wanted to create a robust architecture that would be able to recover from partial failure. As a result, we decided to place the responsibility of the state with the context sources themselves. This has the advantage that it enables us to always have the state accurately represented in a distributed manner, without having to resort to external solutions, like a database, that could introduce more possible points of failure. Since we can always find the context sources; we can always determine their state. Those context sources that cannot be found also are not in a usable state. The RDPs make use of this functionality when they come online by asking each of the context sources about what their state is. Since the context sources emit this state in an identical manner to how they would emit events about state changes, no special procedures are required in the RDP, save for the initial request to receive the updated state. To detect failure, the context sources are asked to re-emit their state information periodically. If an RDP has not received such a state update within a certain time frame, the state entry for that context source is marked as "broken" (with identical semantics to the event with the same name described above), and is subsequently removed if this "broken" state persists.

The event emitters also emit their current state when they first come online. Since they have not received any previous instructions at this state, they can only emit events regarding their prerequisite fulfillment state. This lets an RDP know that the context sources exist, and also lets an RDP issue instructions to the context sources.

These two functionalities make sure that both RDPs and event emitters can recover from the temporary breakage of the other. This leaves the service registry. As an essential component in AMBIENT, the system needs to be able to handle the breakage and subsequent reappearance of the service registry. This is handled by the CMF itself, to a large extent. Any service registered with the service registry can also register an event handler to detect when the service registry goes down and comes up again. AMBIENT gets this "for free" from the CMF. Using this information, AMBIENT can know when the service registry has gone down and come back up again, and thus re-request the state information from the event emitters. The event emitters get the same information, and as a result can update their state with the RDPs. These mechanisms together contribute to the robustness of AMBIENT, as it can deal with the loss and subsequent reappearance of any of its main components.

6 Experiments

We conducted several experiments to evaluate the performance of our algorithms, in particular to determine whether they were capable of creating smaller networks that were capable of delivering performance comparable to that of the full network.

This chapter will provide detail on the goals we set for the experiments (Section 6.1), and provide an overview of the way the experiments were set up (Section 6.2). We also discuss the two sets of experiments we conducted. Some took place in an uncontrolled environment (Section 6.3) and some took place in a more controlled environment (Section 6.4). These experiments took place in the A building of the offices of the Telematica Instituut, over floors 0 to 3, hereinafter referred to as "the building".

6.1 Goals

The goal of the experiments is to determine whether the networks generated by the different optimization algorithms discussed in Chapter 3 can determine location with a performance similar to that of the full network. We will provide these optimization algorithms with different target qualities to aim for when creating these networks. We seek to investigate the following:

- Which optimization algorithms provide the most accurate results? We define a result as being accurate if the determined location is the same as the actual location of some entity. The accuracy of a network is the percentage of accurate location determinations.
- Which optimization algorithms provide the smallest networks for the same target quality?
- For which parameters do these algorithms provide these results?
- Is there a relationship between the target quality fed to the optimizers and the prediction performance of the networks the optimizers create?

We have designed the experiments described below to help us answer these questions.

6.2 Setup Overview

We conducted seven experiments split in to two sets. The first three were conducted in an uncontrolled environment and as such are referred to as the set of "uncontrolled" experiments. The last four were conducted in a more controlled environment, and are thus referred to as the set of "controlled" experiments. Each experiment contains a set of tests. In each test, an optimizer with a specific target quality is used to create a network, and that network is then used to determine the location of the mobile devices. Each set of tests contains one test of the full network, followed by a series of tests of various optimized networks. Table 1 shows which optimizers were used in which experiments. The entropy sort optimizer is abbreviated ES and the worst contributor removal optimizer is abbreviated WCR. These abbreviations can be suffixed with the target quality they were provided with, such as in the case that the entropy sort optimizer was told to aim for a target quality of 0.95, this would be abbreviated ES-0.95.

Before the experiments in Table 1 were conducted, we conducted experiments to determine an appropriate value of the clustering radius R, which is used by the quality metric. These experiments are discussed in 7.1.

Experiment	Controlled	Fingerprint age	Bluetooth version of unknown devices	Tests (optimizers used)
1	No	1 day	1.1	Full, ES-0.95, ES-0.8, ES-0.5, WCR-0.95
2	No	1 day	1.1	Full, ES-0.8
3	No	4 days	1.1	Full, ES-0.95, ES-0.8, ES-0.5, WCR-0.95
4	Yes	<1 day	1.1	Full, ES-0.95, ES-0.8, ES-0.5, WCR-0.95
5	Yes	2 days	1.2	Full, ES-0.95, ES-0.8, ES-0.5, WCR-0.95
6	Yes	3 days	1.2	Full, ES-0.95, ES-0.8, ES-0.5, WCR-0.95
7	Yes	<1 day	1.2	Full, ES-0.95, ES-0.8, ES-0.5, WCR-0.95, WCR-0.8, WCR-0.5

Table 1 - Experiment overview

The optimizers in Table 1 are those discussed in Chapter 3. The reasons we chose to use these target qualities is that we wanted to examine the relative performance of networks optimized with the same optimizer, but with different target quality. This was done to verify that networks with a higher target quality perform better than those optimized with a lower target quality. We also chose to include some tests of different optimizers, but with the same target quality, to be able to compare the relative performance of the optimization algorithms.

Each experiment consists of a set of infrastructure devices deployed around the building, and a set of unknown (or mobile) devices. The infrastructure devices are SITECOM Bluetooth USB Adapter CN-500V2 [11] Bluetooth dongles connected to the desktop PCs of employees in the building. These computers each run an instance of the CMF so that they can provide the context data we need. The infrastructure devices are Bluetooth version 1.2, class 2. Table 1 describes the configuration used for each of the experiments, including the Bluetooth version of the unknown devices.

6.3 Uncontrolled Experiments

This set of experiments used the whole network, and contains experiments one, two and three. The tests were conducted by querying the Bluetooth location context source for the locations of the unknown devices. The Bluetooth location context source uses the Jensen-Shannon divergence between un-normalized fingerprints to create the ranking.

Figure 14 - Bluelon BodyTag BT-002

The unknown devices used for this set of experiments were Bluelon BodyTag BT-002 (Figure 14). These tags use Bluetooth version 1.1. These tags were places in a number of locations inside the building. We took care to spread the devices over the different floors in the building, but other than that, they were placed in a random manner on each floor. The experiments in this set were carried out

by querying the location determination system seven times per experiment, once every five minutes. Each test in each experiment took 30 minutes to conduct.

6.4 Controlled Experiments

This set contains experiments four to seven. The controlled experiments took place on a smaller network of 12 devices, located in adjacent rooms, spread over three floors. We placed four unknown devices in the locations indicated by letters in Figure 15. Each section represents a room with an infrastructure device in it and each row represents a floor. Columns indicate the vertical alignment of the rooms.

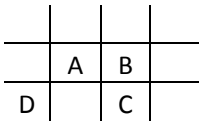

Figure 15 - Device locations during the controlled set

Table 2 shows which devices were used as mobile devices in which experiment, and where they were located.

Experiment	Device	Bluetooth version	Location
4	Bluelon BodyTag BT-002	1.1	A, B, C and D
5	HTC P3300	1.2	A and B
5	QTEK 9000	1.2	C
5	Nokia E60	1.2	D
6,7	HTC P3300	1.2	A, B and D
6,7	QTEK 9000	1.2	C

Table 2 - The location and type of the unknown devices in the controlled set of experiments

Because we took special care to create this controlled environment of 12 devices, we also took steps to make sure that we only used only data from these 12 devices. We did this by modifying the fingerprinting code to only create new fingerprints for these 12 devices, and when doing so, only take inquiry results from the same 12 devices into consideration. Because of this, we created new fingerprints for this set of tests right before the start of experiment four. We also recreated the fingerprints just prior to experiment seven. This can be seen in Table 1. We chose a slightly different way of obtaining our result for this set of tests. We queried the database directly as opposed to using the context source as in the previous set of experiments. This was done for several reasons. Primarily it allowed us to control which devices were part of the calculations. As we only wanted results from our 12 devices, we could write special code to only take those in to account. The algorithms used to perform this localization, however, were the same as in the previous set. Another reason for using a different method was that we wanted to test the relative performance of the divergence calculations using both normalized and un-normalized fingerprints.

While we have limited this set of experiments to these 12 devices, there were still other devices in the environment. The physical environment still contains all the infrastructure devices from the uncontrolled set, as well as WLAN access points and roaming users with Bluetooth-enabled devices. It is possible that any of these factors can have affected the results, but this is outside the scope of this work. The disadvantage is that we cannot measure the impact of these external factors. The advantage is that the experiments were conducted in a realistic environment.

The experiments in the controlled set were carried out a bit differently than those in the uncontrolled set. The experiment time is the same (20-30 minutes), but this time we query the database directly. This is done so that only data from the devices in the controlled set are taken in to account. The database is queried once every five seconds, resulting in approximately 300 observations upon which we base our results.

7 Results

In this chapter, we will present the results of the experiments we conducted. We will first discuss the results of the uncontrolled set of experiments (Section 7.1), including reasons for conducting the set of controlled experiments. The first experiment we conducted, to determine an appropriate value of the clustering radius R for the quality metric is discussed in Section 7.1. After this we will discuss the results of the experiments in the controlled set (Section 7.3). Section 7.4 discusses the performance of the different optimization algorithms. We will finish with a look at how normalization affects prediction outcome (Section 7.5).

7.1 Selecting a value for the clustering radius R

The value for R was chosen by experimentation. The value of R should be relatively small, as a large value for R will make the quality calculations erroneously claim that fingerprints that are actually quite distinct would be mistaken for each other. A small value for R will make the quality calculations erroneously claim that fingerprints that are mistakable from each other are actually quite distinct. Figure 16 shows the performance of the entropy sort optimizer (described in Section 3.2) calculating the quality of different networks, with different values for R. As we can see, values of R around 0.1 and 0.2 would be suitable, as the estimated quality should continually increase as the network grows in size. A value of R of 0.3 or higher returns erratic results, and an R value of 0.7 actually yields an estimated quality that falls as more devices are added to the network. The reason why the curves for R = 0.1 and R = 0.2 stop at an estimated quality of 1.0 is that the optimizer will stop once it has reached this upper limit. The value we chose for R is 0.15, in the range between 0 and 1.

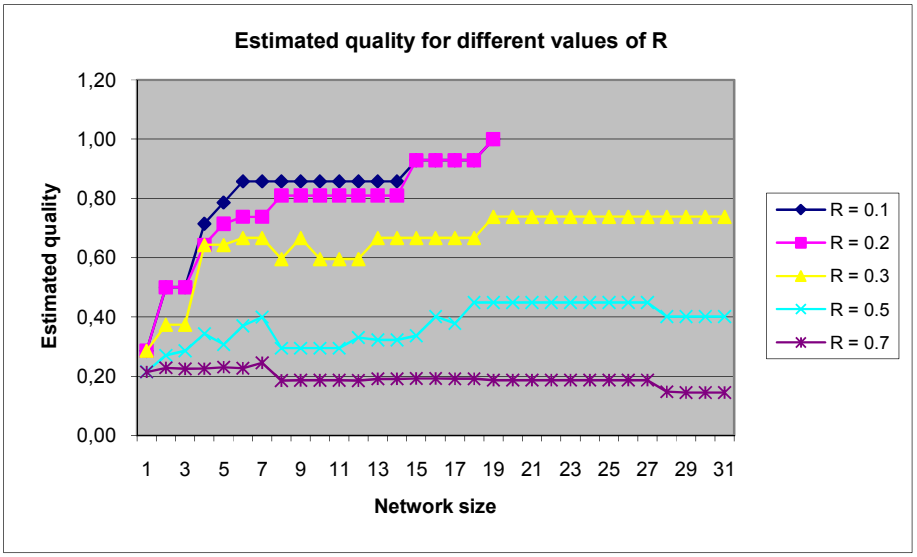

Figure 16 - Estimated quality for different values of R using the Euclidean distance clustering metric

We had initially planned to use the Kullback-Leibler divergence as a basis for quality because this Kullback-Leibler divergence measure was used in early versions of the location determination algorithm, and as such, we would get it "for free", but we discarded this as we had problems with normalization. As the Kullback-Leibler divergence delivers values between 0 and infinity, we could not easily fit the results from it into the range of 0...1. Another quality metric we discarded was to use the percentage of Euclidean distances larger than a particular threshold. This failed because quality would not fall significantly as devices were removed. This can be seen in Figure 17, where T represents the threshold.

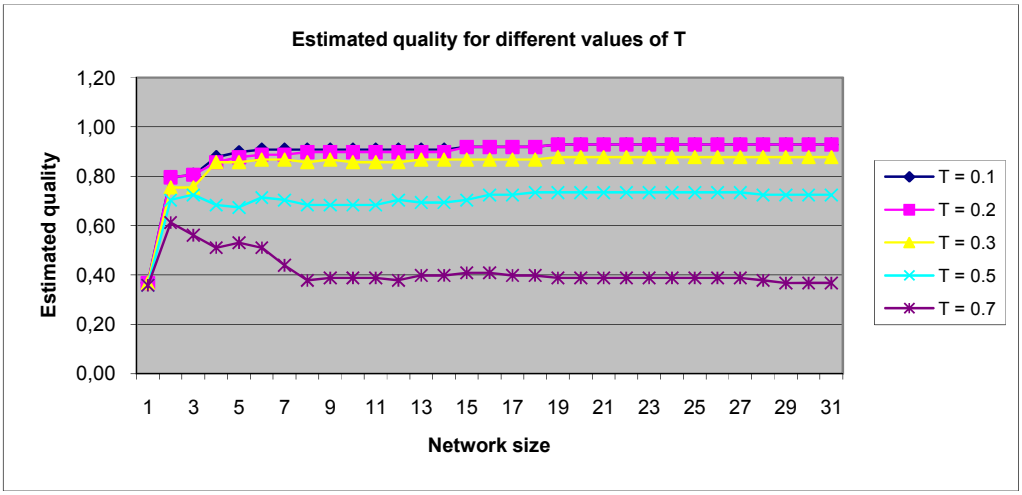

Figure 17 - Estimated quality for different values of the threshold T using the Euclidean distance threshold metric

We tried the same with vector angles rather than Euclidean distances, but this was discarded as vector angles would not take bar height (contribution) in to account. The rather erratic result of this metric can be seen in Figure 18.

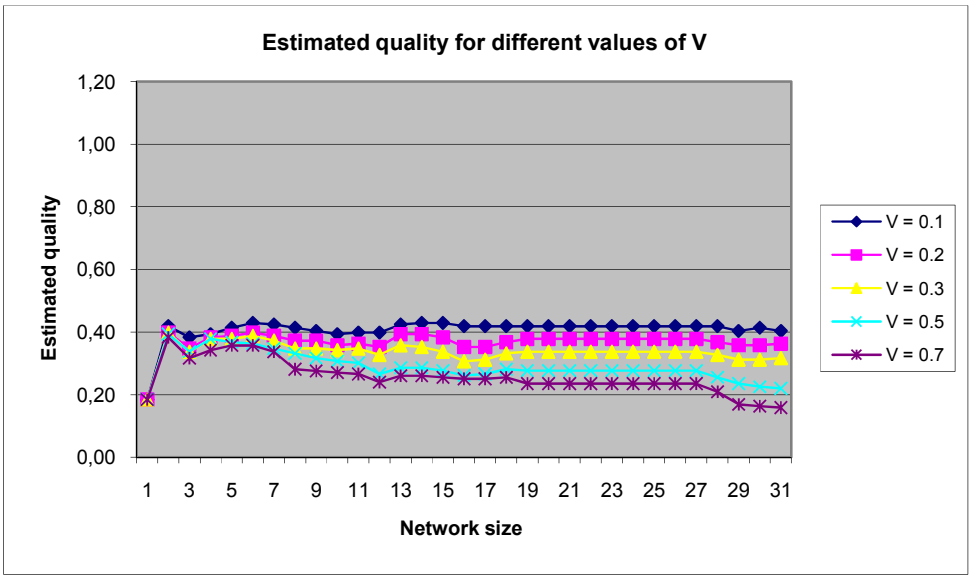

Figure 18 - Estimated quality for different values of the angle V using the Euclidean angle metric

One algorithm that showed some promise was the double average Euclidean distance metric. This metric calculates all the inter-fingerprint distances and uses the average of these as a measure for quality. The idea is that, if the fingerprints are more unique, they will lie further away from each other in Euclidean space. Thus, a higher average inter-fingerprint distance would indicate a higher uniqueness, and thus a higher quality. Another advantage with this metric was that we did not have to supply any parameters, like in the other metrics. This metric was also discarded, however, due to the fact that the estimated quality would not change significantly as the number of active devices changed. These results can be seen in Figure 19, where each line represents a different optimization algorithm. The two optimization algorithms used are the entropy sort optimizer which is discussed in Section 3.2 and the worst contributor removal optimizer which is discussed in Section 3.3.

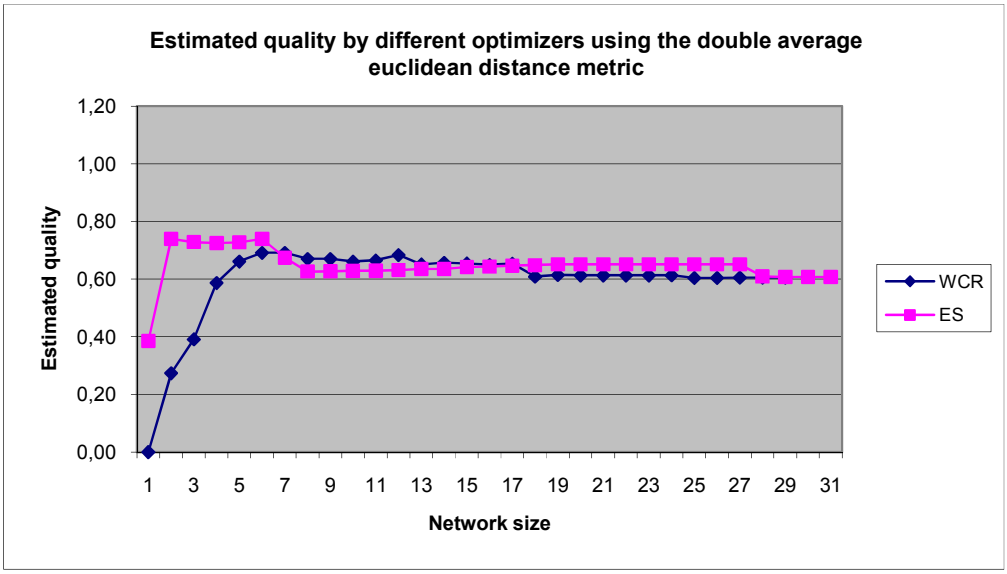

Figure 19 - Estimated quality by different optimizers using the double average Euclidean distance metric

7.2 Uncontrolled Experiments

7.2.1 Performance

Figure 20 shows the results of the first experiment. It shows the prediction correctness, for each location (room, on the x-axis) of each of the optimizers used in the experiment, as well as the prediction correctness of the full network. The y-axis shows the prediction correctness percentage. This is defined as the percentage of times a location was accurately predicted out of a number of observations. A location is accurately predicted if the localization algorithm predicts that the device in question is in the location where the device actually is. For example, if a device is located in room A, and we make 100 observations, and the algorithm determines that the device is in location A in 75 of those observations, and location B in the fourth, this gives us a prediction correctness of 75%. It should be noted that "correct" and "accurate" have the same meaning in this case. Locations without bars indicate that none of the predictions accurately indicated the location in which the unknown device was actually placed (see locations d1, d3, d7, d8 and d9). The different colors indicate different optimizers with different settings. The optimizers are described in Chapter 3, and the settings used for each (as well as their aliases) are defined in Section 6.2. These colors remain the same throughout each prediction performance figure. Prediction performance is defined as the prediction correctness percentage of an algorithm, for a set of locations.

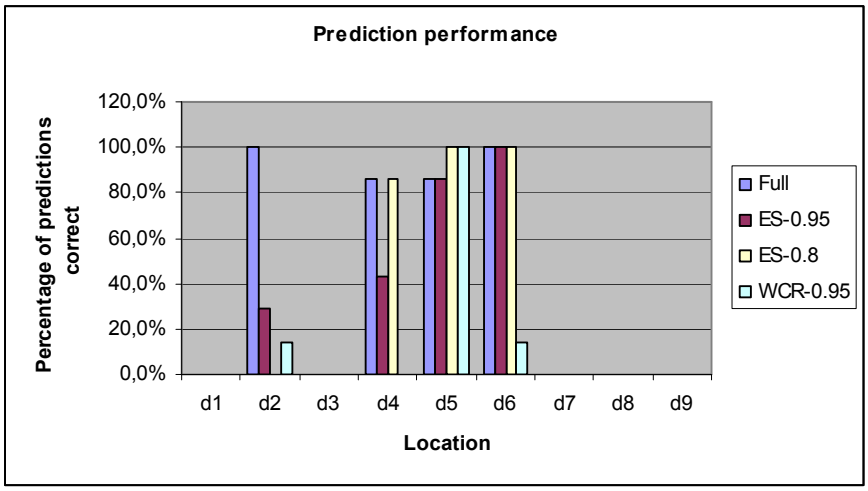

Figure 20 - Prediction performance of the various networks of the first experiment.

Figure 21 shows the averages and standard deviations of the prediction performance in Figure 20. A low average indicates poor prediction performance, as does a high standard deviation. As we can see from the low averages and the high standard deviations of prediction performance in Figure 21, the results of the first experiment were not good. They were characterized not only by the inability of the optimized networks to predict the correct location, but also of the inability of the full network to do the same. In the first experiment, *five out of the nine tested locations were never determined correctly*. This can be seen in Figure 20 as the complete lack of bars for some of the locations.

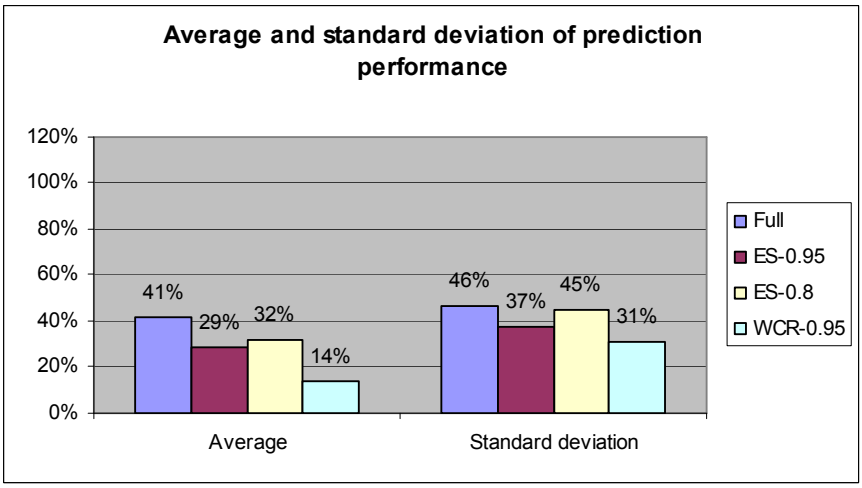

Figure 21 - Average and standard deviation of the prediction results in Figure 20

The primary reason for these poor results was that we had little or no control over when the infrastructure devices joined or left the network. This was because the infrastructure devices were deployed on desktop PCs throughout the building used by the employees, which we are unable to control. It could be the case that computers needed to be rebooted or people would come late or leave early, thus starting up or shutting down their PCs in the middle of the experiments. While the location determination algorithm is designed to take a difference in infrastructure device population between observed fingerprints and recorded fingerprints into account, it would happen that a device that was a valuable contributor to a set of recorded fingerprints would be either offline during determination but online during fingerprinting or vice versa. As a result, the affected fingerprints would become indistinguishable from another set of fingerprints when the state of device availability for each device was taken in to consideration, leading to errors in prediction.

Figure 22 and Figure 23 illustrate this effect. Figure 22 shows two fingerprints (A and B) with contributions from two devices (d1 and d2). Fingerprints A and B are not the fingerprints of devices d1 and d2, but rather fingerprints of some other devices to which d1 and d2 contribute. The fingerprints can be distinguished from each other, due to the difference in contribution of device d2. The contribution of device d2 to fingerprint A is a response rate of 0.7 and to fingerprint B it is a response rate of 0.2.

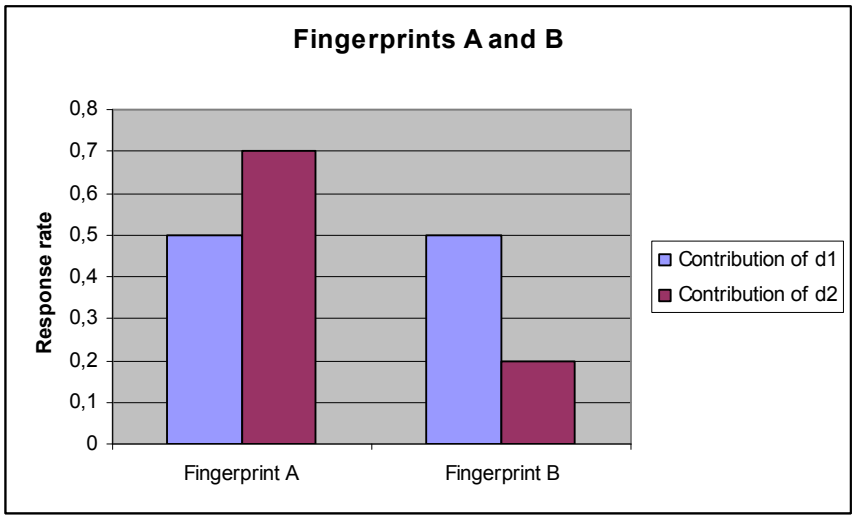

Figure 22 - Fingerprints A and B are clearly distinguishable from each other, due to the contribution of device d2

Figure 23 shows the same fingerprints with device d2 removed. The result is now that the fingerprints are indistinguishable from each other, as the contribution to these fingerprints by device d1 is equal in both fingerprints. Having two fingerprints that are indistinguishable from each other introduces ambiguity in the location determination process, as a fingerprint of a mobile device that matches one of these fingerprints then automatically also matches the other. This leads to an equal probability of predicting that the mobile device is in any of these locations, as opposed to the case when the fingerprints were distinguishable from each other, and a clear "winner" (better match) could be found. While the optimizers also control device availability, they will not make valuable devices unavailable when they are, in fact, needed.

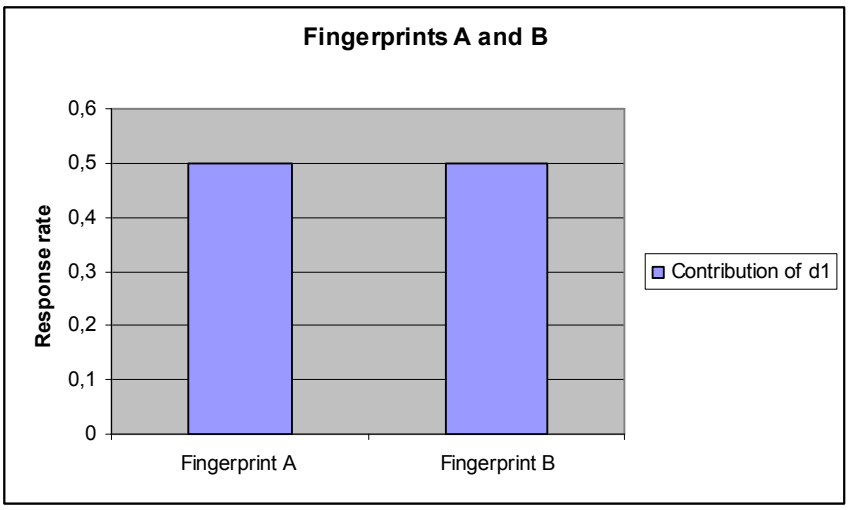

Figure 23 - Fingerprints A and B are indistinguishable from each other, since device d2 is not available

To further show the problems with the uncontrolled environment, the second experiment was aborted when the ES-0.95 optimizer failed to create a network that would satisfy the target quality. Figure 24 and Figure 25 show the results of the third experiment. Comparing these figures with those of the first experiment (Figure 20 and Figure 21), we see that there is little difference in prediction performance.

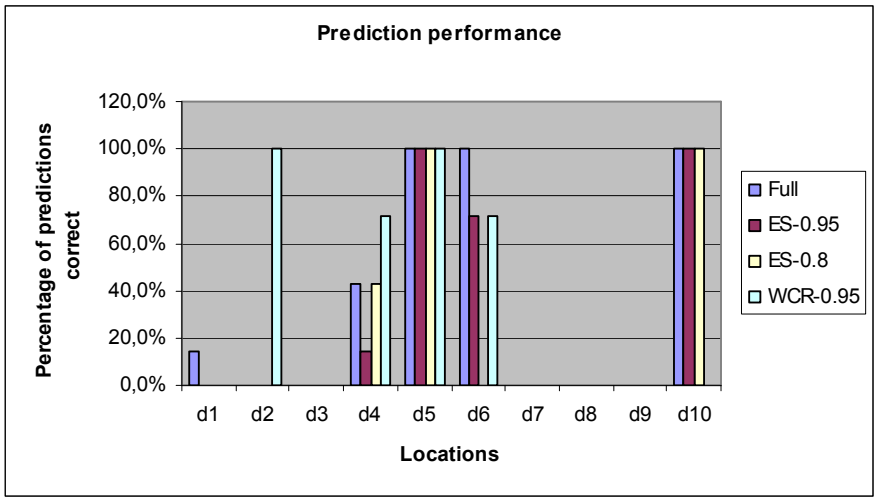

Figure 24 – Prediction performance of the various networks in the third experiment

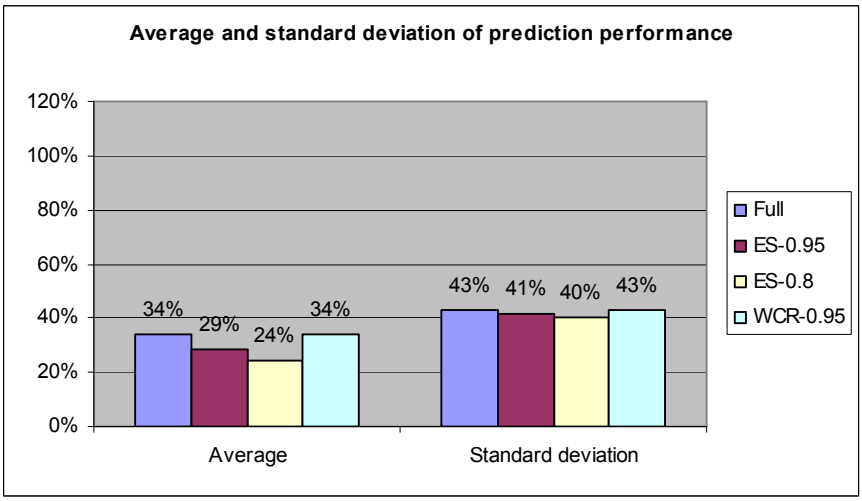

Figure 25 - Averages and standard deviations of the prediction performance in Figure 26

7.2.2 Conclusions

Given the poor performance (indicated by the low averages and high standard deviations in Figure 21) of these experiments combined with the fact that we were unable to properly control the nodes participating in the network, we decided to conduct a new set of experiments. These experiments would take place under more controlled circumstances; the network we used would be smaller, and the devices that made up the network would be guaranteed to not leave at inopportune moments. While this network still uses devices deployed in desktop PCs, we took special care to ensure that the PCs would remain online for the duration of the test by going around in person to the users of these 12 PCs and asking them to please leave their computers on for the duration of the experiments. The problem of deploying on non-dedicated hardware has also been observed elsewhere [3].

7.3 Controlled Experiments

The set of controlled experiments were conducted in the environment described in Section 6.4. This environment contained a set of 12 infrastructure devices, and four unknown devices (Figure 15). This set contains experiments four, five, six and seven. In this section we will discuss the results of each of the experiments.

7.3.1 Performance

Experiment four – Figure 26 shows the prediction performance of the networks created by the various optimizers used in experiment four. The x-axis shows the locations of the four unknown devices, marked A through D, which correspond to the locations indicated in Figure 15. The y-axis shows the prediction correctness percentage as defined in Section 7.1. This experiment used the same Bluelon BodyTag Bluetooth v1.1 devices as unknown devices as were used in the set of uncontrolled experiments. Just as in Figure 20, there are locations that are never predicted correctly.

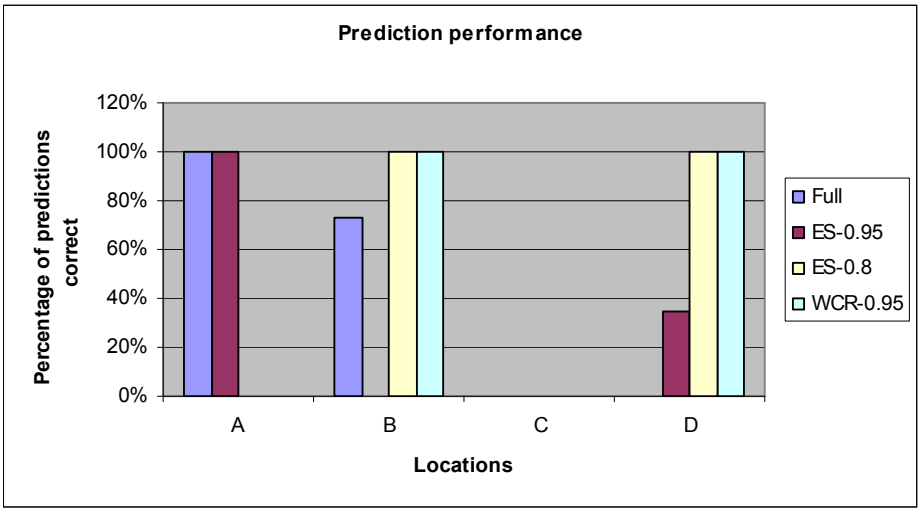

Figure 26 - Prediction performance of the various optimizers for the fourth experiment

Figure 27 shows the average and standard deviation of the prediction performance in Figure 26. Just as in Figure 21, averages are low and standard deviations are high, which indicate poor prediction results.

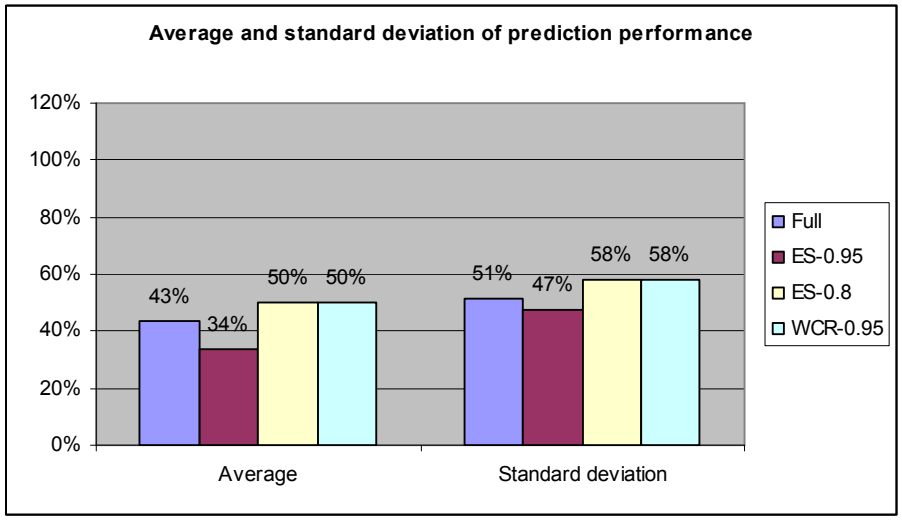

Figure 27 - Average and standard deviation of the prediction performance in Figure 26

Experiment five – Because of the difference between the inquiry procedures between Bluetooth versions 1.1 and 1.2 [12] [13], we decided to use unknown devices running Bluetooth version 1.2 instead of version 1.1 for experiment five. We did this because the infrastructure devices use version 1.2, and we suspected that this version disparity might be a contributing factor to the poor results we had obtained so far.

The results of experiment five were very good. Figure 28 shows the prediction results for experiment five, and as we can see, most of the optimizers perform well by accurately predicting the correct location in a high percentage of the cases. The incorrect determination of location A by optimizer ES-0.8 is due to the fact that the unknown device was placed close to the wall of the adjacent room. As the infrastructure device used to create the fingerprints for this room are at the other end of the room compared to the mobile device, and the infrastructure device used to fingerprint the adjacent room was just on the other side of the wall, this error in prediction is understandable.

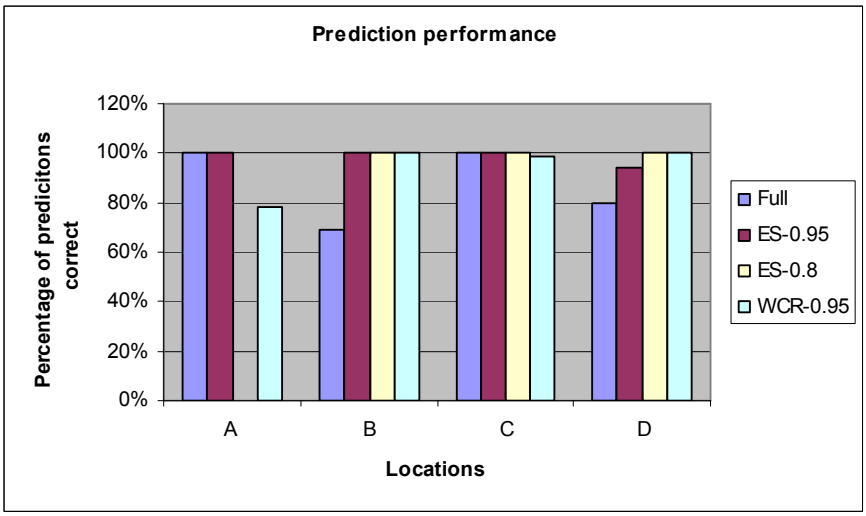

Figure 28 - Prediction performance for the various optimizers in the fifth experiment

Figure 29 shows high averages and low standard deviations of the prediction performance in Figure 28. The high averages and the low standard deviations indicate good performance. The spike in standard deviation for ES-0.8 is due to the fact that the misprediction of one location has a large impact, as there are only four locations used in the experiment.

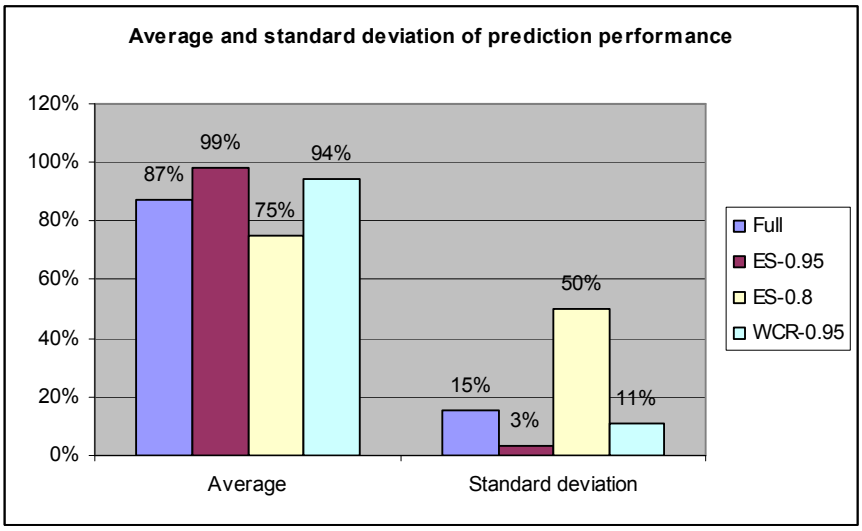

Figure 29 - Average and standard deviation of the prediction performance in Figure 28

Experiment six – To verify that the results of the fifth experiment were not just luck, we conducted experiment six. The configuration was almost identical (some devices were different, but they were all v1.2, as can be seen in Table 2) to the experiment five. As the defining quality we are after is the Bluetooth version of the unknown devices, this difference in device setup should not have any impact on the experiments. The fingerprints in experiment six were the same as in experiment five. This makes the fingerprints used in the experiment six three days old. Figure 30 shows the prediction performance of the networks created by the optimizers in experiment six. As we can see, the results are very poor, as the predictions are only sporadically correct. This is seen by the absence of bars in the difference locations.

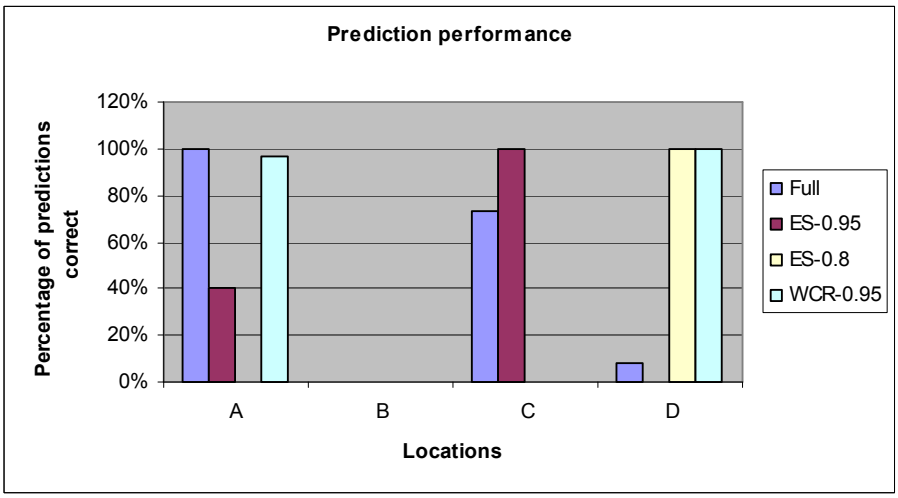

Figure 30 - Prediction performance for the various optimizers in the sixth experiment

The poor prediction performance displayed in Figure 30 becomes even clearer when looking at low averages and high standard deviations shown in Figure 31.

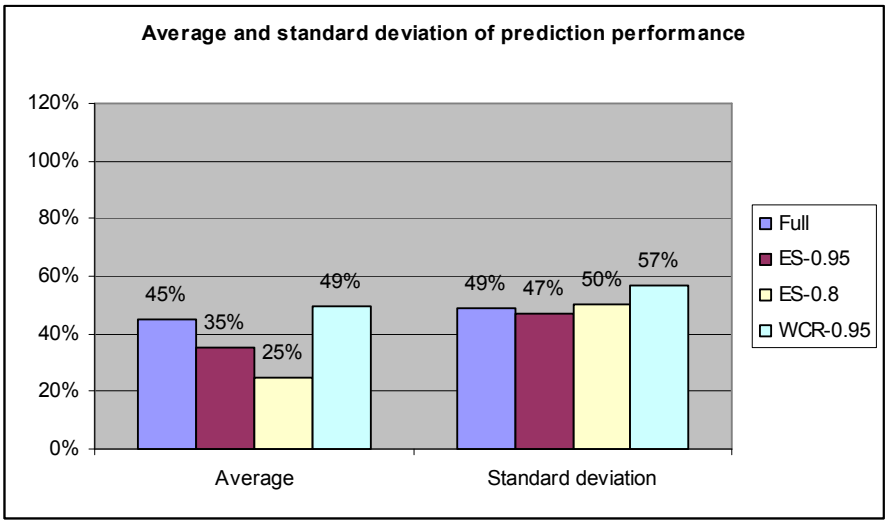

Figure 31 - Average and standard deviation of the prediction performance in Figure 30

Experiment seven – As the fingerprint age was the only thing that varied between experiments five and six, we decided to conduct yet another experiment to determine if this difference in fingerprint age was what caused the comparatively poor results of experiment six. For experiment seven, we created new fingerprints in the morning, and ran the experiment in the afternoon the same day. Figure 32 shows the prediction performance of the optimizers used in experiment seven.

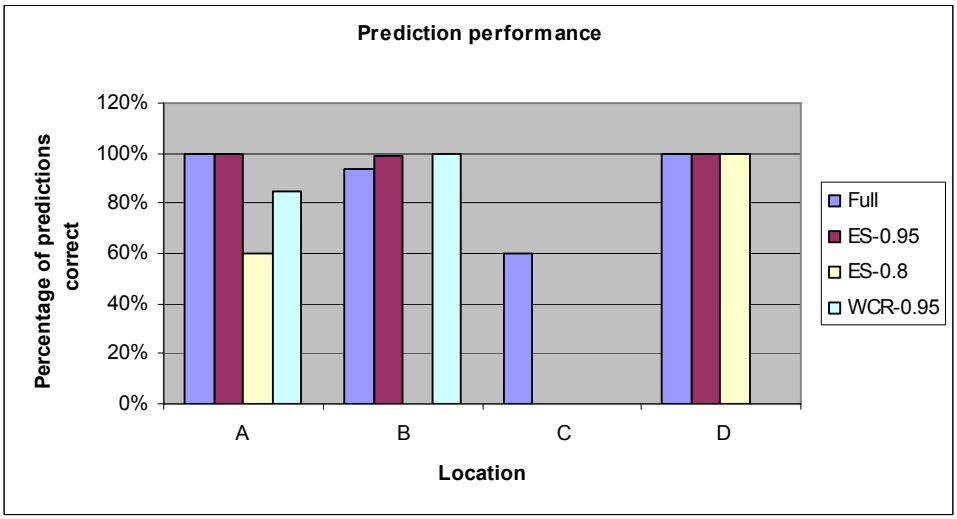

Figure 32 - Prediction performance of different networks in experiment seven.

Figure 33 shows the averages and standard deviations of the predictions in Figure 32. We can see that the prediction averages for the full and ES-0.95 networks are higher than in experiment six (Figure 31), but not as high as in experiment five (Figure 29). A higher average indicates better performance, especially if it is coupled with a low standard deviation. The inverse indicates a worse performance. For example, the average of the ES-0.95 network in Figure 33 is 89%, and the standard deviation is 19%; and in the ES-0.95 network in Figure 31, the average is 45% and the standard deviation is 49%. This higher average and lower standard deviation in Figure 33 than in Figure 31 indicate that the network in Figure 33 has better prediction performance. The inverse can be seen when comparing the ES-0.95 results from experiment five (Figure 29) to the results from experiment seven (Figure 33). Here we see that the average is higher in experiment five than in experiment seven. In addition to this, the standard deviation is also lower.

As the only thing that differed between these experiments were the fingerprints, and in particular the age of the fingerprints, we can say that, as a rule, fresher fingerprints (fingerprints taken closer to the time that they are used) result in better prediction performance. But because experiment seven, where the fingerprints were mere hours old, did not perform as well as experiment five, where the fingerprints were approximately two days old, however, we can say that fingerprint age is not the only thing that matters. We believe that the differences between the experiments are due to the differences in the physical environment between the time the fingerprints are created and the time the fingerprints are used. These differences in physical environment can be anything that alters the radio characteristics of a situation, such as interference or attenuation. The simplest way to attempt to reduce these differences is to create new fingerprints regularly. An extended discussion of this is outside the scope of our work, but we mention a few ideas regarding how to combat this problem Chapter 9, in particular in terms of the impact on the design of the system.

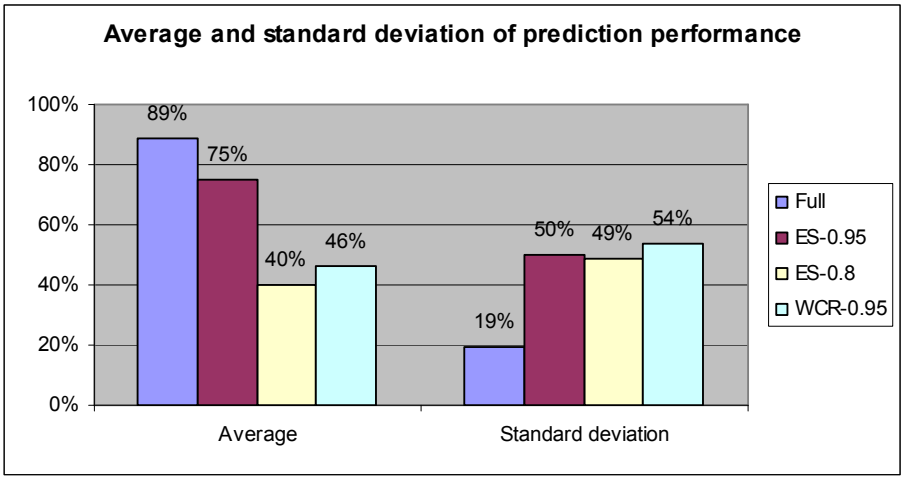

Figure 33 - Average and standard deviation of the prediction results in Figure 32

7.3.2 Conclusions

Experiments in our controlled set let us draw three conclusions.

1. By examining the results in the controlled set of experiments, it becomes clear that it is possible to reduce the number of sensors required in a network for fingerprint-based indoor positioning using Bluetooth, *while* maintaining prediction performance comparable to the "full" network. This is shown by the fact that the predictions of the network optimized with ES-0.95 are even *better* than the results of the full network in experiment five.
2. By looking at the difference between experiments four and five where only the Bluetooth version of the unknown devices differed, we can see that the *Bluetooth version matters*. We cannot say if a network using version 1.1 for both infrastructure and unknown devices would have had similar results, but what we *can* say is that a network that uses v1.2 for both unknown devices and infrastructure devices can produce excellent results, both in full and optimized modes.
3. Another conclusion we can draw by looking at experiments five, six and seven, is that *fingerprint freshness matters*. While we showed that fingerprints created closer to the time they are used are generally better, we believe that that is simplifying the issue somewhat. We believe that the real-world conditions at the time when the fingerprints were created should be as similar as possible to the real-world conditions when the fingerprints are used. The simplest way achieve this is to create fingerprints often.

7.4 Optimization Algorithms

As we can see in Sections 3.2 and 3.3, the approaches taken by the different optimizers are quite different. While the entropy sort optimizer relies on entropy to determine which the most valuable devices are, the worst contributor removal simply looks at the drop in estimated quality that the removal of each device would cause.

The relative performance of ES-0.95 and WCR-0.95 in experiment five (Figure 29) indicate that the prediction performance of the two algorithms are similar, given the same target quality, with ES-0.95 slightly ahead. The same comparison in experiment seven (Figure 33), however, shows that WCR-0.95 is clearly inferior to ES-0.95. Given that the optimizers were given the same target quality, we have to conclude that entropy sort is the better of the two.

Figure 34 shows that the worst contributor removal will generally generate smaller networks than entropy sort, given the same target quality. It also shows that, regardless of original network size, the reduction ratio is similar for any given optimizer with a particular target quality. For instance, the ES-0.95 optimizer delivers a network that contains 55%-65% of the sensors of the original network, regardless of the size of the original network.

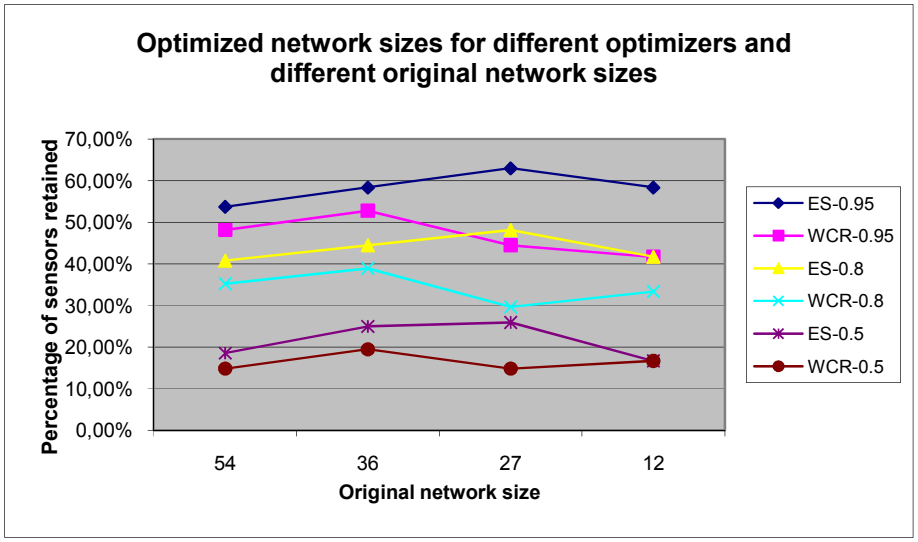

Figure 34 - Optimized network sizes for different optimizers and different original network sizes

Another advantage of the entropy sort optimizer is that it is significantly faster at creating networks than the worst contributor removal optimizer, given the same target quality. This is primarily due to the fact that the entropy sort optimizer carries out quality calculations primarily on smaller network, as it starts from en empty set and builds up, whereas the worst contributor removal optimizer starts with a full network and removes devices to create a smaller network. Figure 35 visualizes the time taken by the various optimizers. The red lines represent the worst contributor removal optimizer and the green lines represent the entropy sort optimizer. We can see that, as the size of the original network increases, so does the runtime of the optimizers. We can also see that the runtime increase of the worst contributor removal is drastically higher than that of the entropy sort optimizer. This increased runtime is due to the fact that the worst contributor removal performs many more quality calculations, and the quality calculations take more time the larger the network is. As the worst contributor removal starts with a full network and removes devices, the quality calculations are more often done on larger networks than smaller networks. The entropy sort optimizer, however, first does not use the quality calculation to select which devices to include, and second primarily performs quality calculations on smaller networks rather than larger. These are the contributing factors to the results seen in Figure 35. The higher runtime or the worst contributor removal optimizer makes it

unsuitable for all but the very smallest networks. The times in Figure 35 are from a machine running an Intel P4 3Ghz with 2GB RAM.

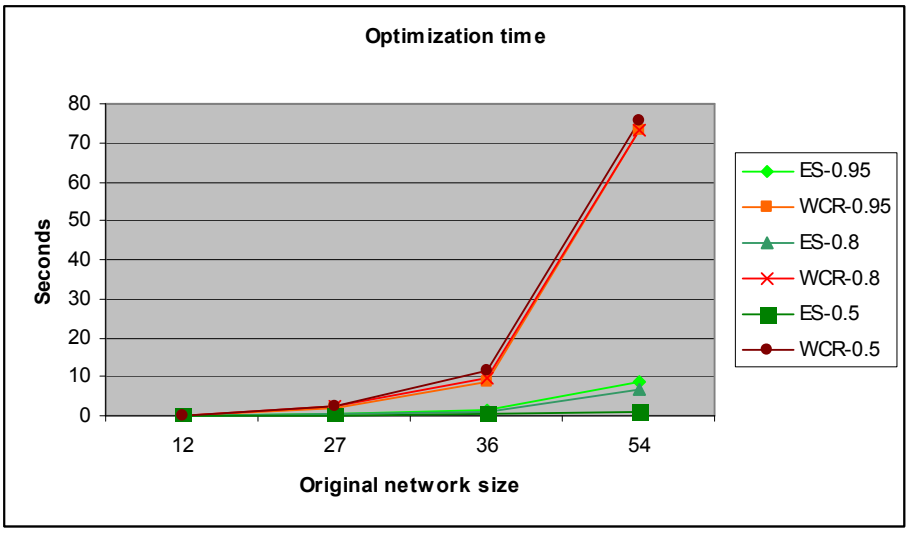

Figure 35 - The optimization time for the different optimizers given different original network sizes.

There is a strong correlation between the target quality provided to the optimizers, and the prediction performance of the network. A network optimized with a target quality of 0.95 is likely to have better prediction performance than one with a target quality of 0.8. This can be seen by looking at the difference in prediction performance of the entropy sort optimizer for different target qualities in Figure 29 and Figure 33. In Figure 36, we have included results from only the entropy sort optimizer in experiment five, using various target qualities. As we can see, the prediction performance falls sharply when a target quality of 0.5 is used.

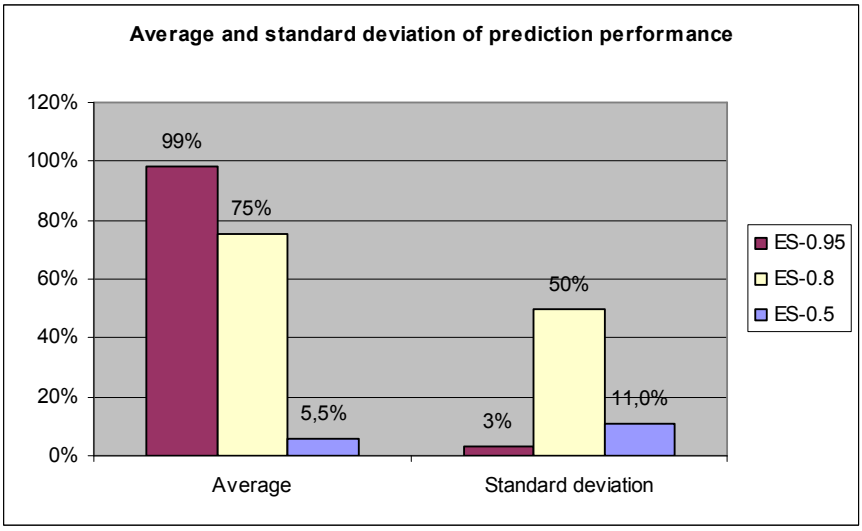

Figure 36 - Average and standard deviation of the prediction performance of the entropy sort optimizer using different target qualities in experiment five

7.5 Effects of Fingerprint Normalization

The location determination calculations in the experiments in the uncontrolled set were performed using un-normalized fingerprints. The calculations in the experiments in the controlled set were conducted using both normalized and un-normalized fingerprints (as described in Section 4.2). We performed calculations using both normalized and un-normalized fingerprints in the controlled set to

determine if there was a difference in the prediction results. We found that there was a difference in prediction performance between using normalized and un-normalized fingerprints and, while the difference is not incredibly large, it is large enough to show that the use of normalized fingerprints in calculations is preferred over the use of un-normalized fingerprints. Figure 37 and Figure 38 show the average and standard deviation of the prediction performance of the different networks using normalized and un-normalized fingerprint calculations, based on the data collected for experiment five. As we can see the averages in Figure 37 are consistently higher when using normalized as opposed to un-normalized fingerprints in the calculations, and the standard deviations in Figure 38 are consistently lower when using normalized as opposed to un-normalized fingerprints in the calculations. As a higher average and lower standard deviation indicate better prediction performance, we can see that normalized fingerprints are preferred over un-normalized fingerprints when used in location determination calculations.

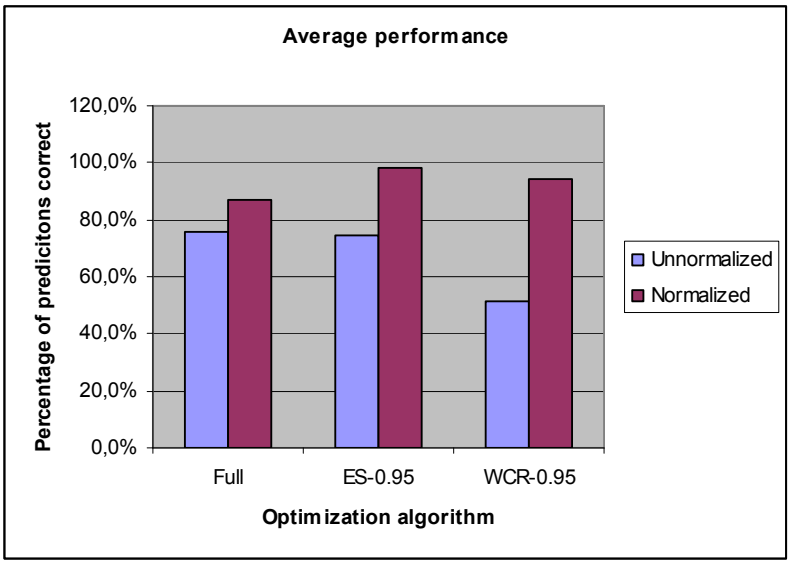

Figure 37 - Average performance of the networks generated by different optimization algorithms using normalized and un-normalized fingerprints.

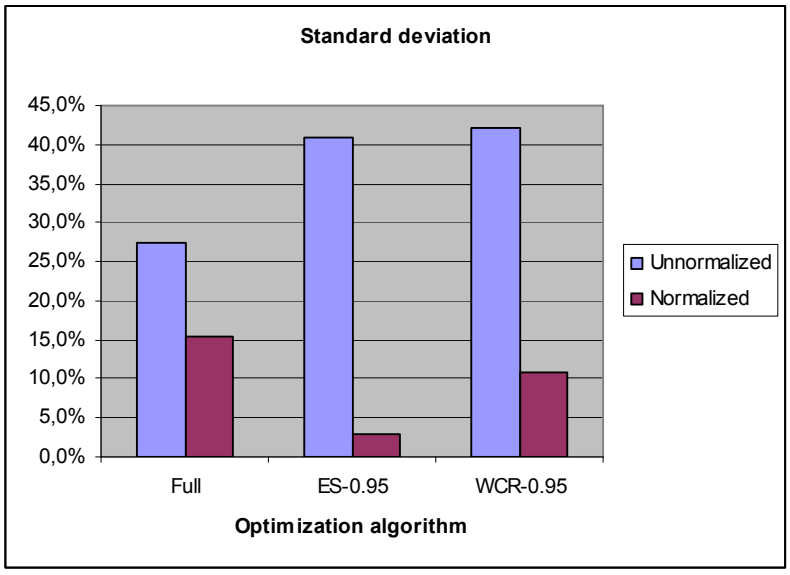

Figure 38 - Standard deviation of the performance of the networks generated by different optimization algorithms using normalized and un-normalized fingerprints

8 Conclusions

In this work we set out to create optimization algorithms for Bluetooth-based sensor networks for localization, and to test the performance of these algorithms. We built a software solution called AMBIENT on top of an existing infrastructure called the Context Management Framework. We added functionality to a part of the Context Management Framework that uses Bluetooth devices for indoor location determination. This functionality allows us to remotely manage the Bluetooth devices involved by changing their settings and, in particular for our purpose, make them available or unavailable. We created four algorithms that can determine which sensors should be available and which should be unavailable, focusing on two of them; the entropy sort optimizer and the worst contributor removal optimizer. The entropy sort optimizer uses entropy to determine the relative value of a device to the network, and the worst contributor removal optimizer uses the reduction in estimated quality caused by the removal of each device as a measure of which devices to remove from the network. We chose these two algorithms because they both satisfy the requirements that they should only use data inherent in the system (fingerprints and device availability), and that they should deliver significantly smaller networks than the full network. The algorithms can use the aforementioned remote management capabilities to affect changes to the network (chance device settings or availability). Finally, we conducted experiments where we asked the optimizers to create new networks, and where we tested these networks to determine whether the smaller networks that the algorithms created would be able to provide prediction performance comparable to that of the whole network.

Our results show that it is indeed possible to reduce the number of sensors in a network while maintaining prediction performance comparable to that of the full network. The results show that *the optimization algorithms will automatically converge on the set of "most valuable" devices in a Bluetooth network*. A valuable device is a device whose contributions to the fingerprints make fingerprints easily distinguishable from each other. Knowing which devices are the most valuable can reduce the amount of manual effort in the planning stages when deciding where to place the sensors. We can simply deploy a large number of sensors in a more-or-less random fashion, and then run the optimizers to see which sensors are the most valuable, and keep only them. This reduces the amount of manual management effort that the system requires (reduces the need to plan where to deploy the Bluetooth devices), which reduces the cost of deploying these systems and fuels the uptake of ubiquitous computing systems in general. Another way this information can be used is that, given limited resources, most of the resources can be spent on the most valuable sensors. For instance, the most valuable sensors could be deployed on dedicated hardware, while less valuable sensors could be deployed in existing infrastructure, such as normal desktop PCs.

Our experiments also let us draw conclusions about the conditions in which it is possible to reduce the number of devices needed. One of these conclusions is that *fingerprint freshness matters*. Our results suggest that the fingerprints vary depending on real-world radio conditions. It is therefore important to have similar conditions when creating the fingerprints as when using them. The simplest way to achieve this similarity is to use recently created fingerprints in when determining location. It is clear, then, that we must create a system that continually creates or updates these fingerprints, so that they are as recent as possible when they are used. As fingerprinting is not instant (it requires 10 minutes per location in its current state), it also becomes clear that some mechanism must be invented to intelligently recreate these fingerprints, or keep tem updated. It could be possible, for instance, to imagine some network segmentation that would allow for parallelization of the fingerprinting process, or possibly some algorithm that would allow the use of historical fingerprint data to be used. As fingerprinting, in its current form, requires the whole network to be online, we cannot just keep creating fingerprints in a round-robin fashion during operation when we are running in an optimized network.

The other conclusion is that the *Bluetooth version matters*. Our experiments show that, regardless of fingerprint age, fingerprints taken of devices using Bluetooth version 1.2 are not usable for determining location of devices using Bluetooth version 1.1. Due to changes in the inquiry

mechanism between these versions, the response rates differ for the two versions in similar situations. Multiple devices inquiring at the same time further exacerbate this problem. As a result, for a system like this to be useful, it is required that the Bluetooth versions used on both infrastructure devices and unknown devices are the same, and preferably version 1.2.

9 Future Work

Due to the fact that fingerprint freshness had such an impact on the performance of the localization algorithm, it is clear that something needs to be done about fingerprint creation. Currently fingerprints are created only when a human initiates the process and, at 10 minutes per location, fingerprinting takes time. While it would be possible to create fingerprints for our test environment of 12 locations each morning, larger networks would require substantially more time to fingerprint. There are some possibilities available to solve the fingerprinting issue. One could imagine that incremental fingerprints would be taken, so that each location would only need a shorter time each day, but that data from the last couple of days would be collected in to a fingerprint. Another possibility is to exploit the symmetry relationship in response rate, if it is indeed there. By the symmetry relationship in response rate we mean that device A has the same response rate when discovering B as vice versa. The reason why we cannot just continually fingerprint each location is that, in its current incarnation, the fingerprinting algorithm needs all devices to be available, and in an optimized network, that is not the case. Other possibilities could include using the RSSI value provided by some Bluetooth v1.2 devices in the inquiry packet. Even though it is in the specification, it is not the case that all v1.2 devices support this. The infrastructure devices currently in place, for example, do not support this feature. Yet another possibility could be to create fingerprint during the night time, but then we run the risk of having environments during fingerprinting and during location determination that are so different that the fingerprints become useless. One option whose viability depends somewhat on the physical layout of the deployment is to subdivide the network into clusters that are known to not be able to see devices outside the cluster. Those clusters would then be fingerprinted in parallel to reduce the time needed to create fingerprints of the whole network.

It might also be interesting to investigate which conditions and properties of the real world have an effect on fingerprint creation. If it would be possible to determine and measure this, as well as find a way to compensate for it, the need to fingerprint often would be lowered, possibly even being completely removable. If, for instance, the barometric pressure or the temperature has a bearing on the fingerprints, it would be possible to incorporate sensors that would deliver this type of context which could then be used by the location determination algorithm to modify the fingerprints to compensate for these differences. If, however, the fingerprint quality depends on factors such as doors being opened or closed, the amount of people in the building or in certain positions at any given time, etc., then it might become very difficult to both acquire this information, as well as quantify it to turn it in to something that the algorithms can use.

To combat the problem of version disparity between the infrastructure and the mobile devices, it might be possible to find a way to convert the fingerprints taken for a version 1.2 device into the equivalent for a version 1.1 device. Given this ability, we still need to be able to determine the version number of the unknown device, which might not be possible. The ideal way of detecting the version is if it can be provided by the Bluetooth protocol somehow. Another way could be to determine the response characteristics of the device, but this adds another layer of uncertainty.

As we saw in Section 3.4, the optimizers we chose are certainly not the only ones. It may very well be that there are other optimizers that perform better, and it would be interesting investigate which those might be, and determine which characteristics they share with the optimizers we chose. It might be possible to combine the sets of best characteristics from each of the algorithms to create a more powerful algorithm. Also, depending on the dynamicity of the network, some algorithms might be more suitable than others. For instance, if the absolute minimum number of devices has to be found, in a network that never changes, it might be beneficial to use the powerset optimizer to be guaranteed to get the best network. In other situations, however, a faster optimizer could be chosen to cope with a network with high dynamicity. It might also be interesting to find an optimizer that does not make use of a quality metric, but rather works based on some other criteria. It could also be possible to parallelize the optimization and/or quality metric calculations, depending on the metric and optimization algorithm.

Yet another area of further research is the quality metric. Just as there can be many different optimizers, there can be many different quality metrics. As we have not yet proven, and also do not know if it would be possible to prove that the quality metric that we use actually indicates the "real" performance if the network, it might be possible to find either a more accurate one, or a faster one. The quality metric calculations are what make the worst contributor removal optimizer so slow compared to the entropy sort optimizer. If we could find a way to speed up these calculations, it would be beneficial to most optimization algorithms.

A combined area of research for both optimizers and quality metrics is giving weight to the individual devices, their characteristics, or groups of devices, or even a hierarchical sub-divisioning of the network. This would, for instance, allow us to create an optimized network where some devices scan less frequently than others, or that some use a different power level. These properties and weights should then also be taken in to account when calculating the quality of the network.

References

[1] Mark Weiser. “The Computer for the 21st Century.” *Scientific American*, 265, September 1991.

[2] Mahtab Hossain, A.K.M, Hien Nguyen Van, Yunye Jin, Wee-Seng Soh, “Indoor Localization using Multiple Wireless Technologies”, *IEEE International Conference on Mobile Adhoc and Sensor Systems*, 2007. MASS 2007, Oct. 2007

[3] Albert Huang, “The Use of Bluetooth in Linux and Location Aware Computing”, Master Thesis, Department of Electrical Engineering and Computer Science, MIT, 2005

[4] Ganek & Corbi. “The dawning of the autonomic computing era”, *IBM SYSTEMS JOURNAL*, VOL 42, NO 1, 2003

[5] H. van Kranenburg, M. S. Bargh, S. Iacob, A. Peddemors, “A Context Management Framework for Supporting Context-Aware Distributed Applications”, *IEEE Communications Magazine*, August 2006, pp. 67-74

[6] http://www.w3.org/TR/rdf-sparql-query/, Checked 2008-06-25

[7] S. Kullback and R. A. Leibler, “On information and sufficiency”, *Annals of Mathematical Statistics*, 22:79-86. 1951

[8] S. Kullback, “Information theory and statistics” (John Wiley and Sons, NY). 1959

[9] S. Kullback, “The Kullback-Leibler distance”, *The American Statistician*, 41:340-341. 1957

[10] J. Lin. Divergence measures based on the shannon entropy. *IEEE Trans. on Information Theory*, 37(1):145--151, January 1991

[11] http://www.sitecom.com/drivers_result.php?groupid=&productid=176&version=V2;001, Checked 2008-06-25

[12] Peterson, Baldwin & Raines, ”Bluetooth Discovery Time with Multiple Inquirers”, *Proceedings of the 39th Hawaii International Conference on System Sciences*, 2006

[13] Peterson, Baldwin, Kharoufeh, ”Bluetooth Inquiry Time Characterization and Selection”, *IEEE Transactions on Mobile Computing*, Vol. 5, No. 9, September 2006

[14] Castro, Chiu, Kremenek & Muntz, “A Probabilistic Room Location Service for Wireless Networked Environments”, *Ubicomp 2001, LNCS 2201*, pp. 18-34, 2001.

[15] http://www.centrak.com/CenTrak_PeopleTracking.asp?menuid=p6, Checked 2008-06-25

[16] http://www.spopos.dk/index.php?id=1460, Checked 2008-06-25

[17] R. Want, A. Hopper, V. Falcão and J. Gibbons, “The Active Badge Location System,” *ACM Trans. Information Systems*, vol. 10, no. 1, Jan. 1992, pp. 91–102.

[18] A. Harter and A. Hopper, “A Distributed Location System for the Active Office,” *IEEE Network*, vol. 8, no. 1, Jan./Feb. 1994, pp. 62–70.

[19] F. Bennett, T. Richardson, and A. Harter, “Teleporting—Making Applications Mobile,” *Proceedings of the IEEE Workshop on Mobile Computer Systems and Applications, IEEE CS Press*, Los Alamitos, Calif., 1994, pp. 82–84.

[20] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward and Paul Webster "The anatomy of a Context-Aware Application", *Wireless Networks*, Vol. 8, pp. 187-197, February 2002.

[21] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43, 2000.

[22] P. Bahl and V. N. Padmanabhan. Radar: An in-building RF based user location and tracking system. *Proceedings of IEEE INFOCOM 2000*, pages 775–784, March 2000.

[23] http://www.spopos.dk/index.php?id=1454, Checked 2008-06-25

[24] Jong-Eon Lee, Si-Ho Cha, Dae-Young Kim and Kuk-Hyun Cho, “Autonomous Management of Large-Scale Ubiquitous Sensor Networks”, *EUC Workshops 2006, LNCS 4097*, pp. 609–618, 2006.

[25] http://w3.antd.nist.gov/wctg/aodv_kernel/aodv_guide.pdf, Checked 2008-06-25

[26] Fransiscani, Vasconcelos, Couto & Loureiro, “Peer-to-Peer over Ad-Hoc Networks: (Re)Configuration Algorithms”, *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS’03)*, 2003

[27] Burgess & Canright, “Scalability of Peer Configuration Management in Partially Reliable and Ad-Hoc networks”, *IFIP/IEEE Eighth International Symposium on Integrated Network Management, 2003*. March 2003
[28] J. Indulska, K. Henricksen, and P. Hu, “Towards a Standards-Based Autonomic Context Management System”, *ATC 2006, LNCS 4158*, pp. 26–37, 2006.
[29] Castaldi, Carzaniga, Inverardi & Wolf, “A Lightweight Infrastructure for Reconfiguring Applications”, *Technical Report, University of Colorado*, 2002
[30] Allesando Genco, “Three Step Bluetooth Positioning”, *LoCA 2005, LNCS 3479*, pp. 52 – 62, 2005.
[31] B. Cărbunar, A. Grama, J. Vitek, O. Cărbunar, “Redundancy and coverage detection in sensor networks”, *ACM Transactions on Sensor Networks (TOSN)*, Volume 2, Issue 1, Pages: 94 – 128, (February 2006)
[32] Shannon, Claude E.: Prediction and entropy of printed English, *The Bell System Technical Journal*, 30:50-64, 1950.
[33] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara, “Accurate GSM Indoor Localization”, *UbiComp 2005, LNCS 3660*, pp. 141–158, 2005.
[34] Pearson, K., "On Lines and Planes of Closest Fit to Systems of Points in Space". *Philosophical Magazine 2*: 559–572, 1901
[35] Mitchell, Tom M. *Machine Learning*. McGraw-Hill, pp. 52-81, 1997
[36] A. Dey, D. Salber, and G. Abowd, “A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications”, *Special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal*, Volume 16 (2-4), 2001, pp. 97-166
[37] Mortaza S. Bargh and Robert de Groote, "Indoor Localization Based on Response Rate of Bluetooth Inquiries", *To appear in the proceedings of MELT'08*, 2008
[38] Ivan. A. Getting. “The global positioning system”, *IEEE Spectrum*, 30(12):36–47, Dec 1993.
[39] Mikkel Baun Kjærgaard, “A Taxonomy for Radio Location Fingerprinting”, *LoCA 2007, LNCS 4718*, pp. 139–156, 2007.
[40] Imola K. Fodor, "A survey of dimension reduction techniques", Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, June 2002
[41] Andrew W. Moore, "Tutorial: Decision Trees", http://www.autonlab.org/tutorials/dtree.html, Checked 2008-07-21
[42] Weisstein, Eric W. "Triangle Inequality." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/TriangleInequality.html, Checked 2008-06-23
[43] B. Schilit, N. Adams, and R. Want. "Context-aware computing applications". *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94), Santa Cruz, CA, US*: 89-101, 1994
[44] Alexander Clemm, “Network Management Fundamentals”, Cisco Press, Nov 21, 2006