INDOOR LOCALIZATION USING WI-FI FINGERPRINTING

by

Saeid Mirzaei Azandaryani

A Thesis Submitted to the Faculty of
The College of Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University
Boca Raton, Florida
December 2013
INDOOR LOCALIZATION USING WI-FI FINGERPRINTING

by

Saeid Mirzaei Azandaryani

This thesis was prepared under the direction of candidate's thesis advisor, Dr. Ionut Cardei, Department of Computer & Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

SUPERVISORY COMMITTEE:

Ionut Cardei, Ph.D
Thesis Advisor

Mihaela Cardei, Ph.D

Shihong Huang, Ph.D

Nurgun Erdol, Ph.D
Chair, Department of Computer & Electrical Engineering and Computer Science

Mohammad Ilyas, Ph.D
Dean, College of Engineering and Computer Science

Barry T. Rosson, Ph.D
Dean, Graduate College

Date

November 22, 2013

ii
ABSTRACT

Author: Saeid Mirzaei Azandaryani

Title: Indoor Localization Using Wi-Fi Fingerprinting

Institution: Florida Atlantic University

Thesis Advisor: Dr. Ionut Cardei

Degree: Master of Science

Year: 2013

Nowadays the widespread availability of wireless networks has created an interest in using them for other purposes, such as localization of mobile devices in indoor environments because of the lack of GPS signal reception indoors. Indoor localization has received great interest recently for the many context-aware applications it could make possible.

We designed and implemented an indoor localization platform for Wi-Fi nodes (such as smartphones and laptops) that identifies the building name, floor number, and room number where the user is located based on a Wi-Fi access point signal fingerprint pattern matching. We designed and evaluated a new machine learning algorithm, K-Redpin, and developed a new web-services architecture for indoor localization based on J2EE technology with the Apache Tomcat web server for managing Wi-Fi signal data.
from the FAU WLAN. The prototype localization client application runs on Android cellphones and operates in the East Engineering building at FAU. More sophisticated classifiers have also been used to improve the localization accuracy using the Weka data mining tool.
INDOOR LOCALIZATION USING WI-FI FINGERPRINTING

LIST OF FIGURES ........................................................................................................ v

LIST OF TABLES AND CHARTS ..................................................................................... ix

CHAPTER 1: INTRODUCTION ...................................................................................... 1

1.1 The Campus 2020 Project ..................................................................................... 3

1.2 Applications and Motivations .............................................................................. 4

1.3 Campus 2020 Special Application Scenario ....................................................... 5

1.4 Problem Statement .............................................................................................. 7

1.5 Localization Methodology .................................................................................. 7

1.6 Thesis Outline ..................................................................................................... 9

CHAPTER 2: BACKGROUND AND RELEVANT TECHNOLOGIES ......................... 11

2.1 Indoor positioning solutions work similar to GPS ............................................. 11

2.2 Other solutions use light or magnetic fields ..................................................... 12

2.3 RFID and inertial systems work very differently ............................................. 12

2.4 Hybrid solutions ............................................................................................... 13

CHAPTER 3: ARCHITECTURE AND ALGORITHMS .................................................... 14

3.1 Architecture ...................................................................................................... 16

3.1.1 The Map Editor .......................................................................................... 17
LIST OF FIGURES

Figure 1. Indoor localization using Wi-Fi AP signal fingerprinting:........................................... 2

Figure 2. The localization methodology; ................................................................................ 8

Figure 3. Gartner’s Prediction for the 2015 Mobile Market...................................................... 16

Figure 4. Overall system architecture for indoor localization. ................................................. 17

Figure 5. Screenshot of the Map Editor application. ............................................................... 18

Figure 6. Screenshot from initializing fragment of android application............................... 20

Figure 7. Screenshot from positioning fragment of the android application ....................... 21

Figure 8. Screenshot of the Weka Explorer program during the “Preprocess” phase ...... 23

Figure 9. Showing RSS data histogram charts with Weka Explorer .................................. 24

Figure 10. KNN selects the majority vote from the vector's K nearest neighbors............. 26

Figure 11. System architecture block diagram .................................................................... 28

Figure 12. Technologies that have been used in the project ................................................. 29

Figure 13. Class Diagram of Our Android Project ............................................................... 31

Figure 15. Sequence Diagram for Saving the RSS Data in Database................................. 35

Figure 16. UML sequence diagram with the logic for the query service ....................... 36
Figure 17. UML sequence diagram for the getLocation service. ................................. 37

Figure 18. UML class diagram the classifier design for. ........................................... 40

Figure 19. UML diagram for the data model, transformation, and classifier evaluator. .. 40

Figure 20. 4th Floor of East Engineering Building.......................................................... 44
LIST OF TABLES AND CHARTS

Table 1. Classification error ratio for K-Redpin for model Data 1. .............................. 46

Chart 1. Classification error ratio for K-Redpin for model Data 1. .............................. 47

Table 2. Classification error ratio for K-Redpin for model Data 2. .............................. 48

Chart 2. Classification error ratio for K-Redpin for model Data 2. .............................. 49

Table 3. Comparing the error ratio K-Redpin across Data 1 and Data 2 for K=5. .......... 50

Chart 3: Comparing K-Redpin across Data 1 and Data 2 for K=5. .............................. 50

Chart 4. The classification error ratio for K-Redpin with K=5,7. .............................. 51

Table 4. Classification error ratio for KNN variants for file Data 1 and k=1,3,5,7,9. ..... 52

Chart 5. Classification error ratio for KNN variants for file Data 1 and k=1,3,5,7,9. ...... 52

Table 5. Classification error ratio for KNN variants for file Data 2 and k=1,3,5,7,9. ..... 53

Chart 6. Classification error ratio for KNN variants for file Data 1 and k=1,3,5,7,9. ...... 53

Table 6. Comparing classification error ratio for KNN between Data 1 and Data 2. ....... 54

Chart 7. Comparing classification error ratio simple KNN, for Data 1 and Data 2. ....... 54

Table 7. Offline classification error for several algorithms. ......................................... 55

Chart 8. Offline classification error ratio for several algorithms. ............................... 56
CHAPTER 1: INTRODUCTION

Localization using radio receivers to find the bearings of a radio transmitter with
the use of simple triangulation was known and used since the invention of wireless
communication. It was actually introduced in World War II to locate soldiers in
emergency situations. In addition, Global Positioning System (GPS) was introduced
during the Vietnam War and became available for commercial applications around 1990
[1].

Nowadays location based applications play an important role in wireless markets.
Indoor positioning is motivated by a number of applications intended for commercial,
public safety, event planning, and military uses [1]. Examples of these applications are
using tracking, positioning, navigating, and some other personal uses.

Most localization applications rely on the cellular network infrastructure meant
for outdoor environments. However, ubiquity of wireless local area networks (WLAN)
makes the opportunity to provide this services in indoor areas. Relying on the existent
network infrastructure for finding a mobile device removes the need to specify a
dedicated structure for this purpose that is very cost-effective.

The main contribution of this thesis is the design, implementation, and
performance evaluation of an indoor localization system for smartphone users that relies
on analyzing RF received signal strength (RSS) from the WLAN access points near users. The fundamental principle I applied for localization is that the ranges of signal values from individual access points seen as a multidimensional vector vary from place to place, such that the signal vector reported by a node's wireless card is essentially a fingerprint that, theoretically, can be matched to a single 3D location in space using signal propagation analysis. In practice, the RF signal value from an access point varies even within a few seconds over a 0-10 dBm range for a static user. Thus, analytic localization methods using signal propagation models (e.g. multilateration or triangulation – in 2D space) are ineffective due to large errors [10].

Figure 1. Indoor localization using Wi-Fi AP signal fingerprinting:

The vector of AP RSS at a user is pattern-matched against an online fingerprint pattern model. The user receives an ordered vector with the most likely positions on the floor map graph.

The method we apply for localization is pattern recognition. The WLAN AP signals are sampled at selected points on a floor map that form a graph and reported to a signal store web service during the training phase. This web service will build a pattern
recognition model and store it online. This model implements a supervised classification machine learning technique, where sampled RSS vectors are classified to location identifiers. A user that needs to identify its location on the floor map for which there exists an up-to-date pattern recognition model reports one or more signal vectors, as they are gathered, to the localization web service. This service will use a classifier such as KNN or support vector machine to infer the most likely user locations. This likely location vector with location IDs is reported back to the user's terminal to be displayed graphically on the floor plan in a user-friendly way. The user likely location vector is also stored in the Campus2020 user state database, to be used for implementing other context-aware applications relying on location and trajectory.

1.1 The Campus 2020 Project

The Campus 2020 project investigates new means for improving the campus experience for students, faculty, and staff, by streamlining their interaction with the campus environment, and also by optimizing common activities. This person-centered project aims at environmentally-friendly interactive cyber-physical campus, for both knowledge and action. Its goal is to provide multi-source information and interface between human and natural/man-made environments, using mobile tech devices as well as allowing for real-time sensor-based decisions and actions [2].

Key elements of the Campus2020 project's ability to offer context-aware services are localization and navigation services for people on campus. Outdoor GPS-based localization and navigation is being addressed by a another team. In this thesis we present our work on the indoor localization service – essential for finding people, for obtaining
directions, and for navigation across the entire campus. User location is also necessary for providing a fully integrated campus experience, combining social networking elements with university-centered social events into a single platform. More benefits of having an indoor positioning and navigation system are discussed in the next section.

1.2 Applications and Motivations

Finding special places like water fountains, rest rooms, stairways, conference rooms and etc. is a big demand in big buildings. Following information signs is not easy for everyone, like blind people and people who are in rush. In addition, sometimes we must know possessions of a building before searching inside it.

Finding faculties and their rooms, seminars, conferences and other events is an everyday activity for people on campus. Having indoor positioning system can arrange meetings for students and faculties. Events locations can be loaded to the map, this way there is no need for extra activities and time consuming announcements.

In case of emergency, people can be guided through the service to safe places. This is one of the most important aspects of position-based services because it can save lives. For example in case of fire application can guide people to outside stairways, or in case of earthquake, it can guide you to the safe places.

There are some special applications for campus 2020 project that the indoor localization is a requirement for it.
1.3 Campus 2020 Special Application Scenario

For the following we define two user roles: faculty and student. The system encompasses the communication/computing infrastructure, mobile terminals, protocols, and application interfaces.

**Faculty scheduler.** A faculty member enters the EE building and the system detects their arrival. Upon arrival, the system notifies the faculty on his phone app of immediate or near term items on the schedule, such as meetings to attend, seminars, or classes. Location, participants, and the ability to contact them via phone or text messaging is provided by the mobile application. The system sends reminders during the work day for the scheduled activities, and also provides generic calendar-type features for activity scheduling. The faculty scheduler could allow users to organize meetings by giving availability windows and picking the most feasible time. The scheduler system would integrate with the user's regular calendar application, such as Google Calendar or Zimbra.

**Student scheduler.** A student arrives at the University. The system detects the arrival and notifies the student about immediate and near-term scheduled activities (with location and timing), such as classes, meetings, tutoring sessions, seminars, etc. The same scheduling system used for faculty also supports student users, providing similar services. In addition, the student scheduler could be enhanced with integration with Blackboard, for instance to pull homework deadlines, tasks, etc. The scheduler can be used to keep track of students’ degree progress and notify of necessary paperwork. Academic advisors would like to have such as tool.
**Student social.** Students spend a lot of time on campus and can use the scheduler described above to also help their social and academic life in ways that Facebook and similar applications do not support yet. Depending on the context (time/location/scheduled activities) students can be notified of pertinent activities they have not sought actively. The system has the ability to learn preferences, patterns and, use the student schedule. The system can track user location and extrapolate trajectory based on history/schedule.

Some example uses: notify when a seminar with a subject related to the user's interests (e.g. degree program) is taking place nearby; when colleagues/project team members or friends are located nearby (e.g. hanging out at the gym, faculty club, or having lunch at the cafeteria); when other relevant events extracted from the FAU/College/Department event schedule occur.

**Status query.** A user queries the system about the location and schedule of a faculty. The system can provide information on whether faculties are in their office, in a meeting, or in a class. The weekly course schedule (available online) lists location and course time. A personal scheduler system keeps track of office hours and other time-specific teaching/research activities.

That subject user controls the level of access to location and schedule information offered by the system about a user in order to maintain an acceptable level of privacy. The subject user can modify access permission to this information at any time using a mobile application or from the web.
1.4 Problem Statement

Position-based services like Google maps uses GPS for triangulating position of a devices. Here we face a problem because GPS doesn’t work indoors or at least it work poorly. Another problem is that services like Google maps don’t have all building maps. Without having a precise map, navigation and positioning mean nothing.

There are so many answers to this problem by different companies and algorithms. Using RFID [3], Bluetooth [9], Wi-Fi signals [6,7] and etc. Wi-Fi network infrastructure is found in many public facilities and can be used for indoor positioning. In addition, the ubiquity of Wi-Fi-capable devices makes this approach especially cost-effective.

In the chapter 2, we will show more solutions addressing this problem in detail.

1.5 Localization Methodology

The processes involve in the localization system are shown in Figure 2.
Figure 2. The localization methodology;

Involves floor map graph editing, RSS sampling throughout a building at specific locations forming a graph's node, model training, and position classification based on RSS sampling at runtime.

1. Floor map Graph Editing. It is impractical to sample the RSS at all grid positions through a building because of the large floor size and because of quite large differences in probability that users are located at any grid coordinate. Instead, the floor map graph is made of nodes corresponding to relevant locations on the floor map, such as rooms, multiple points on corridors, in front of offices, meeting areas, or on stairways. Larger rooms are subdivided in areas, with a node per area. Each node has the following attributes: (x, y) position coordinate, floor number, and building name. Two graph nodes are connected by a bi-directional edge if one can walk from a node to the other, directly, or through a door. The graph normally is planar, but this is not mandatory.

2. Wi-Fi AP RSS Sampling. Periodically, the AP signals are sampled by project volunteers or through crowdsourcing using the Campus2020 localization app. The
smartphone's Wi-Fi Manager component generates a periodic report with RSS vectors for APs in range (up to about -98 dBm). This RSS vector is sent together with the actual user location identified by the graph node to the web server for storage and pattern matching model training.

3. Localization (Client Sampling). A user who wants to finds its location on the floor map starts an app that reports to the localization web service the current AP RSS vector.

4. Localization Algorithm (classifier). This process matches an RSS vector from a client with its RSS fingerprint model using one or more classification algorithms and generates a vector with likely locations that is sent back to the localization client. This location vector is also stored in the Campus 2020 user state database to be used for context-aware services, such as “friend proximity alert” or indoor navigation assistance.

5. Location and Map GUI. This is the Android app that displays the likely location vector on the current floor map.

1.6 Thesis Outline

This thesis has five chapters. In this chapter we introduced the area that we are working on in summary. In Chapter 2 we will discuss efforts that has been done by academics and companies for indoor localization in detail. In Chapter 3 our project will be examined completely. Software requirements and architectures are shown and explained. In Chapter 4, system evaluation and experiment results of different algorithm that we have used are illustrated. And in Chapter 5 we discuss our next steps toward the
future of the Campus 2020 project and present conclusions and lessons learned from this project.
CHAPTER 2: BACKGROUND AND RELEVANT TECHNOLOGIES

As mentioned in previous chapter, location-based services are very useful for many different applications. Thus a lot of researches have been made regarding to achieve this goal. There are so many commercial systems that have been introduced to address this need.

One methodology for indoor positioning system is using a special hardware for this purpose that has been demonstrated a high accuracy. There are some studies that use RFID [3], infrared [4], or ultrasound [5] that has been successful with a high accuracy, but installation of the hardware is needed that is not cost-effective.

Another approach toward indoor positioning is to use already installed hardware like Wi-Fi infrastructures [6,7], GSM signals [8], and also Bluetooth signals [9] in buildings by measurement of received radio signals. The issue with this approach is the time consuming training phase.

2.1 Indoor positioning solutions work similar to GPS

Locata [11] provides beacons that send out signals that cover large areas and can penetrate walls. Locata receivers work similarly to how GPS receivers work. The U.S. Department of Defense is an early Locata user [12].
Some companies like Nokia [13] use beacons that send out Bluetooth signals. While any Bluetooth device can read them but the problem is they only cover a few square meters.

Many companies use Wi-Fi signals. Google [14], Navizon [15] and Skyhook [16] are among the leaders in this area. Using a good floor map a Wi-Fi signal receiver device can be positioned indoors.

TruePosition[17], which offers a cell tower locating solution, they use TV signals for location determination.

2.2 Other solutions use light or magnetic fields

ByteLight [18] uses unsteady light patterns from its LED light fixtures. This way by receiving the light code using camera on a phone and sending it to the server, they can say that the phone is under the special “light” and can locate the device.

IndoorAtlas [19] reviews buildings using their internal magnetic map. By getting some magnetic field fingerprints and using cell phone’s internal digital compass and a server much like ByteLight’s procedure, they provide an indoor positioning system [20].

2.3 RFID and inertial systems work very differently

By using Passive radio frequency identification tags (RFIDs) and knowing the starting location of a device it is possible to have an indoor localization system. These sensors use accelerometers, gyroscopes and other sensors including clocks to track orientation and distance to keep track of location in near real-time. The latest inertial solution, from DARPA [22], is a chip smaller than a penny [21].
2.4 Hybrid solutions

For getting a better precision in indoor localization using more than one sensor or solution probably can be useful. There are some companies and academic works that use Wi-Fi and magnetic field for this purpose [23].
CHAPTER 3: ARCHITECTURE AND ALGORITHMS FOR INDOOR LOCALIZATION

In this chapter we present our web-based architecture for indoor localization, its main components, and localization algorithms. We begin with a summary of the system services that are essential for implementing the methodology described in the Introduction.

The Campus2020 web-based indoor localization system needs to provide three main services:

1. Wi-Fi Fingerprint Storage Service
2. Wi-Fi Fingerprint Query Service
3. Localization Query Service

1. The first service saves Wi-Fi RSS fingerprints captured on a smartphone and delivered to the service via HTTP to a database. The RSS fingerprint database is used to build and train the Wi-Fi fingerprint machine learning model, necessary for the location classification algorithm.
The second service allows another program to query the database for a set of RSS fingerprints based on a variety of parameters, such as timestamp, Wi-Fi channel number, and frequency. The selected fingerprints are returned CSV or JSON format.

3. The third service receives from the client one or more scan results sets (RSS fingerprints), computes client likely locations using a classification algorithm, and returns these results back to the client. For testing, crowdsourcing sampling, and for performance evaluation, the client can also add to the request the user’s actual location (entered manually on the user app). In that case, the server computes the client’s feasible locations, stores the actual and predicted feasible locations for the future performance analysis, saves the sent Wi-Fi fingerprint to the scan result tables, and finally returns the predicted locations to the client. One requirement of this service is to turn back results in less than 5 seconds and also with accuracy of at least 75%. For more information about these services read the Campus 2020 indoor localization service specification from Appendix A.

Nowadays we can find smartphones in everybody’s hands. Apple, Google, Microsoft, Samsung, Nokia, HTC, and some other well-known companies are market leaders.

Apple iOS, Google Android, and Windows Mobile Operating System are among the most popular platforms chosen by users and developers. Recently Apple has put out its new Operating System iOS7, Google has published Android 4.3 Jellybean, and Microsoft has released Windows Phone 8.
According to the statistics from Gartner, Inc. [Figure 3] Android will be the dominant OS till 2015, meaning that a majority of people will own Android phones. In addition, Android software is written in Java and the platform has tremendous support for developers from Google and a variety of other software providers.

![Gartner’s Prediction for the 2015 Mobile Market](image)

**Figure 3. Gartner’s Prediction for the 2015 Mobile Market**

Consistent with the above information we decided to use Android as our platform for developing the Campus 2020 indoor localization system.

In this chapter we will discuss the application features and requirements and the system architecture in detail.

### 3.1 Architecture

This section presents the overall system architecture. In the Campus 2020 project we want to give students the ability to find their position using smart phones inside the
campus buildings. To reach this goal we have implemented the methodology from Figure 2 using the system architecture shown in Figure 4.

**Figure 4. Overall system architecture for indoor localization, with the essential components and services.**

The main components are:

1. Map editor
2. Android localization client
3. Android application for Wi-Fi RSS sampling
4. Data Mining Classification algorithms for Wi-Fi localization
5. Java web services for fingerprint storage and classification

Below we explain each one of our applications in detail.

### 3.1.1 The Map Editor

As we mentioned in first chapter for indoor localization we need indoor maps with details but also a graph structure with location nodes and edges for navigation. The map editor desktop application takes a floor map image as an input. The user chooses relevant reference points in the map, creates graph nodes for them, and gives them
meaningful names, that include building name/floor/room number. The application saves the map graph to an XML file that contains map details, like the graph structure (nodes, edges) and also information about the map imaging scaling, such as real-world coordinates of the coordinate system and image pixel density (pixels/meter). Figure 5 shows a screenshot of the Map Editor application.

Figure 5. Screenshot of the Map Editor application that is used to create map graphs for building floors.

Here is an example XML chunk from the fourth floor map file of the East Engineering Building at FAU:

```xml
<mapgraph>
    <imageURI>http://10.15.5.213/indoorloc/EE96-floor-4.jpg</imageURI>
    <pixelDensity>24.3</pixelDensity>
    <cornerX>-284.70588235294116</cornerX>
    <cornerY>-10.588235294117647</cornerY>
    <nodes>
        <n id="EE96/4/403/3">
            <bld>EE96</bld>
            <fl>4</fl>
            <a>403</a>
        </n>
    </nodes>
</mapgraph>
```
The XML file contains the address of the image, reference points as nodes, and edges including two points.

### 3.1.2 Android Client Application

This application has two roles, RSS sampling and positioning. It gives a data technician the ability to gather Wi-Fi data from reference points on the map graph and to make send them to the RSS storage service. Figure 6 shows a screenshot of this application while executing RSS sampling.
As seen in Figure 6 the app loads the map’s image and the map XML file graph created using the map editor and shows them on the screen. The data technician can choose between different maps stored on the web server. The username can be chosen and also the IP of the server that he/she wants to send the data to. By touching each green node on the graph view, it turns to red, and app begins sampling and sending Received Signal Strength vectors from nearby access points to the server using the service that saves the fingerprints data. Also the app shows the number of data rows in the database and this way technician can make sure that the process is going well. By touching the red point again the sampling process can be stopped.

The other function of this app is indoor positioning. Using this app the user can visualize its likely positions – nodes identifying building, floor, and the area of the device as returned by the location classification algorithms.
Figure 7. Screenshot from positioning fragment of the android application.

Showing the localization screen. Likely positions are shown with different shades of blue. Nodes with higher quality score are drawn with a darker color.

The localization service returns a vector of likely positions sorted based on a quality score that depends on the classification algorithm. As shown in Figure 7, by touching the scan button the user can see the most probable positions of the device indicated with different shades of blue. The darkest color node is the most likely as it has the best quality score. By touching the Scan button the user can contribute to the RSS map training process, we discussed earlier in this chapter. By tapping the Location button the user runs the localization function of the application.

One other usability feature of this app is the ability to pinch and zoom on the displayed map structure that has been implemented very well. This way the user can see more details on the map or deal with a map that is much larger then what the 4-5” device screen would allow.
3.1.3 Machine Learning Algorithms for Wi-Fi based Localization

For the localization web service we use several machine learning algorithms that match RSS fingerprints to RSS patterns associated with nodes (locations) in the map graph. For this part of project we designed and implemented a framework for pluggable classification algorithms and for their offline evaluation. We implemented two supervised classification algorithms, K-Redpin and KNN. The first algorithm, K-Redpin, is an improved version of the Redpin algorithm [24] that relies on majority voting. The second one is the standard KNN algorithm, with and without weighting.

During this project we wanted to try to find a reasonable outcome by testing several algorithms with the sampled data. In addition to these two algorithms, we have also classification algorithms from the Weka data mining tool [25].

We used the query service of the RSS database to generate a CSV file with training data to be used for classification. We used this file for the algorithms we implemented in Java, K-Redpin and KNN. The Weka toolset has the Explorer program that provides a user-friendly way for analyzing data. The purpose of Explorer is to explore various ML algorithms and to find out quickly the best one and its optimal parameters. Weka gives users options for normalizing and filtering data, for adjusting variables in all algorithms. Figure 8 shows a screenshot in Weka Explorer displaying the RSS data.
For this data set we have 348 instances of data, with 100 attributes, and 29 distinct classes. Each class has almost 12 instances. If we want to see how our attributes look like we can click on the **Visualize All** button the Explorer displays the screenshot from Figure 9.
3.1.3.1 The K-Redpin Algorithm

The original Redpin algorithm has been implemented for indoor localization at ETH Zurich [24]. Redpin maintains a set of RSS fingerprints associated with locations and for a localization request it iterates through all stored fingerprints and calculates a similarity score between the fingerprint to classify and those from the RSS database. The similarity score takes into consideration the number of common access points (NCAP) points, the number of non-common access points (NNAP), and also the difference between RSS levels from the same AP. The Redpin source code can be found here [26]. Redpin uses a weighted combination of the vector distance (as in KNN, with K=1) and the AP similarity [24].
To understand how does Redpin works, here is the formulation of the similarity score, as presented in article [27].

Considering the following function,

$$\delta(X_i) = \begin{cases} 
0, & X_i = 0 \\
1, & X_i \neq 0 
\end{cases}$$

The NCPA of two fingerprints X, Y is:

$$NCPA = \sum_{i=1}^{n} \delta(X_i) \cdot \delta(Y_i)$$, Where • represents the dot product

The NNAP of RSS vectors X, Y is:

$$NNAP = \sum_{i=1}^{n} \delta(X_i) \oplus \delta(Y_i)$$, Where ⊕ is the exclusive disjunction

The generalized similarity score of RSS vectors X, Y can be defined as:

$$S(X,Y) = \alpha \times NCPA - \beta \times NNAP + \gamma \times C(X,Y)$$

where $C(X,Y)$ is a function that calculates the contribution of RSS closeness between vectors X and Y. Alpha, beta, and gamma are bonus and penalty weights that can be adjusted. In our implementation, $C(X,Y) = |X - Y|$ and $\gamma < 0$: the closer two RSS vectors are, the greater the similarity score.

The K-Redpin algorithm computes the similarity score for the vector sampled at the user's current location with a set of selected RSS sample vectors and takes a simple majority vote to determine the class of the current RSS vector to match. In case of a tie,
the class with the best (highest) score wind. We performed a search for the optimal values for the beta, gamma, and K parameters for K-Redpin.

3.1.3.2 K Nearest Neighbor (KNN)

In machine learning and pattern recognition one of the simplest and more effective algorithms is KNN. It is a non-parametric method for both classification and regression. It predicts a vector's class membership based on to K (must be a positive integer) closest training samples. KNN is a lazy learning algorithm, meaning it defers all the computation until the classification is actually needed. KNN computes the K closest neighbors by iterating over a sample set for all classes and storing for each sample, its class and its vector distance (in d-dimensional space). The concept is shown in Figure 10.

![KNN Concept](image)

**Figure 10.** KNN selects the majority vote from the vector's K nearest neighbors.

For example suppose K is equal to 3. For an object with two closest neighbors from class A and with one neighbor from class B, the object will classify as an A category. The most common distance measurement function for continuous variables is Euclidean Distance, for dimension 2:
where $a$ is the vector to classify and $b_i$ are vector samples, $i$ is the access point index, and $n$ is the number of all access points we have in our database table.

### 3.1.3.3 Support Vector Machine (SVM)

SVM is a supervised learning method offering learning algorithms for both regression and classification. SVM is a non-probabilistic binary method that computes a decision boundary to separate objects of binary classes. SVM are extended to non-linear classification using the so-called kernel trick that simply maps the input into a higher dimension feature space.

In our case study we are not looking for a binary classification. SVM can do multiclass classification by converting the multiclass problem into several binary class problems.

The Weka implementation for SVM is called SMO and is derived from John Platt's *Sequential Minimal Optimization* algorithm from [2].

### 3.1.4 Java Servlet Services

As mentioned earlier in this chapter, there are three major web services that the localization relies on: storing RSS fingerprints, query RSS fingerprints based on several attributes, and the localization web service. These services are used by the RSS sampling Android app and the localization app. We have implemented these services using Java.
servlets. The services' requirements and interface specification is available in Appendix A. Their architecture is described in the following pages.

3.2 System Architecture

In this Section we present the overall system architecture, technologies used, software design and API’s. Figure 11 illustrates the block diagram of the whole system.

![System architecture block diagram.](image)

Figure 11. System architecture block diagram.

Figure 12 shows the different technologies used and how they map onto the architecture.
Figure 12. Technologies that have been used in the project.

The following subsections explain all architecture components in detail and how they work together.

3.2.1 The Map Graph and the Map Editor Tool

For each floor of a building that supports the Campus 2020 localization system we use the Map Editor tool to create and edit a graph data structure. Nodes are created for each relevant location on a floor, such as in rooms, labs, corridors. Larger rooms may have several nodes defined. An edge connects two nodes if one can walk from a node to the other. The edge weight is equal to the physical distance between the points. The graph should be planar, but there is no restriction on that.
The graph structure, with its navigable edges, is better suited for supporting in the future an indoor navigation function on the smartphone app than a grid-based location representation.

The Map Editor is a desktop tool written in the Scala language that one can use to create and edit the map graph overlayed on a building's floor plan. A screenshot from the Map Editor is shown in Figure 5. In this Figure, the map graph, with its nodes and edges, are shown with green color, superimposed on the floorplan image.

### 3.2.1 Android Client Application

This application is written in Java and uses the Android 4.3 libraries and APIs. Its code is split in four packages addressing different duties in the application:

- Dialogs
- Fragments
- Indoormap
- Wifireport

The dialogs package is responsible for setting username and server IP address or hostname, and for navigating between maps. The fragments package includes training fragment and positioning fragment, where an Android fragment is the object that displays and manages a user interface for an app. The Indoormap package contains the classes for loading the map images and relevant XML files, and classes implementing the image view, with pinch and zoom support. In addition, this package has the classes that handle the graph data structures, such as nodes and edges. Finally, the Wifireport
package has different classes for implementing RSS sampling using the Android wireless networking API (WiFiManager). Below we can see some important class diagrams of the app Figure 13.

Figure 13. Class Diagram of Our Android Project
3.2.2 The RSS Samples Database Schema

A MySQL database stores on the server side the RSS samples, AP information (MAC address, BSSID, frequency), and user-related configuration (user name, machine type). The server-side code relies on the Hibernate library and code generation to implement persistence. The classes generated by Hibernate for object persistence are shown in the UML class diagram from Figure 14. This class diagram also reflects the database schema.

Figure 14. WiFi scans data model class diagram. Reflects the design of the classes generated by Hibernate.
One entry in the Scan table describes a scan operation on a sampling client. Its attributes include the scan location (building, floor, area), timestamp, user ID, plus sensor information identifying the phone's position (azimuth, pitch, and roll). The Scan table points to the machine type used by the user. Be believe this is relevant for improving the location accuracy as different platforms (e.g. Apple vs. Samsung vs. Motorola) may have slightly different RF receiver sensitivity and WiFi sensor interfaces.

An RSS sample scan involves one or more scan results, stored in the ScanResults table. A scan result describes the signal power (dBm), and the frequency for the signal from a single access point. An access point is defined by an entry in the AccessPoints table. Its properties are BSSID (WLAN name) and SSID (MAC address).

It is noteworthy to mention that a typical campus wireless intranet may use access point boxes that run at the same time multiple basic service sets (WLANs), on both the 2.4 GHz and 5 GHz bands. Scan results from the WiFiManager Android component do not differentiate the signals coming physically from the same box. We believe receiving multiple samples from the same AP box and consider them as separate attributes for machine learning does not impede classification, although this remains to be investigated in the future.

3.2.3 Localization Web Services

To serve multiple clients and to reduce the storage and computational load on the smartphone clients, we implemented the sampling storage and the localization services as web services, accessible to clients via JSON-encoded HTTP request/reply messages. The web services are written in Java and use the Java EE Servlet framework.
Storage for RSS samples, access points, and other necessary data is provided by a MySQL database running on the server. The Hibernate library converts object-based persistence operations to SQL statements that run on the MySQL server process.

The three web services are:

a. saveWiFiData: storage service. This service is used by a client that must store sampled RSS vectors on the Campus 2020 web server. On Android platforms, the RSS samples are obtained from scanning the 2.4 and 5 GHz bands using the WiFiManager system component. RSS sample vectors, together with user ID, a timestamp, current values for the compass and accelerometer sensors, and machine information are encoded in JSON notation and sent via HTTP to the web server using the GET or POST messages. The SaveWiFiData servlet executes the storage request using the Hibernate library.

The UML sequence diagram for the saveWiFiData service is in Figure 15. It shows the service initiation logic on the client side from the MapView object reacting to a user tapping on a location node. The controller starts WiFi scanning using the WiFiManager component. The LearnerScanner object formats the JSON request and sends it via the HTTP POST method to the service. The saveWiFiData service than saves the incoming samples to the database using the Hibernate library.
b. getWiFiData: the sample query service. This service is used to query RSS samples stored on the server based on several parameters, such as user id, timestamp interval, location attributes (e.g. building and floor). The data are returned either in CSV format (useful for machine learning tools) or encoded in JSON, to be consumed on the client side. This service is useful to create experimental training models to evaluate different ML algorithms.

The UML sequence diagram for this service is shown below, in Figure 16.
c. getLocation: the localization service. This is the main service used by localization clients. A client captures RSS sample vectors, encodes to JSON the sample data plus user ID, timestamp, and optionally, the user's location, and then sends the request via HTTP to this service running on the web server. The service decodes the request, extracts the parameters, and performs location classification using the RSS vector on a selected trained model with a user-configurable classification algorithm. The classifier returns to the service the set of K “best” results (locations), ordered on their “quality” (e.g. distance for KNN), encodes these in JSON and sends this back to the client as an HTTP reply.

The UML sequence diagram for the client-service interaction is displayed in Figure 17.
The scenario from the figure begins when the smartphone user initiates scanning by tapping on a map graph node. For this scenario, the user has selected to also report her current location. The `PositionScanner` object relies on the WiFiManager Android component to get an RSS sample scan. The request is encoded in JSON and sent via HTTP to the web server, which passes the request to the `getLocation` service. For this scenario the service is configured to use the Weka SMO support vector machine classifier. The classifier computes the best set of feasible locations with the `classify()` call. The current scanned RSS vector is saved to the database, and the classification results are also stored for future performance analysis.

A basic authentication method prevents these services from being used without authorization. As future work, a thorough security solution with HTTPS and strong authentication will be implemented for all Campus 2020 services.
3.2.3 Localization Software Design

For the localization service we implemented an extensible Java framework that allows us to develop custom classification algorithms and plug them into the web servlet architecture with minimal effort. We used this framework to implement and test KNN and K-Redpin, classification algorithms discussed earlier in this thesis.

Another advantage of this approach is that it facilitates code reuse and simplifies integration. To evaluate rapidly a large number of classification algorithms and to find their optimal parameters, we created an offline performance evaluation program for the localization problem using this framework. Classification performance results are summarized in Chapter 4.

The UML class diagram for the server-side classifier design is shown in Figure 18, below. The key concepts in this diagram are:

1. The Classifier interface, with methods for:
   a) setting/getting the data model
   b) training the classifier on a data model (actually, on different subsets of vectors from the data model)
   c) classify a vector: classify() returns the best class; classifyOrd() returns an ordered list of classes, starting with the best one, in an object of class Result.
2. The *NoopClassifier* class is an abstract class implementing basic functions of the classifier interface that can be reused by subclasses, such as dealing with the data model. Its design follows the Template Method design pattern.

3. The *Scorer* interface has one method, *score()*, that computes the classification score for one vector to classify using the data model associated with a classifier. Scorer subclasses for the KNN and K-Redpin algorithms are implemented, using a Strategy pattern.

4. The *KVotingClassifier* and *KWeightedClassifier* classes extend the *NoopClassifier* abstract class and implement simple majority vote class decision or a score-weighted class decision algorithm, relying both on *Scorer* objects, as strategy. Thus, the code to create a KNN classifier object is:

   ```java
   Classifier knnClassifier =
   new KvotingClassifier(..., new KNNDistanceScorer());
   ```
Our design includes additional elements for managing ML models, normalization, and classifier evaluators, as seen in the class diagram in Figure 19.

Figure 19. UML diagram for the data model, transformation, and classifier evaluator components.
The data used for training is stored at runtime in an object implementing the DataModel interface. This interface provides comprehensive access to sample vectors (instances) and their properties. BasicDataModel is a DataModel class that loads the training instances from a CSV file.

The DataTransformer interface is used to implement transformation algorithms on the data in the training model, such as filtering, normalization, data limiters – for setting upper/lower bounds, etc.

To evaluate offline a classifier performance, one must implement the DMEvaluator interface. It has methods for evaluating a classifier on the associated training data using a custom method (evaluate()), and to access the resulting error metric and confusion matrix. We implemented three classifier evaluators:

1. DataFileEvaluator: evaluates test vectors from a separate CSV file

2. CrossValidationEvaluator: uses the training data from the data model for cross validation. For instance, for 10-fold cross validation, 90% of the vectors in the data model are used for training, 10% are classified. Then the 90%-10% split is rotated nine more times and the aggregate error ratio and confusion matrix are computed. This is a standard ML evaluation technique, used by default in Weka. Prior to cross validation, a random shuffle of the vectors in the data model is required.

3. NFoldEvaluator: uses all but one (n-1) vectors from the data model for training, and classifies the remaining vector. This is then repeated for all other vectors from
the data model. Such an evaluator used with smaller training data sets generally yields a lower error ratio due to using more vectors for training.

Our design also integrates classifiers from the Weka library.
CHAPTER 4: SYSTEM EVALUATION

In this chapter we will review the results from the performance evaluation of our system. The main performance metric for measuring localization performance is the error ratio defined as the ratio of the number of misclassifications divided by the total number of classifications. Ideally, the error ratio should be as close to 0 as possible.

We have evaluated the system in two phases, offline evaluation and online evaluation. The offline evaluation phase was used to select the best performing classification algorithms and its configuration parameters. For offline algorithm evaluation we used two sets of sampled data in CSV format and evaluated several classification algorithms in “batch mode” using 10-Fold cross validation over a multidimensional range of runtime parameters. The online system evaluation, we used the localization app to record the actual location (ground truth) and then had the getLocation web service compute and store the location to the database. This way we measured the error rate of the system as used in practice.

4.1 Offline Evaluation

We ran our experiments on 4th floor of the FAU East Engineering building. On the floor map shown in Figure 20, we put 44 reference points using the map editor program. For sampling the RSS fingerprints we used the Android client application. Two
different datasets were captured. The first dataset has 12 samples for each node (location) and 29 nodes have been used, with a total of 348 RSS vectors dataset number 1. Each data row has 99 features or attributes. These correspond to the RSS in dBM received from each access point. This way, each row may have some missing values. Different algorithms have different optimal default value for these missing points. For SMO it is -94 dBM.

A second dataset was sampled for the same floor plan with 30 nodes, with 20 samples for each node captured with the user facing four orthogonal orientations. The second dataset has 600 different rows.

Figure 20. 4th Floor of East Engineering Building

We have used K-Redpin, KNN, some other algorithms using our own original code and Weka data mining tool. Depending on algorithm we use we have to set some parameters. In our study we have tried to find the optimal parameters for each algorithm in our case. In the following we will see evaluation of each one.

4.1.1 The K-Redpin Algorithm
The K-Redpin algorithm has 4 parameters that must be adjusted, alpha, beta, gamma, and K. The original Redpin algorithm determines the best score matching class and returns that. The K-Redpin algorithm takes the majority vote among K best scoring classifications. The alpha, beta, and gamma parameters have been introduced in the original algorithm to weigh the contributions to the overall score of the number of common/not common access points and the signal closeness function $C()$:

$$S(X,Y) = \alpha \times NCAP - \beta \times NNAP + \gamma \times C(X,Y)$$

We found that the best combination is $\alpha = 1$, $\gamma = 12$, and then we determined beta and K separately.

In Table 1, beta and K are variables using the dataset including 348 samples. The results are summarized in Chart 1.
<table>
<thead>
<tr>
<th>Beta</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.46176470</td>
<td>0.39705882</td>
<td>0.38529411</td>
<td>0.38529411</td>
<td>0.37058823</td>
</tr>
<tr>
<td>0.75</td>
<td>0.37058823</td>
<td>0.26764705</td>
<td>0.28235294</td>
<td>0.28235294</td>
<td>0.25588235</td>
</tr>
<tr>
<td>1</td>
<td>0.35294117</td>
<td>0.24117647</td>
<td>0.25588235</td>
<td>0.25</td>
<td>0.25588235</td>
</tr>
<tr>
<td>1.25</td>
<td>0.33529411</td>
<td>0.25294117</td>
<td>0.24411764</td>
<td>0.23235294</td>
<td>0.25294117</td>
</tr>
<tr>
<td>1.5</td>
<td>0.32941176</td>
<td>0.26470588</td>
<td>0.25294117</td>
<td>0.23529411</td>
<td>0.25294117</td>
</tr>
<tr>
<td>1.75</td>
<td>0.32941176</td>
<td>0.26764705</td>
<td>0.25</td>
<td>0.24117647</td>
<td>0.25588235</td>
</tr>
<tr>
<td>2</td>
<td>0.32941176</td>
<td>0.28529411</td>
<td>0.24705882</td>
<td>0.24705882</td>
<td>0.27058823</td>
</tr>
<tr>
<td>2.25</td>
<td>0.32941176</td>
<td>0.29117647</td>
<td>0.25</td>
<td>0.25588235</td>
<td>0.26176470</td>
</tr>
<tr>
<td>2.5</td>
<td>0.33235294</td>
<td>0.28823529</td>
<td>0.25294117</td>
<td>0.25588235</td>
<td>0.26176470</td>
</tr>
<tr>
<td>3</td>
<td>0.33235294</td>
<td>0.28235294</td>
<td>0.24705882</td>
<td>0.25</td>
<td>0.27058823</td>
</tr>
<tr>
<td>5</td>
<td>0.32352941</td>
<td>0.28823529</td>
<td>0.25882352</td>
<td>0.25882352</td>
<td>0.27352941</td>
</tr>
<tr>
<td>7</td>
<td>0.32058823</td>
<td>0.29411764</td>
<td>0.26176470</td>
<td>0.26470588</td>
<td>0.27647058</td>
</tr>
<tr>
<td>10</td>
<td>0.31470588</td>
<td>0.29705882</td>
<td>0.26176470</td>
<td>0.27058823</td>
<td>0.28529411</td>
</tr>
<tr>
<td>12</td>
<td>0.31470588</td>
<td>0.29705882</td>
<td>0.26470588</td>
<td>0.27352941</td>
<td>0.29117647</td>
</tr>
<tr>
<td>15</td>
<td>0.31470588</td>
<td>0.3</td>
<td>0.27058823</td>
<td>0.27352941</td>
<td>0.28529411</td>
</tr>
<tr>
<td>18</td>
<td>0.31764705</td>
<td>0.3</td>
<td>0.27058823</td>
<td>0.28235294</td>
<td>0.30294117</td>
</tr>
<tr>
<td>20</td>
<td>0.31764705</td>
<td>0.29705882</td>
<td>0.27058823</td>
<td>0.28529411</td>
<td>0.30588235</td>
</tr>
<tr>
<td>22</td>
<td>0.31764705</td>
<td>0.29705882</td>
<td>0.27352941</td>
<td>0.29117647</td>
<td>0.31176470</td>
</tr>
<tr>
<td>25</td>
<td>0.31764705</td>
<td>0.3</td>
<td>0.27647058</td>
<td>0.29411764</td>
<td>0.31470588</td>
</tr>
<tr>
<td>30</td>
<td>0.31764705</td>
<td>0.3</td>
<td>0.27941176</td>
<td>0.3</td>
<td>0.32058823</td>
</tr>
<tr>
<td>40</td>
<td>0.32058823</td>
<td>0.3</td>
<td>0.28235294</td>
<td>0.30294117</td>
<td>0.32352941</td>
</tr>
<tr>
<td>50</td>
<td>0.32352941</td>
<td>0.3</td>
<td>0.27647058</td>
<td>0.30294117</td>
<td>0.32352941</td>
</tr>
</tbody>
</table>

Table 1. Classification error ratio for K-Redpin for model Data 1 depending on parameters K and beta.
Chart 1. Classification error ratio for K-Redpin for model Data 1 depending on parameters K and beta.

We did the same experiment with dataset 2, which has 600 samples, and the results are shown in Table 2, and summarized in Chart 2.
<table>
<thead>
<tr>
<th>Beta</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.301666667</td>
<td>0.281666667</td>
<td>0.26</td>
<td>0.261666667</td>
<td>0.261666667</td>
</tr>
<tr>
<td>0.75</td>
<td>0.205</td>
<td>0.176666667</td>
<td>0.151666667</td>
<td>0.151666667</td>
<td>0.153333333</td>
</tr>
<tr>
<td>1</td>
<td>0.205</td>
<td>0.155</td>
<td>0.138333333</td>
<td>0.146666667</td>
<td>0.136666667</td>
</tr>
<tr>
<td>1.25</td>
<td>0.2</td>
<td>0.153333333</td>
<td>0.133333333</td>
<td>0.133333333</td>
<td>0.136666667</td>
</tr>
<tr>
<td>1.5</td>
<td>0.196666667</td>
<td>0.15</td>
<td>0.133333333</td>
<td>0.125</td>
<td>0.133333333</td>
</tr>
<tr>
<td>1.75</td>
<td>0.191666667</td>
<td>0.145</td>
<td>0.133333333</td>
<td>0.135</td>
<td>0.138333333</td>
</tr>
<tr>
<td>2</td>
<td>0.188333333</td>
<td>0.148333333</td>
<td>0.133333333</td>
<td>0.14</td>
<td>0.15</td>
</tr>
<tr>
<td>2.25</td>
<td>0.186666667</td>
<td>0.155</td>
<td>0.136666667</td>
<td>0.156666667</td>
<td>0.148333333</td>
</tr>
<tr>
<td>2.5</td>
<td>0.19</td>
<td>0.153333333</td>
<td>0.135</td>
<td>0.158333333</td>
<td>0.156666667</td>
</tr>
<tr>
<td>3</td>
<td>0.186666667</td>
<td>0.16</td>
<td>0.141666667</td>
<td>0.161666667</td>
<td>0.156666667</td>
</tr>
<tr>
<td>5</td>
<td>0.198333333</td>
<td>0.163333333</td>
<td>0.156666667</td>
<td>0.168333333</td>
<td>0.17</td>
</tr>
<tr>
<td>7</td>
<td>0.2</td>
<td>0.166666667</td>
<td>0.163333333</td>
<td>0.166666667</td>
<td>0.166666667</td>
</tr>
<tr>
<td>10</td>
<td>0.2</td>
<td>0.171666667</td>
<td>0.161666667</td>
<td>0.173333333</td>
<td>0.176666667</td>
</tr>
<tr>
<td>12</td>
<td>0.203333333</td>
<td>0.178333333</td>
<td>0.163333333</td>
<td>0.178333333</td>
<td>0.178333333</td>
</tr>
<tr>
<td>15</td>
<td>0.201666667</td>
<td>0.178333333</td>
<td>0.168333333</td>
<td>0.18</td>
<td>0.183333333</td>
</tr>
<tr>
<td>18</td>
<td>0.2</td>
<td>0.178333333</td>
<td>0.166666667</td>
<td>0.18</td>
<td>0.186666667</td>
</tr>
<tr>
<td>20</td>
<td>0.2</td>
<td>0.178333333</td>
<td>0.168333333</td>
<td>0.18</td>
<td>0.186666667</td>
</tr>
<tr>
<td>22</td>
<td>0.2</td>
<td>0.178333333</td>
<td>0.168333333</td>
<td>0.18</td>
<td>0.186666667</td>
</tr>
<tr>
<td>25</td>
<td>0.201666667</td>
<td>0.178333333</td>
<td>0.168333333</td>
<td>0.181666667</td>
<td>0.188333333</td>
</tr>
<tr>
<td>30</td>
<td>0.203333333</td>
<td>0.176666667</td>
<td>0.17</td>
<td>0.18</td>
<td>0.188333333</td>
</tr>
<tr>
<td>40</td>
<td>0.205</td>
<td>0.18</td>
<td>0.168333333</td>
<td>0.181666667</td>
<td>0.191666667</td>
</tr>
<tr>
<td>50</td>
<td>0.205</td>
<td>0.18</td>
<td>0.171666667</td>
<td>0.181666667</td>
<td>0.195</td>
</tr>
</tbody>
</table>

Table 2. Classification error ratio for K-Redpin for model Data 2 depending on parameters K and beta.
The first conclusion from this study is that the larger training set decreases the error rate very well. Error rate for the first dataset with 12 samples per each class was at best %23.63. By increasing the samples to 20 for each class the accuracy increased almost by %10. Although the time for evaluation has increased by making the training set larger, it is worth it because the reduced error by 10%, with minimum impact on running time. Table 3 summarizes the obtained error ratio data for K-Redpin running on data sets 1 and 2 with different K and beta parameters.

Chart 2. Classification error ratio for K-Redpin for model Data 2 depending on parameters K and beta.
<table>
<thead>
<tr>
<th>Beta</th>
<th>Data 1, K=7</th>
<th>Data 2, K=7</th>
<th>Data 1, K=5</th>
<th>Data 2, K=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.385294118</td>
<td>0.261666667</td>
<td>0.385294118</td>
<td>0.26</td>
</tr>
<tr>
<td>0.75</td>
<td>0.282352941</td>
<td>0.151666667</td>
<td>0.282352941</td>
<td>0.151666667</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.146666667</td>
<td>0.255882353</td>
<td>0.138333333</td>
</tr>
<tr>
<td>1.25</td>
<td>0.232352941</td>
<td>0.133333333</td>
<td>0.244117647</td>
<td>0.133333333</td>
</tr>
<tr>
<td>1.5</td>
<td>0.235294118</td>
<td>0.125</td>
<td>0.252941176</td>
<td>0.133333333</td>
</tr>
<tr>
<td>1.75</td>
<td>0.241176471</td>
<td>0.135</td>
<td>0.25</td>
<td>0.133333333</td>
</tr>
<tr>
<td>2</td>
<td>0.247058824</td>
<td>0.14</td>
<td>0.247058824</td>
<td>0.133333333</td>
</tr>
<tr>
<td>2.25</td>
<td>0.255882353</td>
<td>0.156666667</td>
<td>0.25</td>
<td>0.136666667</td>
</tr>
<tr>
<td>2.5</td>
<td>0.255882353</td>
<td>0.158333333</td>
<td>0.252941176</td>
<td>0.135</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
<td>0.161666667</td>
<td>0.247058824</td>
<td>0.141666667</td>
</tr>
<tr>
<td>5</td>
<td>0.258823529</td>
<td>0.168333333</td>
<td>0.258823529</td>
<td>0.156666667</td>
</tr>
</tbody>
</table>

Table 3. Comparing the error ratio K-Redpin across Data 1 and Data 2 for K=5.

The same data are shown in Chart 3 and Chart 4.

Chart 3: Comparing K-Redpin across Data 1 and Data 2 for K=5
Chart 4. The classification error ratio for K-Redpin with K=5,7, for the two data sets, depending on Redpin parameter beta.

By comparing the classification performance in above charts we notice the significant negative effect beta has for low values, beta<0.75. Beta regulates the contribution to the similarity score of the “dissimilarity” between two signal vectors. We conclude that the optimal weight of the beta parameter is beta=1.5 and the optimal value for K is 7. With these parameters, the error ratio was only 12.5%.

4.1.2 The K-Nearest Neighbor Algorithm

Another offline study that has been done is using KNN algorithm using Euclidean distance method to measure its offline performance. As mentioned earlier, KNN is one of the fastest machine learning algorithms because it is “lazy” and defers all calculations to the classification step. We have implemented a voting version for KNN and a weighted
version. In this section we present results for KNN with different K parameters, on the two datasets, in Tables 5, 6, and 7.

<table>
<thead>
<tr>
<th>K</th>
<th>No weighting</th>
<th>1- weighting</th>
<th>1/weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.344117647</td>
<td>0.344117647</td>
<td>0.344117647</td>
</tr>
<tr>
<td>3</td>
<td>0.294117647</td>
<td>0.291176471</td>
<td>0.291176471</td>
</tr>
<tr>
<td>5</td>
<td>0.255882353</td>
<td>0.320588235</td>
<td>0.320588235</td>
</tr>
<tr>
<td>7</td>
<td>0.282352941</td>
<td>0.335294118</td>
<td>0.335294118</td>
</tr>
<tr>
<td>9</td>
<td>0.282352941</td>
<td>0.308823529</td>
<td>0.308823529</td>
</tr>
</tbody>
</table>

Table 4. Classification error ratio for KNN variants for file Data 1 and k=1,3,5,7,9.

Chart 5. Classification error ratio for KNN variants for file Data 1 and k=1,3,5,7,9.

As shown in Table 4, the best value for K for KNN is K=5. As we see KNN is faster than K-Redpin, but the accuracy of K-Redpin is higher. Considering the small difference in evaluation time, that is in the order of hundreds of milliseconds, we can say that the K-Redpin performs better than KNN on all fronts.
<table>
<thead>
<tr>
<th>K</th>
<th>No weighting</th>
<th>1- weighting</th>
<th>1/weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.211666667</td>
<td>0.211666667</td>
<td>0.211666667</td>
</tr>
<tr>
<td>3</td>
<td>0.188333333</td>
<td>0.19</td>
<td>0.19</td>
</tr>
<tr>
<td>5</td>
<td>0.175</td>
<td>0.185</td>
<td>0.185</td>
</tr>
<tr>
<td>7</td>
<td>0.18</td>
<td>0.193333333</td>
<td>0.193333333</td>
</tr>
<tr>
<td>9</td>
<td>0.175</td>
<td>0.178333333</td>
<td>0.178333333</td>
</tr>
</tbody>
</table>

Table 5. Classification error ratio for KNN variants for file Data 2 and k=1,3,5,7,9.

Chart 6. Classification error ratio for KNN variants for file Data 1 and k=1,3,5,7,9.

Table 5, shows the performance of KNN on the larger data set 2. It is clear that non-weighted performs better in this case with k=5.
Table 6. Comparing classification error ratio for KNN between Data 1 and Data 2 for K=1,3,5,7,9

<table>
<thead>
<tr>
<th>K</th>
<th>Data 1</th>
<th>Data 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.344117647</td>
<td>0.211666667</td>
</tr>
<tr>
<td>3</td>
<td>0.294117647</td>
<td>0.188333333</td>
</tr>
<tr>
<td>5</td>
<td>0.255882353</td>
<td>0.175</td>
</tr>
<tr>
<td>7</td>
<td>0.282352941</td>
<td>0.18</td>
</tr>
<tr>
<td>9</td>
<td>0.282352941</td>
<td>0.175</td>
</tr>
</tbody>
</table>

Chart 7. Comparing classification error ratio simple KNN, for Data 1 and Data 2 for K=1,3,5,7,9.

According to Table 6 and Chart 7, we can see that the optimal value for K is 5. In general, by study of these experiments we can say that the best value for K in our case is K=5 for sure, and KNN is better than two weighted versions. Comparison between K-Redpin and KNN shows that K-Redpin performs better in matter of accuracy. We also notice that a larger training data set reduces the classification error by 8-10%.
4.1.3 Algorithms from the Weka Library

For making our study complete we have evaluated several algorithms from the Weka machine learning library to see how they perform with our data. In this section we will compare these algorithms based on their classification error ratio obtained on the two datasets.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Data Set 1</th>
<th>Data Set 2</th>
<th>Delta error:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bayes Net</td>
<td>0.2413</td>
<td>0.1416</td>
<td>0.0997</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>0.339</td>
<td>0.22</td>
<td>0.119</td>
</tr>
<tr>
<td>SMO</td>
<td>0.2212</td>
<td>0.1216</td>
<td>0.0996</td>
</tr>
<tr>
<td>IBK-1</td>
<td>0.3244</td>
<td>0.2166</td>
<td>0.1078</td>
</tr>
<tr>
<td>IBK-3</td>
<td>0.2844</td>
<td>0.1833</td>
<td>0.1011</td>
</tr>
<tr>
<td>IBK-5</td>
<td>0.2902</td>
<td>0.1816</td>
<td>0.1086</td>
</tr>
<tr>
<td>IBK-7</td>
<td>0.3017</td>
<td>0.1783</td>
<td>0.1234</td>
</tr>
<tr>
<td>K-Star-False</td>
<td>0.2988</td>
<td>0.205</td>
<td>0.0938</td>
</tr>
<tr>
<td>K-Star-True</td>
<td>0.2787</td>
<td>0.2</td>
<td>0.0787</td>
</tr>
<tr>
<td>Bagging-10</td>
<td>0.3045</td>
<td>0.19</td>
<td>0.1145</td>
</tr>
<tr>
<td>Bagging-100</td>
<td>0.2701</td>
<td>0.1566</td>
<td>0.1135</td>
</tr>
<tr>
<td>Random-Forest</td>
<td>0.2758</td>
<td>0.21</td>
<td>0.0658</td>
</tr>
<tr>
<td>KNN</td>
<td>0.25588</td>
<td>0.175</td>
<td>0.08088</td>
</tr>
<tr>
<td>K-Redpin</td>
<td>0.238235294</td>
<td>0.131666</td>
<td>0.106569294</td>
</tr>
</tbody>
</table>

Table 7. Offline classification error for several algorithms, measured with 10-fold cross-validation.

The Bayes Net algorithm implements Bayesian network inference. SMO is a support vector machine algorithm. IBK-K implements KNN with the indicated value for K. K-Star is an instance-based classifier that relies on a similarity score using an entropy-based distance function. Bagging-x is a bagging ensemble classifier using the fast decision tree classification algorithm.
As shown in Table 7 and Chart 8, there are three algorithms that perform better than KNN, and SMO beats K-Redpin by 1%. For our online experiment we have used the best classifier, SMO (with error ratio of 12.16%), which is a support vector machine algorithm. In the next section we will discuss our online experience.

Chart 8. Offline classification error ratio for several algorithms, measured with 10-fold cross-validation.

4.2 Online Evaluation

For the online experiment we have used the SMO classifier from Weka. For measuring the performance of the system in a real setting, we used the app to find locations at locations on the 4th floor of the East Engineering building. The getService web service has saved the ground truth data (actual location) to a database table and we could then determine the prediction error ratio. For 597 queries, the error ratio was 49%.
When considering the two best scoring locations, the error ratio was 23.4%. In only 5.3% of tests was the actual location not identified by any of the four best scoring classification results. The lower classification accuracy can be explained by the facts that sampling was not performed at node locations and that the phone and user stance and orientation may have not been identical to those when the RSS samples were taken for data set 2. As discussed, the RSS is highly sensitive to user position and orientation on a floor due to signal absorption and multipath reflection.
CHAPTER 5: CONCLUSIONS

In this thesis we describe an architecture for indoor user localization using Wi-Fi RSS sampling that can be used by laptop and smartphone users equipped with IEEE 802.11 network cards. The client application runs on Android smartphones and the localization service is implemented as a web service. The software architecture is expandable to integrate with new location classification algorithms, as they become available. The client app allows users to contribute to improving the RSS fingerprint database. The localization performance was analyzed in an offline mode and online mode, with a user waking and testing the app throughout the building's 4th floor.

The online performance evaluation indicated a localization error in typical usage scenario of 46% considering the best score location only. This result is not sufficient for effective context-aware computing applications. For future work we plan to improve the system accuracy by developing more sophisticated localization algorithms that use the distribution of probable user positions in time to remove false positives from our results.

We will also investigate analytical method for multilateration using indoor signal propagation models.
In the long term the localization system will be expanded to providing indoor and outdoor navigation service to FAU students and visitors, spanning through multiple buildings and the campus outdoor space.
APPENDIX A

Send comments and questions to icardei@cse.fau.edu

Architecture Requirements

Client platform: Android >=2.3

Server OS: Linux

Webserver: Apache Tomcat

Service platform: Java EE Servlets

Database: MySQL

Useful References

- Android Java class android.net.wifi.WifiManager
- Android Java class android.net.wifi.ScanResult
- HTTP Status Code Definitions http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Wi-Fi Fingerprint Storage Service

This web service saves a set of Wi-Fi fingerprints captured on a smartphone to a SQL database.

- Service URL
/saveWiFiData

- Request String
?tok=<authentication code>&req=<JSON request>

<authentication code>: for now, use a short hardcoded password string to avoid attacks
The <JSON request> string has the following format representing a JSON object:

```
[  
  {'t':timestamp_ms_since_epoch, 'user':userName, 'building':building,  
   'floor':floornumber, 'loc':'location', 'pos':'position',  
   'machine': {'name':'name', 'brand':brand'},  
   'scanresults': [  
                   {'ap': {'bssid':BSSID, 'ssid':SSID}, 'f':frequency, 'level':RSSI_dBm}, .... 
               ],  
  },...,  
]
```

A JSON request is an array of scans objects. Each scan object encapsulates the results of a Wi-Fi scan using the Android WiFiManager.

Notation “...” above indicates 0 or more elements in a JSON array.

- **Request Parameters**

  t: type: long

  timestamp in ms since the epoch (1/1/1970, 0:00 GMT).

  This is a Java `long` number obtained using:

  ```java
  new Date().getTime()
  ```

  Example: '1347332465112' means Tue, 11 Sep 2012 03:01:05.112 GMT

  user: type: string

  id of the user who runs the scanning application. Must be enclosed between ' and '

  building: type: integer

  building floor number, starting with 1 for the ground floor.

  floor: type: integer

  61
loc: type: string

Symbolic code of the location where the scan occurred. Example: 'EE414'. Must be enclosed between ' ' and ' '.

pos: type: string

Format is 'x,y', where x and y are coordinates on the floor map. Must be enclosed between ' ' '.

scanresults: type: JSON array

This array holds objects describing the Wi-Fi scan results. It can have 1 or more elements of the following type:

bssid: type: string

Access point BSSID, i.e. MAC address. Example: '12:34:56:78:9A:BC'. This is the field android.net.wifi.ScanResult.BSSID. Must be enclosed between ' ' and ' '.

ssid: type: string

The network name (SSID). Example: 'fauwpa2' or 'fauguest'. This is the field android.net.wifi.ScanResult.SSID. Must be enclosed between ' ' and ' '.

f: type: int

The frequency in MHz of the channel over which the client is communicating with the access point. E.g. 2412. This is the field android.net.wifi.ScanResult.frequency.

dBm: type: double

This is the RSSI[dBm] (received power expressed in dBm). E.g. -86. This is the field android.net.wifi.ScanResult.level.

• **Reply Format**

**HTTP STATUS CODE:**

As described below, and complying with the standard at http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

**Reply message JSON format:**

{ 'code': errorcode, 'message': diagnostic_message }
where
  code: type: integer.
    0 indicates no errors
  diagnostic_message: type: string
    a string describing the result of the operation or the error

Reply for Successful Operation:

  HTTP CODE: 200 OK
  Reply message JSON format:
    {'code': 0, 'message':'OK'}

Reply for Failed Operation:

  HTTP CODE: 200 OK
  Reply message JSON format:
    {'code': error_code, 'message':'a descriptive message of the error that occurred'}

  code: integer >0, to be defined

Wi-Fi Fingerprint Query Service

This service:
  1. Receives query parameters in a JSON string
  2. Constructs a SQL SELECT string using the incoming parameters to filter Wi-Fi scan results
  3. Builds a reply in the requested format (JSON or CSV)
  4. Sends back reply string

  • Service URL
    /getWiFiData

  • Request String
    ?tok=<authentication code>&req=<JSON request>

    <authentication code>: for now, use a short hardcoded password string to avoid attacks.

    The <JSON request> string has the following format representing a JSON object:
    {

'format': replyFormat,
'starttime': timeStart, 'endtime': timeEnd,
'building': building, 'floor': floornumber, 'loc': location,
'bssid': BSSID, 'ssid': SSID, 'apid': AP_ID,
'user': username,
'fields': [field1, field12, ...],
'orderby': [field1, field12, ...]
}

• **Parameter Description:**

format: type: string

Format encoding for the returned data. Accepted values: 'CSV' and 'JSON'.

starttime: type: ms since the epoch

OPTIONAL. Return Wi-Fi data at least since this timestamp. If missing, return data from the beginning.

endtime: type: ms since the epoch

OPTIONAL. Return Wi-Fi data up to this timestamp. If missing, return data captured up to now.

If both starttime and endtime are missing, don't use capture time as a SELECT query filter.

building: type: string

OPTIONAL. Standard FAU building name. If missing, do not use building as a query filter.

group: type: integer

OPTIONAL. Floor number. Ground floor is number 1. If missing, do not use for query filter.

loc: type: string
OPTIONAL. If present in the request object, use the supplied node location value as a query filter.

bssid: type: string

OPTIONAL. If present, use the supplied BSSID as a query filter.

ssid: type: string

OPTIONAL. If present, use the supplied SSID as a query filter.

apid: type: integer

OPTIONAL. This is the AP ID key of the AP table. If present, use as query filter.

user: type: string

OPTIONAL. User name. Use as query filter if present.

fields: type: string array

OPTIONAL. Columns to include (with names from the DB schema) from the scans and scanresult tables. Data must be returned in the order given in the array. (Format the SELECT query string with the fields array.)

If this array is missing from the request, then return the following (field names don't match DB column names):

\[timestamp, scanID, userName, AP_ID, frequency, building, floor, location, position, level\]

orderby: type: string array

OPTIONAL. Order the returned data first by field1, then by field2, etc. if missing, order by:

\[scanID, AP_ID\]

• **Reply for Successful Operation:**

HTTP CODE: 200 OK

• **Format for Timestamps:** Timestamps are returned in MM/DD/YYYY HH:MM:SS format, where HH is military hour time, i.e. use hour 23 instead of 11PM.
• **CSV Data Format**

If the `format` request parameter is CSV, return the requested fields in the following format:

```
field1, field2, field3, field4
val1, val2, val3, val4
val1, val2, val3, val4
.....
```

The first line is the header line, with field names are DB column names. The data must include the fields required in the fields parameter in the JSON GET request.

The rows are ordered according to the 'orderby' request parameter.

The field separator is COMMA (,).

The line separator is NEWLINE (\n). Append a '\n' at the end of each line.

• **JSON Data Format**

If the `format` request parameter is JSON (or json), send back a reply in the following format:

```json
{
  'code': 0, 'message': 'OK',
  'header': [field1, field2, ...],
  'data': [
    [val1, val2, ...], ...
  ]
}
```

header:  type: array of string

Array with the fields requested by the user and returned here. It should be the same as the `fields` request parameter.

data: type: array of array of strings.

Data from the DB. NOTE that ALL values are returned as strings, including numeric values. This is because in JSON array elements must be of the same type.

Example:

```json
'data':[['11/20/2012 14:20:58', '10', '210', 'username', 'EE-96', '4', 'EE413', '11', '-47']]
```

• **Reply for Failed Operation:**
As in Section 0.

**Map Graph Storage Service:**

The purpose of this service is to create and store the location graph for indoor localization and navigation.

The location graph for a building or site is \( G = (V,E) \), where \( V = \{v| v \text{ is an indoor location } \} \) is the set of map nodes and \( E = \{(u,v)| u,v \in V \} \) is the set of edges representing segments connecting two locations that can be reached from one to another. For a building map, if \((u,v) \in V\) then \((v,u) \in V\).

Each building has its own map graph that spans all floors. This web service provides an operation for storing a map graph on a web server.

- **Service URL**
  `/saveMapData`

- **Request String**
  `?tok=<authentication code>&req=<JSON request>`

  `<authentication code>`: for now, use a short hardcoded password string to avoid attacks

  The `<JSON request>` string has the following format representing a JSON object:

  `[ …………………………………………]`

**Map Graph Query Service**

[Work in progress]

**Localization Query Service**

This web service receives from the client one or more WiFi scan result sets (fingerprints), computes the client's feasible locations and returns these back to the client.

For testing and performance evaluation, the client can also add to the request the user's location (entered manually on the user app). In that case, the server computes the client's feasible locations, stores the actual and predicted feasible locations (for future performance analysis), saves the sent Wi-Fi fingerprint to the scanresults tables, and finally returns the predicted locations to the client.
Service URL

/getLocation

Request String

The request string is very similar to the one used for the WiFi Fingerprint Storage Service described in Section ·.

It has in addition a request ID (string), an optional location (obtained from the app UI, string).

Format:

?tok=<authentication code>&req=<JSON request>

<authentication code>: for now, use a short hardcoded password string to avoid attacks

The <JSON request> string has the following format representing a JSON object:

{
    'reqid': request_id,
    'user': userName,
    'machine': {'name': name, 'brand': brand},
    'knownLocation': (OPTIONAL)
        {
            'building': building,
            'floor': floornoumber,
            'loc': 'location',
            'pos': 'position'
        }
    'scans':
        [
            
        ]
}
't': timestamp,
'scanresults':
[
    {'ap': {'bssid': BSSID, 'ssid': SSID}, 'f': frequency, 'level': RSSI_dBm},
    ...
]
],
...
]
}

Notation “…” above indicates 0 or more elements in a JSON array.

- **Request Parameters**

  'reqid': request_id, type string

  a unique string to identify this localization request. It will be returned to the client to prevent confusion when multiple requests from the same client are outstanding (e.g. when network is slow)

  user: type: string

  id of the user who runs the scanning application. Must be enclosed between ' and '.

  'machine': JSON object that identifies the user's platform

    'name': model name (string, example “iPhone 1")

    'brand': brand or maker (string, example “Apple")

    'knownLocation': type: JSON object
OPTIONAL field that indicates the user's current location

'building': building name (string)

'floor': int

'loc': location ID (string)

'pos': 'x, y' coordinate position on map, if available

'scans': type JSON array of objects of the following type:

t: type: long

timestamp in ms since the epoch (1/1/1970, 0:00 GMT).

This is a Java long number obtained using:

    new Date().getTime()

Example: '1347332465112' means Tue, 11 Sep 2012 03:01:05.112 GMT

'scanresults': type: JSON array

This array holds objects describing the WiFi scan results. It can have 1 or more elements

of the following type:

'bssid': type: string

Access point BSSID, i.e. MAC address. Example: '12:34:56:78:9A:BC'. This is

the field android.net.wifi.ScanResult.BSSID. Must be enclosed

between ' and '.

'ssid': type: string

The network name (SSID). Example: 'fauwpa2' or 'fauguest'. This is

the field android.net.wifi.ScanResult.SSID. Must be enclosed between ' and '.

'f': type: int
The frequency in MHz of the channel over which the client is communicating with the access point. E.g. 2412. This is the field android.net.wifi.ScanResult.frequency.

'level': type: double

This is the RSSI[dBm] (received power expressed in dBm). E.g. -86. This is the field android.net.wifi.ScanResult.level.

- **Reply Format**

HTTP STATUS CODE:

as described below, and complying with the standard at http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Reply message JSON format:

```json
{
  'reqId': request_id,
  'code': errorcode,
  'message': diagnostic_message,
  'locations':
    [
      {
        'building': building,
        'floor': floornumber,
        'loc': location_ID,
        'pos': position,
        'quality': prediction_quality
      },
      .......
    ]
}
```

where
code: type: integer.
0 indicates no errors

diagnostic_message: type: string
an a string describing the result of the operation or the error

'reqid': request_id, type: string
a a unique string to identify this localization request. It was sent by the client with the localization

'relocations': array of JSON objects describing feasible location predictions. Standard ML algorithms
(like KNN, Redpin, and weka algs. return just one class (i.e. one location prediction). The service must support new algorithms that return a list of predictions. The 'quality' parameter could be a probability metric or could indicate the estimated prediction maximum error.

'relocation' array element type:

'building': building (string)

'floor': floor number (number)

'loc': location ID (string) from XML map document

'pos': 'x,y' when available

'quality': prediction_quality (floating point number)

• Reply for Successful Operation

HTTP CODE: 200 OK
Reply message JSON format:

{'code': 0, 'message': 'OK', ........}

• Reply for Failed Operation

HTTP CODE: 200 OK
Reply message JSON format:

{'code': error_code, 'message': 'a descriptive message of the error that occurred'}

code: integer >0, to be defined

Sending Request
• **HTTP Request Methods**

POST and GET

• **Content Type**

Use request header: `Content-type: application/json`

• **User Agent Header Field**

The client's browser sends also the *user agent* description to the HTTP server. The HTTP header field is

User-Agent.

Example:

*User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0*

The service should store a code for the user agent for each measurement that identifies at least the hardware platform, such as iPhone4, Samsung Galaxy III, etc. We need to come with a more systematic approach for this in the DB schema to be able to identify the platform type (iPhone, Android), the manufacturer (e.g. Apple, HTC, Samsung), and the product name (Apple iPhone 5).
REFERENCES


