ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

BlueLocation II - A Localization Infrastructure for Bluetooth Enabled Mobile Devices

Semester Thesis (WS 05/06)
SA-2006-03

Oliver Keiser, Philipp Sommer

January 9, 2006

Advisors: Vincent Lenders, Bernhard Tellenbach
Supervisors: Dr. Martin May, Prof. Dr. Bernhard Plattner

Abstract

This thesis describes the design and implementation of a system to localize the position of a mobile user with a Bluetooth enabled device in an indoor environment. Positioned nodes with a Bluetooth radio periodically scan their environment for other devices and transfer this information to a central database unit. We propose algorithms to estimate the users position and compare them in terms of accuracy. Additionally, we propose a design and implement a Java application running on mobile devices to interact with the BlueLocation system. This application offers the possibility to obtain information about other users within proximity and get their current position.

Diese Semesterarbeit beschreibt das Design und die Implementierung eines Systems, dass den gegenwärtigen Aufenthaltsort eines mobilen Benutzers mit einem Bluetooth-fähigen Gerät feststellen und anzeigen kann. Fix installierte Knoten mit einem Bluetoothadapter scannen periodisch ihre Umgebung nach anderen Bluetoothgeräten und übermitteln diese Informationen an eine zentrale Datenbank. Wir entwerfen Algorithmen um den Aufenthaltsort eines Benutzers zu bestimmen und vergleichen diese auf Genauigkeit. Zusätzlich entwerfen und implementieren wir eine Java-Anwendung, die auf mobilen Geräten läuft und mit dem BlueLocation System interagiert. Diese Anwendung erlaubt einem Benutzer nach anderen BlueLocation Benutzern zu suchen und ihren Aufenthaltsort abzufragen.

Contents

1 Introduction **6**
- 1.1 Motivation . . . 6
- 1.2 Summary . . . 6

2 Related Work **8**
- 2.1 Atlantis: Location Based Services with Bluetooth . . . 8
- 2.2 Positioning with Bluetooth . . . 8
- 2.3 JSR 179 . . . 8

3 Design **9**
- 3.1 The BlueLocation System Architecture . . . 9
- 3.2 Fixed Nodes . . . 9
- 3.3 Central Unit . . . 10
- 3.4 Phone Application . . . 12
- 3.5 Localization Algorithms . . . 19

4 Implementation **26**
- 4.1 BlueFramework . . . 26
- 4.2 Third Party Contributions . . . 28
- 4.3 Phone Application . . . 29
- 4.4 Fixed Node Application . . . 34
- 4.5 Fixed Node Emulator . . . 35
- 4.6 Central Unit . . . 36
- 4.7 Protocols . . . 40

5 Evaluation **45**
- 5.1 Evaluation of the Bluetooth Inquiry Process . . . 45
- 5.2 Measurements with Mobile Phones and the Deployed Infrastructure 46
- 5.3 Fixed Node Limitations . . . 52
- 5.4 Mobile Device Limitations . . . 53

6 Outlook and Conclusion **54**
- 6.1 Further Extensions to the BlueLocation System . . . 54
- 6.2 Conclusion . . . 55

A Task Description **59**

B Project Timeline **63**

C Measurement Results **64**
C.1 Detections of Fixed Nodes by other Fixed nodes 65
C.2 Algorithm Results for the Measurement Positions 69

D Deliverables **85**
D.1 CD-ROM 85

E Installation Guide **86**
E.1 Installation of a Fixed Node 86
E.2 Installation of the Central Unit 87
E.3 Installation of the mobile device application 88
E.4 Installation of the Web Interface 88

F Used Software Tools **90**

G Used Mobile Devices **92**

Chapter 1

Introduction

1.1 Motivation

Most modern mobile phones feature Bluetooth for short range communication. If Bluetooth is enabled on a phone, it can be used to establish a connection with another device, e.g. for downloading an MP3 file to the phone or uploading the latest pictures to a PC. Besides, the phone can also be detected and identified by other Bluetooth enabled devices within range limits. If we build up an infrastructure of pre-positioned devices (fixed nodes), that are periodically scanning their proximity for other devices, we can evaluate which fixed nodes can recognize other mobile devices, and furthermore, can deduct their location. For a fixed node can only detect devices in a limited range (typically $\leq$ 10m) the position of mobile devices can be pinpointed to quite a small area.

1.2 Summary

1.2.1 Previous Work

BlueLocation I

BlueLocation I [2] was a semester thesis at TIK in SS05. The outcome of that thesis was that the general layout for the BlueLocation system was designed and part of the fixed node and central unit were implemented. The fixed node did a periodical scan for Bluetooth devices and reported it to the central unit, which stored the information in a database. A SSL encryption of the link between the fixed nodes and the central unit and some key management tools were also implemented.

1.2.2 Conducted Work

The present semester thesis resulted in the implementation of BlueLocation as a working system consisting of

- Mobile Devices
- Fixed Nodes

- Central Unit

The implementation has been successfully completed and the interaction between mobile devices and fixed nodes on one hand and fixed nodes and central unit on the other hand works stable.
The application for the mobile device was started from scratch. Although some of the application layer protocols were applied by BlueLocation I, they had to be completely redesigned and most of the Bluetooth protocols were newly introduced during this thesis.
At first, localization algorithms had to be designed and evaluated. All of them have been implemented successfully.
During implementation of the Bluetooth protocols and the localization algorithms most of the fixed node and central unit software was adapted to our needs and largely extended.
After the first version of the complete system was working, extensive tests were conducted. During the course of testing, the localization algorithms were further improved and fine tuned. On the mobile side, the need for additional features surfaced, so that the design was extended. One by one, these features were implemented and improved during further testing.
Finally, we added a web interface to handle administrative tasks on the database.

1.2.3 Results

The successfully implemented localization algorithms made it possible to deliver (based on the acquired data by the fixed nodes) an estimation for the position of a mobile device back to the requesting client. The quality of this decision depends on the position of the device and the used algorithm. The main problem is the lack of any information about the signal strength between the device and the fixed nodes. It could be shown empirically that algorithms with additional knowledge of properties of the environment produce the best results, but need more time for fine tuning. Additional research in this part of the thesis should probably lead to better results and may trigger new ideas for improved algorithms.

Chapter 2

Related Work

2.1 Atlantis: Location Based Services with Bluetooth

In this thesis [5] done at the Brown University a framework for Bluetooth based localization is developed and tested. A fixed Bluetooth sensor network is used to connect to a device. The distance from the base station is extrapolated from the value of the Received Signal Strength Indicator (RSSI). If data from at least 3 base stations are available a triangulation algorithm is used to determine the expected position of the device.

2.2 Positioning with Bluetooth

This paper [6] evaluates different methods for localizing a Bluetooth enabled device. With the direct method the client asks another Bluetooth device if it offers a Bluetooth positioning service and tries to obtain the position from there. With the indirect method the other Bluetooth devices act passively. The unique Bluetooth addresses of devices in the neighborhood are used as keys to lookup the position of the device in a database. If there are more than one positions possible then triangulation is used.

2.3 JSR 179

The Location API for J2ME [7] addresses location based services for mobile devices. It is possible to use several different technologies, like GPS, Cell-ID or Bluetooth, as data sources. This API is more a toolbox to manage location information and location based services. The implementation of the algorithms to calculate the location from the source data is not standardized and left to be implemented by the hardware manufacturer. There are very few mobile phones on the market that support this API.

Chapter 3

Design

3.1 The BlueLocation System Architecture

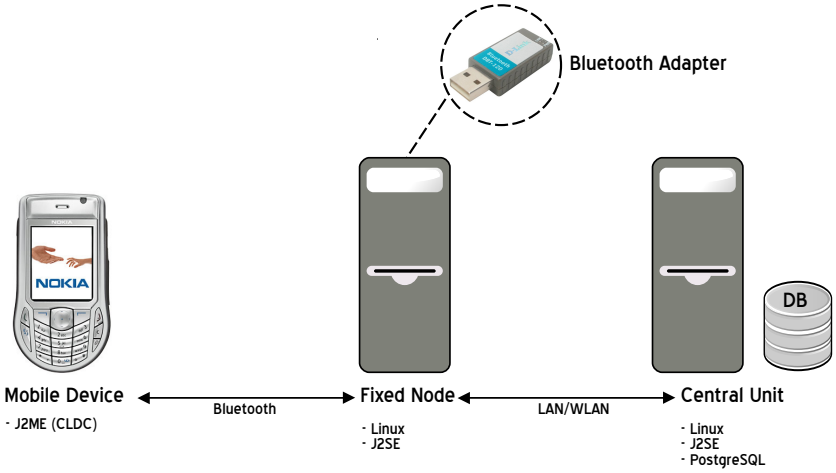


Figure 3.1: BlueLocation system overview

3.2 Fixed Nodes

Fixed nodes form a distributed system and are positioned at different places all over the area where the BlueLocation service is offered. They are equipped with a Bluetooth receiver and are connected to a backbone network (Ethernet/WLAN).
The fixed nodes are responsible for:

Figure 3.2: D-Link DBT-120 USB Bluetooth dongle

- periodically scanning their neighborhood for other Bluetooth devices and report them to the central unit.
- offering the BlueLocation service to the phones and forward request between the phones and the central unit.

The hardware for the fixed nodes should be as small as possible so that it could be placed on the wall or at the ceiling of a room. Alternatively the software should also be runnable on a standard personal computers when they are already available at the desired location.

3.3 Central Unit

The central unit must store all information about Bluetooth devices reported by the fixed nodes. It consists of two parts: the server software which offers a communication channel to the fixed nodes and the database system which is responsible for the data storage. This offers the possibility to access data not only through the server software, but also directly from the database with other tools. The server software has to handle requests from the phones and delivers information about users/devices based on the stored data. To be able to locate the current position of a user it has to use one or more localization algorithms.

3.3.1 Communication between Fixed Nodes and Central Unit

Each fixed node reports his list of devices found during inquiry to the central unit. For this purpose it sends data over the backbone network to the central unit. The communication between the fixed nodes and the central unit has to use a secured channel with authenticated communication partners.

3.3.2 Communication between Mobile Devices and Central Unit

Because the central unit has no direct possibility to communicate with mobile devices, all communication has to be routed over a fixed node. The central unit must handle the following requests:

- list all users who are currently active
- check if a user is currently online

- find the position of a user using a localization algorithm
- search the name of a user
- register a new user or change its properties
- handle permissions of a user

3.4 Phone Application

3.4.1 Overview

The mobile part of BlueLocation consists of a program running on a handheld device, e.g. a mobile phone. We want to limit text input from the user to as little as possible, instead we are using a graphical user interface (GUI), where the user can choose actions from lists or drop-down menus. By this approach we can reduce false input and facilitate handling by the user.

- The basic features of the mobile part consist of:
  - Connecting to a fixed node
  - Querying the central unit for a list of currently active users
  - Locating a specific active user
  - Buddylist
  - Searching for a user (active or inactive)
  - Disconnecting from a fixed node
  - Log

- Optional features are:
  - BlueMessage
  - Options to be set by the user
  - Detecting the loss of connection to a fixed node

3.4.2 Fixed Node Connection Setup

While a scan from a fixed node can find mobile phones, whether they have BlueLocation running or not, a mobile device cannot start querying the database before a connection to a fixed node is established. Therefore, the first thing BlueLocation has to do after start-up is connecting to a fixed node. There are basically two possibilities to achieve this: active and passive connection setup. Figures 3.3 and 3.4 visualize the algorithms described in the next paragraphs.

Active Connection Setup

In the active connection mode, the mobile device will first scan its environment for other Bluetooth devices. Then it will scan every device found for the BlueLocation UUID, which every fixed node has published.
From the list of found fixed nodes the user can choose one, e.g. the one closest. Optionally, BlueLocation can also automatically choose one.
Finally, a Bluetooth connection is established to the fixed node chosen before, which has a thread waiting for connections.

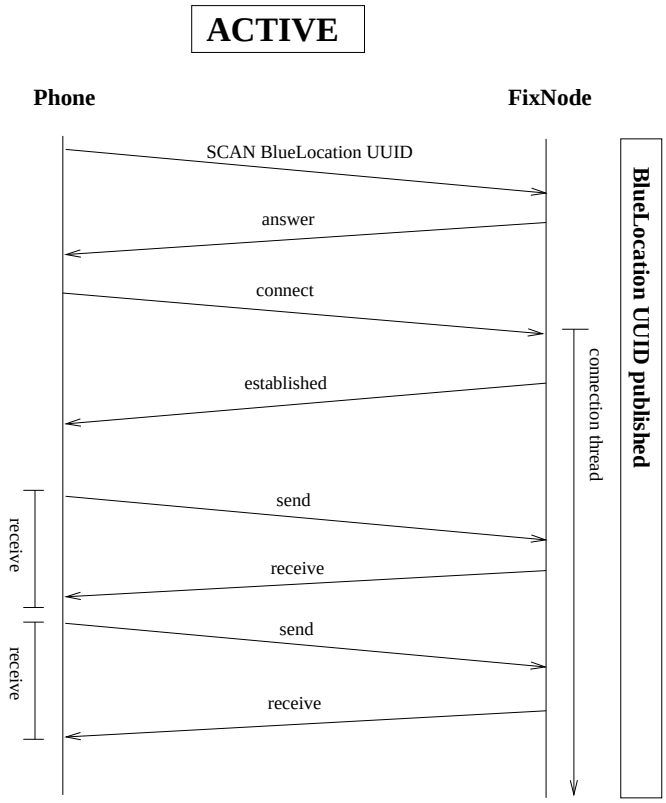


Figure 3.3: Active connection setup

Passive Connection Setup

While in passive mode, our mobile device will not scan for other devices itself, but publish the BlueLocation Connection Request UUID and then listen for a fixed node to connect. Since listening for incoming connections is a blocking task, it has to be run as a thread.
The basic task of the fixed node is to scan its environment for devices running BlueLocation. Now another thread has to be added, which will do a service scan for the Connection Request UUID and all devices found in the basic task. To enhance response time this should be done in a much shorter interval than the device scan. Since a service scan only takes a fraction of the time needed for a device scan, timing should not be a problem.
Thus, once a fixed node has found a mobile device requesting a connection, it will start the connection and the listening thread will accept it.
In both cases, we now have a connection from the mobile device to the fixed node, a two way byte pipe, and are ready to exchange messages.

Comparison of the two modes

The advantage of the active mode is its autonomy. A mobile device can start a connection setup at any time and doesn't have to wait for the next scan by a fixed node.

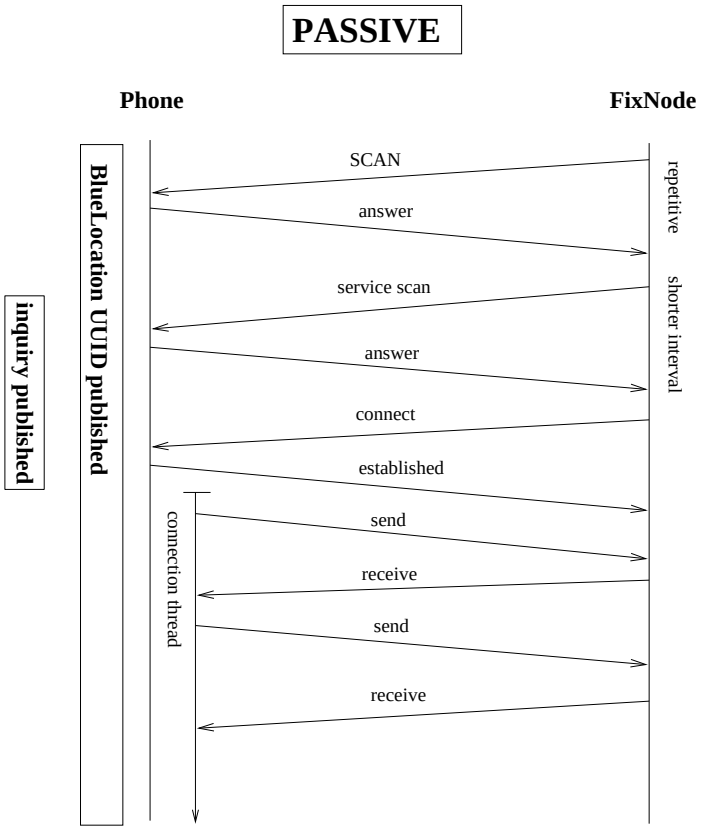


Figure 3.4: Passive connection setup

A problem can arise, when more than one mobile device within Bluetooth radio range starts to scan for fixed nodes at approximately the same time. Since they are sending radio signals on the same frequencies, interferences might occur, and therefore no device could be found. A related problem is the collision of a mobile device's scan with a scan from the fixed node which should be found by the mobile device. Since the mobile device's scan is only started once at start-up and again if it moves out of the fixed node's range, such collisions shouldn't occur too often. But even if they occur, it is likely that the user will initiate another scan a little bit later, which then has a good chance to be successful.

3.4.3 List of Active Users

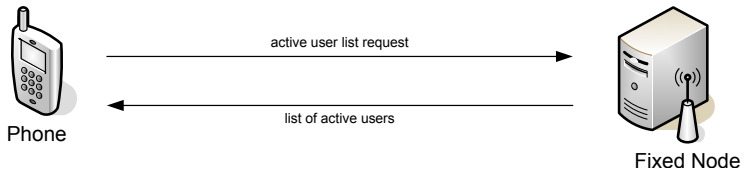


Figure 3.5: Requesting a list of active users

If we want to find out the location of another mobile device, we need to know its Bluetooth address (BTID) to uniquely identify the device. Since a BTID has the same format as a MAC address (6 byte, presented as hexadecimal numbers), it is not easily memorizable by the users. We wanted to spare BlueLocation users from asking other users for their BTID and so decided, that the mobile device should first get a list of currently active users from the central unit, then one can be selected to receive detailed information about its location.
Once a connection is established to a fixed node, the method for querying the central unit for the list becomes available. The request is sent as a string via the connection to the fixed node, which forwards it via Ethernet to the central unit. The central unit then generates a list of recently discovered BlueLocation devices and returns it to the fixed node, which then forwards it to the mobile device. From the mobile devices point of view, the communication works as depicted in Fig. 3.5. What is happening beyond the fixed node is not important at this scope.
This design works as long as the number of users stays small. On a large system with more than 10–20 active users, scrolling through the long list is inefficient. The design has to be changed in a way, that the first letters of name can be specified and the central unit returns a list of all users, whose name starts with these letters.

3.4.4 Localizing

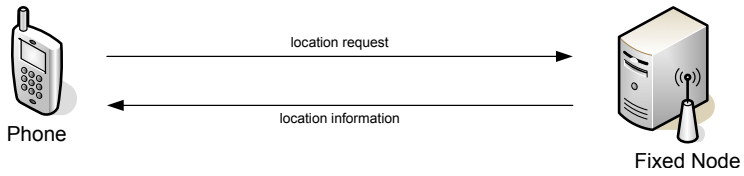


Figure 3.6: Requesting location information

After we have received a current list, the user can select one of the entries and request information about this user. A Location Request is then sent to the fixed node and forwarded to the central unit, from which an answer will be received, as shown in Fig. 3.6. The information received is the name of the user, its current location and the time in minutes since he has last been seen. (fixed nodes don't scan continuously)
Now the user of the mobile device has the option to add located users to his Buddylist.

3.4.5 Buddylist

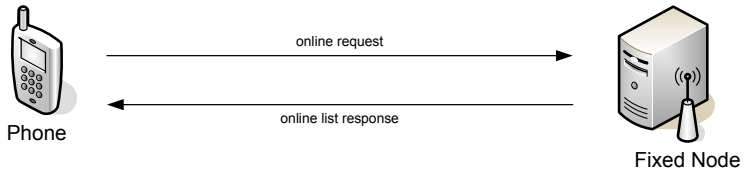


Figure 3.7: Checking which buddies are online

The goal of the Buddylist is to have information about your friends saved persistently on your mobile device. Once connected to a fixed node, instead of requesting a list of active users and manually looking for friends, the Buddylist can be called. The mobile device then checks whether the users in the Buddylist are online, according to Fig. 3.7, and presents this information on a single screen similar to an instant messenger.
An active user out of this list can be selected and information about its location is then requested from the central unit. The answer includes the current position and how long ago the user has last been seen.
As friendship don't last forever, the possibility to remove a user from the Buddylist must be available.

3.4.6 User Search

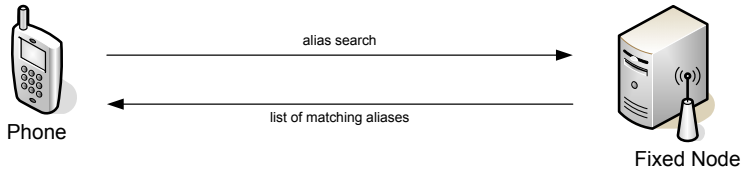


Figure 3.8: Searching for users

With the above features only active users can be added to the Buddylist. To enable the user to arrange his Buddylist without having to wait for all his friends to be online, searching for any user who has ever been discovered by the BlueLocation system must be possible. Similar to the design of the extended active user list request, a textfield will be presented, where the user can enter some letters. After receiving the answer from the central unit, all users whose name starts with those letters will be presented in a list and can be added to the Buddylist. (see Fig. 3.8)
A search with an empty textfield will display all users ever seen by BlueLocation. If too many names match the search, the user will be asked to further specify his search, until a reasonable number of results are returned.

3.4.7 Disconnecting from a Fixed Node

This command is straightforward, it closes the connection to the fixed node. The user can call this command, if he wants to connect to another fixed node, since only one connection with a fixed node can be active at a time. It is however not

necessary to disconnect before closing the BlueLocation application, the closing command takes care of this.

3.4.8 BlueMessage

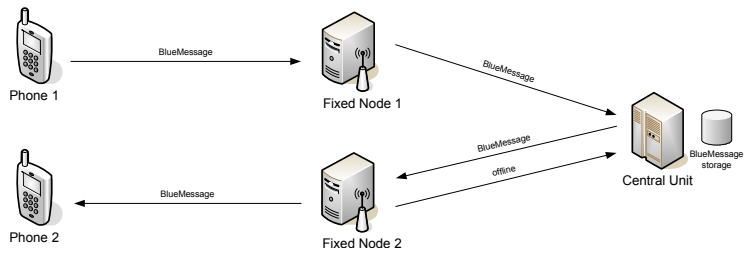


Figure 3.9: Sending a BlueMessage

It is not a basic requirement for the BlueLocation system to support peer-to-peer connections, i.e., from one mobile device to another, but nevertheless the backbone hardware could be put to sound use that way. The functionality is like SMS, the user can type a short text on his mobile device and send it to the fixed node, which forwards it via the central unit to the fixed node, where the receiver has an active connection. This fixed node will then forward the message via Bluetooth. If the receiver was not active at the time the message was sent, it could be stored centrally until the receiver had again an open connection to a fixed node. Figure 3.9 sketches the involved nodes.
The advantage of BlueMessage over SMS is that it is faster and of course free of charge.

3.4.9 Options

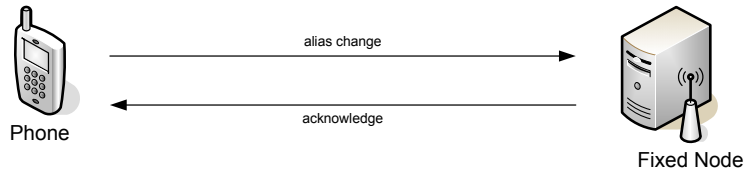


Figure 3.10: Updating an alias

In the options menu, the user will be able to change settings, which will not be asked interactively during operation.

- The alias will only be asked from the user the first time he starts BlueLocation and is then saved persistently. A later change of the alias can be done using the options menu, resulting in a data exchange as seen in Fig. 3.10.
- The standard connection mode is active. If a lot of users are active simultaneously, changing to passive mode might produce better results. The user can do so in the options menu.

3.4.10 Detection of Connection Loss

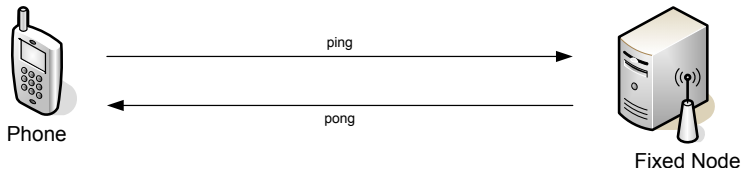


Figure 3.11: Testing the connection

Normally communication over the Bluetooth connection is stable, since it is implemented as an end-to-end-bytepipe on the application layer and transmission errors are handled on lower layers. However, the mobile device might move out of range or the fixed node might crash, resulting in a loss of connection.
Such a loss is only detected when data is exchanged, the send or receive routine terminates with an Exception or hangs waiting on data. Our design includes a thread which is periodically pinging the fixed node to verify that the connection is uninterrupted, see Fig. 3.11. Upon detection of connection loss, it informs the user with a pop-up and sets the global state of BlueLocation to disconnected.
Now the user can either manually try to reestablish a connection with a fixed node or the application automatically scans for other fixed nodes available and connects to the first one found. The user should then again be notified that a new connection exists.

3.5 Localization Algorithms

3.5.1 Introduction

The BlueLocation infrastructure in our laboratory setup consists of multiple standard PCs running Linux, equipped with an USB Bluetooth device. Each fixed node performs a Bluetooth inquiry periodically. The result of this inquiry is a list containing the Bluetooth addresses of all active Bluetooth devices in the fixed nodes neighborhood. The acquired data are submitted over a secured TCP connection to the central unit where the server software writes this information into the database. This information will be used to find out the position of a Bluetooth enabled device at a certain moment.

3.5.2 The Bluetooth Inquiry Process

During the inquiry process a Bluetooth device will try to discover other Bluetooth devices. The device enters the INQUIRY state and will choose 32 out of the 79 frequencies of the Bluetooth spectrum to form a hopping sequence. This selection depends on the local clock and the used inquiry access code. In our case the Generic Inquiry Access Code (GIAC) is used because we are looking for all types of discoverable devices. Inquiry packets are broadcasted periodically over these frequencies in the hopping sequence.
A device which is discoverable will enter periodically the INQUIRY SCAN state and listens for inquiry packets. If it detects an inquiry packet it will respond to the inquiring device.

3.5.3 Input Data

The results of the Bluetooth inquiry do not include an estimation of the signal strength (RSSI) or link quality between the fixed node and this device. This information is only available if a Bluetooth connection is successfully established with this device. It is in general not advisable to connect to each device in your neighborhood after every inquiry because a connection setup is time and energy consuming. This could be done if you need additional information such as the signal strength to perform a better localization. Therefore, we can only say if we have detected or not this device from a fixed node during Bluetooth inquiry. The elements of the observation vector $\vec{x}$ take on only binary values.

$$\vec{x}[i] = \begin{cases} 1 & \text{if device can be detected by fixed node i} \\ 0 & \text{else} \end{cases}$$

If we can detect a device from only one fixed node our decision is very simple. We assume that this device is located very close to this fixed node and estimate its location at the same position as the fixed node is.
If we can detect a mobile device from more than one fixed nodes during inquiry we have to use a localization algorithm to decide where the user is located.

Algorithm Operation Overview

Whenever a user of the BlueLocation service wants to know the location of another user, the phone application sends a localization request to the central

bluetooth address	fixed node A	fixed node B	fixed node C
00:13:46:05:A2:C7	found	not found	found
00:09:DD:10:46:7B	not found	found	not found

Table 3.1: Sample data set for two bluetooth devices

unit (as described in the protocol section). The software running on the central unit has to do the following steps before it can send back an answer to the user:

1. Find all entries for the chosen Bluetooth device and time in the database
2. Construct an observation vector from these data and pass it to the appropriate localization algorithm
3. Further processing of the result (if necessary)
4. Send back the result to the user

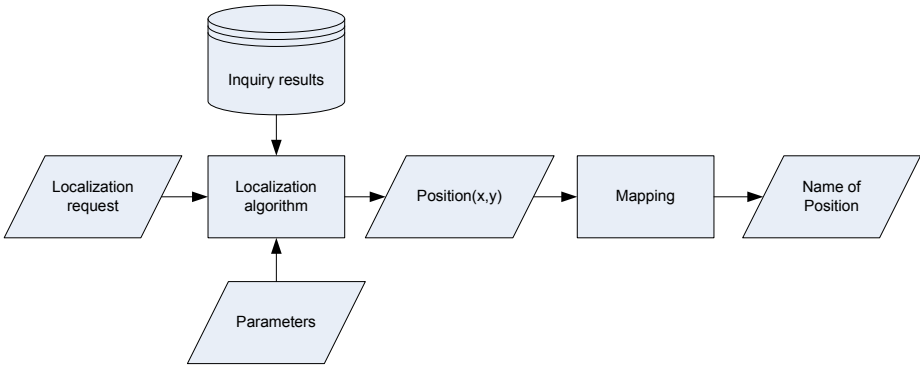


Figure 3.12: Algorithm operation overview

A localization algorithm can be fed with input data, the observation vector, and should return an estimation of the location of a device.
This estimate can be a name of a location such as the name of a fixed node or it could be a position in a coordinate system. In the latter case we have to map these coordinates to a human readable location.

3.5.4 Random Decision Algorithm

Maybe the simplest approach is to make a random decision based on the actual observation vector.
Choose one of the fixed nodes where the corresponding entry $\vec{x}[i]$ in the observation is 1. If we do so, we assume that the a-priori-probability of the device location is equally likely everywhere. With this algorithm we can only assign a device to the neighborhood of a fixed node.
The probability that we make a mistake in our estimation of a fixed node is given by

$$P[\hat{y} \neq y|y] = 1 - P[\hat{y} = y|y] = 1 - \frac{1}{n} \tag{3.1}$$

where n denotes the number of fixed nodes which received an answer from the device during Bluetooth inquiry.

3.5.5 Triangulation

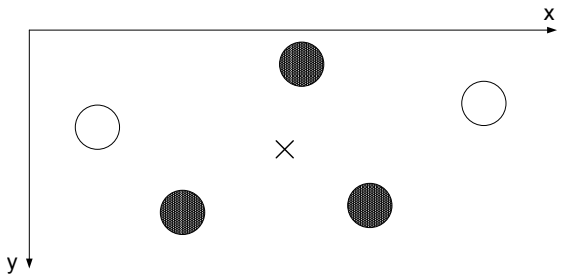


Figure 3.13: Triangulation example: circles indicate the position of the fixed nodes, a filled circle indicates that this fixed node can detect the device, the cross indicates the position estimated by the triangulation algorithm

Because we have no information available about the signal strength of a device, we cannot estimate the distance from the fixed node to the mobile device. Therefore, we try to localize the position of the device as the mean value of both x and y coordinates. We only take into consideration the coordinates of the fixed nodes which can detect the mobile device during inquiry.

$$x_{trian} = \frac{1}{n}\sum_{k=1}^{n} x_k \qquad y_{triang} = \frac{1}{n}\sum_{k=1}^{n} y_k$$

As result of the Triangulation algorithm we obtain a coordinate tuple (x_{triang}, y_{triang}) which gives us the triangulated position. This position can be at any location and is not necessarily located directly near a fixed node. We have to map this position to a name of a room or location which we can return to the user. This mapping is done using a simple look-up table (see example). The name of the point in the map with shortest Euclidean distance to the estimated position is returned.

```
#######################################
#  BlueLocation - Central Unit - Map  #
#######################################
# example: x,y,"name"
35,4,"ETZ G60.1"
37,9,"ETZ G99 (coffee break)"
37,15,"ETZ G97"
32,15,"ETZ G96"
28,15,"ETZ G95"
24,15,"ETZ G94"
21,15,"ETZ G93"
```

Table 3.2: File map.dat: sample of a look-up table

3.5.6 Decision based on Data of Reference Nodes

The former two algorithms needn't to know any information about the topology of the environment, but obviously the signal strength of a Bluetooth device is reduced if there is a wall or another obstacle in the line of sight.
Imagine the following situation:

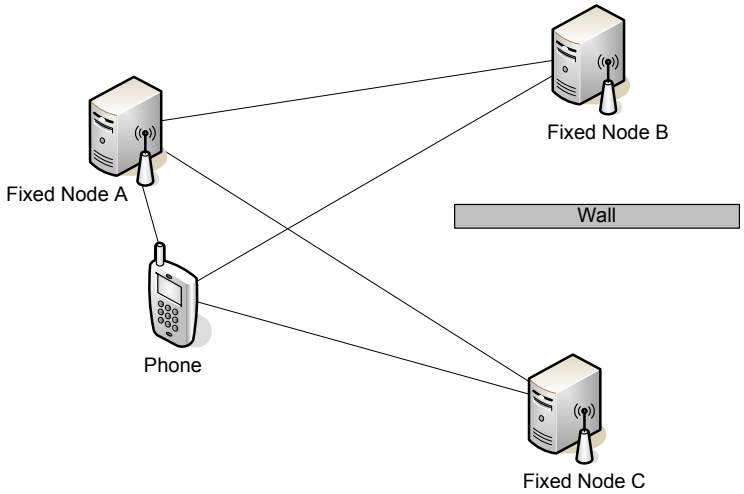


Figure 3.14: Example of the Reference Node Algorithm

We have three fixed nodes A, B and C and a mobile device (phone). A line indicates that the two devices can find each other during Bluetooth inquiry. The mobile device can be found by every fixed node. Additionally, a fixed node can find other fixed nodes in our infrastructure because they behave like every other discoverable Bluetooth device. A can detect B and C, B detects A but not C, C detects A but not B.

Fixed Node	A	B	C	Phone
A	-	√	√	√
B	√	-	-	√
C	√	-	-	√

Comparing the entries of the other fixed nodes with the mobile device's entries reveals that its properties are similar to fixed node A's and it must therefore be located near A.

3.5.7 Decision Based on Data of Reference Positions

A mobile device is put on a specific location and we take a look at the results of the Bluetooth inquiry over a long enough time period. This information is stored and can be used for further decisions. For every fixed node in the system a value between 0 and 1 is assigned. This value corresponds to the percentage of detection by the fixed node if the phone is positioned at the given position (see sample data below). Given an observation vector, we can compare these data with the reference data. We decide in favour of the reference point with the most similar data.

In this model, the specific topology of the environment is implicitly considered in the measured data. However, it is very time consuming to make all these reference measurements. It is necessary to take new measurements every time changes to the infrastructure are made (add/remove fixed node).

```
# reference data for position 2
(23,13)
"ETZ G60.1" 0
"ETZ G93" 1
"ETZ G94" 1
"ETZ G95" 0.95
"ETZ G96" 0.91
"ETZ G97" 0
"ETZ G99" 0.38
```

Table 3.3: File reference.dat: Reference data for position 2

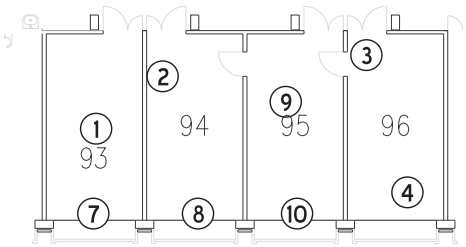


Figure 3.15: Map with reference positions

3.5.8 Decision Based on a Wall Graph

Bluetooth devices can typically communicate with each other if they are not more than a few meters apart. If there are walls on the line of sight between the two devices this distance will be reduced. It is therefore a good idea to include our knowledge about the topology of an area into a localization algorithm. We need to build a model of the walls and obstacles in our fixed node infrastructure and represent the topology as a weighted unidirectional graph. The nodes in our graph represent a location or room and the weight of the edges corresponds to the number of walls between these two positions.

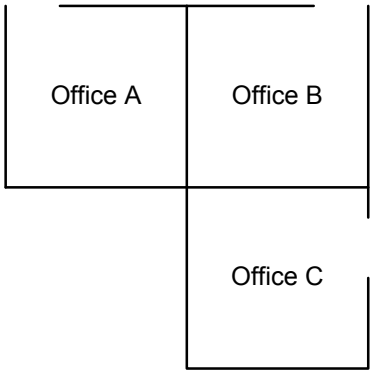


Figure 3.16: Map of the situation

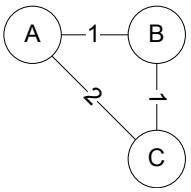


Figure 3.17: Graph representation

From previous measurements we know that a fixed node can find another mobile device during inquiry, if there are not more than one or two walls between them, so we set the variable *threshold* to one or two, respectively. Now the problem reduces to finding all adjacent nodes with a distance smaller or equal to the *threshold* value from those fixed nodes that can detect the device. We obtain a list of candidates for every fixed node which can detect the device. If we compare these different lists to the list of fixed nodes which can detect the device, we can decide for the fixed node whose list of candidates matches best.

Example of the Wall Algorithm

We look at the following situation:

- The phone is located at Office B
- The phone is detected by fixed nodes in Offices A, B and C

- The threshold value is set to one

For every fixed node where we can detect the phone (indicated by the green circles) we search for all adjacent fixed nodes with a distance smaller than the threshold value one (indicated by the blue circles). These fixed nodes should also be able to detect the device if it is positioned at the originating fixed node (green circle). For every fixed node in the graph which has assigned a blue circle and can detect the device we increase the counter variable of the originating fixed node. If we have done this procedure for every fixed node where the device is detected, we decide for the fixed node with the biggest value of the counter variable. In this example node A and C have both a counter variable of value one and node B has a counter variable of value two so the algorithm decides correctly for node B.

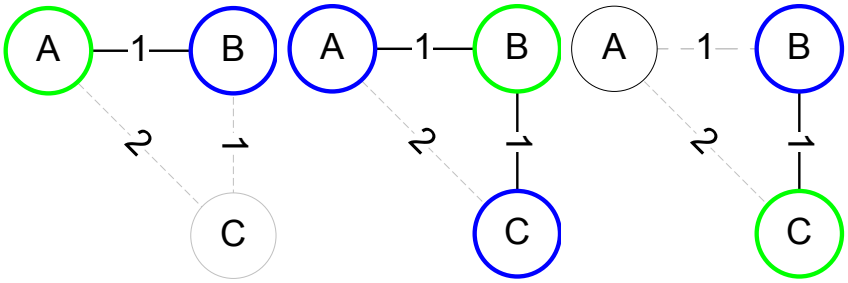


Figure 3.18: Example of the wall algorithm

3.5.9 Decision Based on the Frequency of Detection by Fixed Nodes

If you assume that the fixed node located in the same room as the mobile phone will detect the phone during almost every inquiry and another fixed node further away will detect the same phone only occasionally (because of stronger interferences), then the algorithm decides for the fixed node with the biggest frequency of detection. This algorithm only makes sense to use over a longer time period (more than one inquiry, e.g. 10 inquiries). For this algorithm looks at the history of detections, in the implementation it is called the 'History Algorithm'.
Given the sample data below this algorithm locates the phone near the fixed node "A".

Fixed node	Detections during inquiry
A	10/10
B	8/10
C	3/10
D	0/10

Table 3.4: Number of detections during ten inquiries

Chapter 4

Implementation

4.1 BlueFramework

4.1.1 Introduction

BlueFramework [3] was a Master thesis at TIK in SS05. It is a framework based on J2ME [17] and offers several modules often used for applications running on Bluetooth enabled phones. It offers a straightforward approach for building the user interface in MIDlets and for Bluetooth device discovery and communication.
The modules of this framework are actually designed for use in a system of mobile devices only. In our case, where we have mobile and fixed devices, some adjustments had to be made, since BlueFramework doesn't run on fixed devices, where J2SE [18] instead of J2ME is used.

For our project we used the following modules of BlueFramework:

- User Interface Module
- Discovery Module
- Communication Module
- Log Module

Further we used the `RecordStoreAccess` class to save data persistently in record stores.

4.1.2 User Interface Module

The User Interface Module consists of a tool, which takes a XML file as source and produces a Java class which implements elements of the library designated `javax.microedition.lcdui`. This tool really helped with the programming of the Graphical User Interface (GUI). The GUI layout can be described in XML in a hierarchical way. This XML design is then checked against an XML Schema for correctness to assure compatibility with the `lcdui` library.
With this tool, understanding the use of J2ME's GUI classes works in a learning-by-doing way, i.e. after analyzing the generated Java class against the XML

design, one can implement further GUI Screens directly in Java. This saves a lot of time, which would have been used, reading through all the JavaDoc of the `lcdui` library.

4.1.3 Discovery Module

The Discovery Module is responsible for finding Bluetooth devices with a certain service. Unfortunately, if applied according to the documentation and all methods are called from the same thread, a deadlock occurs. Also the Discovery Module is designed for use with mobile devices on both sides, i.e. that BlueFramework runs on the device discovering and the one to be discovered. In our case, where the other side is not a mobile device and therefore not running BlueFramework, some changes had to be implemented. The main adaption was to split the method calls into two threads to avoid locking. In the code of the class `BluetoothClient` we made some changes with semaphores, since it seemed to lead to racecondition.
With these changes, discovery of our fixed nodes worked stable but still slow. Sometimes several scans were necessary until a device was found. Since we were unsatisfied with these results, we stopped using the DiscoveryModule of BlueFramework and implemented our own solution. During the course of programming, we noticed, that the thread handler of the Nokia 6630 is not working in the expected manner. After calling the `wait` method on the main MIDlet thread, the CPU is not handed over to other threads in a runnable state and therefore timeouts occurred. When `wait` is called in its own thread however, the control switches between the threads and communication between them works. Our discovery implementation now works a lot faster and finds fixed nodes with the same stability as before.
Up to now, the result of discovery scans was a Vector of BTIDs. To improve user readability we added the readout of the remote devices friendly name to the scan. Now the Discovery Module returns a Vector of friendly names of the same length as the BTID Vector with corresponding entries. The implementation includes the new class `DeviceInfo` and changes in the classes `DiscoveryModule` and `BluetoothClient`.

4.1.4 Communication Module

Out of this module we only used the method to establish a connection. The existing sending and receiving methods were designed for use with other mobile devices and, as above, communication with a non BlueFramework partner was not planned. To fit our needs, we added methods to send and receive unprocessed datastreams, which work on a lower layer. We also put a semaphore on those methods to ensure that different threads don't interfere with each other while communicating. These changes concern the class `BluetoothCommunicator`.

4.1.5 Log Module

In the first phase of developing our BlueLocation MIDlet, we could test and debug it on the Sun J2ME Emulator [11]. For debug output, we could simply use the `System.out.print` routine.

As soon as we proceeded and were implementing features which included interaction with a fixed node, the emulator wouldn't do its job, since we needed a live communication partner. Unfortunately, the design of mobile phones doesn't offer a console, so debug output is a tricky thing. Here we could count on the Log Module which simplifies the task of writing information to persistent memory. So even after crashes we could see where the program stalled and remove these bugs.
What we are missing is the possibility to clear all entries from the log, since scrolling through a long list of text gets inefficient.
Finally we did a minor change to enhance readability on the screen.

Figure 4.1: The Log Screen with some entries

4.1.6 Record Store

Record Stores are an easy way to save data persistently to the mobile devices memory, without having to mess with files and filesystems. The mobile device needs to have the PDA Profile for J2ME (JSR 75) implemented to support Record Stores, which are accessed with the `javax.microedition.rms` package. The `RecordStoreAccess` class of BlueFramework facilitates the work with Record Stores even further, as it takes care of creating, opening and closing. Up to now BlueFramework only supported writing information to and reading from Record Stores. We added a method to also remove an entry from a Record Store.

4.2 Third Party Contributions

4.2.1 StringTokenizer

For some reason J2ME neither includes the method `matches` nor `split` usually found in the `java.lang.String` class. We needed some method to parse the String received from the central unit as reply to our request and split it into smaller Strings at the delimiter characters for further processing. The StringTokenizer class of the OstermillerUtils package (`com.Ostermiller.utils`) offered exactly the functionality we were looking for.

4.3 Phone Application

For our implementation of BlueLocation to work on a mobile device, the following specifications are required:

- CLDC 1.1 or 1.0
- MIDP 2.0
- Bluetooth for J2ME API(JSR 82)
- PDA Profile for J2ME API(JSR 75)

For a list of phones check [21].

4.3.1 Main Menu

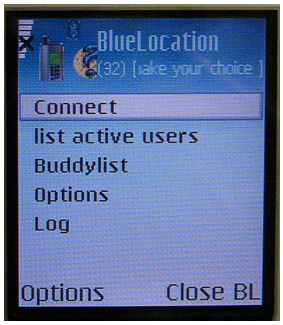


Figure 4.2: Main menu of BlueLocation

After starting BlueLocation, the main menu is displayed. (Fig. 4.2) From here, the user can access all basic features of BlueLocation. Right now only `Connect`, `Options` and `Log` are active. To make the other commands available, the user has to connect to a fixed node first.

4.3.2 Connect

If fixed nodes have been found on previous inquiries, they are now loaded from cache, the device discovery is omitted. By executing the command `refresh` an inquiry scan can be started manually. We chosed this workaround, because device discovery may take a long time and doesn't always find all fixed nodes. This problem is further explained in Sec. 5.4. Another problem was, that our device was delayed in the inquiry process by the many BTNodes, that were active all the time.
Our final implementation doesn't use the DiscoveryModule of BlueFramework anymore due to the reasons explained in Sec. 4.1.3. Our own implementation uses two threads, which work together on the same semaphore. The control thread starts the inquiry scan in another thread, waits on its results and then starts a service scan for every found device. The search thread is of class `DiscoveryAgent` of the `javax.bluetooth` package and handles lower level communication with the Bluetooth device. Upon successful search, our

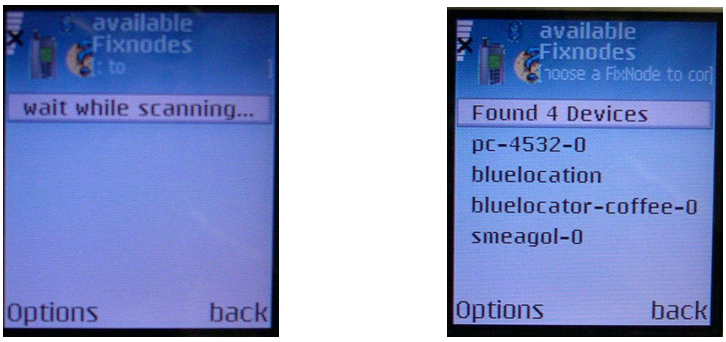


Figure 4.3: Screen during discovery and after fixed nodes have been found

`DiscoveryListener` is called, which is responsible for managing the results. After these threads have been processed, the results can be read out of two `Vectors`, one containing the BTIDs and the other one the friendly names of the found fixed nodes, with their indices corresponding.
Besides presenting a list of results on the display, the above vectors are also serialized and saved persistently using a record store.
The next step is to select one of the fixed nodes in the list and execute the connect command. The index of the selected entry is read and the content of the BTID vector at this index is forwarded to an instance of `BluetoothSettings`. This instance is then passed to another thread, which calls the `DiscoveryModule` of BlueFramework, which tries to establish a connection to the fixed node. If successful, the user is notified with a pop-up. The reason for using threads here is that a timer is started before attempting to connect. The method used for connection setup might lock if it is unsuccessful, but by using threads control is always returned to the main class after the timer runs up.
Once a connection to a fixed node is established, three more commands become available:

Figure 4.4: Pop-up confirming a connection is established

- List of Active Users
- Buddylist
- Changing alias

4.3.3 List of Active Users

This command executes a method, which sends a request string to the central unit and waits for a reply, according to our protocol (Sec. 4.7).
This reply is parsed for the delimiters ":", ";" and "," to split up the information. Two vectors are again used to store the userIDs and aliases. All the aliases are then listed on screen.

Figure 4.5: A list of all active users

4.3.4 Localizing

The method for receiving location information extracts the userID from the vector at the index of the selected alias in the list above. A request string with this userID is sent to the central unit and the reply again parsed for delimiters. A pop-up with the received information is then displayed.

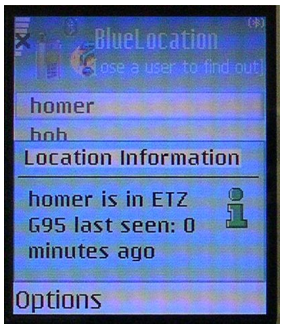


Figure 4.6: Pop-up informing about the location of a user

4.3.5 Buddylist

To ease handling with buddies a class `Buddy` was created. Every buddy is an instance of this class holding userID, alias and real name. Information about a users buddies are stored in two places: persistently in a record store as a serialized object and volatile in a vector of `Buddy` objects. The latter could be called the working copy.
Further there are five methods performing the following operations:

- Loading information from the record store to the `Buddy-Vector`
- Adding an entry to the record store
- Removing an entry from the record store
- Displaying the list of buddies on screen
  This is done by sending an online request with the userIDs of all buddies to the central unit. The answer is parsed as usual to get the status of all buddies. For every buddy the alias is written to the screen accompanied with either the online or the offline picture.
- Presenting the current location
  This is done in the same fashion as localizing above.

Figure 4.7: Example of a Buddylist

4.3.6 User Search

The user search is implemented using a `Form` with a `TextField` where the search string can be typed. The request is sent over the bytestream to the central unit and the response is again parsed for aliases and userIDs, which are stored in a vector each. The aliases are then displayed in a list.
If an entry is selected to be added to the buddylist, the content of the vectors at the selected index is read and added to the record store.

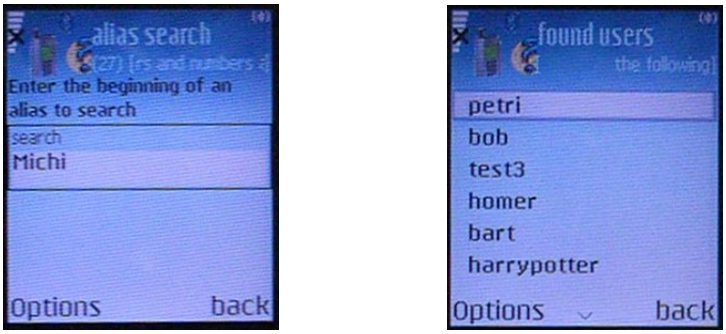


Figure 4.8: Search Form and search result List

4.3.7 Change of Alias

The only item under Options being implemented is the change of alias. A `Form` is displayed with the current alias and a `TextField` appears where a new alias can be entered. A request with the new alias is sent to the central unit. If it is accepted, the alias is also updated in the record store and a pop-up confirms the change. If the alias already exists or another error occurrs, the user is prompted to choose another alias. The old alias can be left unchanged if the `back` button is pressed.

Figure 4.9: Example of a Buddylist

4.3.8 Detection of Connection Loss

This feature is implemented as a thread, which is started as soon as the connection to the fixed node is established. On every cycle the thread sends a ping and waits for the pong from the central unit. As soon as the pong arrives, the thread is put to sleep for 10 seconds. After wake-up a new cycle is started.
If a `CommunicationException` occurs, the state in the main class is set to disconnected and a pop-up informs the user about the lost connection.

Figure 4.10: The pop-up displayed after a connection loss

4.4 Fixed Node Application

The fixed nodes are running the Debian GNU/Linux operating system [9]. The fixed node software is implemented in the Java programming language. The design of the fixed node application was taken from BlueLocation I, but large parts of the software were modified and extended to the new requirements.
After starting up the program the `FixedNode` Class creates two different threads called `DeviceDiscoveryAgent` and `BluetoothListener`.

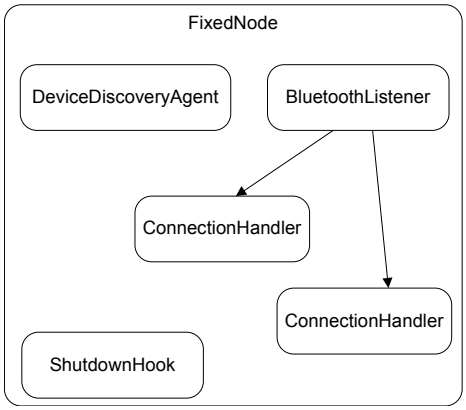


Figure 4.11: Different threads and classes in the fixed node

4.4.1 DeviceDiscoveryAgent

The `DeviceDiscoveryAgent` thread will periodically start a Bluetooth inquiry process and sleeps for the rest of the time. The list of devices found during inquiry is passed back to the main program from where it will be sent to the central unit.

4.4.2 BluetoothListener

The `BluetoothListener` thread publishes the BlueLocation service. This service uses the Bluetooth Serial Port Profile (btspp) for communication with the phones. The Thread blocks until a connection from a phone is established. Then a new `ConnectionHandler` thread is started for handling the connection with the phone. This ConnectionHandler communicates over Bluetooth with the mobile device on the one side and over a secured TCP connection with the central unit on the other side. The first string submitted to the central unit after the connection setup is the Bluetooth address of the phone. It will be stored in the central unit and assigned with this connection.

4.4.3 ShutdownHook

This thread is registered within the Java virtual machine(JVM) as a ShutdownHook. It will be called before shutdown of the JVM and is our last chance to do some cleanup. Unfortunately, our published BlueLocation Bluetooth service is

not removed automatically from the Bluetooth stack and, therefore, this service remains advertised to other devices. Thus, within our ShutdownHook we have to call the `shutdown()` method of the BluetoothListener class which will remove the service record of the BlueLocation service from the Bluetooth stack.

4.5 Fixed Node Emulator

Because debugging the communication process or protocols on the phone device is rather complicated we developed a simple emulator script for testing the functionality of the fixed node and the central unit. Running the `runEmulator` script in the fixed nodes directory on the computer passes a request to the fixed node program as it would come directly from the phone over a Bluetooth connection. The emulator validates the request and sends it to the central unit. The reply from the central unit is printed directly to the console. Program settings such as the server IP or the Bluetooth address of the (emulated) phone can be set in the `runEmulator` shell script.

```
Compilation: fixedNode$ ./compileEmulator
Normal usage: fixedNode$ ./runEmulator <request>
```

4.6 Central Unit

4.6.1 Database

Database Connectivity

The BlueLocation system uses the PostgreSQL [15] database system for data storage. The connection between the central unit software and the database is done through the PostgreSQL Java Database Connectivity (JDBC) [16] Driver. The database driver uses a TCP connection to the database so it would also be possible to run the central unit and the database on different physical systems or even to run multiple central units.

Database Layout

The `bluelocation` database must store all data collected by the fixed nodes and all information available about the users of the system. It contains the following four tables. The SQL script to generate the database can be found on the CD.

table	description
fixednodes	information about fixed nodes
inquiries	information about inquiries
deviceslocations	devices found during inquiries
lastseen	devices found during inquiries (cached for 30 minutes)
users	information about BlueLocation users
permissions	user permissions (granter/grantee)

Table 4.1: List of database tables

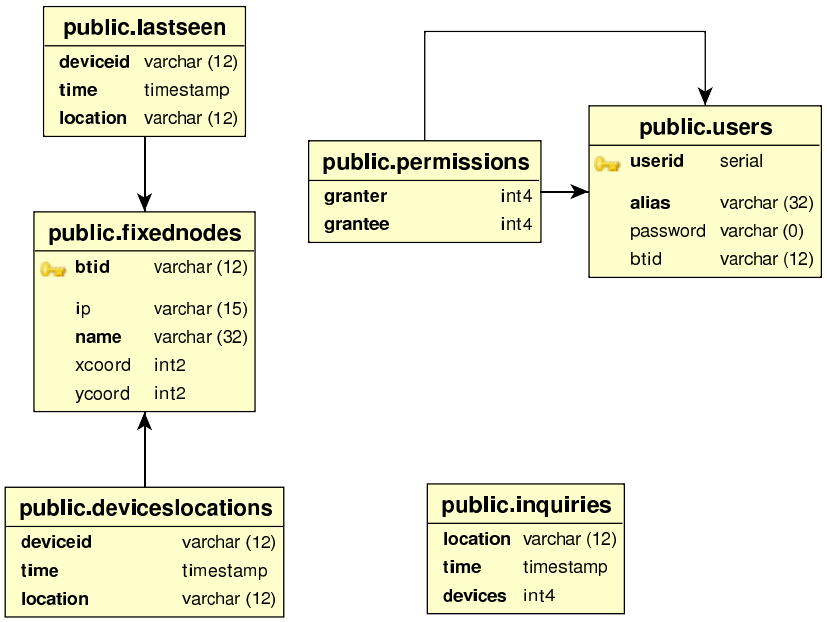


Figure 4.12: Database structure

4.6.2 Central Unit Software

Server Class

The central unit software running in Java is started by the `./runServer` shell script. The database password must be passed to the script as the first command line argument. The `Server` class is started by the script and initializes first the database driver and the SSL socket on port 4223. Then it blocks in a `while` loop until there is a new incoming connection from a fixed node. A new instance of `ConnectionHandler` is created and from now on responsible for the processing of this connection.

ConnectionHandler Thread

The ConnectionHandler uses a `BufferedReader` and `BufferedWriter` to read and write data from/to the fixed node. It obtains a new `RequestHandler` instance which will handle all requests associated with this connection. It waits in a loop for further requests from the fixed node until the connection has been closed.

RequestHandler Class

The RequestHandler is responsible for handling all requests as described in the protocol section. When the RequestHandler Class is constructed a new database connection is obtained and used during all requests associated with this instance. The connection will be closed when the `cleanup()` method is called. This method is called by the ConnectionHandler when the connection has been closed.

The method `doRequest(String request)` takes a request String as argument and returns the reply to this request. This method checks first if the request is valid and calls the appropriate method which can handle this type of request.

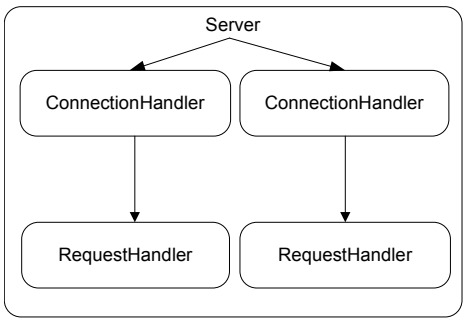


Figure 4.13: Different threads and classes in the central unit

Localization Algorithms

Each localization algorithm is implemented in a separate Java class file in the `localization.algorithm` package.

Random Algorithm	RandomAlgorithm.java
Triangulation	TriangulationAlgorithm.java
Reference Node Algorithm	ReferenceNodeAlgorithm.java
Reference Point Algorithm	ReferencePointAlgorithm.java
Wall Algorithm	WallAlgorithm.java
History Algorithm	HistoryAlgorithm.java

Table 4.2: Java classes for the localization algorithms

Additionally, some classes and methods are used for handling the required data structures. They are part of the `localization` package. At the moment only the Triangulation algorithm is used to perform a localization in the current implementation of the central unit. Please refer for further details of the implementation to the JavaDoc on the CD.

Statistic Tool

The Statistic Tool is used to generate the graphs for the measurements. You can give a list of the different positions where you placed the phone during the measurement and the program will write the results in a text file. A linux shell script (plot.sh) generates the graphs in the eps and pdf format using the gnuplot tool.

4.7 Protocols

4.7.1 Overview

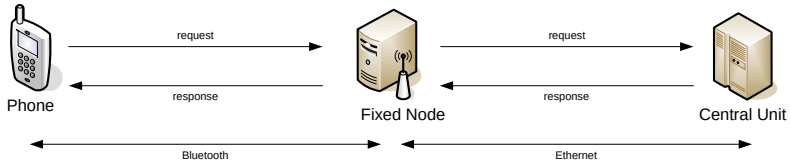


Figure 4.14: Communication within the BlueLocation system

The BlueLocation system follows a client/server model. The phone acts as the client and sends requests to the central unit (server). The reply to this request is sent back to the phone. Direct communication between the phone and the central unit is not possible. A fixed node has to act as a gateway. Communication between the phone and a fixed node is done over a Bluetooth connection (btspp), and between the fixed node and the central unit over a TCP (SSL) connection.
The phone sends a `request` String to the fixed node. The fixed node sends the request to the central unit. The central unit processes the request and sends the reply back to the fixed node and from there it goes to the phone.
The communication between the phone and the fixed node and between the fixed node and the central unit is handled over a dedicated Bluetooth or TCP connection. When the phone connects to the fixed node also the connection between the fixed node and the central unit is set up. The fixed nodes sends first the Bluetooth address of the phone to the central unit where this address is assigned to this connection for further reference.

4.7.2 Type of Replies

Each reply from the central unit starts either with ”ok” or ”error” to indicate if the request could be handled successfully or not. The actual reply starts after these two keywords and the additional colon character ”:”.

4.7.3 Communication between Fixed Node and Central Unit

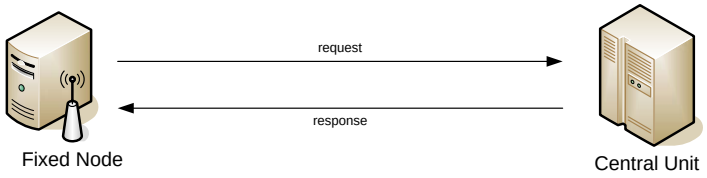


Figure 4.15: Communication fixed node–central unit

Report devices found during Bluetooth inquiry

Each fixed node periodically performs a Bluetooth inquiry. The list with the devices found during the last inquiry is sent to the Central Unit. If no devices are found an empty list will be sent. This is to inform the central unit that this fixed node is still active and scanning.

```
request: locationupdate?[deviceBTID;deviceBTID;deviceBTID...]
reply: ok
```

4.7.4 Communication between Phone and Central Unit

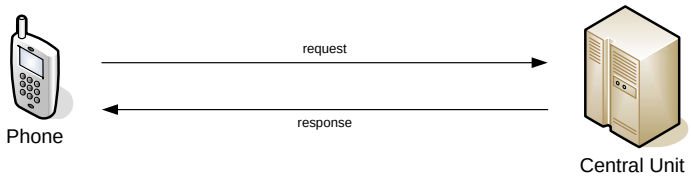


Figure 4.16: Communiation phone–central unit

Ping request

The `ping?` request is used to test if the connection between the phone and the BlueLocation system is still valid. The central unit sends the `pong` reply back.

```
request: ping?
reply: pong
```

List with currently active BlueLocation users

The `listactive?` request is used to obtain a list with all users of the BlueLocation system currently active. Pairs of userid and alias are sent back to the phone. An error message is returned in case there are no users present.

```
request: listactive?
response: ok:userid,alias;userid,alias...
or
response: error:no users active
```

Localization of a user

The `locate?` request is used to locate a user. This will start the localization algorithm on the central unit. A string with a human-readable name of the current position of the user is sent back. The time parameter is the time in minutes when the user was last seen by the system.

```
request: locate?userid
response: ok:location,time
or
response: error:notfound
```

Search for users registered with BlueLocation

The `searchuser?` request is used to search for users who have registered with the BlueLocation service (have an alias). The central unit searches the database for all aliases starting with the submitted `name` argument. One or more results are returned. If nothing is found an error message is returned. If the `name` argument is the empty string then all users in the database are returned.

```
request: searchuser?name
response: ok:userid,alias;userid,alias...
or
response: error:no results
```

Register a new user/phone

The `setname?` request registers a new alias for the sending device in the BlueLocation system.

```
request: setname?alias
response: ok:
or
response: error:
```

Check if a user is currently online

The buddylist functionality of the phone application requires information about the current status of a BlueLocation user. Each user can be in two different status: online or offline. Users are online if their phones are detected by an inquiring fixed node during the last few minutes.
The phone sends a list of userids to the central unit. The status of each user is checked in the database and the reply is returned. The reply is a list of ones and zeros. One indicates that the user is currently online, zero offline. The reply is in the same order as the userids in the request.

```
request: online?userid;userid;userid..
response: ok:1;1;0...
or
response: error:
```

Grant/revoke permission to/from another user

The `allow?` request updates the `permissions` table in the database. Permission is granted to the user with the specified userid.
Permission management is not implemented in the current version.

```
request: allow?userid
response: ok:
```
or
```
response: error:
```

The `revoke?` request updates the `permissions` table in the database. Permission is revoked from the user with the specified userid.

```
request: revoke?userid
response: ok:
```
or
```
response: error:
```

4.7.5 The BlueLocation Web Interface

To offer a web interface to the BlueLocation system was not included in our design. It is a last minute addition and is just a really simple possibility to interact with the system using a browser instead of the phone. The web interface is implemented in PHP [20] and runs on the Apache2 webserver [19] at the central unit. The main purpose is the ability to show a list of registered users, to add a new user to the system or to remove one. Additionally, all currently active users are visible and it is possible to send a request to the central unit as it would come directly from the phone. Other functions could be easily implemented if needed.

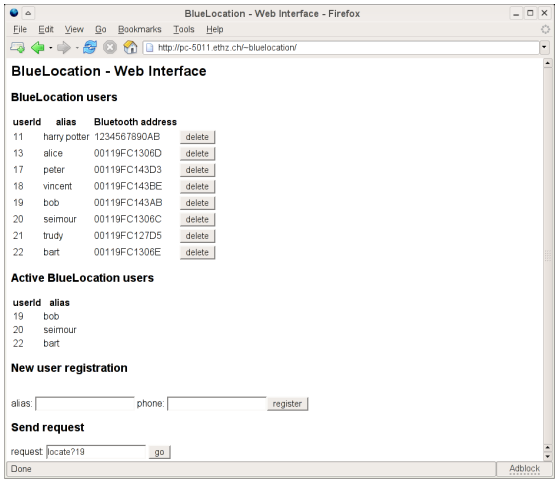


Figure 4.17: Screenshot of the BlueLocation web interface

Chapter 5

Evaluation

5.1 Evaluation of the Bluetooth Inquiry Process

5.1.1 Simultaneous Inquiries by Different Fixed Nodes

We wanted to know if it leads to any problems if more than one fixed node is performing a Bluetooth inquiry at the same time. Therefore, we conducted an experiment with the following setup:

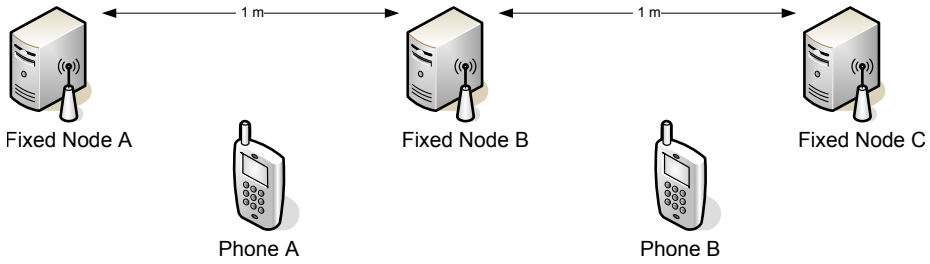


Figure 5.1: Inquiry test setup

There are three fixed nodes and two mobile phones in our test setup. Each fixed node runs on a dedicated computer with all clocks synchronized by a time server. All three fixed nodes perform simultaneous Bluetooth inquiries. The phones have Bluetooth activated and are set to get discovered by other devices.

	fixed node		
	A	B	C
inquiries	34	34	34
detected phone A	32	31	34
detected phone B	34	27	34

Table 5.1: Results of simultaneous inquiries

The result of this experiment showed that it is in most cases possible for the fixed nodes to receive an answer from the phones. However, the fixed nodes did not answer to other fixed nodes while they are inquiring. Therefore, two different fixed nodes should not start inquiry at the exactly same moment if

they should be able to discover each other as it is required for the reference node algorithm.

5.2 Measurements with Mobile Phones and the Deployed Infrastructure

5.2.1 Fixed Nodes Infrastructure

Our test infrastructure consists of seven fixed nodes running on standard desktop PCs. We have placed a fixed node inside every room and one on the floor (see Fig. 5.2).

coordinates	name	Bluetooth address	host	OS
(20,13)	ETZ G93	00134605A2D2	pc-3528	Linux 2.4.27
(23,12)	ETZ G94	00134605A2D4	pc-3640	Linux 2.4.27
(27,12)	ETZ G95	0020E07B41E0	nb-4788	Linux 2.6.7
(32,14)	ETZ G96	00134605A2D3	pc-3255	Linux 2.4.27
(35,18)	ETZ G97	00134605A317	pc-4209	Linux 2.4.31
(37,9)	ETZ G99	00134605A2C7	pc-3296	Linux 2.4.27
(37,4)	ETZ G60.1	00134605A2D1	pc-3641	Linux 2.4.27

Table 5.2: List of fixed nodes

5.2.2 Measurement Procedure

We placed the phones (Nokia 6630) on each test position for one hour. The fixed nodes were performing a Bluetooth inquiry once per minute. We took care that the inquiries of the different fixed nodes did not overlap to much.
The evaluation of the algorithms was done using a Java program (StatisticTool.java) which splits the test period into intervals of one minute and analyzes the results of the different algorithms. For each test position the closest fixed node was assigned (shortest Euclidian distance). Thus, it was in some cases possible that the closest fixed node was not the fixed node in the same room but in an adjacent room. The triangulation and reference point algorithms returned a position which was also mapped to the closest fixed node. If both the closest fixed node for the test position and the algorithm result were equal, then the decision was counted as correct. The results of this evaluation were written into different files and plotted with `gnuplot`. It has to be stated that some algorithms (topology, history, wall, reference point) can decide for more than one fixed node simultaneously (e.g. when two fixed nodes are equally likely) and therefore the sum of the bars in the graph was sometimes more than one hundred percent.

position (coordinates)	nearest fixed node	date/time	device address
1 (22,15)	ETZ G93	23.12.05 09:00-10:00	00119FC143AB
2 (23,13)	ETZ G94	23.12.05 09:00-10:00	00119FC1306D
3 (30,12)	ETZ G96	23.12.05 09:00-10:00	00119FC127D5
4 (32,17)	ETZ G96	23.12.05 09:00-10:00	00119FC143D3
5 (37,14)	ETZ G97	23.12.05 09:00-10:00	00119FC143BE
6 (33,5)	ETZ G60.1	23.12.05 09:00-10:00	00119FC1306E
7 (21,18)	ETZ G93	23.12.05 10:15-11:15	00119FC143AB
8 (24,18)	ETZ G94	23.12.05 10:15-11:15	00119FC1306D
9 (28,14)	ETZ G95	23.12.05 10:15-11:15	00119FC127D5
10 (28,18)	ETZ G96	23.12.05 10:15-11:15	00119FC143D3
11 (36,18)	ETZ G97	23.12.05 10:15-11:15	00119FC143BE
12 (37,6)	ETZ G60.1	23.12.05 10:15-11:15	00119FC1306E
13 (20,7)	ETZ G94	27.12.05 10:00-11:00	00119FC1306E
14 (31,6)	ETZ G99	27.12.05 10:00-11:00	00119FC1306C
15 (36,8)	ETZ G60.1	27.12.05 10:00-11:00	00119FC143AB

Table 5.3: Measurements details

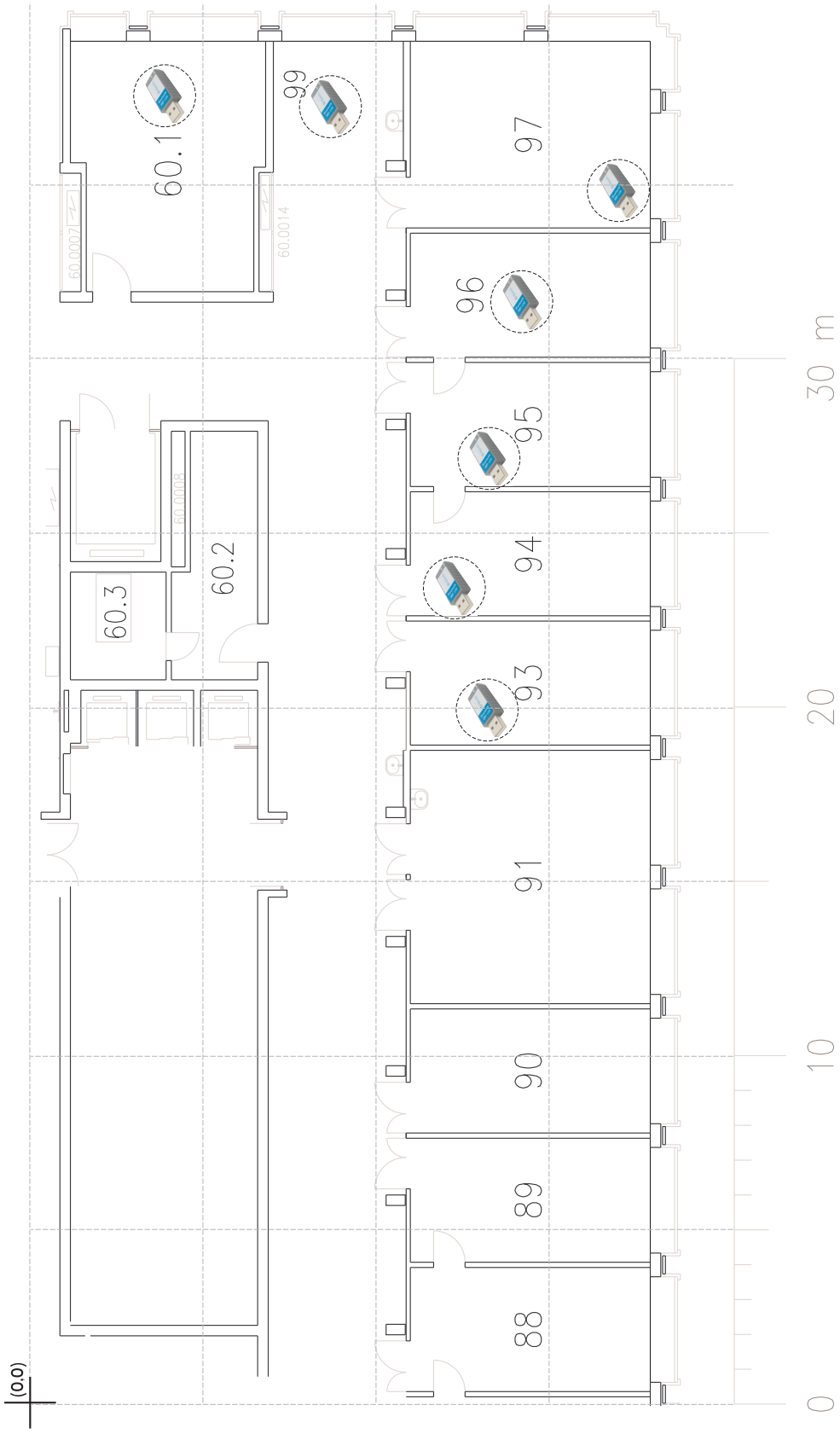


Figure 5.2: Position of the fixed nodes

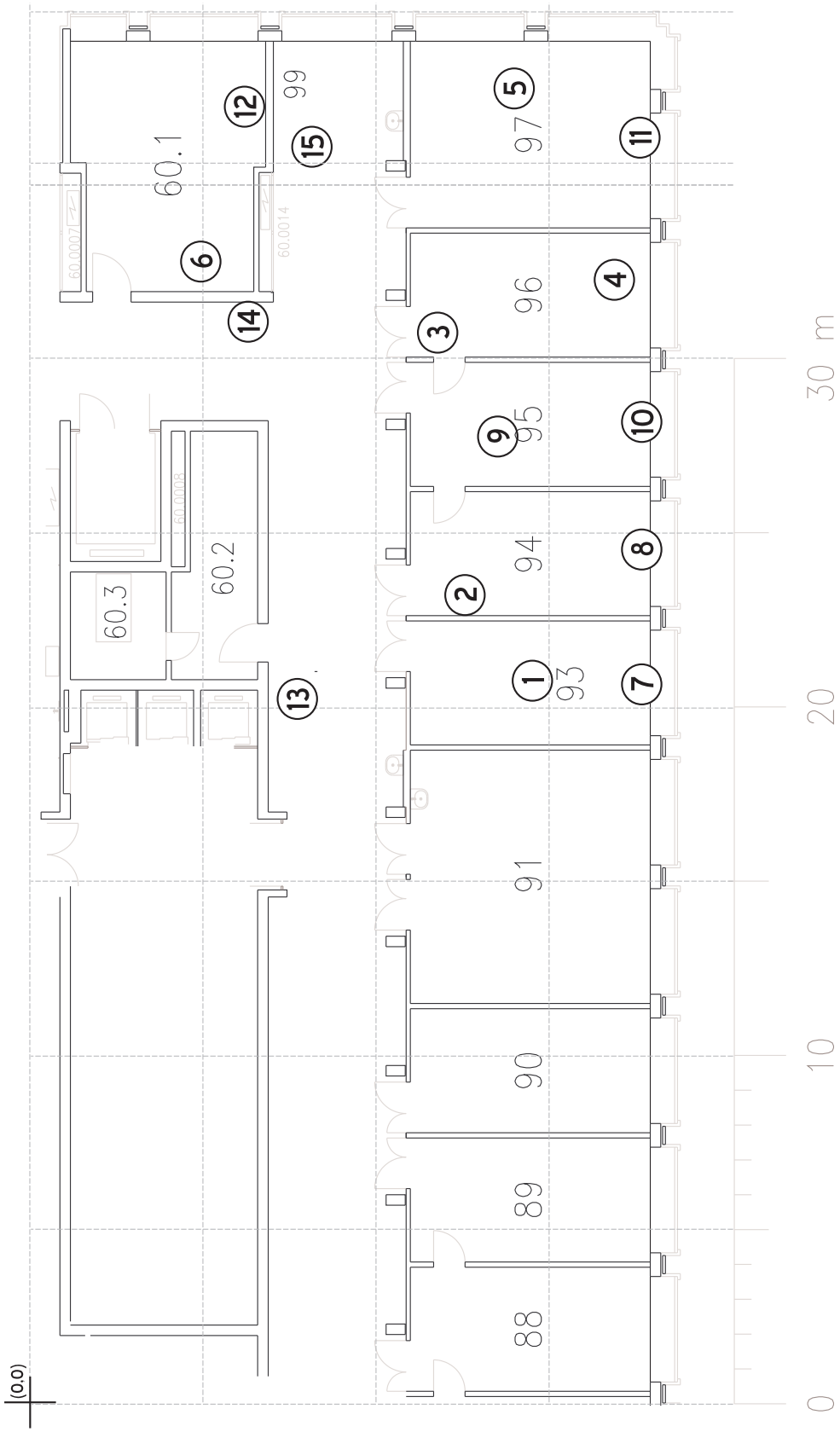


Figure 5.3: Map of test positions

5.2.3 Measurement Results

Detections by Fixed Nodes

Looking at the number of detections by the fixed nodes, it is obvious that the fixed nodes located in the same room as the phone were able to detect the phone almost everytime. The fixed nodes in the adjacent rooms were able to detect the device during most of the inquiries.
Please note, that although fixed nodes ETZ G95 and ETZ G99 were positioned more exposed, they could not always find devices in their closer neighborhood. This was due to a limitation of the fixed nodes (see Sec. 5.3.1), by which not all devices can be found, if there are too many devices around. So it may occur, that the fixed node cannot find the device although it is positioned very close to the fixed node.

Reference Point Algorithm

The reference point algorithm operates on data obtained from previous measurements at different known positions. Reference data are generated for all 15 positions where a phone was placed during the measurement procedure. Based on the number of detections by the different fixed nodes an expected value for the probability of detection was stored in the file `reference.dat`, which was later used by the reference point algorithm. This algorithm performed on most occasions very well. Sometimes it made a decision for another reference point in the same room which leads also to a decision for the correct fixed node. It can be stated that one reference point in every room is sufficient as the difference between two positions in the same room can normally be neglected.

Reference Node Algorithm

The precision of this algorithm depends on two different factors:

1. detections of the mobile device by the fixed nodes
2. detections of fixed nodes by the other fixed nodes in the system

How often each fixed node is detected by other fixed nodes is shown in the appendix C.

Wall Algorithm

The wall algorithm uses additional information about rooms and the number of walls between them to construct a graph of the situation. For the test measurement we used the following configuration:
The weight of the edges in the graph is the number of walls on the direct line between the two fixed nodes. All walls (and doors) have been assigned the weight one, except for the walls around room ETZ G60.1 which have been assigned weight two. The distance between two fixed nodes is not taken into account, but this could be a further extension. The threshold value used in the algorithm is set to two because our measurements showed, that a mobile device can be detected, if not more than two walls are between it and a fixed node.
A lot more time for tuning parameters (weight of edges, threshold value) could

be invested and could probably lead to better results. However, for large environments or for the usage for a specific event only, this would become too much complicated.

History Algorithm

The history algorithm operates over a longer time period than the other algorithms. In general, longer periods lead to better results because the probability of detection by a fixed node increases over a longer time period. The diagrams in appendix C are plotted for a time period of ten minutes. It is assumed that the position of a device does not change during the measurement.

5.2.4 Number of Correct Decisions

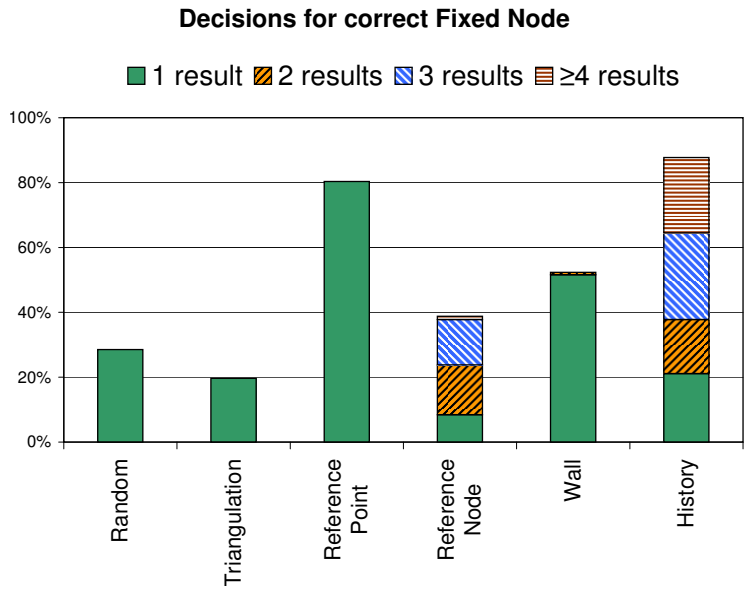


Figure 5.4: Correct decisions for different algorithms

Figure 5.4 shows how often the different algorithms decided for the fixed node which was positioned next to the phone during the measurement. Additionally, it is indicated for how many different fixed nodes (results) the algorithm decided simultaneously. Note that only one of these fixed nodes represents the correct fixed node but for the algorithm they are all correct decisions. If only a single result has to be returned to the client then a further algorithm has to be used.

5.2.5 Combination of different Algorithms

Comparing the number of decisions for the correct location made by the different algorithms that were implemented during this thesis, it was obvious that there was no single algorithm which did a correct decision for every position. It should therefore be evaluated if the combination of different algorithms leads to better results. Some algorithms (e.g. reference node algorithm) can decide for more than one fixed node at the same time. This list of fixed nodes can then be

the input vector for another algorithm, e.g. the triangulation algorithm. The triangulation algorithm returns always only one result, and this result can be sent back to the client.

5.2.6 Scalability of the System

Our last measurement was a crash test with 45 Bluetooth devices in a small area, interfering with the devices we wanted to find. The BlueLocation system was able to handle the load, nothing crashed and the discovery scan did not throw any exceptions. The drawback was, that not all Bluetooth devices were discovered during a scan. The following figure shows the results obtained from an example position. We can explain this result as a consequence of the hardware limitation discussed in Sec. 5.3.1. The total number of Bluetooth devices, including the fixed nodes, is 45. The average fixed node should be able to detect slightly more than half of them, out of which only the first ten are returned, so $\frac{1}{3}$ is about the probability of detection. Without the hardware limitation, we forecast that the results would be more or less the same as during normal operation. This will have to be tested when work on BlueLocation is continued.

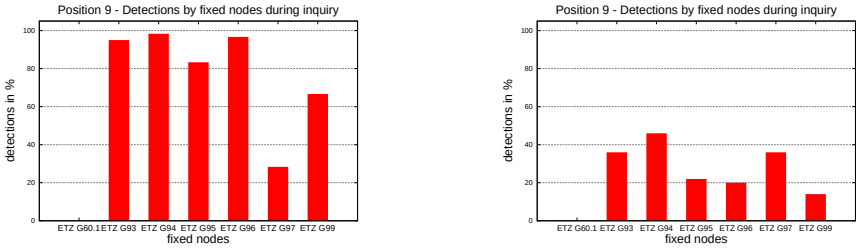


Figure 5.5: Detectability of device at position 9 during normal operation (left) and crash test (right)

5.3 Fixed Node Limitations

5.3.1 Maximum Number of Devices Found

During the crash test with a lot of Bluetooth noise emitted from mobile devices, we discovered that our Bluetooth adapters could find a maximum of ten devices per inquiry scan. If more devices are active, the scan is stopped after the 10^{th} device. The maximum number of devices found depends on the hardware and can not be addressed by the Java interface.

5.3.2 Master-Slave Role Switch

Bluetooth connections can only be established in a piconet. A piconet consists of a master and up to seven active slaves. While it is theoretically possible for a Bluetooth device to be active in more than one piconet, all of our Bluetooth adapters couldn't handle any other connections once they were connected in the slave role. This is due to the limited resources in the adapters, the calculation power of the onboard chip is probably too weak to handle another piconet.

The only solution for a fixed node to handle several mobile devices is for it to be master and all the mobiles to be slaves. When our Nokia 6630 initiated a connection they were always master and the fixed node slave. A role switch was therefore necessary after the connection was established. We were able to trigger such a role switch manually with the HCI Tool, but the Java implementation was not able to do that switch, altough the necessary master and slave flags were set.
As a result of this limitation, only one mobile device can be connected to a fixed node at a time.

5.3.3 Discovery Scan During Connection

In the Bluetooth standard, there are different states a device can be in. Connected and device discovery are two of them. While it is again theoretically possible for a device to be in more than one state at a time, our Bluetooth adapters were only able to start a device discovery, when they were in master role. Unfortunately, due to the last limitation our fixed nodes are always in slave role, therefore device discovery is interrupted as soon as a mobile device is connected to it.
A role switch would solve this problem as well.

5.4 Mobile Device Limitations

5.4.1 Inquiry Scan

When BlueLocation is started, a connection to a fixed node has to be set up before the application can be used. If no BTIDs of fixed nodes are stored in cache or a refresh is called, an inquiry for devices with the BlueLocation UUID is started. If this scan happens to coincide with a scan from a fixed node, no devices are found, since the fixed node is busy and doesn't answer to the phones scan. The Bluetooth hardware doesn't offer the possibility to listen for ongoing scans before starting one itself, so collisions cannot be prevented (compare to CSMA).
Something similar happens if the mobile device tries to connect to a fixed node, which is scanning at that time. Another attempt has to be started a couple of seconds later. Once a device is connected to a fixed node and tuned into its frequency hopping succession, a scan does not limit communication with the fixed node. Data can be sent and received during the scan.

5.4.2 Exit Command on Nokia Devices

On our mobile device testbed, the Nokia 6630, as well as on other Nokia devices of the series 60 generation, a command called "Exit" is automatically added to every Screen, without the possibility to remove or alter this command. When selected, the MIDlet is terminated without any control involved, and no defined cleanup is done. There is no other way, than to warn the user not to use this command. On other mobile phones or on the J2ME emulator, for example, no default commands are added to screens.

Chapter 6

Outlook and Conclusion

6.1 Further Extensions to the BlueLocation System

6.1.1 Inquiry with RSSI

The current BlueLocation system can not perform a Bluetooth inquiry with Received Signal Strength Indicator (RSSI) because the used Bluetooth hardware on the fixed nodes does not support this functionality at this time. The RSSI would give us an estimation of the signal strength to a Bluetooth device already during the inquiry process. This would help us to improve the algorithms (especially the triangulation algorithm). At the moment we only know if a fixed node has found or not a phone during inquiry. But we do not know if the signal from the device is very strong or very weak.

6.1.2 Communication from the Central Unit to the Fixed Nodes

In the current system architecture it is only possible to start a connection from a fixed node to the central unit (inquiry results update) and not otherwise. It would be useful if the fixed node would also listen on a special port for incoming connections from the central unit. In this case, it would be possible to dynamically change parameters of the system such as the interval between subsequent Bluetooth inquiries or tell the fixed nodes to suspend/restart its activities. Also the fixed nodes could be instructed to download a new software version from a Subversion (SVN) repository and restart themself afterwards. This would allow easy maintenance of the distributed fixed nodes.

6.1.3 Waiver of Limitations

For BlueLocation to be used in a larger context, the limitations of the fixed node discussed in the last chapter must be removed. New Bluetooth hardware has to be found that supports the three features that the present Bluetooth adapters are lacking.

6.1.4 Mobility Models of Users

From the data collected by the BlueLocation system a mobility model of individual users could be developed. Questions such as 'Who was where for how long?' could be answered easily.

6.1.5 Web Interface to the BlueLocation System

The already started development of a web interface to the BlueLocation system could be continued and further features implemented. A search function for BlueLocation users could be implemented and the current position of a user could be shown on an interactive map.

6.2 Conclusion

This semester thesis clearly proofed that a distributed system to localize people based on data from Bluetooth radio is possible. We demonstrated that the location of a Bluetooth device can be estimated with satisfying accuracy, although no information about signal strength is available from the scanning fixed nodes. The accuracy of the different algorithms, however, depends heavily on the position of the mobile devices and the fixed nodes. In our small test environment, there were only two rooms which didn't suffer from the edge effect. The triangulation algorithm for example is only reliable in the non-edge area. To better characterize our algorithms further tests in a larger environment are necessary. Another aspect of algorithm evaluation is the fact that the reference node and the wall algorithm need to be fed with information about the environment. The more information is available and, therefore, the better the algorithm parameters are adapted to the environment, the more exact the calculated results are.

The communication between the mobile device and the fixed node, once established, works very reliable. It was quite unexpected how fast, not in terms of bandwith, but reaction time, data were sent over the Bluetooth connection. Since we are only transmitting short streams a couple of bytes long, this is exactly what is relevant in our scope. To a human being, between sending a request and receiving the answer on screen, no delay is perceivable.

List of Tables

3.1 Sample data set for two bluetooth devices 20
3.2 File map.dat: sample of a look-up table 21
3.3 File reference.dat: Reference data for position 2 23
3.4 Number of detections during ten inquiries 25

4.1 List of database tables 36
4.2 Java classes for the localization algorithms 39

5.1 Results of simultaneous inquiries 45
5.2 List of fixed nodes 46
5.3 Measurements details 47

List of Figures

3.1 BlueLocation system overview 9
3.2 D-Link DBT-120 USB Bluetooth dongle 10
3.3 Active connection setup 13
3.4 Passive connection setup 14
3.5 Requesting a list of active users 15
3.6 Requesting location information 15
3.7 Checking which buddies are online 16
3.8 Searching for users 16
3.9 Sending a BlueMessage 17
3.10 Updating an alias 17
3.11 Testing the connection 18
3.12 Algorithm operation overview 20
3.13 Triangulation example: circles indicate the position of the fixed nodes, a filled circle indicates that this fixed node can detect the device, the cross indicates the position estimated by the triangulation algorithm 21
3.14 Example of the Reference Node Algorithm 22
3.15 Map with reference positions 23
3.16 Map of the situation 24
3.17 Graph representation 24
3.18 Example of the wall algorithm 25

4.1 The Log Screen with some entries 28
4.2 Main menu of BlueLocation 29
4.3 Screen during discovery and after fixed nodes have been found . 30
4.4 Pop-up confirming a connection is established 30
4.5 A list of all active users 31
4.6 Pop-up informing about the location of a user 31
4.7 Example of a Buddylist 32
4.8 Search Form and search result List 32
4.9 Example of a Buddylist 33
4.10 The pop-up displayed after a connection loss 33
4.11 Different threads and classes in the fixed node 34
4.12 Database structure 37
4.13 Different threads and classes in the central unit 38
4.14 Communication within the BlueLocation system 40
4.15 Communication fixed node–central unit 40
4.16 Communiation phone–central unit 41

4.17 Screenshot of the BlueLocation web interface 44

5.1 Inquiry test setup 45
5.2 Position of the fixed nodes 48
5.3 Map of test positions 49
5.4 Correct decisions for different algorithms 51
5.5 Detectability of device at position 9 during normal operation (left) and crash test (right) 52

C.1 Detections of the fixed nodes by other fixed nodes (27.12.05 10:00-11:00) 66
C.2 Detections of the fixed nodes by other fixed nodes (23.12.05 13:40-14:30) 68
C.3 Results for Position 1 70
C.4 Results for Position 2 71
C.5 Results for Position 3 72
C.6 Results for Position 4 73
C.7 Results for Position 5 74
C.8 Results for Position 6 75
C.9 Results for Position 7 76
C.10 Results for Position 8 77
C.11 Results for Position 9 78
C.12 Results for Position 10 79
C.13 Results for Position 11 80
C.14 Results for Position 12 81
C.15 Results for Position 13 82
C.16 Results for Position 14 83
C.17 Results for Position 15 84

Appendix A

Task Description

Institut für Technische Informatik und Kommunikationsnetze
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Communications Systems Research Group

Winter 2005/06

Semester Thesis

for

Oliver Keiser (D-ITET), Philipp Sommer (D-ITET)

Advisors: Vincent Lenders and Bernhard Tellenbach
Supervisors: Dr. Martin May and Prof. Dr. Bernhard Plattner

Issue Date: 24th October 2005
Submission Date: 7th February 2006

BlueLocator II - A Location Infrastructure for Bluetooth Enabled Mobile Phones

1 Motivation

Most of today's mobile phones feature Bluetooth for short range communication. In addition of using Bluetooth for communication purposes, it is also imaginable to use Bluetooth for locating purposes. The position of a mobile user can easily be determined by checking the connectivity of a user to the fixed Bluetooth nodes that have been deployed at pre-determined positions. The connectivity information directly indicates the proximity of a user to a fixed node. Many applications such as an (indoor) navigation or people location systems for Bluetooth enabled mobile phones require this position information. Another usage of a locating infrastructure is to establish mobility patterns by collecting position information of mobile users at various places over time. The latter usage is specially interesting to study the behavior of mobile users which is of high interest in the networking research community.

2 Background

In a previous semester thesis [2] conducted at the CSG lab, a location infrastructure using Bluetooth was designed and implemented. The goal of this thesis is to use this infrastructure to (i) test different location algorithms and (ii) to extend it to allow users to query the location of other users with their mobile phones. This thesis is part of the Blue* project[1].

3 Assignment

The students should start working together and solve the common tasks. Then, Task I and II should be worked out separately.

3.1 Common Tasks

- Familiarize yourself with the previous semester thesis BlueLocator I[2].
- Setup the BlueLocator infrastructure at the CSG lab. The infrastructure consisting of the database server and a few fixed nodes is already working. The task of the students is to verify that everything is still working properly and to install additional fixed nodes.

3.2 Task I

- Design different localization algorithms. The goal of these algorithms is to estimate the location of a device when it is seen by multiple fixed nodes simultaneously.
- Implement the designed algorithms in the programming language of your choice.
- Evaluate the accuracy of each algorithm by making active measurements with mobile phones and the deployed infrastructure.

3.3 Task II

- Familiarize yourself with BlueFramework[3]. The BlueFramework should be used to implement the application on the mobile phones.
- Design the application for the mobile phones.
- Implement the application.
- Integrate the localization algorithm with the best performance of Taks I into your application.
- Test the application.
- If time is remaining, identify a new hardware platform for the fixed nodes and port the code for the fixed nodes to this new platform. The new platform should be much smaller and cheaper than standard desktop PCs as we are using today.

3.4 Deliverables

- At the end of the first week, a detailed time schedule of the semester thesis must be given and discussed with the advisors.
- At half time of the semester thesis, a short discussion of 15 minutes with the professor and the advisors will take place. The students have to talk about the major aspects of the ongoing work. At this point, the students should already have a preliminary version of the written report, including a table of contents. This preliminary version should be brought along to the short discussion.
- At the end of the semester thesis, a presentation of 15 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work and the demonstrator should be presented at this time.
- The final report may be written in English or German. It must contain a summary written in both English and German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Four copies of the final report must be delivered to TIK.

References

[1] *The BlueStar Project*, http://www.csg.ethz.ch/research/running/Blue_star, October 2004.

[2] Katrin Bretscher, *BlueLocation*, TIK-SA-2005-17, ETH Zürich, Switzerland, September 2005.

[3] Nicole Hatt, *BlueFramework - Application Framework for Bluetooth Enabled Mobile Phones*, TIK-MA-2005-16, ETH Zürich, Switzerland, October 2005.

18th October 2005

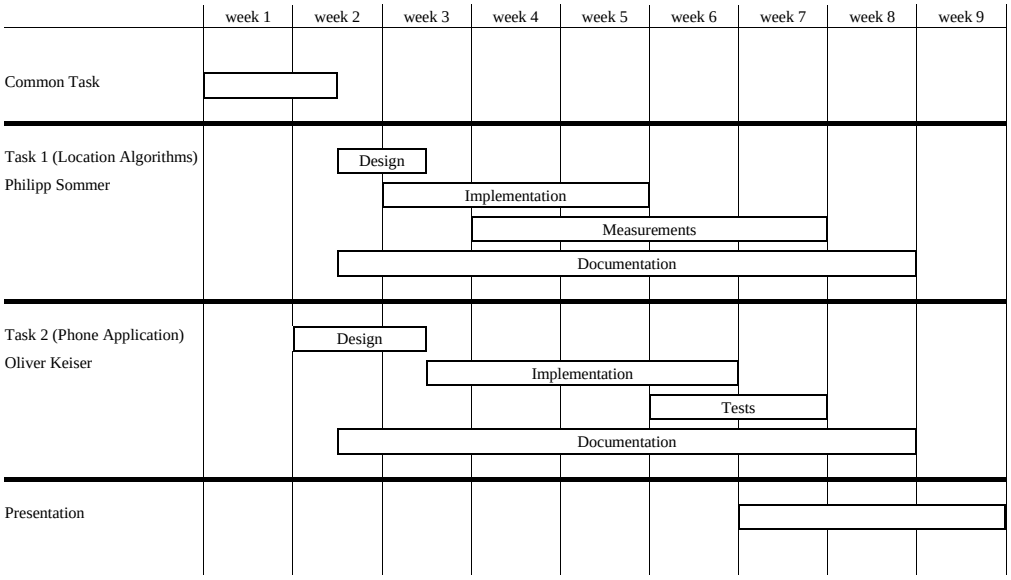
Prof. B. Plattner

Appendix B

Project Timeline

This is our project timeline as submitted at the end of the first week.

Semester thesis "BlueLocation" - Project timeline



Appendix C

Measurement Results

C.1 Detections of Fixed Nodes by other Fixed nodes

C.1.1 15 Bluetooth Devices in the Covered Area

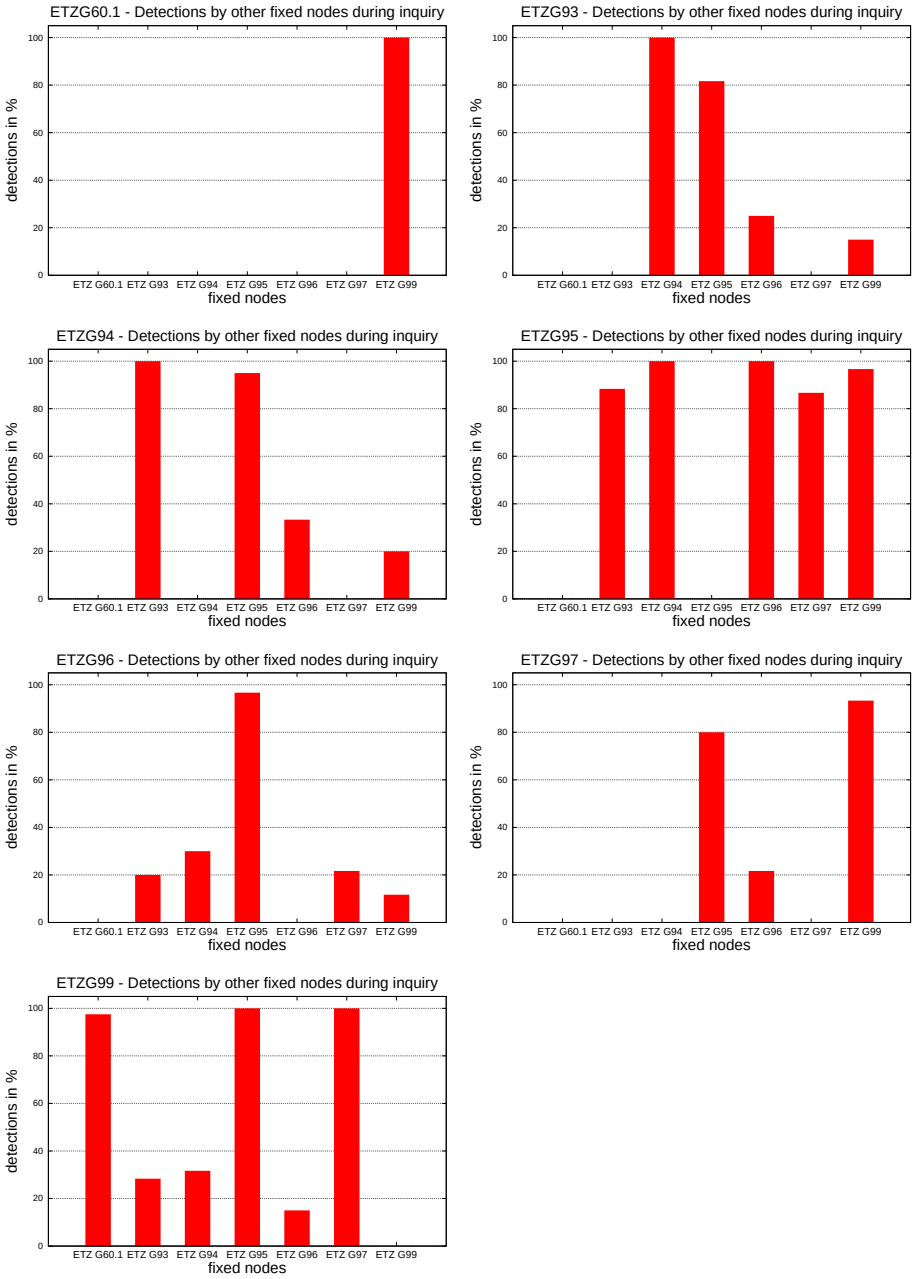


Figure C.1: Detections of the fixed nodes by other fixed nodes (27.12.05 10:00-11:00)

C.1.2 45 Bluetooth Devices in the Covered Area

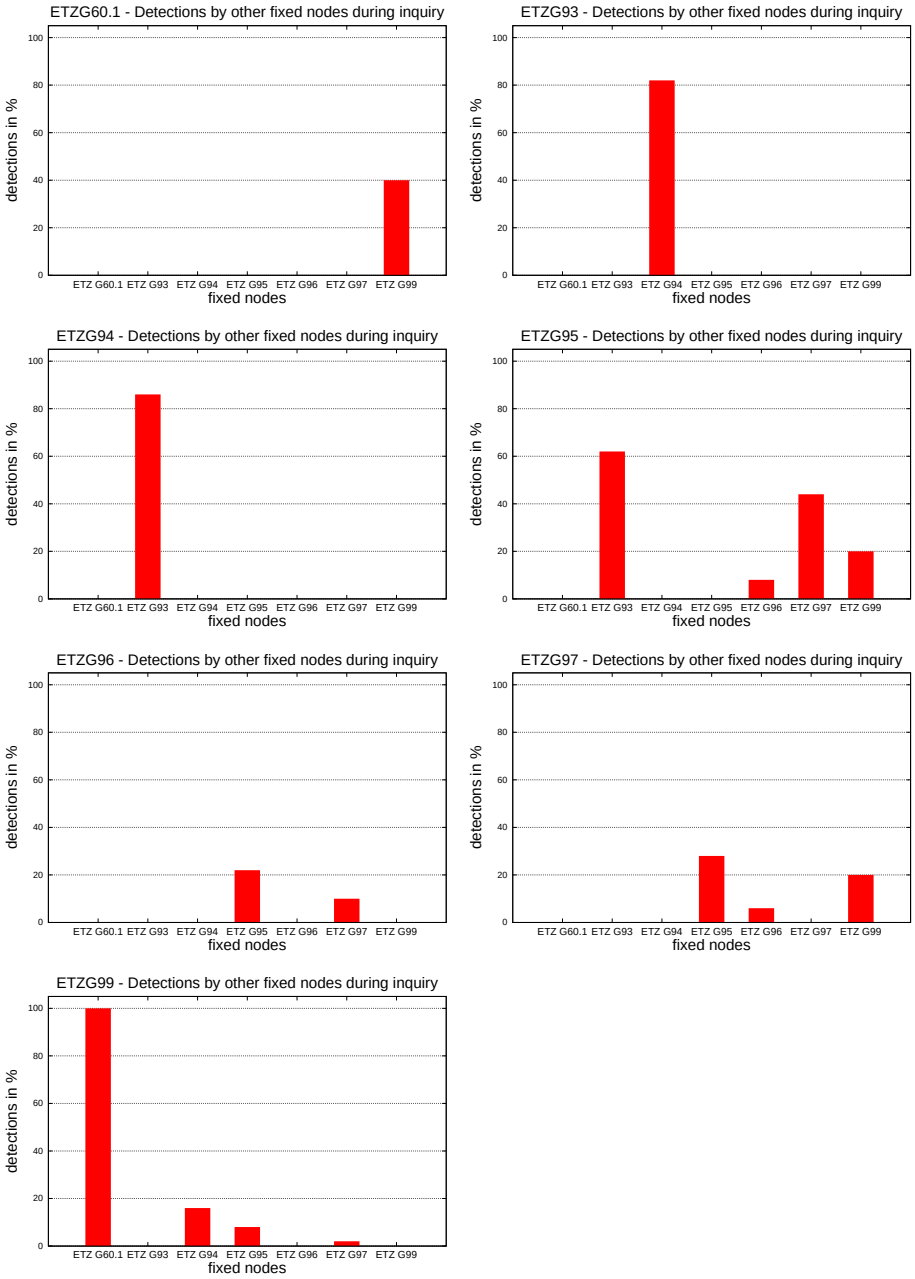


Figure C.2: Detections of the fixed nodes by other fixed nodes (23.12.05 13:40-14:30)

C.2 Algorithm Results for the Measurement Positions

The following pages show the analysis of the measurements with the phones placed at a fixed position and the BlueLocation infrastructure. There is a histogram with the percentage of detection by each fixed node. Additionally you will find a graph with the results for each algorithm. It shows how often each algorithm has decided for which fixed node. Note that some algorithms(Wall, Reference Node, Reference Point, History) can decide simultaneously for more than one fixed node and therefore the bars sum up to more than one hundred percent. The last graph shows how often each algorithm has decided for the correct fixed nodes (based on the position of the phone). The results are stacked together. It is indicated for how many fixed nodes the algorithm decided simultaneously (only of them was correct).

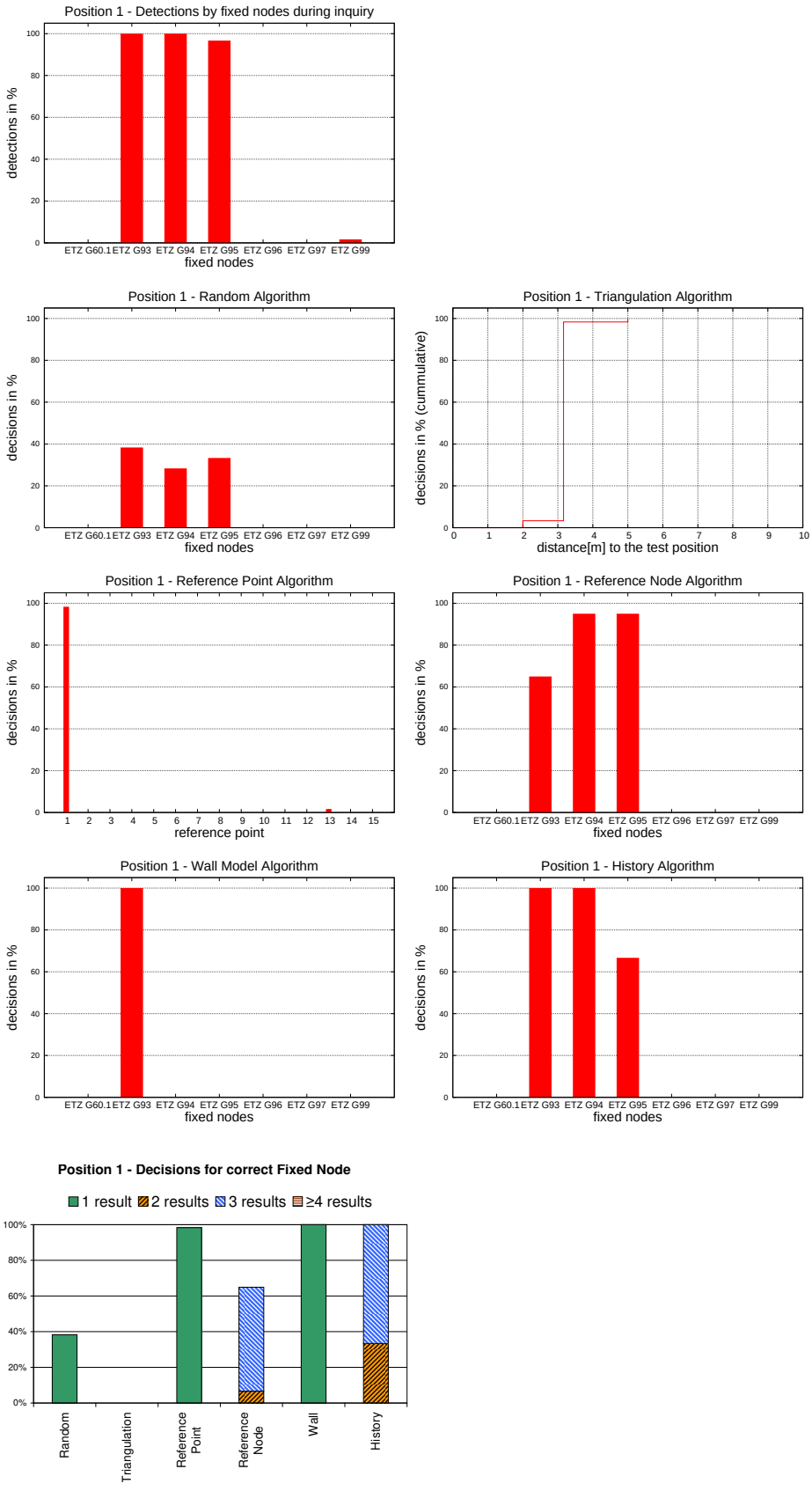


Figure C.3: Results for Position 1

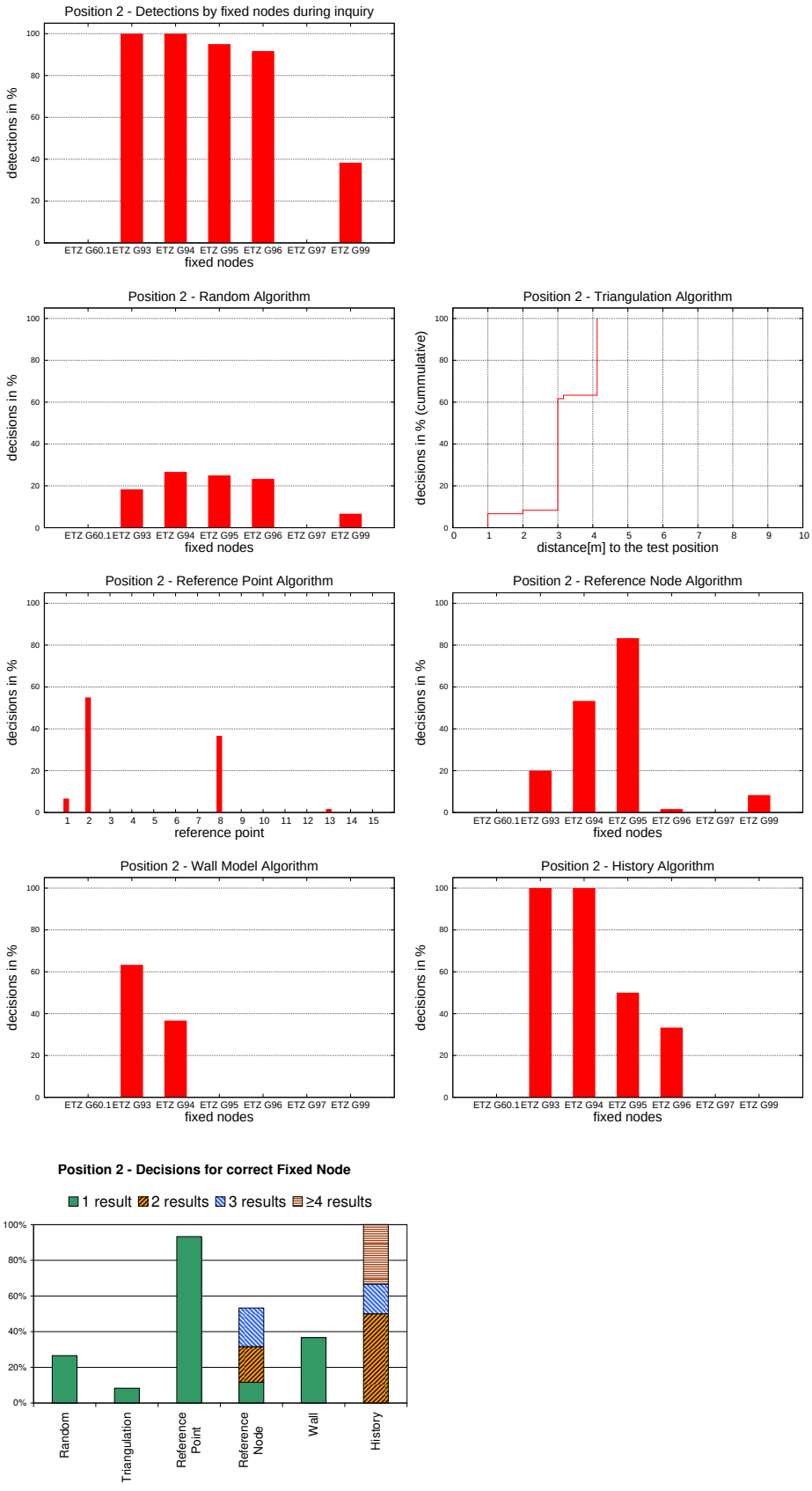


Figure C.4: Results for Position 2

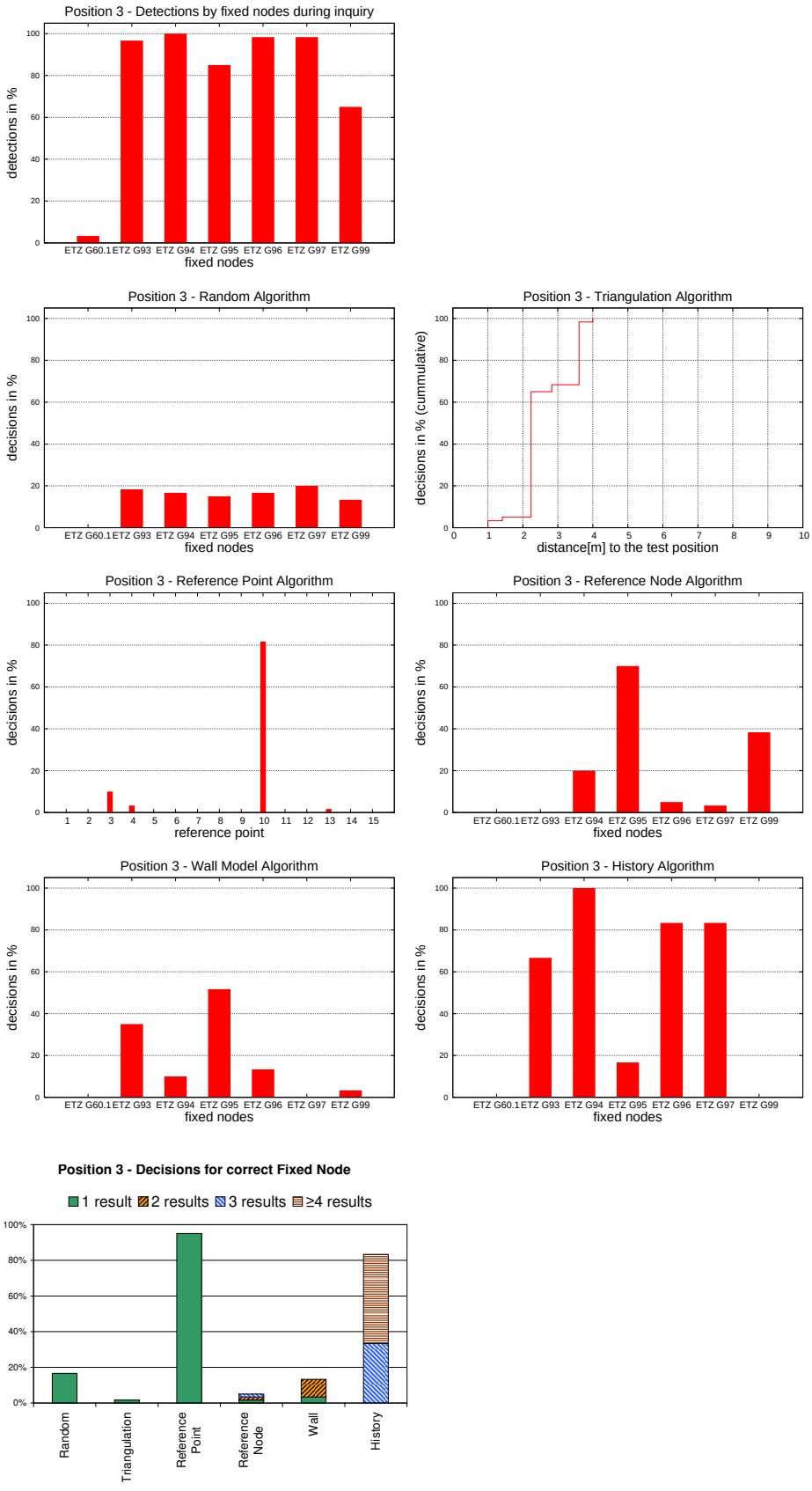


Figure C.5: Results for Position 3

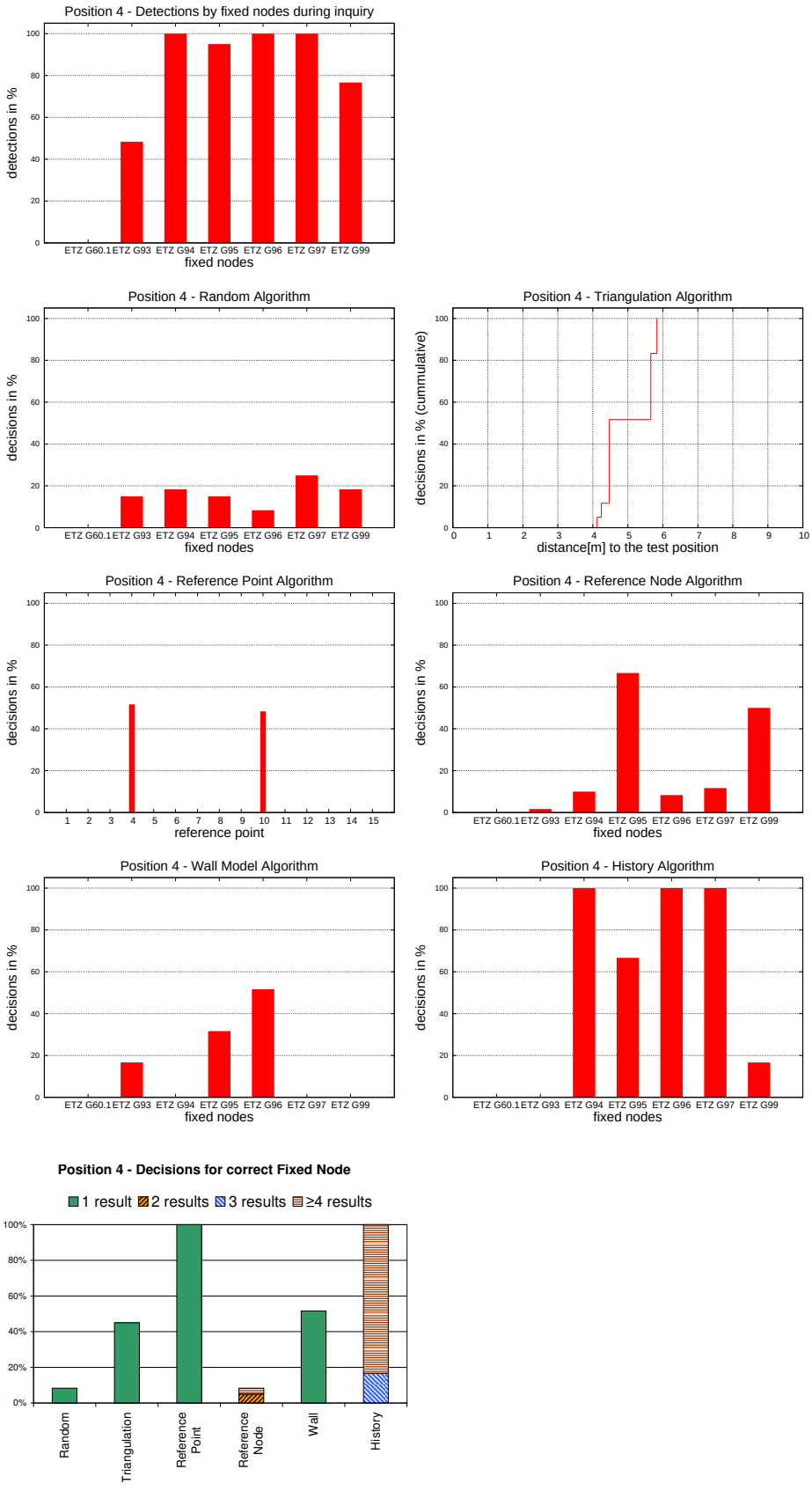


Figure C.6: Results for Position 4

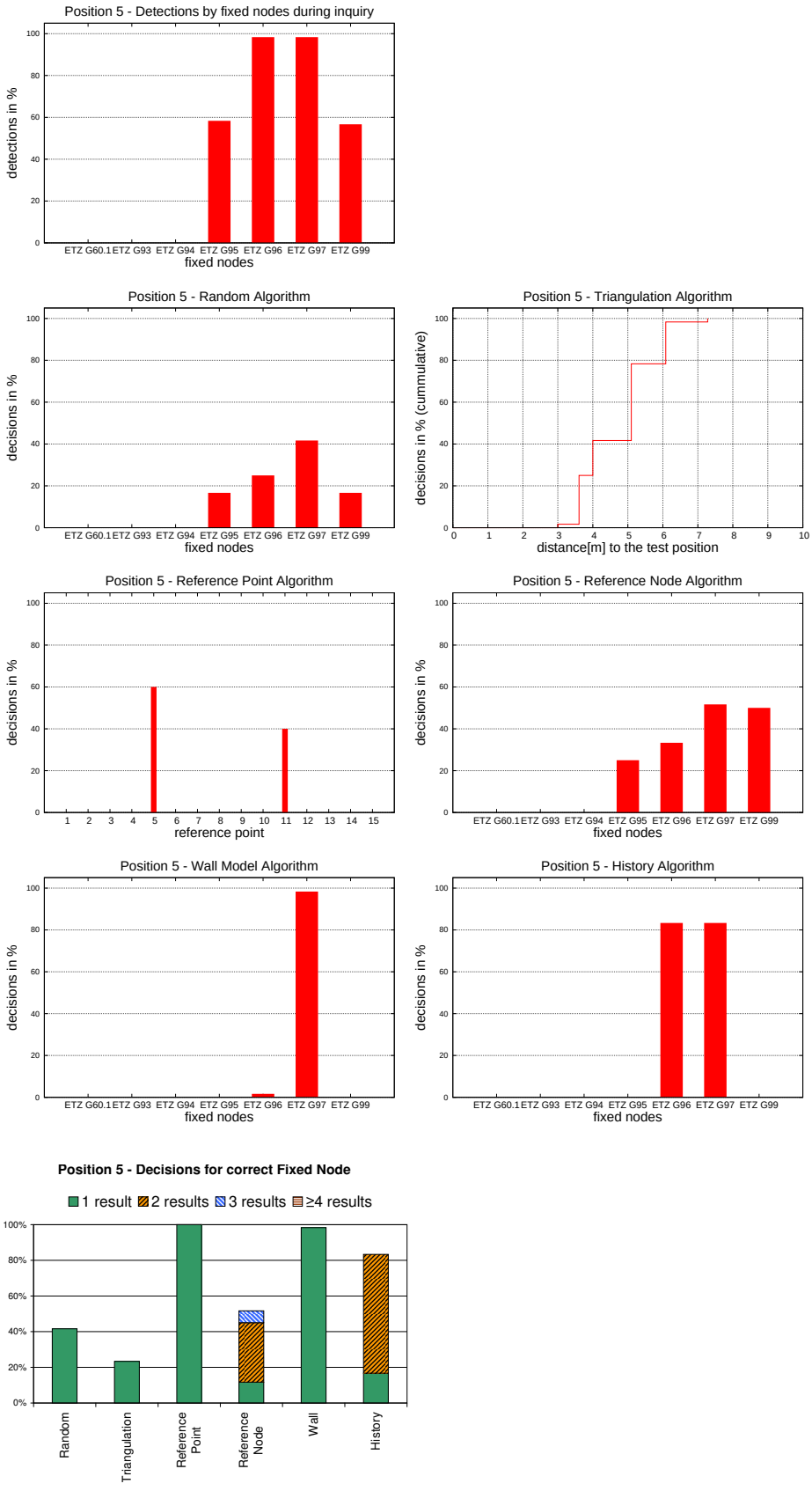


Figure C.7: Results for Position 5

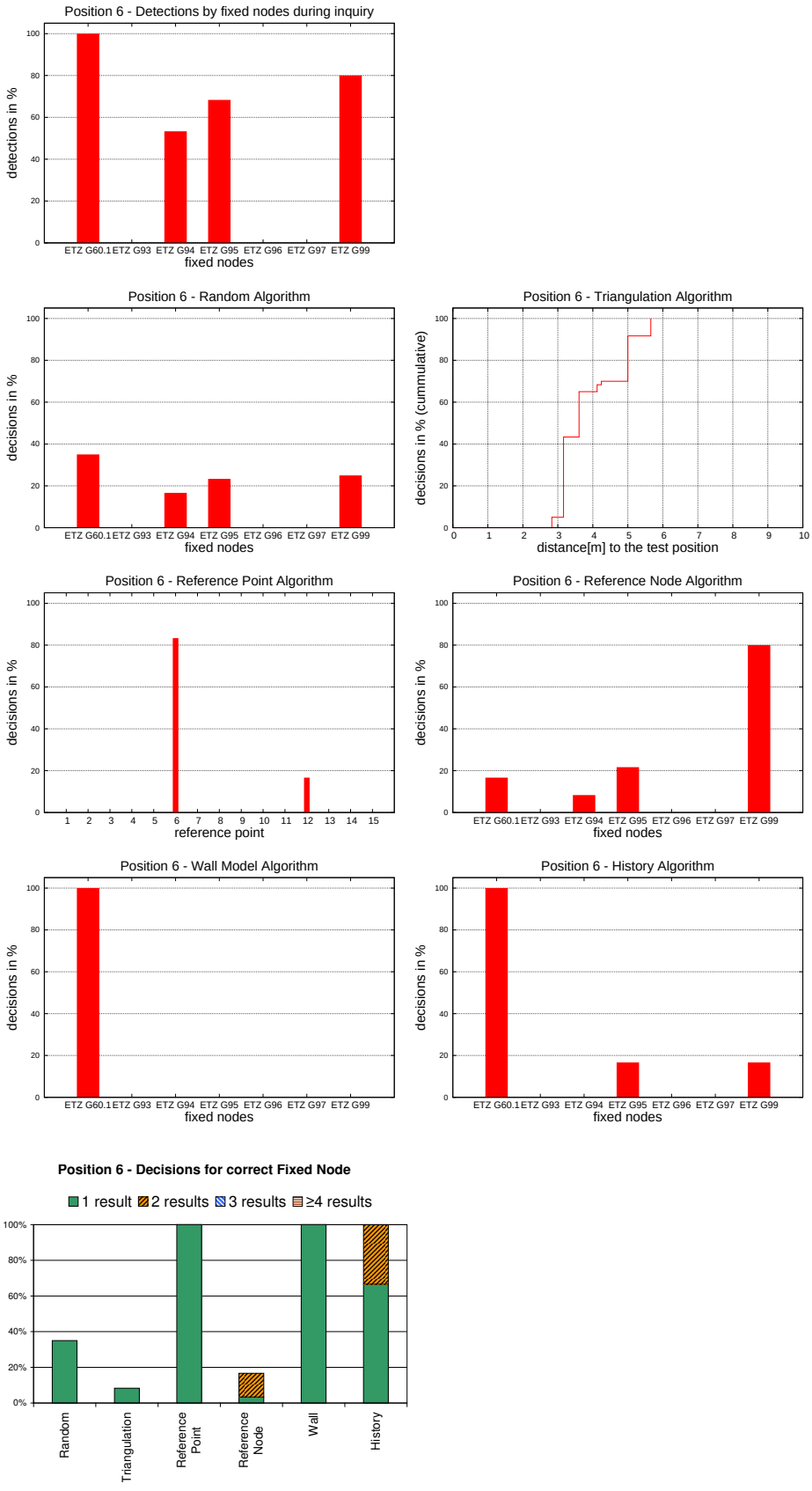


Figure C.8: Results for Position 6

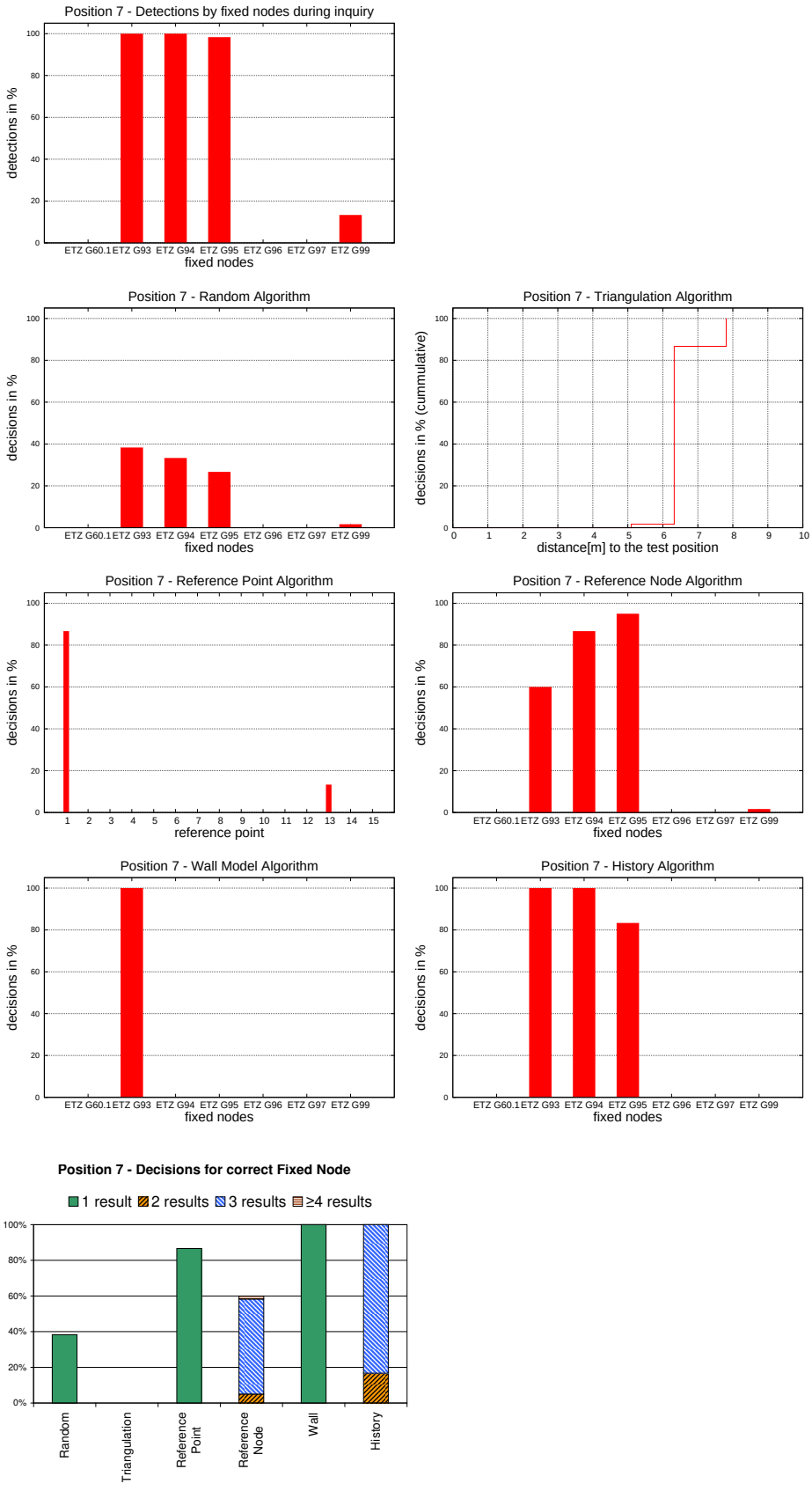


Figure C.9: Results for Position 7

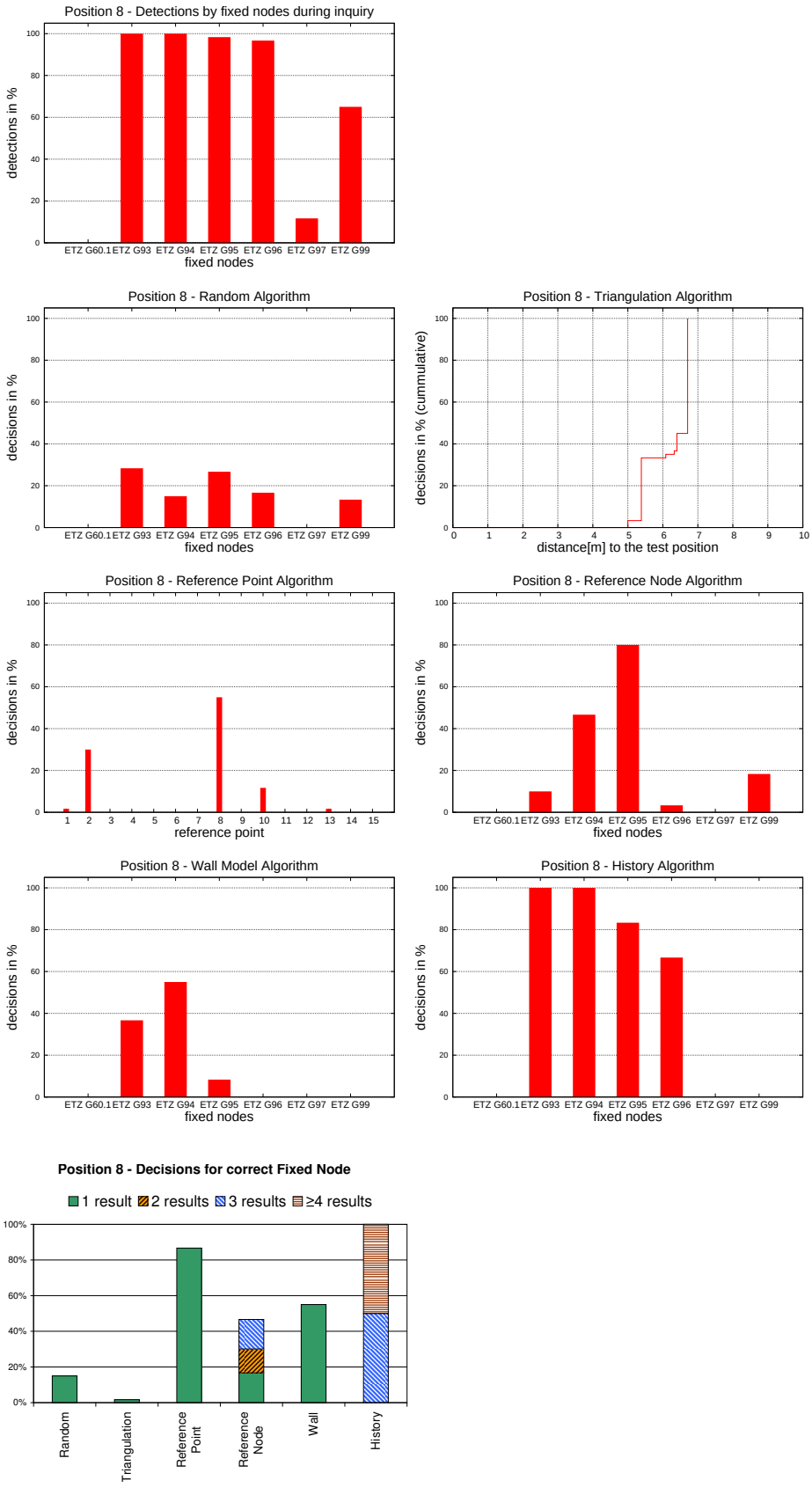


Figure C.10: Results for Position 8

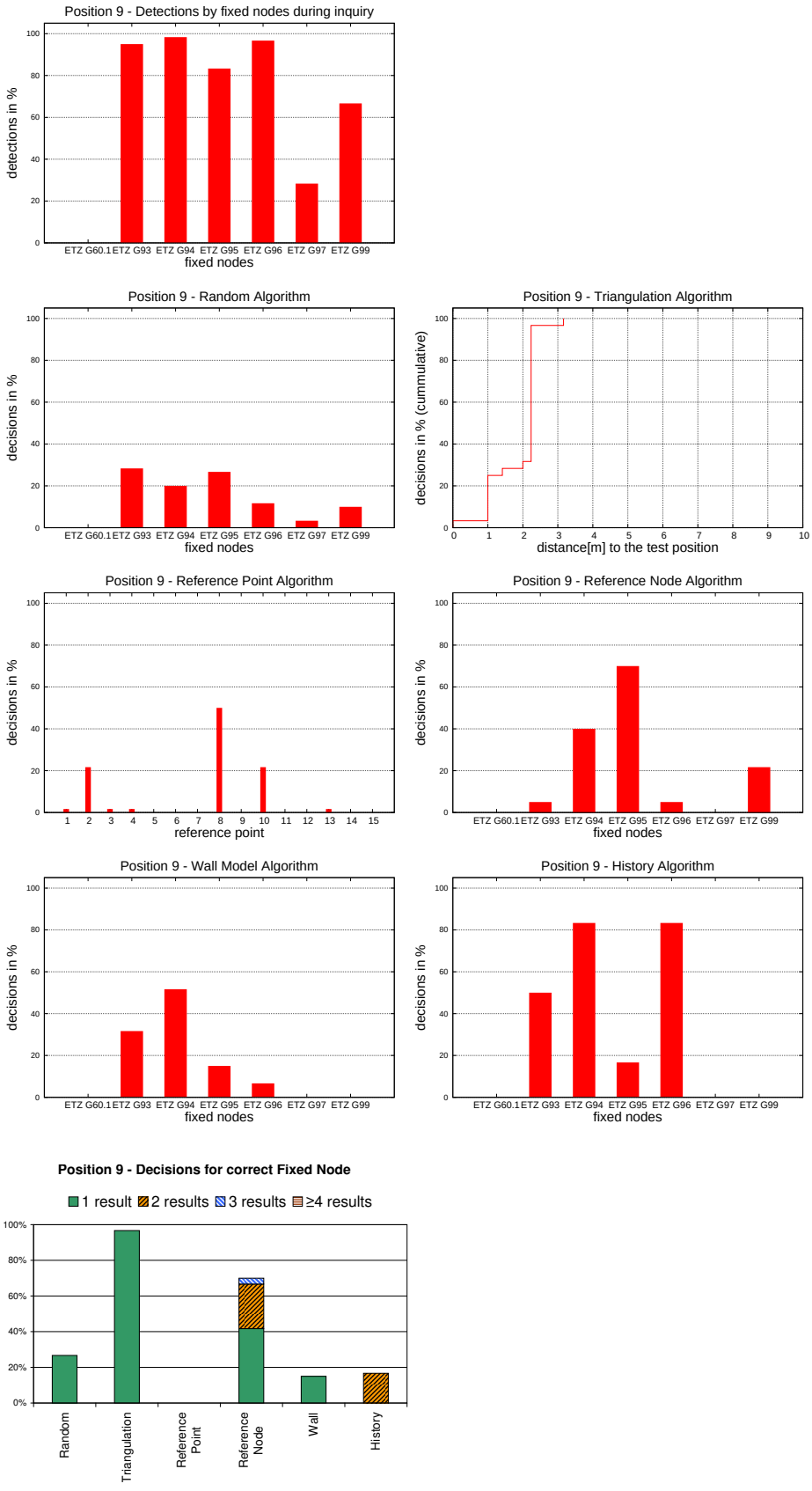


Figure C.11: Results for Position 9

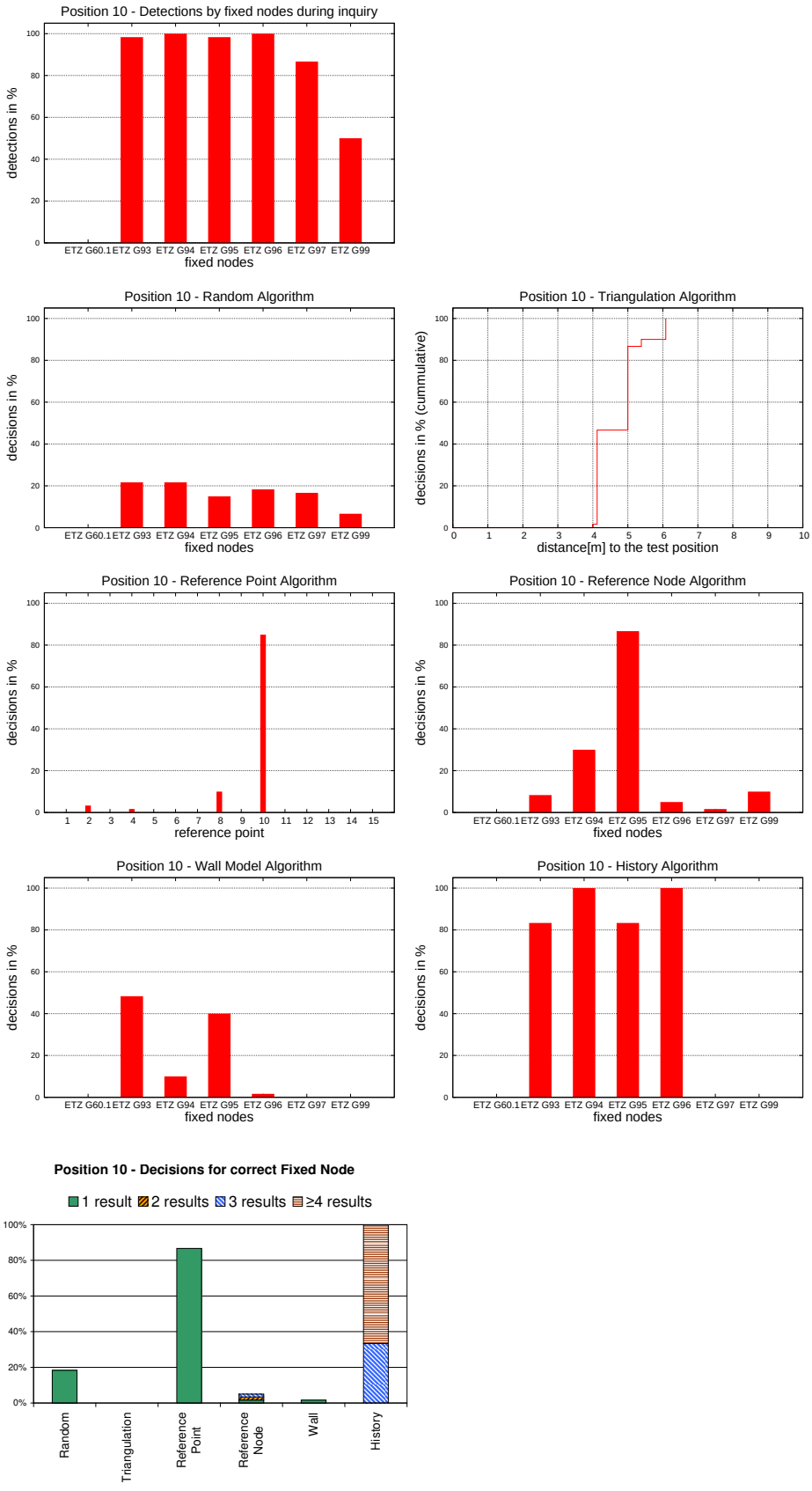


Figure C.12: Results for Position 10

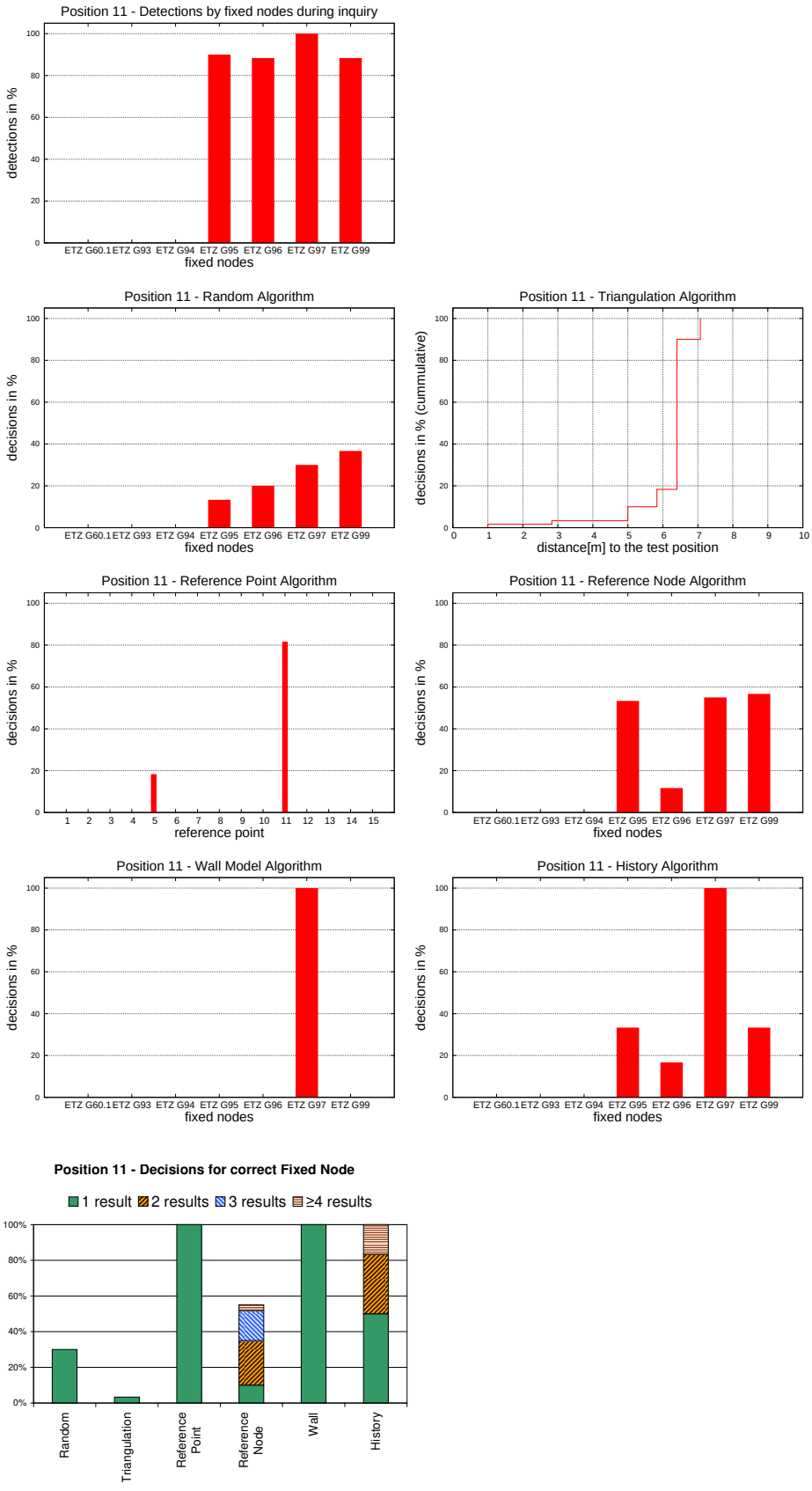


Figure C.13: Results for Position 11

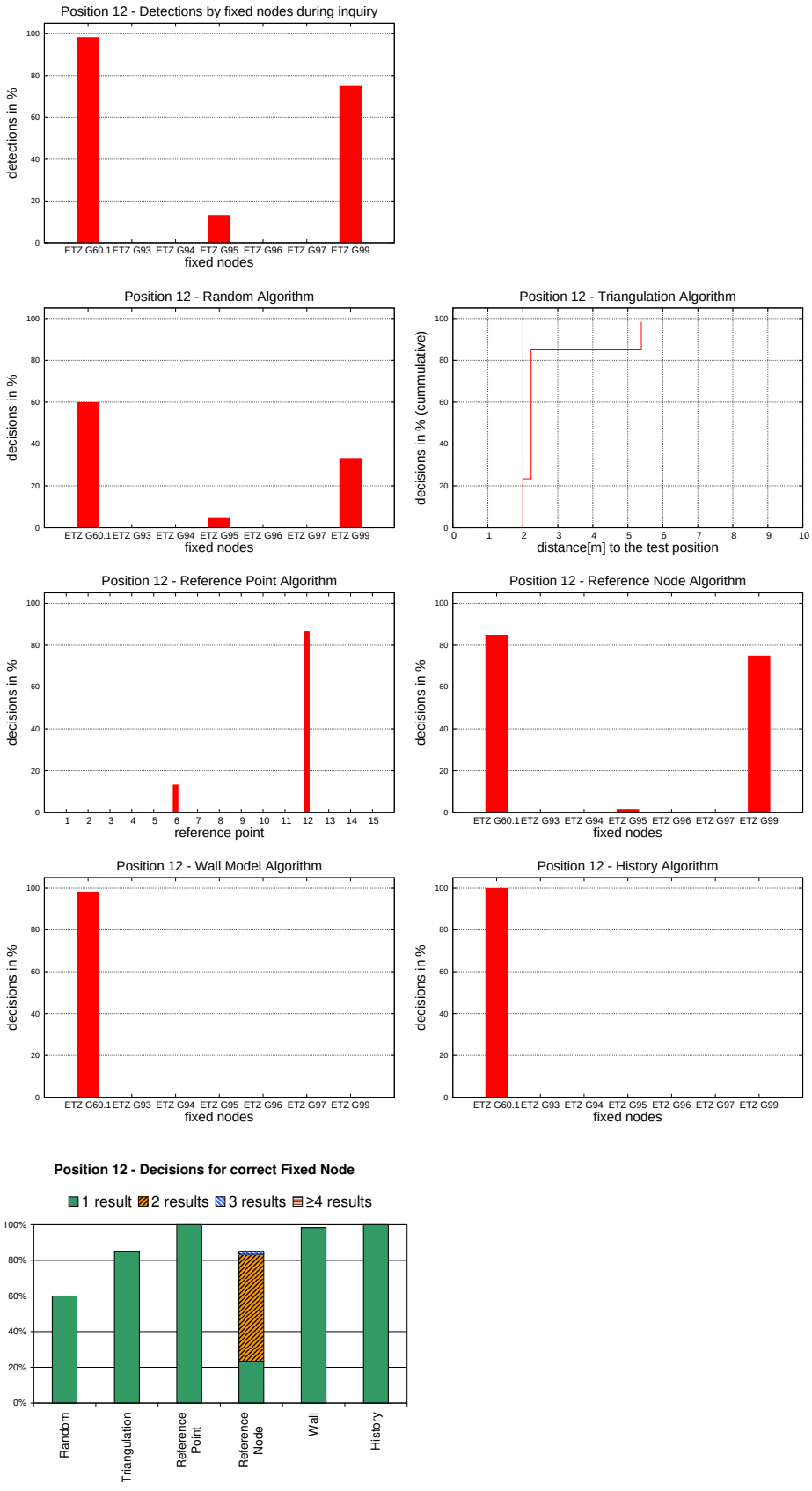


Figure C.14: Results for Position 12

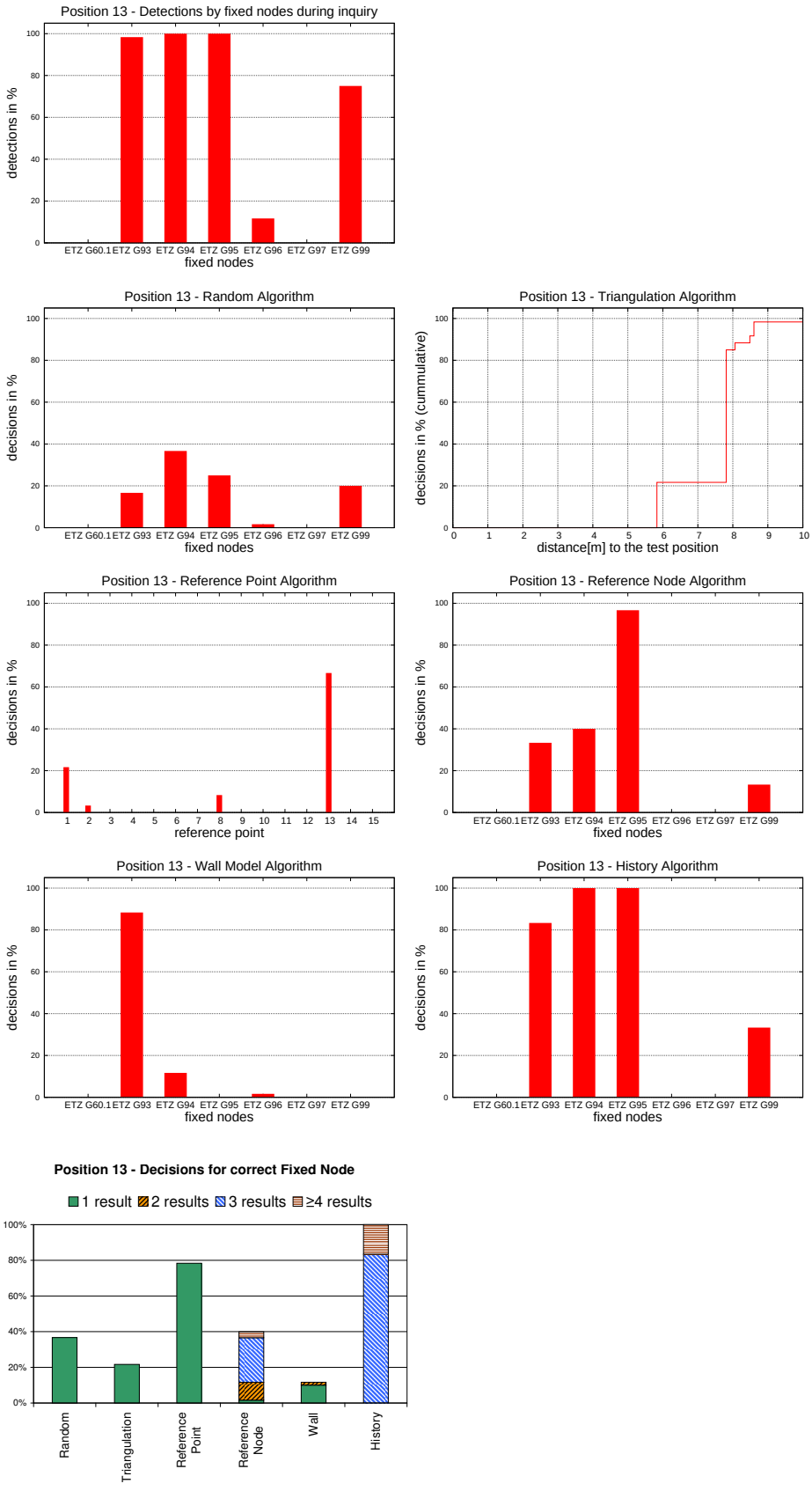


Figure C.15: Results for Position 13

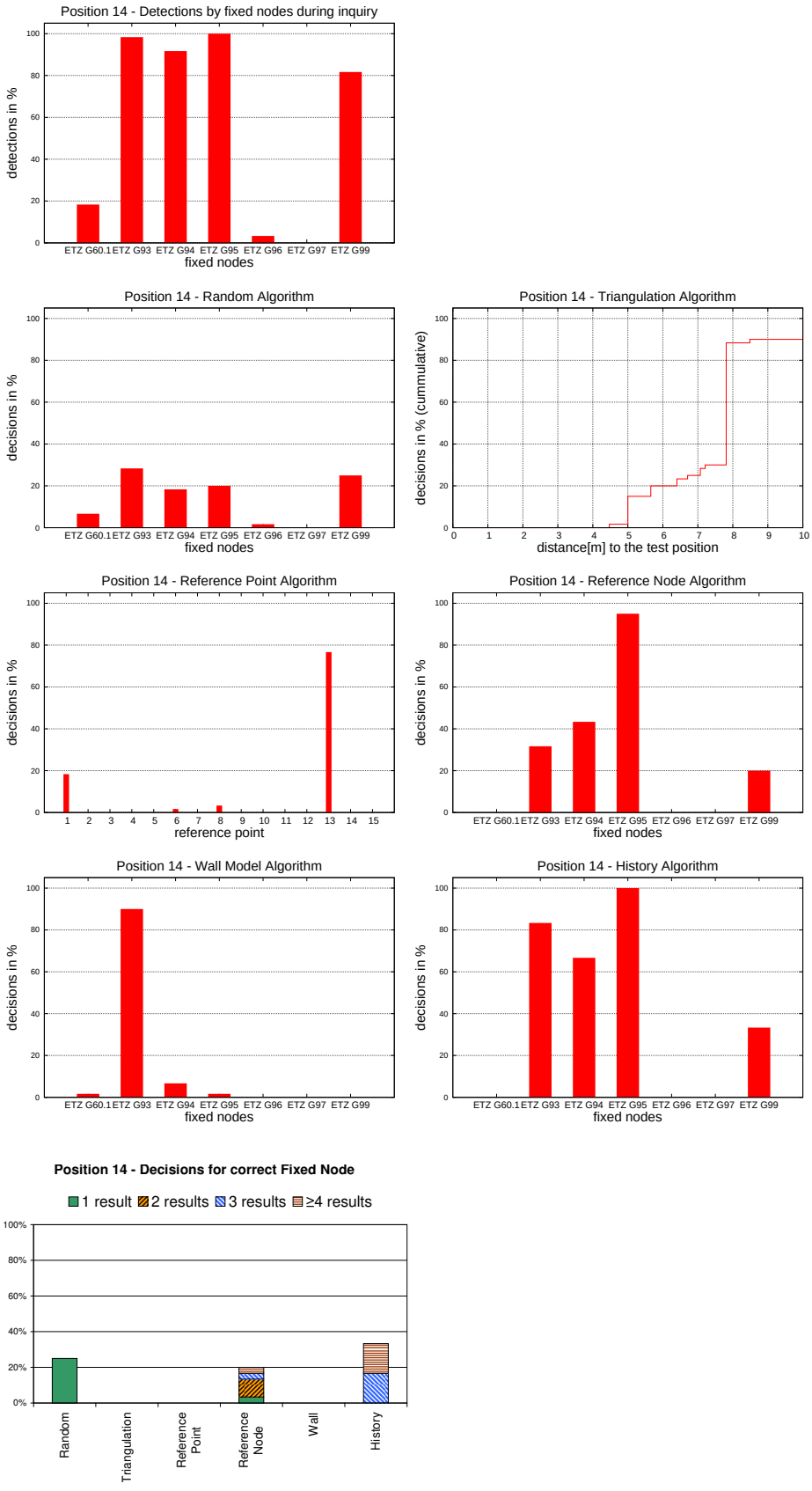


Figure C.16: Results for Position 14

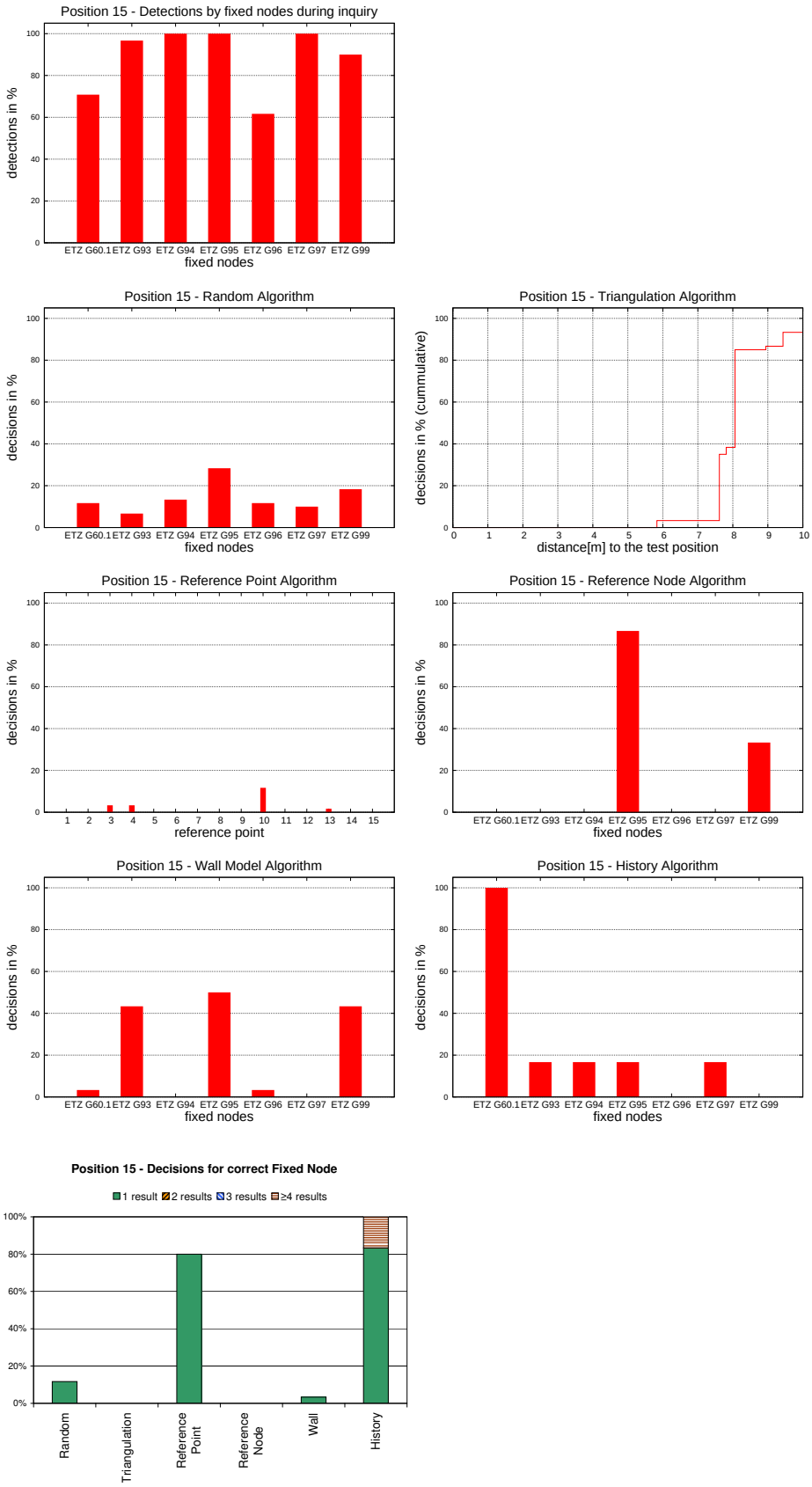


Figure C.17: Results for Position 15

Appendix D

Deliverables

D.1 CD-ROM

Content of the CD-ROM

directory	content
avetanaBluetooth-20051023	Avetana Bluetooth
blueframework	BlueFramework
centralUnit	Java source for the central unit
database	database definition file (SQL)
documention	this documentation
fixedNode	Java source for the fixed node
javadoc	JavaDoc (HTML) of all Java classes
measurements	measurement data (SQL)
phone	Java source for the phone application
web	web interface script

Appendix E

Installation Guide

E.1 Installation of a Fixed Node

E.1.1 Installation of the Java Software Development Kit (SDK)

We recommend to use the latest version of Sun's Java, although the application is reported to run on Java implementations from other vendors. You need to install a Java Runtime Environment(JRE) to run the software. If you want to recompile the software on the fixed node you need the complete Java SDK. You can obtain the Java software from the official Java website [8].

Installing Java "The Debian Way"

Java is not part of the standard Debian distribution due to licence policies. You have to install it on your own. These instructions can be found also [10].

1. Make sure that "contrib" is in your `/etc/apt/sources.list`
   example:
   `deb http://debian.ethz.ch/debian/ testing main contrib non-free`
   `deb-src http://debian.ethz.ch/debian/ testing main contrib non-free`
2. Update the package list:
   `apt-get update`
3. Install `java package` to build a Java package for Debian
   `apt-get install java-package`
4. Download the latest Java version from the Sun website [8]
5. Build your own Java Debian package `fakeroot make-jpkg <downloaded file>.bin`
6. Install the generated package on your system `dpkg -i <package>.deb`

E.1.2 Linux Bluetooth Stack (BlueZ)

BlueZ is the official Bluetooth protocol stack for Linux and part of the Linux kernel since version 2.4.6.

bluez-utils and Bluetooth libraries

BlueZ comes with some user-space tools called the bluez-utils. They are required for computers running the fixed node software. You can download the bluez-utils package from the BlueZ website [12] or get it from your Linux distribution. For a Debian system you can install them running `apt-get install bluez-utils` as root. Additionally, the `libbluetooth1` and `libbluetooth1-dev` libraries are required.

E.1.3 AvetanaBluetooth

AvetanaBluetooth [13] is an implementation of the JSR-82 specification. It is freely available for the Linux platform at SourceForge.net [14]. We were using version 20051023. Follow the build instructions which can be found in the downloaded package. A pre-compiled version of the software can be found on the CD. The build scripts will create two files: `AvetanaBT.jar` and `libavetanaBT.so`. You have to copy them into the fixed node's root directory or to somewhere else on the Java classpath.

E.2 Installation of the Central Unit

E.2.1 Installation of the Java SDK

You need to have an installed version of the Java SDK to compile and run the central unit software. Build instructions can be found in the fixed node section.

E.2.2 Setting up the Database

Create a new system user `bluelocation`
```
root#: adduser bluelocation
```

Installation and Configuration of PostgresSQL

Install the PostgreSQL database either directly from their website [15] or use the Debian package system (preferred):

```
root#: apt-get install postgresql postgresql-client
```
This will install PostgreSQL and will create the administrative user `postgres`.

We will next create the database user `bluelocation`
and the database `bluelocation`
```
root#: su postgres
  postgres$: createuser --pwprompt --no-adduser\
             --no-createdb bluelocation
```

```
postgres$: createdb --owner bluelocation bluelocation
postgres$: exit
```

You have to edit the file `/etc/postgresql/pg_hba.conf` to allow the bluelocation database user to access the database. At the end of this file you will find the section with the client names.
After the following line:
```
# TYPE  DATABASE    USER   IP-ADDRESS   IP-MASK    METHOD
```

add the entry for the bluelocation user to allow local connections:
```
local   all    bluelocation      ident sameuser
```

add the entry for the bluelocation user to allow connections by TCP sockets (required for the JDBC driver)
```
host  bluelocation  bluelocation 127.0.0.1  255.255.255.255  md5
```

Restart the PostgreSQL service.

Now you can login as user bluelocation and create the structure of the bluelocation database `root#: su bluelocation`
```
bluelocation$:  psql bluelocation -f <database definition file>
bluelocation$:  exit
```

E.3 Installation of the mobile device application

On the CD directory `phone/bin` you will find two files named `BlueLocation.jar` and `BlueLocation.jad`. The .jad file contains a description of the application and the .jar file contains all classes in a compressed format.
The installation of BlueLocation on a mobile device really depends on the manufacturer. The easiest way is to upload these two files using OBEX. Then the .jar file is executed, which should install the application
If this method doesn't work, many manufacturers supply special cable and utility software. This software can be used to upload the two files over the cable. Consult the manual for details.

E.4 Installation of the Web Interface

System requirements:

- Apache Webserver with PHP support (Debian packages `apache2`, `php4`, `libapache2-mod-php4`)
- PostgreSQL module for PHP (Debian package `php4-pgsql`)

E.4.1 Configuring Apache

Uncomment the line `#UserDir public_html` in `/etc/apache2/apache2.conf` to enable webpages for users. You can now access the `public_html` directory in your home at `http://<hostname>/~<username>`.
Note: There are no access restrictions configured by default. Refer to the apache website [19] for further instructions.

E.4.2 Installation

Copy the content of the directory `web` on the CD to the `public_html` directory.

Appendix F

Used Software Tools

Debian GNU/Linux

Debian Sarge, Version 3.1, http://www.debian.org/

Java 2 Standard Edition (J2SE)

Sun Microsystems, Version 1.5, http://java.sun.com/

AvetanaBluetooth

Avetana GmbH, latest CVS version from http://sourceforge.net/projects/avetanabt/

PostgreSQL

PostgreSQL Global Development Group, Version 7.4, http://www.postgresql.org/

pgAdmin III - PostgreSQL tools

pgAdmin Development Team, Version 1.2.2, http://www.pgadmin.org/

Wireless Took Kit including J2ME Emulator

Sun Microsystems, Version 2.2, http://java.sun.com/products/sjwtoolkit/

StringTokenizer

OstermillerUtils, Version 1.12, http://ostermiller.org/utils/

XMLSpy

Altova, Version 2006 Home Edition, http://www.altova.com/products_ide.html

Eclipse SDK

The Eclipse Foundation, Version 3.1.1, http://www.eclipse.org/

Xfig

Supoj Sutanthavibul, Version 3.2.5, http://www.xfig.org/

Dia

Alexander Larsson, Version 0.94, http://www.gnome.org/projects/dia/

Microsoft Office Visio

Microsoft Corporation, Version 2003, http://office.microsoft.com/

Gnuplot

Thomas Williams, Colin Kelley and many others, Version 4.0, http://www.gnuplot.info/

Scribus - Desktop Publishing

Version 1.2.2.99, http://www.scribus.net

BlueZ-utils

BlueZ Project, Version 2.19, http://www.bluez.org

Apache2 Webserver

Apache Software Foundation, Version 2.0.54, http://www.apache.org/

PHP - Hypertext Preprocessor (including the PostgreSQL module for PHP)

The PHP Group, Version 4.3.10, http://www.php.net/

Appendix G

Used Mobile Devices

TIK-Number	Name	Bluetooth address
2640-4820	Test 1	00119FC1306C
2640-4929	Test 2	00119FC143BE
2640-4931	Test 3	00119FC127D5
2640-4821	Test 4	00119FC1306E
2640-4818	Alice	00119FC1306D
2640-4927	Bob	00119FC143AB
2640-4826	Peter	00119FC143D3

Bibliography

[1] ETH Zurich, TIK: The Blue* project
`http://www.csg.ethz.ch/research/projects/Blue_star/`, December 2005

[2] Katrin Bretscher: *BlueLocation*, TIK-SA-2005-17, ETH Zurich, Switzerland, September 2005
`ftp://www.tik.ee.ethz.ch/pub/students/2005-So/SA-2005-17.pdf`

[3] Nicole Hatt: *BlueFramework:* Application Framework for Bluetooth Enabled Mobile Phones, TIK-MA-2005-16, ETH Zurich, Switzerland, October 2005
`ftp://www.tik.ee.ethz.ch/pub/students/2005-So/MA-2005-16.pdf`

[4] Bluetooth: The Official Bluetooth Website
`http://www.bluetooth.com/`, December 2005

[5] Jason Yipin Ye: *Atlantis: Location Based Services with Bluetooth*, Departement of Computer Science, Brown University, Providence RI, U.S.A.
`http://www.cs.brown.edu/publications/theses/ugrad/2005/jye.pdf`, December 2005

[6] Josef Hallberg, Marcus Nilsson, Kåre Synnes: *Positioning with Bluetooth*, Department of Computer Science and Electrical Engineering, Luleå University of Technology, Sweden
`http://media.csee.ltu.se/publications/2003/hallberg03positioning.pdf`, December 2005

[7] JCP, JSR 179
`http://www.jcp.org/en/jsr/detail?id=179`, December 2005

[8] Sun Developer Network - *Java Technology*
`http://java.sun.com/`, December 2005

[9] Debian GNU/Linux, The Debian Project
`http://www.debian.org/`, December 2005

[10] Installing Java on Debian - Linuxquestions.org Wiki
`http://wiki.linuxquestions.org/wiki/Java-Debian`, December 2005

[11] Sun Microsystems, Wireless Took Kit including J2ME Emulator
`http://java.sun.com/products/sjwtoolkit/`, December 2005

[12] BlueZ - Offical Linux Bluetooth protocol stack
Project Website `http://www.bluez.org/`, December 2005

[13] Avetana GmbH: AvetanaBluetooth JSR-82 implementation
`http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.xml`, December 2005

[14] AvetanaBluetooth JSR-82 implementation - SourceForge.net Project Page
`http://sourceforge.net/projects/avetanabt/`, December 2005

[15] PostgreSQL: Open source relational database system
`http://www.postgresql.org/`, December 2005

[16] JDBC: Java Database Connectivity
`http://java.sun.com/products/jdbc/`, December 2005

[17] Java 2 Platform, Micro Edition (J2ME)
`http://java.sun.com/j2me/`, December 2005

[18] Java 2 Platform, Standard Edition (J2SE)
`http://java.sun.com/j2se/`, December 2005

[19] Apache2 - Webserver
`http://www.apache.org/`, December 2005

[20] PHP - Hypertext Preprocessor
`http://www.php.net/`, December 2005

[21] A list of which phones have which technologies implemented can be found at:

- `http://www.javabluetooth.com/jsr82devices.html`, December 2005
- `http://www.benhui.net/modules.php?name=Midp2Phones`, December 2005

The latter has more recent updates.