Design and implementation of location based services on a PDA with MIPv6

Vredeveld, D.

Award date:
2002

Design and implementation of location based services on a PDA with MIPv6

M.Sc. thesis

By D. Vredeveld

Dennis Vredeveld
IMST GmbH
Kamp-Lintfort, Germany, July 2001 – April 2002
Supervisors:
Ir. M.B. Hijdra (IMST GmbH)
Prof. ir. A.M.J. Koonen (TU/e)
Ir. J.J.B. Kwaaitaal (TU/e)

Abstract

This report has been written to finish the activities performed as a graduation project at IMST GmbH, located in Kamp-Lintfort, Germany. This nine-month project is the final part of my study at the faculty of electrical engineering at the Eindhoven University of Technology (TU/e). It has been performed under supervision of ir. M.B. Hijdra of the group Communication Systems and Networks (CSN) of the Systems department of IMST. The supervisors at the TU/e were prof. ir. A.M.J. Koonen and ir. J.J.B. Kwaaitaal of the department Telecommunication Technology and Electromagnetism (TTE).

Future wireless networks are likely to offer high-bandwidth communication possibilities to their users. In addition, it will become possible to determine the position of the user accurately, either through network services, or by means of additional sensing devices. These characteristics of future mobile environments will make it possible to cope with the growing demand for location based services. The IMST ISIDOR project researches the possibilities of this new type of service.

The assignment's main target is the design and implementation of location based services, based on a Personal Digital Assistant (PDA). To achieve this, the PDA has to be equipped with a suited operating system supporting mobile IPv6, as well as a suited wireless access technology. The most suited operating system turns out to be the Linux Familiar distribution. As wireless access technology, wireless LAN is chosen. Support for Bluetooth is planned, but currently not yet feasible. Next step is to investigate a client-server implementation of location based services, based on the techniques stated above. This requires three entities, each consisting of an application and an interface to the other entities. These are the home agent, the database server and the mobile node. The database server store all information available in the location based services network and should be capable of handling multiple connections simultaneously. By monitoring packets sent and received at the interfaces of the database server, information on its performance can be acquired. To complete these measurements, the round trip delays between the different entities have to be estimated first.

Throughout this report it becomes clear that the basic functionality of Linux with MIPv6 on a PDA is good, as the essential requirements with respect to mobility and connectivity are all met. The interaction between mobile nodes and the database server functions as expected and generation of notifications and response messages by the database server is fast, compared to the other two entities. However, there are a lot of recommendations to improve speed and efficiency of the applications. Once these have been implemented, scaling to a larger LBS network should be possible.

Contents

List of figures

List of tables

Preface

Completing a task like a graduation assignment is never easy. It is almost always the first time one has to deal with an assignment of this magnitude, both in duration and complexity. At first it is important to develop a general view on the assignment and the work it involves. This stage is followed by a period where literature and other information have to be gathered in order to successfully accomplish the rest of the assignment. The final period is always more stressful than anticipated, but it is very satisfactory having finished this period.

Working in an environment where both hardware and software configuration are experimental is usually a guarantee for unknown and unexpected problems. This can be both very educational and frustrating. Luckily, the first feeling has dominated during the nine months of my assignment.

This would never have been the case without proper supervision. So, I would like to thank the Communication Systems and Networks (CSN) group of the IMST for giving me the opportunity to do my graduation project here and giving me an insight in their daily business. In particular, my supervisor ir. Martijn Hijdra has done an excellent job in giving me all the information I needed to complete my assignment and in assisting me whenever there were problems I was not able to solve on my own. Finally, I would like to thank ir. Kwaaitaal and prof. ir. Koonen for being my supervisors from the Eindhoven University of Technology.

Kamp-Lintfort, Germany, April 2002.
Dennis Vredeveld.

Chapter 1. Introduction

This report is written to inform on the activities of my master's thesis, performed at IMST GmbH, located in Kamp-Lintfort, Germany. IMST is a research institute on mobile and satellite communication and was founded in 1992. At the department Communication Systems and Networks (CSN) research on location based services (LBS[1]) and wireless access technologies is done.

It is widely anticipated that future mobile users will have access to high-bandwidth wireless communication systems. This can be realised in many forms, for instance by means of third generation (3G) public networks or by means of private access points based on technologies such as IEEE 802.11. Irrespective of the exact implementation, the overall effect will be that new classes of communication-oriented mobile applications will be enabled.

In parallel with the emergence of these high-bandwidth wireless communication systems, there will be a growing demand for new types of services. Location based services form such a new type of service. With LBS, the delivered content depends on the location of the user. This can be, for instance, a guide through a museum. The user uses a wireless and handheld device to retrieve and view this content. Depending on the user's location within the museum, he receives information concerning his current position. This information can also be used for navigation through the museum.

The ISIDOR project is one of the projects within the CSN group, in which LBS are researched. ISIDOR stands for *Information Systems Improved by location Dependent Services Over Radio*. The project's goal is to implement location based services. It uses standards like mobile IPv6 (MIPv6) as network protocol and wireless LAN (WLAN[2]) and Bluetooth for wireless communication. Applications are implemented in Java to ensure portability.

In most LBS implementations presented so far, the mobile nodes have no knowledge of the location of other mobile nodes. In the ISIDOR project, we want to provide this information to all mobile nodes present in the network. This enables them to contact each other based on their location. Different infrastructures can be used to implement this functionality.

This report describes the most important aspects regarding location based services with respect to the ISIDOR project's implementation. First, a closer look at the project's background and the assignment are given, followed by the most important aspects of location based services, as well as three possible service architectures to implement these. Chapter 4 gives some background information on mobile IPv6. Next, the relevant

[1] Instead of LBS, the terms 'Location Dependent Services' (LDS) and 'Location Aware Services' (LAS) are also used. They refer to the same concepts. The term LBS is currently the most common one and will therefore be used throughout this document.

[2] Wireless LAN (WLAN) cards are also known as WaveLAN cards. Strictly speaking, the last term is the name of a former Lucent product line. However, both terms are used in this document.

information of MIPv6 with respect to the Linux environment is given. The configuration of the handheld devices is also discussed in this Chapter. In Chapter 6, the communication protocol and the actual implementation are discussed. The measurements performed in order to retrieve some information on the functionality of the implementation can be found in Chapter 7. Finally, the general conclusions and recommendations are presented in Chapter 8.

Chapter 2. Project description and assignment

This chapter gives a short description of the graduation assignment as it was mutually agreed on before its start in July 2001, together with some information and the status of the project it belongs to.

2.1 The ISIDOR project

This section describes the ISIDOR project and its status at the start of my graduation project. All the work described in this section has been done before the start of my assignment and serves as background information, as well as illustration of the project's requirements. This information forms the starting point of my assignment, which will be described in section 2.2.

2.1.1 Location based services

In the past few years, telecommunication has found its way into the mass market. The enormous growth of mobile phone purchases illustrates this fact. Now that mobile speech communication has become common wealth, the demand for data communication is also likely to take a leap. This is already being indicated by the success of Short Message Service (SMS). New techniques and protocols open the way to mobile multimedia communication, for instance with the new Universal Mobile Telecommunication System (UMTS). Location based services may be an important part of the newly developed services and platforms for mobile multimedia communication [1].

The combination of wireless multimedia communication with a handheld device makes it possible to receive any information at any place, on any time. Especially the services that depend on the physical position of the user are believed to have great potential [1]. As can be seen in Figure 2-1, three out of the four most wanted services by mobile customers are location based.

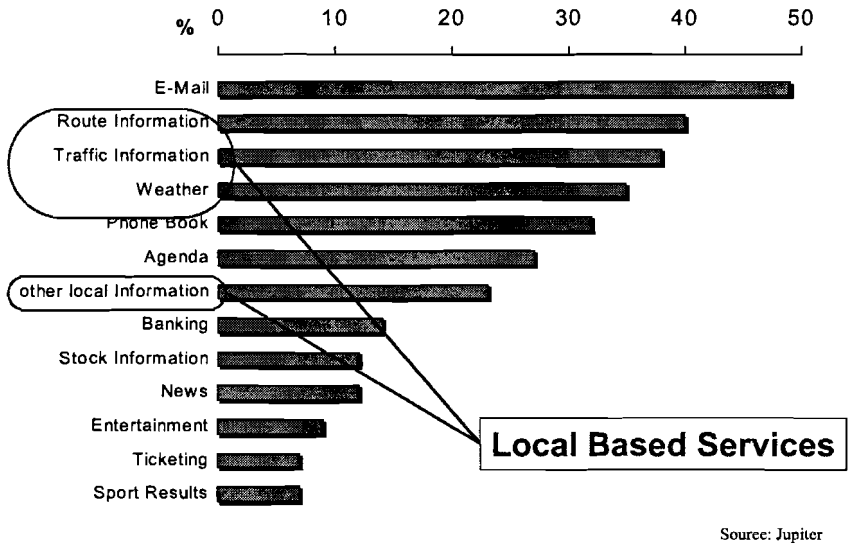


Figure 2-1. The wishes of users of mobile services.

2.1.2 ISIDOR project background and requirements

2.1.2.1 Project background

The aim of the ISIDOR project is the development of a scalable 'Information System Improved by location Dependent services Over Radio' for different environments. The project runs in the section Communication Systems and Networks (CSN) of IMST[3]. Its target is the provision of location based information to spontaneous user groups, e.g. in exhibitions. Instead of proprietary solutions, publicly available standards like mobile IPv6, XML (Extensible Mark-up Language) and Java are used in the ISIDOR project.

To achieve this target, a demo application of location based services will be developed (in Java). One of the project's innovative concepts is the support of distributed spontaneous communities instead of centralised service providers. This enables every node to exchange information with every other node, based on its position. Traditional centralised concepts only offer information regarding the mobile node itself; no information on the network structure and its users can be obtained. The location information and services run on top of IP and are based on cell identifiers of the access technology used. The service provisioning is also based on this identification. Connectivity must be guaranteed during a change of cell (a hand over) and current connections should not be interrupted.

2.1.2.2 Mobile device requirements

Location based services are most valuable when the user and his communication device are mobile. The communication device can be any portable, wireless and handheld device, for instance a cell phone or notebook. For the ISIDOR project, a Personal Digital Assistant (PDA) is used as mobile device. A PDA is, in general, smaller than a notebook or laptop, but bigger than a mobile phone. It lacks a keyboard. Mostly, a touchscreen is

[3] See www.imst.de/mobile/section/com-sys.htm for more information.

used for input. Its display is considerably larger than that of a mobile phone and usually displays an image at a higher resolution with more colours.

The PDA of our choice is the Compaq iPAQ 3630. It contains 16 MB of Flash ROM and 16 or 32 MB of RAM. Its 320*240 screen is capable of displaying 16 million colours. It can be expanded by a (optional) sleeve (a so-called backpaq), which contains an extra battery and a PCMCIA slot. This PCMCIA slot can be equipped with a wireless LAN (IEEE 802.11b, to be more specific) or Bluetooth card. See Figure 2-2 for a picture of the iPAQ 3630 with a wireless LAN expansion card.

Figure 2-2. The Compaq iPAQ 3630 with WLAN card.

The iPAQ PDA is out of the box equipped with the Windows CE operating system (OS). While this is sufficient for common applications, it is not the operating system of our choice when it comes to implement and develop location based services. Main reason for this is the fact that Windows CE is not open source software, restricting configuration possibilities and collective development of, for instance, mobile IPv6 support. This leads to the first task of the assignment, which consists of choosing, installing and configuring an OS (and possibly a special distribution) that supports mobile IPv6, combined with a suited wireless technology. The choice of wireless technology will be discussed in 2.1.3. The choice of the OS is mostly hardware dependent. For that reason, every PDA with the same hardware as the iPAQ 3630 can be used. In practice, the CPU (in case of the iPAQ an Intel StrongARM at 206 MHz.) is the most important factor.

2.1.2.3 Java

The implementation of a location based service will be done in Java. Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler

to use than C++ and enforces an object-oriented programming model. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build a small application module or applet for use as part of a Web page. Applets make it possible for a Web page user to interact with the page. The following characteristics of Java are important for the ISIDOR project:

- The created programs are portable in a network. The source program is compiled into so-called *bytecode*, which can be run anywhere in a network on a server or client that has a Java Virtual Machine (JVM). The JVM interprets the bytecode into code that will run on the real computer hardware. This means that individual computer platform differences such as instruction lengths can be recognized and accommodated locally just as the program is being executed. Platform-specific versions of program are no longer needed. This means no recompilation is necessary to run a developed Java application on the iPAQ.
- In addition to being executed at the client rather than the server, a Java applet has other characteristics designed to make it run fast. This is important, as the hardware of a PDA is relatively slow compared to modern personal computers.

2.1.2.4 The WINEGLASS project

The ISIDOR project runs time wise concurrently with a bigger, European project called WINEGLASS. This stands for *Wireless IP Network as a Generic platform for Location Aware Service Support.* This project started in 1998 and is finished in Q1 2002. The WINEGLASS project researches the possibilities of using IP-based wireless mobile multimedia with UMTS and wireless LAN and involves several companies. The emphasis lies on suitable (software) platforms for location based services. Mobile IPv6 is one of the issues considered within this project.

The ISIDOR project only copes with the private or small area scope, while the WINEGLASS project focuses on the bigger environment. The private scope means that the location based services will only be meaningful in certain designated places, for instance between the walls of a museum. The scope of the public area however, is often only limited by the reach of the chosen transmission method, like the coverage of a GSM[4] network. This can be seen in Figure 2-3, together with some examples of possible applications.

In the case of the ISIDOR project, the term "location" has a geographical, as well as a content-related meaning. It is geographical in the sense that it is only available in a certain and fixed area, for instance a museum or exhibition. In that case, the content loses its meaning once the user leaves the museum or exhibition. The term "location" is also content-related, as it offers services that are only available within this area. In this case, the location can be seen as a community, formed by a group of users using the same services. Whenever the community physically moves, the area in which their services are available moves as well.

[4] Global system for mobile communication.

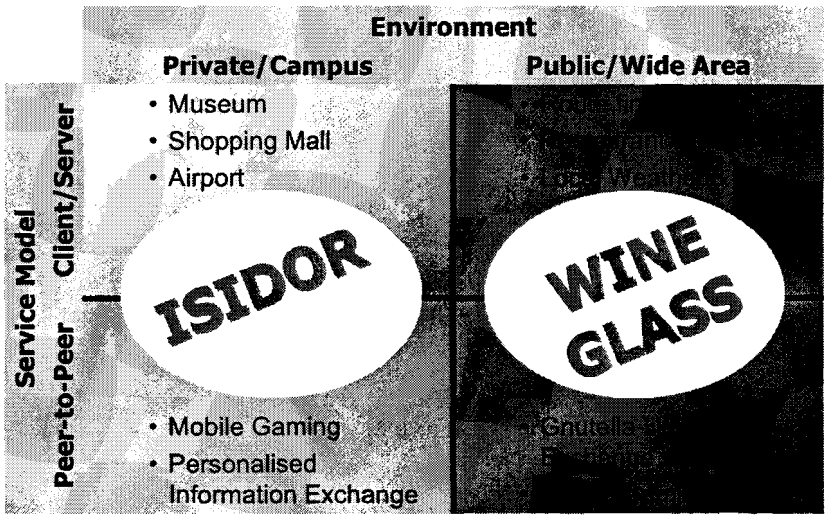


Figure 2-3. The different forms of location based services within IMST.

2.1.3 Wireless network choice

As the ISIDOR project is limited to a small and well-known area, location determination will not be done by means of technologies used in wide-area networks (WANs) such as GSM or the Global Positioning System (GPS). Instead, WLAN or Bluetooth is used to determine the user's location. This has several reasons:

- First of all, the location determination of wide area networks is usually not as accurate as in local area networks (LANs). The better the determination of location takes place, the more the offered services can be focused on this location. With WLAN the location can be determined with an accuracy of approximately 10^2 meter[5]; with Bluetooth 10^1 meter can be achieved. With GSM this kind of accuracy is only possible with use of triangulation techniques. These techniques are not part of the core network and have to be implemented by the user. GPS does not have this limitation, but because of the large distance to the transmitter, it is virtually impossible to determine locations in the z-direction. However, this can be necessary to differentiate the floors of a building.
- Another disadvantage of wide area networks are the variable costs. As these networks mostly have an operator, access is charged. This is done either by connection time or by amount of transmitted data. With WLAN of Bluetooth, no extra costs are involved once the private infrastructure is present.
- Finally, there is the problem of quality of services. Within a local LBS network, the infrastructure can be adapted and extended to the amount of users to ensure connectivity and bandwidth requirements. This is not possible with national or global networks as the operators control their configuration and quality of service offered. This may not be satisfactory for of the users of the LBS network.

However, there is one problem with the wireless technologies of our choice. The development of Bluetooth implementations has not yet advanced enough to reliably implement a network based on IPv6 with Bluetooth based access points. Therefore,

[5] If location is determined on the basis of the access point the mobile device is communicating with. See section 6.2 for more details on location determination.

wireless LAN is the only wireless technology currently used. Bluetooth will be integrated in a later stadium of the project. The much smaller power consumption of a Bluetooth module, compared to a wireless LAN card, is one of the reasons that this delayed implementation will still be feasible. Another reason is the higher accuracy of the location determination of the Bluetooth technology.

2.1.4 ISIDOR project status

Different devices and operating systems have been used to verify the concept of mobility using the WLAN technology. Three different combinations have been tested: an iPAQ with Windows CE, and a Sony Vaio laptop with Linux and with Windows 98.

In the initial phase of the ISIDOR project, an attempt has been made to use the Dynamic Host Configuration Protocol (DHCP) to develop a demonstration of a location based application. DHCPv4 is used to centrally manage and automate the assignment of unique IPv4 addresses in a network. The tests have been done as follows: the iPAQ or laptop is started in cell one on the first floor. Next, the DHCP negotiation takes place, after which a browser displays a web page containing information regarding the current location. This page automatically refreshes every ten seconds. The next step is to take the device under test to the other cell (the area covered by access point 2) and DHCP negotiation is started again in the background. Normally, the browser times out and a new IP address is assigned by means of DHCP.

The following conclusions can be drawn with respect to the different configurations and operating systems:

- Sony Vaio with Windows 98:
  Windows98 and DHCP with wireless LAN do not work properly. The DHCP negotiation in cell two fails and stops. The only way to complete the DHCP IP address configuration is to remove and insert the network card. This combination is not suited to make a location based system, as there is no cell-change possible without any user intervention;
- Sony Vaio with Linux:
  Linux and DHCP with wireless LAN do not work properly either. Only by removing the WLAN network card and inserting it again, the IP address configuration finishes successfully. So, this combination is also not suited to make a location based system as user intervention is again required for a cell change;
- Compaq iPAQ with Windows CE:
  The basic functionality is working on the iPAQ, but the cell switch time is very poor (over one minute). Good part is that the WLAN card does not have to be removed from the iPAQ. The problem originates from the refresh mechanism: whilst moving to a new cell the DHCP process starts and assign a new IP address to the PDA. When the same page is refreshed from the new location it is not possible to track the user easily because its IP number has changed;

This lead to the conclusion that mobile IP may be a better solution to offer location based services whilst meeting the project's requirements. There are two different versions of mobile IP: v4 and v6, also known as MIPv4 and MIPv6. A number of different implementations of mobile IPv4 exist. They are all based on IPv4 tunnelling. Both Windows and Linux are being used as OS. For links and abstracts of the documentation, see Appendix B. Mobile IPv6 is the successor of mobile IPv4. It offers more functionality, is better integrated with IP itself and is considered to be the mobile Internet protocol of the future. Therefore, mobile IPv6 is chosen for use within the ISIDOR project. This choice is

the point where the first task of the graduation assignment comes in. The assignment will be described in detail in 2.2.

Mobile IPv6 can be used both with Windows and Linux. Drawback of using Windows is the fact that this OS is not open source and therefore no adaptations can be made to the source code. Also, it is more expensive. Linux is freely available and can be customised to one's own needs and wishes. So Linux is the OS of our choice. One of the first tasks to be done is to choose the best Linux distribution to use on the iPAQ. This issue is dealt with in section 5.3. For a more detailed overview of MIPv6, see Chapter 4 or [2].

2.1.5 Summary

In this section, the requirements and status of the ISIDOR project have been described. Some experiments have already been done on various types of mobile devices (iPAQ PDA and Sony Vaio laptop) combined with various operating systems (Windows CE, Windows 98 and Linux). From these experiments can be concluded that a PDA together with mobile IPv6 and the Linux operating system is the preferred way for use within the ISIDOR project. WLAN is currently the wireless technology of our choice; in a later stadium Bluetooth support can be added. Bluetooth has several advantages over WLAN, but its implementations have not developed enough to use in this project from the beginning.

From this point, the first step is to choose and install the most suited Linux distribution for the iPAQ in order to have a mobile node running on MIPv6, with WLAN as wireless technology. This will be described in Chapter 5.

2.2 The assignment

The main target of the assignment is the design and implementation of location based services, based on a mobile device equipped with MIPv6. The mobile device is chosen to be a Compaq iPAQ PDA. The PDA must be equipped with an operating system supporting MIPv6 to develop these services. The choice has been made to use Linux for this task. As a consequence, the iPAQ must be equipped with a Linux distribution supporting MIPv6. This is currently not yet commercially available.

The impact on the Quality of Service (QoS) by scaling to a large network (QoS negotiation) is another aspect that can be researched. In a further stage, integration with the WINEGLASS project (see paragraph 2.1.2.4) can be researched.

2.2.1 Specific tasks

Specific tasks for this assignment are:

- Equip the Compaq iPAQ with Linux and IPv6;
- Extend the Linux kernel with mobile IPv6;
- Research a way to offer location information efficiently through an application program interface (API);
- Research a client-server implementation of location based services. Which part should run on the server and which part can run on the client? What does the data traffic look like on large networks (congestion, traffic control, etcetera)?

- Implement a simple location based service in Java, e.g. a virtual guide through a building or exposition;
- Research how to offer different types of media, for instance text, sound, images and video, on this service platform and describe the so-called service classes;
- Research the hand-over performance;
- Research security aspects in wireless IPv6 networks;

2.2.2 Possible extensions

It may be desirable to expand and/or redirect the assignment. Possible ways to do this are:

- Develop service concepts. How does an efficient location based service look like? What about the decomposition in objects?
- Implement communication from PDA to PDA (peer-to-peer communication) or voice over IP (VoIP). Determine who is at which location and who is registered on the network?
- Integration with the WINEGLASS project;
- Optimise the MIPv6 implementation with, for instance, quick handovers or broadcast timers;
- Research interoperability with other MIPv6 implementations;

Chapter 3. LBS concepts and requirements

3.1 Introduction

The LBS network has two major functions. The first and most essential function is the *provision of mobile communication*. This requires a radio interface that ensures the mobility of the user with as less user interaction as possible. For instance, aspects like hand-over or roaming should be transparent to the user. A hand-over is the process where a mobile node changes his point of attachment from one access router to another. Additionally, there has to be a way to determine a mobile node's current location in a sufficient accurate way. The amount of accuracy required depends on the size and scope of the network. What this means for the ISIDOR project, has been explained in paragraph 2.1.3.

The second function is the *provision of location based services*. The service architecture can be based on different concepts, for instance stand-alone, client-server or peer-to-peer. These concepts will be discussed in section 3.2. In section 3.4 will be evaluated which service model is suited best to be used within the ISIDOR project.

3.2 Service architecture

Three common service architectures (stand-alone, client-server and peer-to-peer) will be explained in more detail in this section.

3.2.1 Stand-alone architecture

Within the *stand-alone* architecture, all information needed for an application (this means all the required software plus all the data related to a certain location) is loaded once onto the mobile node. This has the drawback that the content can easily get out of date, as the information is stored locally. It is also more difficult to load new data into the mobile node. Refreshing the data by means of a software update when the mobile node is at its base location may compensate this shortcoming. A model where mobile devices are regularly updated yields the client-server architecture, which will be discussed in the next paragraph.

The clear advantage of the stand-alone architecture is that there is very little network traffic, as no data has to be transported from a centralized server to the mobile node. A drawback is that the data may not be very recent, thus reducing the use in a very dynamic and content changing environment.

3.2.2 Client-server architecture

In the *client-server* architecture, the content providers distribute all content. The providers act as servers and transfer all the data (content) needed for their services to the client. The platform operator is connected to both the content providers and the clients to administrate the content provision. It is also possible that the platform operator acts as its own content provider. Note that there is no direct interaction between the individual users possible. They may not even know of each other's presence in the same location. See Figure 3-1 for an example of a client-server scenario.

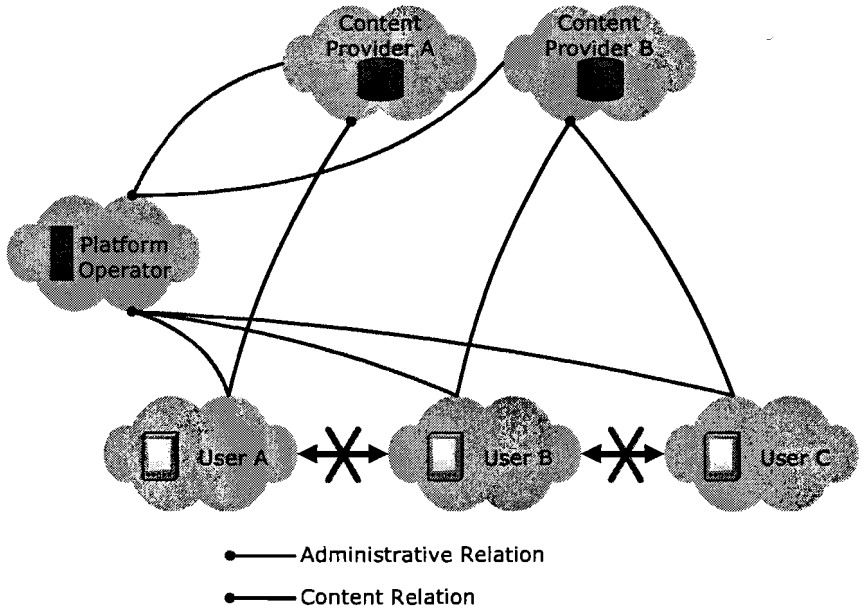


Figure 3-1. Client-server architecture scenario.

This architecture has a clear advantage: all the content is stored in one unique location, making it very easy to maintain and update a service. The drawback is the increased network traffic for transporting the content to the node. A way to reduce this is to cache the content at the mobile node and only fetch data when it is newer or not available in the local cache.

3.2.3 Peer-to-peer architecture

The *peer-to-peer* architecture has a completely different approach. Within this architecture, every user can interact directly with every other user. Services can be exchanged, or a device may serve as a bridge to connect to other devices in a different community. Communities are formed by a group of users and are separated by the size of the radio cells. This way, two different communities can be connected through a common user. The common user hereby acts as a service proxy between the two different communities. In the peer-to-peer scenario, there are no dedicated content providers, nor are there any platform operators. Content can be discovered by lookup actions between two adjacent users. Figure 3-2 gives a graphical representation of a possible peer-to-peer scenario.

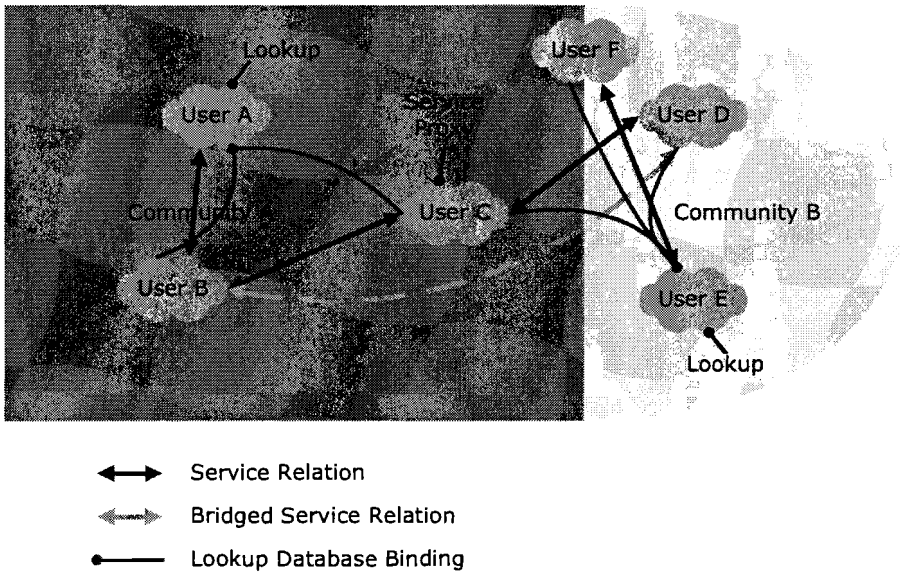


Figure 3-2. Peer-to-peer architecture scenario.

The peer-to-peer variant has currently not yet been implemented in (global) mobile networks, although it is highly flexible and scalable. But it is also more complex, due to its distributed nature. It requires some kind of (automatic) configuration of the present mobile devices. Another reason it is not commonly used, is that it is not as profitable as the client-server model, as the network operator has no direct control over the distribution of the content. Therefore, traffic cannot be billed by means of a central administration. The fact that the amount of research in this area is small compared to the traditional client-server model offers good opportunities to develop a product that is unlike any other seen so far.

3.3 Network topology

3.3.1 Configuration

To be able to test different scenarios regarding IPv6 networks, IMST has its own IPv6 test network. It is used for different projects and purposes within the company:

- The ISIDOR project;
- The WINEGLASS project;
- Competence development within the CSN department;
- Investigation of UMTS IP core network mechanisms;

Figure 3-3 depicts the current configuration of the IPv6 test network. It consists of four access points (that each cover one cell), a router rack and several other routers and switches. The access points are placed on different floors of the building. The routers use a set of protocols to discover and exchange routing information. For IPv4 intra-domain routing, Open Shortest Path First (OSPF) [3] is used. With OSPF, every host obtains a routing table. As soon as it detects a change in the table or in the network, it multicasts this information to all other hosts in the network. This keeps every host's routing table up to date. Unfortunately, OSPF does not work very well with IPv6 and the current router configuration, so the Routing Information Protocol (RIP) [4] is used. Unlike the OSPF

protocol, in which only the part of the routing table that has changed is sent, the RIP sends the entire routing table to all other routers on its subnet on a regular basis.

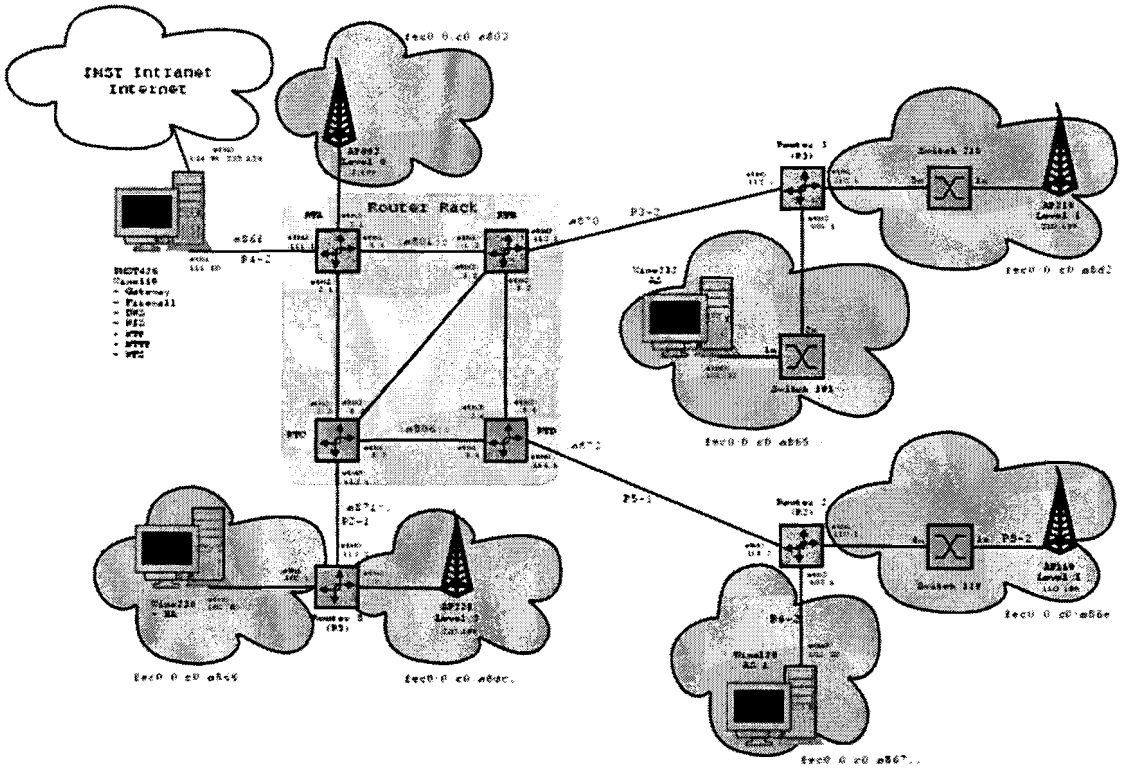
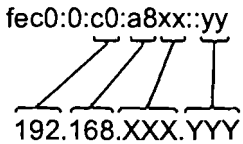


Figure 3-3. IPv6 network topology.

For the IPv6 routers and hosts, the 2.4.0 Linux kernel with a mIPv6 patch has initially been used. For the patch, the MIPL[6] implementation, which is being developed at the Helsinki University of Technology (HUT), is used. It is an implementation of MIPv6 as a Linux kernel module. Kernel 2.4.0 with MPLS[7] is used for the routers in the router rack. Note that all work done on MIPL should be considered work in progress.

3.3.2 Addressing

To have a comfortable addressing scheme, the IPv4 addresses are mapped onto IPv6 addresses. The hexadecimal IPv6 addresses starting with "fec0:0:c0:a8xx:" are (local) subnet prefixes. The 4-byte IPv4 address can be mapped onto the 16-byte IPv6 version in the following way:



[6] More information on MIPL can be found on www.mipl.mediapoli.com.

[7] Multiprotocol label switching.

Where XXX is the decimal equivalent of the hexadecimal digit xx. This number indicates the cell. Likewise, YYY is the decimal equivalent of yy and is used to uniquely identify each interface. For details on IPv6 addressing, see [5].

3.4 Evaluation

All the architectures described in paragraph 3.2.2 through 3.2.3 have advantages as well as disadvantages. The stand-alone architecture is not flexible enough in a LBS network. More frequent updates are required, which leads to the client-server architecture. Within this architecture it is relatively easy to maintain and update the services and the contents, depending on the amount of centralisation. Biggest disadvantage is the lack of direct communication possibilities between the different mobile nodes, which is at its turn the major advantage of the peer-to-peer architecture. One of the project's requirements was that the user should be able to retrieve information on each other's presence. This makes the traditional client-server architecture not suited for usage within ISIDOR project.

When we look at the network topology, we see that with WLAN it is not possible for a node to communicate in two different cells simultaneously. Though a node can have one primary and multiple other care-of addresses on the network layer, it can use only one uplink at a time. So the concept of service proxy is therefore at a physical level not possible. This may change when Bluetooth is used as access technology, because a Bluetooth node is capable of joining different cells (called piconets) quasi-simultaneously[8]. Connecting different piconets yields a so-called scatternet. With scatternets, one can create a virtually unlimited network. However, this also introduces different routing problems for which currently no standards are available. As concluded in 2.1.3, the combination of Linux on an iPAQ, together with MIPv6 and Bluetooth support is not feasible.

After evaluating the different approaches discussed in section 3.2, it is best to use some aspects of all models. More specifically, this would look like a client-server model with content distribution and with the possibility to retrieve information on the current network topology and the other nodes in the network. The available information may be limited to the users in their own cell and/or a fixed amount of users they are able to trace at any moment and place. Due to the mentioned limitations of MIPv6 over wireless LAN, the service proxy concept cannot be used to combine these wishes. So, what we eventually need is a centralised entity (a server) that takes care of the distribution of all the mobile node information. This information can be stored in a database on the server. Next step may then be to extend the client-server implementation with peer-to-peer communication possibilities between mobile nodes.

To develop and test such a concept, the IMST IPv6 network should be used. Therefore, all needed elements have to be divided into essential building blocks. These are:

- Home agent(s), for a correct functioning of mobile IPv6;
- Mobile node(s), to ensure mobility of the user;
- A database server, to centrally store all relevant mobile node and network information;
- A content server, to provide location based content to the mobile nodes;

[8] This is not completely simultaneous, as it only has certain time slots available for sending and receiving data in each cell it is a member of. The master of the piconet manages the assigning of time slots among its slaves.

Each of these building blocks –or entities- consists of an application and an interface to one or more of the other entities. The application will be implemented in software. Because of the fact that the implementation of location based content is time-consuming and beyond the scope of this thesis, it will not be discussed.

3.5 Service mapping

A mobile IPv6 LBS platform with different entities as described in 3.4 can be mapped on the network topology in the way shown in Figure 3-4. The elements are connected to each other by using the mobile IPv6 network and WLAN access points.

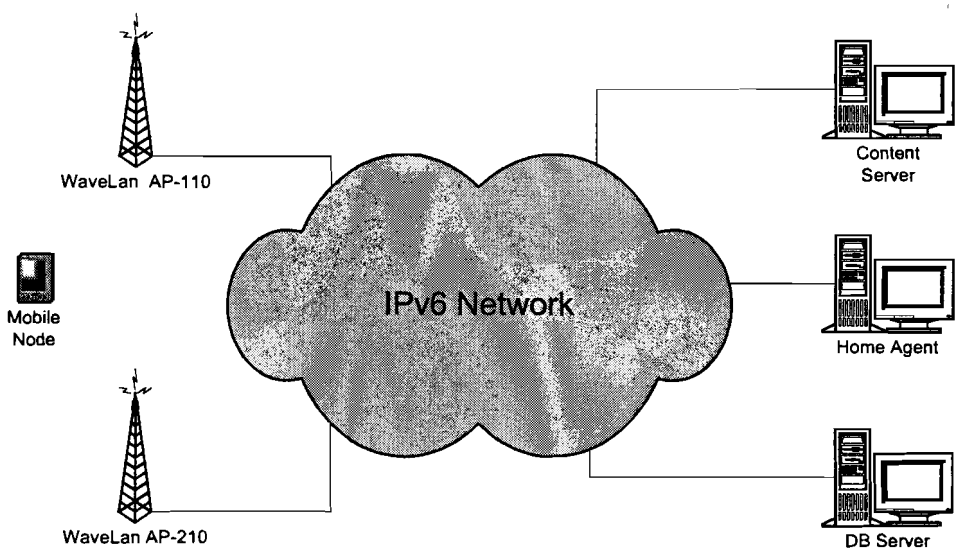


Figure 3-4. Mobile IPv6 LBS topology.

The next Chapter provides more detailed information on the functioning of mobile IPv6. The mobile node's configuration is explained in Chapter 5. Chapter 6 and Chapter 7 describe a way to implement and test the database server, as this is the most important and complex entity. This is because of the fact it has to be able to interface with the other entities simultaneously.

Chapter 4. Overview of mobile IPv6

This section provides some background information on the working and origin of the mobile IPv6 protocol. As it is still under development, most documentation on this protocol is provided in so-called Request For Comment (RFC) documents[9], implying that all information regarding this topic is subject to change. However, it is quite safe to state that the major concepts described in this Chapter are quite definite and only details will be adjusted prior to the release of the final MIPv6 specification.

4.1 History of IPv6

IPv6 is the successor of IPv4, which is still widely used today. IPv4 was developed in a time where the Internet was mainly used by universities, the military and the government. But this group is getting bigger and bigger. More people make use of the Internet every day and they have different needs compared to the early users. People want to be mobile while being connected. And by the convergence between computer, amusement and communication equipment, virtually every device may be an Internet node in the near future. This varies from a cellular phone to a television and from a handheld PC to a fridge. This means billions of new IP addresses, which is more than IPv4 can offer.

It is mainly because of these circumstances that IPv4 is not sufficient anymore. The IETF got aware of these problems around 1990. They started to work on a newer version of IP. Their targets were [6]:

1. Be able to work with billions of hosts, even with an inefficient addressing technique. Therefore, an IPv6 address consists of 16 bytes, instead of 4 bytes in the case of IPv4;
2. Reduce the size of the routing tables by using headers with a fixed size;
3. Make the protocol less complex, to enable routers to process packets faster. This was, amongst others, achieved by reducing the number of fields in the header and by offering more options;
4. Take better care of security (authentication and privacy) compared to IPv4;
5. Give more attention to service types, especially to real-time data;
6. Make multicasting easier by allowing scopes to be specified;
7. Make it possible for a host to travel without changing its address;
8. Allow the protocol to develop itself in the future;
9. Make it possible to co-exist with the previous version.

As stated in point 3, the header of IPv6 is less complex than its IPv4 predecessor, though it is longer. One of the most important differences is that the *options* field is dropped in IPv6. It is infrequently used in IPv4 because it introduces some complications. The options field increases the packet's processing time, as each router must process all of the options before forwarding the packet. Therefore, the extension header mechanism was developed for IPv6 to satisfy the need to specify options more efficiently. Because of

[9] RFC is a specification by the Internet Engineering Task Force (IETF).

these extension headers, only the IPv6 header with a fixed size of 320 bits (40 bytes) has to be processed by routers. The routers consider the extension headers as part of the packet payload and will therefore not analyse them. This improves forwarding performance. See Figure 4-1 for a comparison between the header of IPv6 and IPv4.

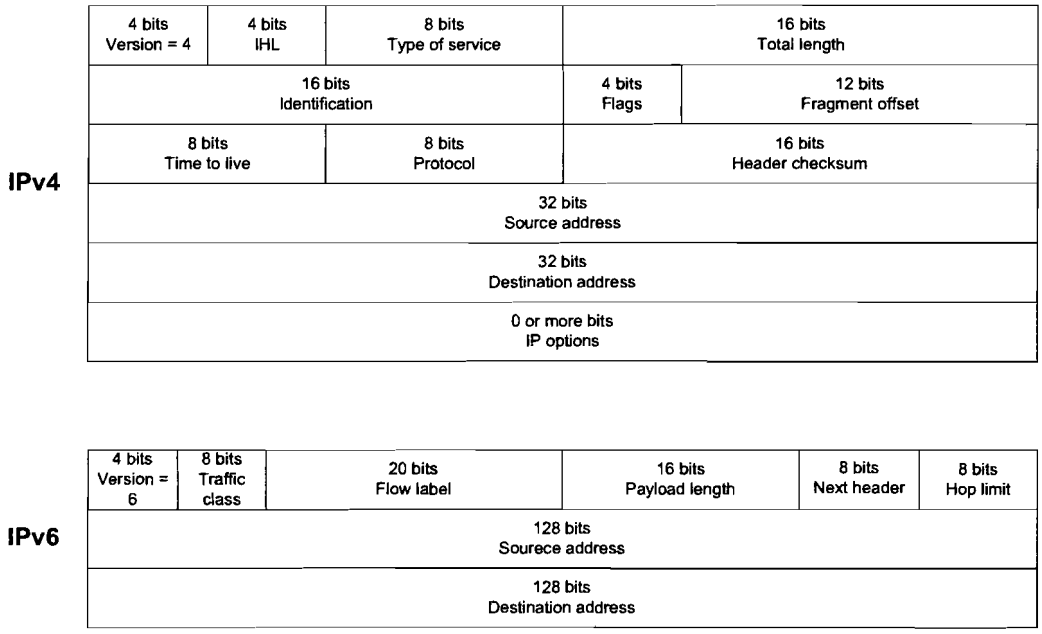


Figure 4-1. IPv6 header compared to IPv4 header.

In IPv6, each extension header start with a *next header* field to indicate which type of extension header follows. This forms a so-called *header-chain*. These headers have a fixed order, which can be seen in Figure 4-2. More information on extension headers can be found in [7].

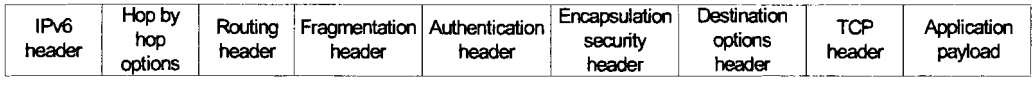


Figure 4-2. IPv6 header extension format.

4.2 Mobility support

As stated in the previous section, mobility support for Internet devices is important. This will probably only become more important in the future [1]. But roaming with mobile devices yields several complications. The problem lies in the current Internet routing mechanisms. It starts when a mobile node is disconnected from the Internet and tries to connect again in a different subnet. This is only possible after the user has configured the system with a new IP address, the correct net mask and a new default router. There is, more or less, a topological relation in the IP addresses of linked computers. It is implicitly assumed that every node always has the same point of attachment to the Internet. Furthermore, the node's IP address indicates the link on which the node resides. So, if a node moves without changing its address, existing routing protocols will not be able to deliver packets correctly. To solve this problem, mobile IP has been developed.

Both IPv4 and IPv6 have mobility support, but it in case of IPv4 it is not fully integrated into the protocol itself. Mobility support for IPv4 is defined in [8]; for the IPv6 version, see [2].

4.3 Basic operation of MIPv6

A mobile node is always addressable by its *home address*, whether it is currently attached to its home link or is away from home. While a mobile node is at home, packets addressed to its home address are routed using conventional Internet routing mechanisms in the same way as if the node were never mobile.

While a mobile node is attached to a foreign link away from home, it is also addressable by one or more *care-of addresses*, in addition to its home address. A care-of address is an IP address associated with a mobile node while visiting a particular foreign link. The subnet prefix of a mobile node's care-of address is the subnet prefix (or one of the subnet prefixes) on the foreign link being visited by the mobile node. If the mobile node is connected to this foreign link while using that care-of address, packets addressed to this care-of address will be routed to the mobile node in its location away from home.

The association between a mobile node's home address and care-of address is known as a *binding* for the mobile node. A mobile node typically acquires its care-of address through either stateless [9] or stateful (e.g. DHCPv6) address auto configuration, according to the methods of *IPv6 Neighbour Discovery* [10]. Other methods of acquiring a care-of address are also possible, such as static pre-assignment by the owner or manager of a particular foreign link, but details of such other methods are beyond the scope of this thesis.

While away from home, a mobile node registers one of its care-of addresses with a router on its home link, requesting this router to function as the *home agent* for the mobile node. This binding registration is done by sending a packet containing a *binding update* from the mobile node to the home agent. The home agent replies to the mobile node by returning a packet containing a *binding acknowledgement*. The care-of address in this binding, registered with its home agent, is known as the mobile node's *primary* care-of address. The mobile node's home agent thereafter uses *proxy neighbour discovery* [10] to intercept any IPv6 packets addressed to the mobile node's home address on the home link. Next, it tunnels each intercepted packet to the mobile node's primary care-of address. To do this, the home agent encapsulates the packet using IPv6 encapsulation [11] with the outer IPv6 header addressed to the mobile node's primary care-of address.

When a mobile node changes from one care-of address to another on a new link, it is desirable for packets arriving at the previous care-of address to be tunnelled to the mobile node's new care-of address. Since the purpose of a binding update is to establish exactly this kind of tunnelling, it is specified to be used (at least temporarily) for tunnels originating at the mobile node's previous care-of address in exactly the same way that it is used for establishing tunnels from the mobile node's home address to the mobile node's current care-of address.

It may be desirable for a mobile node to use more than one care-of address at the same time. However, a mobile node's primary care-of address is distinct among these in that the home agent maintains only a single care-of address registered for each mobile node, and always tunnels a mobile node's packets intercepted from its home link to this mobile

node's registered primary care-of address. The home agent thus need not implement any policy to determine the particular care-of address to which it tunnels each intercepted packet. The mobile node alone controls the policy by which it selects the care-of addresses to register with its home agent.

It is possible that while a mobile node is away from home, some nodes on its home link may be reconfigured, such that the router that was operating as the mobile node's home agent is replaced by a different router serving this role. In this case, the mobile node may not know the IP address of its own home agent. Mobile IPv6 provides a mechanism, known as *Dynamic Home Agent Address Discovery* (DHAAD) [7], that allows a mobile node to dynamically discover the IP address of a home agent on its home link with which it may register its (primary) care-of address while away from home. The mobile node sends an *Internet Control Message Protocol* (ICMP) [12] "home agent address discovery request" message to the mobile IPv6 home agents anycast address for its own home subnet prefix [13] and thus reaches one of the (possibly many) routers on its home link currently operating as a home agent. This home agent then returns an ICMP "home agent address discovery reply" message to the mobile node, including a list of home agents on the home link. This list of home agents is maintained by each home agent on the home link through use of the home agent (H) bit in each home agent's periodic unsolicited multicast router advertisements.

4.4 IPv6 destination options

As mentioned before, new IPv6 destination options are defined for mobile IPv6. They will be briefly described in this section.

4.4.1 Binding update

The mobile node uses a binding update option to notify a correspondent node or the mobile node's home agent of its current binding. Together with the binding acknowledgement and binding request options, it is used to allow IPv6 nodes communicating with a mobile node, to dynamically learn and cache the mobile node's binding. The binding update sent to the mobile node's home agent to register its primary care-of address, is marked as a *home registration*. Any packet that includes a binding update option MUST[10] be protected by some authentication data to guard against malicious binding updates. This is very important, as malicious binding updates could be used to intercept packets meant for the mobile node.

4.4.2 Binding acknowledgement

The binding acknowledgement option is used to acknowledge receipt of a binding update, if such an acknowledgement was requested in the binding update. Any packet that includes a binding acknowledgement option MUST be protected by some authentication data to guard against malicious binding acknowledgements.

[10] Words in this document written in small caps are compliant to [14].

4.4.3 Binding request

The binding request option is used to request a mobile node to send to the requesting node a binding update containing the mobile node's current binding. A correspondent node typically uses this option to refresh a cached binding for a mobile node, when the binding is in active use but its lifetime is close to expiration. No authentication is required for the binding request option.

4.4.4 Home address

The home address option is used in a packet sent by a mobile node to inform the recipient of that packet of the mobile node's home address. For packets sent by a mobile node while away from home, the mobile node generally uses one of its care-of addresses as the source address in the packet's IPv6 header. By including a home address option in the packet, the correspondent node receiving the packet is able to substitute the mobile node's home address for this care-of address when processing the packet, thus making the use of the care-of address transparent to the correspondent node. If the IP header of a packet carrying a home address option is covered by authentication, then the home address option MUST also be covered by this authentication, but no other authentication is required for the home address option.

One reason to use this option is that many routers implement security policies that do not allow forwarding of packets that have a source address, which appears topologically incorrect. By using the care-of address as the IPv6 header source address, the packet is able to pass normally through such routers, yet filtering rules will still be able to locate the true topological source of the packet in the same way as packets from non-mobile nodes. By also including the home address option in each packet, the sending mobile node can communicate its home address to the correspondent node receiving this packet, allowing the use of the care-of address to be transparent above the mobile IPv6 support level (e.g., at the transport layer).

Chapter 5. Mobile IPv6 for Linux

5.1 Introduction

This Chapter explains important aspects of MIPv6 in Linux and describes some tools to configure and monitor an IPv6 network. In contradiction to the iPAQ, a Linux MIPv6 distribution for desktop PC's and laptops is already available. For the iPAQ, a different Linux distribution must be used, as will be described in 5.3. To make the choice for a distribution, several aspects are considered, for instance Java and WLAN support.

5.2 Linux system files and tools

Based on this configuration, several tools and commands can be used to monitor the existing links and to configure all the necessary IPv6 parameters of a node. The actual MIPv6 configuration and status can be read from the Linux `/proc` file system. This section discussed the most important files and tools for this purpose.

5.2.1 Kernel system files

For a home agent:

- `/proc/net/mip6_bcache` contains the home agent's binding cache;
- `/proc/net/mip6_home_agents` contains the home agent's home agents list;
- `/proc/net/mip6_stat` contains home agent statistics, like the number of binding messages sent and acknowledged;

For a mobile node:

- `/proc/net/mip6_bcache` contains the mobile node's binding cache;
- `/proc/net/mip6_bul` contains the mobile node's binding update list (see end of paragraph for more information);
- `/proc/net/mip6_stat` contains different mobile node statistics;

For a correspondent node:

- `/proc/net/mip6_bcache` contains the correspondent node's binding cache;
- `/proc/net/mip6_stat` contains different correspondent node statistics;

Configurable parameters for a home agent:

- `/proc/sys/net/ipv6/mobility/debuglevel`
  contains the home agent's debug level;

Configurable parameters for a mobile node:

- `/proc/sys/net/ipv6/mobility/debuglevel`
  contains the mobile node's debug level
- `/proc/sys/net/ipv6/mobility/home_address`
  contains the mobile node's home address;
- `/proc/sys/net/ipv6/mobility/home_agent_address`
  contains the mobile node's home agent's address;

Configurable parameters for a correspondent node:

- `/proc/sys/net/ipv6/mobility/debuglevel`
  contains the correspondent node's debug level;

The *binding cache list* is maintained by each IPv6 node and contains bindings for other nodes. It contains, amongst others, the following fields for each binding:

- The home address of the mobile node for which this is the binding cache entry. This address is used as the key for searching the binding cache when routing packets. If the destination address of a packet matches a home address in the entry, this entry SHOULD be used whilst routing the packet. This means that the corresponding care-of address is used in a routing header, as described in section 4.1;
- The care-of address of the mobile node with the home address of the previous point;
- A lifetime value, indicating the remaining lifetime for this entry. When this lifetime expires, the entry MUST be deleted from the binding cache, as this binding is not valid anymore;
- Two flags, indicating whether or not this entry is a home registration entry, and whether or not the entry represents a mobile node that should be advertised as a router in proxy neighbour advertisements [10];
- The binding security association (BSA) to be used when: 1. Authenticating binding updates that are received for this entry and 2. Calculating authentication data for inclusion in binding acknowledgements in response to binding updates that are received for this entry;

Each mobile node maintains a *binding update list*. It records information for each binding update sent by this node, if and only if the lifetime has not yet expired. It is used by the Mipdiag tool, which will be described in paragraph 5.2.2.1.

5.2.2 Network and IPv6 tools

The following tools can be used to configure and monitor the mobile IPv6 network.

5.2.2.1 Mobile IPv6 diagnostics and configuration tool

The mobile IPv6 diagnostics and configuration tool (*Mipdiag*) for the mobile IPv6 kernel module gets statistics and state information and sets runtime parameters. It can also show the binding cache of an IPv6 node and the binding update list of a mobile node. Furthermore, it can be used to show and modify several mobile IP runtime parameters, such as the mobile node's home (agent) address.

Synopsis:

```
Mipdiag[-acls]  [-M keymd5]  [-S keyshal]
        [-h ipv6-address/prefix-length]
        [-H ipv6-address/prefix length]  [-d integer]
        [-r access-control-rule]  [-R file-name]  [-t boolean]
```
[11]

5.2.2.2 IPv6 router advertisement daemon

The IPv6 router advertisement daemon (*radvd*) sends router advertisements as described in [10]. With these advertisements, hosts can automatically configure their addresses and some other parameters. They can also choose a default router, based on these advertisements.

Synopsis:

```
Radvd  [-vh]  [-d debuglevel]  [-C configfile]  [-m method]
       [-l logfile]  [-f facility]
```

5.2.2.3 IPv6 router advertisement trace tool

The IPv6 router advertisement trace tool (radvdump) is a tool to trace and dump router advertisement packets.

Synopsis:

```
Radvdump    [-vh]  [-d debuglevel]
```

5.2.2.4 Network traffic monitoring (Ethereal)

Ethereal is used to interactively browse network traffic on any chosen network interface. For instance, if we look at the interface of the router that is connected to the access point of level 2 (this is eth1 of Router2, see Figure 3-3 for details), we can exactly monitor what packets are between this AP and the router and who are the source and destination of these packets. By applying a filter, only a distinct number of protocols can be made visible. In the case of Figure 5-1, only IPv6 packets are shown. This can be seen by the *ipv6.version == 6* statement in the bottom of the screen. Furthermore, each shown packet is automatically decomposed into the different parts (i.e., headers and payload). Of each part shown, the hex dump is made visible in the bottom window. This makes it very easy to track what is happening on the link.

Synopsis:

```
ethereal    [-B byte view height]  [-c count]
        [-f filter expression]  [-h]  [-i interface]  [-k]  [-m font]
        [-n]  [-o preference setting]  [-p]  [-P packet list height]
        [-Q]  [-r infile]  [-R filter expression]  [-S]  [-s snaplen]
        [-T tree view height]  [-t time stamp format]  [-v]
        [-w savefile]
```

See Figure 5-1 for a screenshot of Ethereal. This shows a trace of the eth1 interface of Router 3, as the laptop was taken from level one to level two.

[11] The –M and –S options may not be available in older versions of Mipdiag.

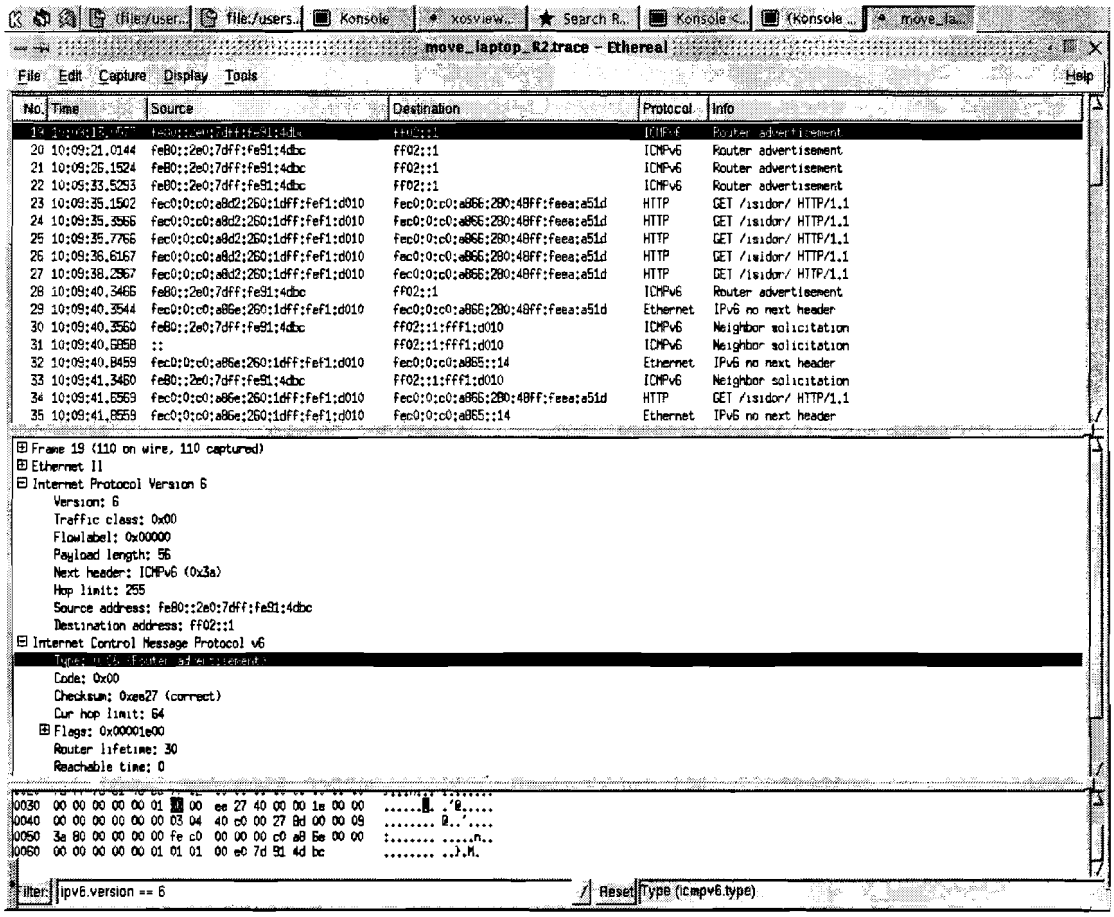


Figure 5-1. Screenshot of Ethereal.

5.2.2.5 Wireless networking configuration tool

A wireless networking configuration tool, called *iwconfig*, can be used to review and configure parameters of the available wireless extensions (lo, eth0, etc.). These parameters include bit rate, sensitivity and the use of an encryption key. The default values for these parameters are stored in the file `/etc/pcmcia/wireless.opts`. For more information on this file, see Appendix C.

Synopsis:

```
Iwconfig    interface  [essid {NN|on|off}]  [nwid {NN|on|off}]
        [freq {N.NNNN|k|M|G}]  [channel N]  [sens N]  [nick N]
        [rate {N|auto|fixed}]  [rts {N|auto|fixed|off}]
        [frag {N|auto|fixed|off}]  [enc NNNN-NNNN]
        [power {period N|timeout N}]  [txpower N {mW|dBm}]
```

5.3 Mobile node configuration[12]

The mobile node requires a special software configuration. The only test done so far with the iPAQ was with Windows CE (see paragraph 2.1.4 for details). As concluded in that paragraph, the DHCP solution does not work as required. And as stated in paragraph 2.1.5, MIPv6 combined with the Linux OS is chosen in order to accomplish better results. Until recently, this had to be done in two steps:

1. Equip the iPAQ with a suitable Linux distribution;
2. Expand this distribution with MIPv6;

Ad 1. Most of the mainstream Linux distributions are compiled for the Intel x86 processor architecture. The iPAQ is based on an Intel StrongARM processor and therefore needs a specialised distribution[13]. This will be discussed in detail in paragraph 5.3.1 and 5.3.2.

Ad 2. For this purpose, the MIPL implementation can be used, as has been done for PC's in the IPv6 network. See section 3.3 for details.

5.3.1 Requirements

Several Linux distributions are available and suited to install on the iPAQ, each with their own advantages and drawbacks. In our case, there are several requirements and wishes that need to be considered. These are:

1. MIPv6 support;
2. WLAN and/or Bluetooth support;
3. Java support;

First of all, MIPv6 has to be integrated in the kernel, or it has to be possible to expand the kernel with MIPv6. The integrated option is preferred, as this is less prone to errors and requires less work. Secondly, there are two possibilities for wireless communication by means of the included PCMCIA sleeve: WLAN and Bluetooth. But, as concluded in 2.1.3, the combination of Bluetooth and MIPv6 is not yet possible. However, any information on Bluetooth support is valuable, as it may be useful in the future. Furthermore, it is desirable to have support for the Lucent WaveLAN cards, as these have already been purchased for the IPv6 test network. This is not a problem for the majority of the distributions, however. The last point of interest is the support for Java. As it is the intention to implement a location based service in Java, the Java Runtime environment (JRE) is required to achieve this task. The JRE contains the Java Virtual Machine, to interpret the Java bytecode, as well as the necessary libraries.

All of the aspects mentioned above are to some degree kernel dependent. So, the kernel version can already tell us a lot on which requirements it meets. The following table summarizes the most important aspects of all freely available[14] distributions found on the Internet.

[12] All information in this paragraph is as of August 2001, unless otherwise stated.

[13] See Halsall, C., *Linux on an iPAQ*, The O'Reilly Network, January 2001.

[14] Except *LISA*, which is the only distribution found that is not freely available.

Table 5-1. Linux iPAQ distributions comparison.

Name	Kernel	Java	(m)IPv6	WaveLAN	Notes
Compaq 0.21					Dropped in favour of Familiar
Familiar v 0.4	Based on Linux 2.4.0-test1	JVM	MIPv6 supported	Lucent supported.	Update to 2.4.7 possible
Intimate	2.4.3-rmk2-np1	JVM	IPv6 supported[15]	Lucent supported.	Microdrive required
Midori 1.0.0-beta3	2.4.6	Unknown[16]	IPv6 supported in RH 7.0.	PCMCIA 3.1.25	Compiles on Red Hat 7.1
LISA 0.9	2.4.0-test 8	JavaScript, no Java	Unknown	Supported. (Lucent recommended)	Future support for BT Not free of charge
PocketLinux 1.0	Variable	Intended to run Java-applications	Unknown	WLAN supported	No real Linux environment

5.3.2 Distributions

Compaq's own distribution has been the reference distribution since its release in May of the year 2000. Because of the fact that it was a reference design, other distributions based on Compaq's were developed shortly after.

After a year of improvement Compaq started to focus on the kernel and other critical parts, and gradually **Familiar** became the new reference. The current, stable version of Familiar is 0.4; version 0.5 is to be released soon. It is based on kernel 2.4.0, but can be upgraded to 2.4.7. MIPv6 support is included in the most recent kernels, as well as support for the Lucent WaveLAN card. The JVM is also part of this version; the complete JRE is not natively supported, unfortunately. However, it is possible to install any graphical user interface (GUI) on top of familiar-for instance PocketLinux- as long as it is suited to run on the iPAQ together with Linux. Another advantage of Familiar is its good support and documentation, compared to the other candidates. There are several discussion boards and HOW-TO's available on the Internet that make debugging easier and ensure a continuing development.

Intimate is an extension of Familiar. The major difference with Familiar is that Intimate requires at least an extra 340 MB of storage space to be available, usually by means of a Microdrive. In contradiction to Familiar, which is designed to fit in a standard iPAQ, Intimate wants to offer a full-size iPAQ distribution. As IMST does not posses a Microdrive and there is no clear advantage in the extra packages included in this distribution, there is also no reason to prefer Intimate to Familiar.

Midori is a relatively new distribution available. Midori claims to promote the development of low-cost, energy-efficient and small devices by using standard Linux open source software. The current release (1.0.0-beta3) is based on kernel 2.4.8. Midori compiles on Red Hat 7.1, which has the advantage that IPv6 has been included since v7.0. It also supports PCMCIA cards. Java support is a problem, however. There still has to be a JVM

[15] Upgrade to MIPv6 possible.

[16] See Midori discuss digest mailing, volume 1 #65, 09-08 2001 on www.geocrawler.com/archives/3/9297/2001/.

embedded, though a lot of work on this topic is currently being done. Installation of a complete JRE may be a good option.

LISA is another relatively new Linux distribution for mobile devices. It has been developed specifically for the iPAQ. It claims to be the first fully Internet capable distribution for handhelds, but there is nothing known on IPv6 support. As LISA uses GSM, GPRS or HSCSD[17] to achieve a mobile Internet connection, it is not likely that IPv6 is natively supported. WaveLAN cards are fully supported, however. Support for the JVM is also available; the complete JRE is not. Another drawback is that LISA is the only distribution suited for a PDA that is not freely available for download. The complete software packet, named "*LISA mLinux 0.9 iPAQ Edition for IPAQ H36XX series*", comes on one CD-ROM and costs about € 40. This does not encourage the kind of public development that is so successful with, for instance, Familiar. It could also be a problem if any kind of commercial product, based on this distribution, would emerge from the ISIDOR project.

PocketLinux is also developed specially for mobile devices. Its main distinction compared to the other distributions is that it is completely based on Java. This includes JRE support, which is a big advantage during the development of location based services. Unfortunately, there is nothing known on native IPv6 support in PocketLinux. As it lacks a Linux runtime environment, almost nothing can be derived from the packages that usually come with the kernel, either. WaveLAN support appears to be no problem, however. Interesting aspect is that the PocketLinux GUI can also be installed on top of another distribution. This would enable us to combine the best aspects of both preferred kernel and GUI. This combination would yield a kernel with MIPv6 support and a graphical environment based on Java; at before hand almost the ideal situation.

5.3.3 Evaluation and conclusions

The previous section discussed the most important aspects of each distribution considered. Familiar offers a complete distribution, which currently only lacks the proper Java support. An attempt can be made to solve this problem by installing the JRE of PocketLinux or its complete graphical interface. As Familiar is still considered to be the reference, support and development of Familiar is a big advantage.

Compared to Familiar, the Intimate distribution only has an additional disadvantage in the form of the required extra space. Midori is also interesting as it offers roughly the same support and features as Familiar, though not yet as far developed and debugged. LISA possesses an up to date kernel with all its advantages, but the fact that it is not freely available makes it not suited for our applications. PocketLinux, finally, offers excellent Java support, which is a major advantage in developing location based services. However, the other important aspects (such as IPv6 support) are somewhat unclear. The fact that it is also possible to use only the most interesting part of this distribution (the JRE or the complete graphical interface) makes the usage of the complete distribution less interesting.

With respect to Bluetooth support, the most important aspect seems to be the frequency of kernel upgrades or the possibility to compile a new kernel yourself, as this will probably suffice to add support in the near future.

[17] High Speed Circuit Switched Data (HSCSD) is a new technology now being implemented in several GSM cellular networks. HSCSD makes it possible to use several time slots simultaneously when sending or receiving data. See www.nokia.com/phones/cardphone2_0/hscsd.html for details.

Considering these points, Familiar seems to be the best choice for our purposes. It is the most complete and best-supported distribution currently available. The only difficult part may be the installation and configuration of the JRE or the PocketLinux GUI. However, the vast amount of developers of Familiar, may be of great help with this task.

5.3.4 Familiar installation

The installation of Familiar on the iPAQ consists of three parts. First, we back-up the existing Windows CE Operating System. By doing this, it is at any point possible to restore the original OS. Next, we have to install a bootloader. The bootloader is the program that first gets control of the machine when it is powered on. It initialises and manages the raw hardware. One of its main jobs is to copy the operating system from non-volatile memory (Flash) to DRAM and pass control to the OS. The bootloader performs platform specific hardware configuration for the operating system. In addition, the bootloader provides a simple command-line interface that allows the user to download new versions of the operating system, user code, and parameters into Flash via the Xmodem protocol. The installation of the bootloader is the most critical step, as an installation failure at this point could render the iPAQ (temporarily) unusable. Compaq Research ensures they can and will fix a non-booting iPAQ, but this could take a while and should therefore be avoided. The bootloader is being used to make the entire procedure more bullet proof.

The final step consists of the actual installation of Familiar into the Flash ROM of the iPAQ. There is also an optional fourth step. This is a post-installation script that uses an NTP[18] server to adjust date and time and installs one or more true-type fronts from the Microsoft website. The correct network and PCMCIA settings must have been applied to complete this step. It is also possible to manually install and configure the elements of this fourth step. The precise steps taken to complete the Familiar installation can be found in Appendix C. This also includes the necessary steps to configure the iPAQ as mobile node in the IPv6 test network

5.3.5 Java installation

The Java runtime environment was needed in order to develop an application that demonstrates a location based service. As mentioned before, it is possible to use the Java package that is also used in PocketLinux. This package is called *tvt-kaffe-ipaq*[19]. It consists of all the Java packages used in the PocketLinux distribution. As the installation is an additional task, the complete PocketLinux package can be uninstalled any time in favour of another GUI. The extra installation steps for installing PocketLinux can also be found in Appendix C.

A first test to verify the Java functionality is a small application that displays a different page on the screen of the iPAQ, depending on its current care-of address. This application is based on the stand-alone architecture discussed in paragraph 3.2.1. So, effectively, entering a new cell displays new information on the screen. This information is already stored on the iPAQ. See Figure 5-2 and Figure 5-3 for screenshots of the information displayed in two different cells.

[18] Network Time Protocol.

[19] See www.transvirtual.com/ftp/pocketlinux/dists/.

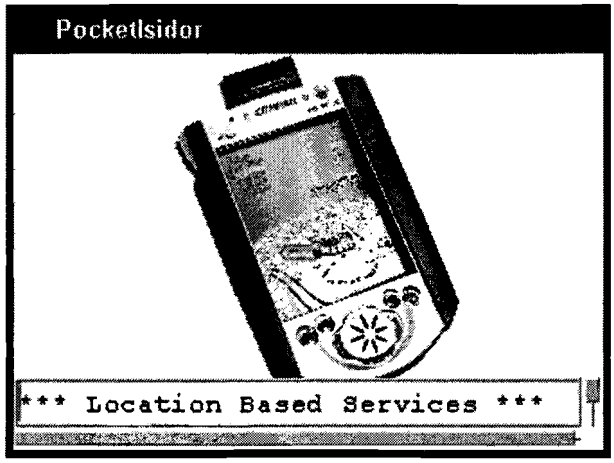


Figure 5-2. Information displayed in the first cell.

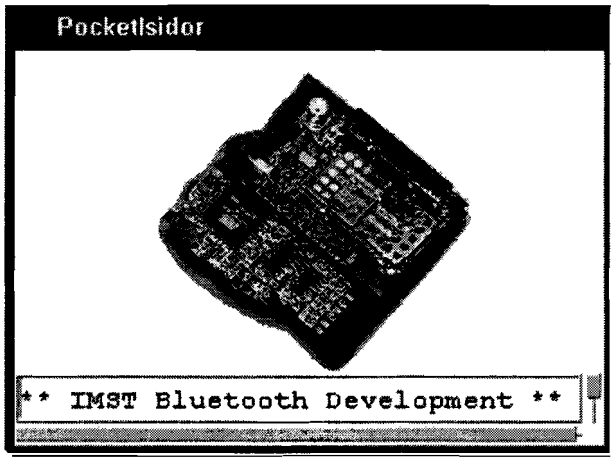


Figure 5-3. Information displayed in the second cell.

Chapter 6. Implementation

6.1 Introduction

This Chapter describes the implementation of the concepts of Chapter 3. This implementation consists of three logical entities: the home agent(s), the mobile nodes and the database server. Each entity consists of an application and an interface to the other entities. There may be multiple home agents and mobile nodes in the LBS network, but there should be only one database server to make sure all information is stored at a central place. The database server application may run on the same host as a home agent, but also on a separate host. It should be accessible through an IPv6 connection, as the mobile nodes are configured to use only IPv6. However, the home agent and database may also communicate through an IPv4 connection. See Figure 6-1 for an overview.

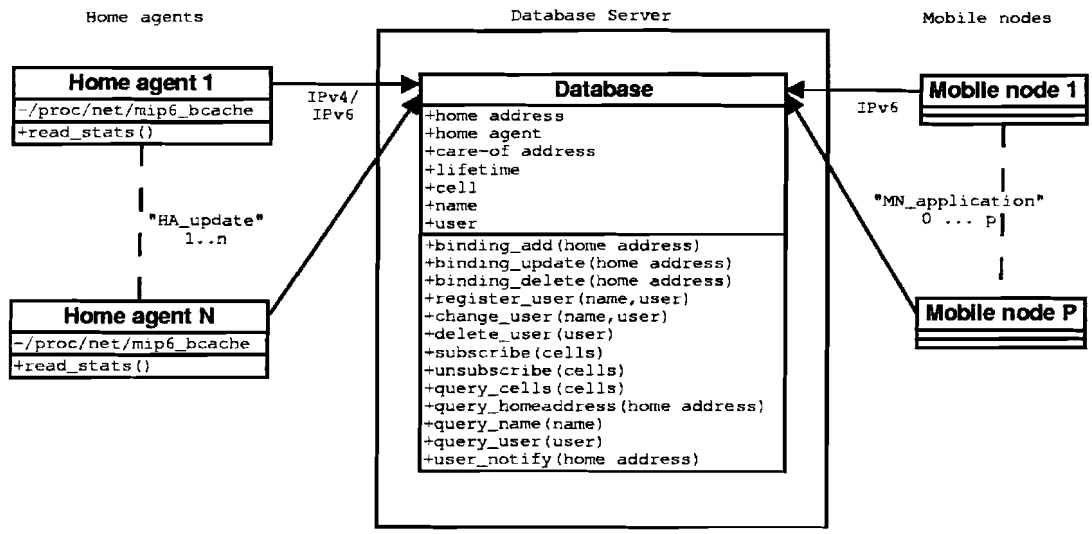


Figure 6-1. Logical entity overview.

Figure 6-1 shows each entity with its attributes and its operations. All attributes and operations are public, except for the *mip6_bcache* file on the mobile nodes. This attribute is private, which means it only has a meaning within the home agent entity. Public attributes and operations also have a meaning outside the entity they belong to. The working of the different entities and their operations will be explained in the following paragraphs. Note that the mobile nodes also communicate with their home agent, but this communication only takes place on the mobile IP-level.

6.1.1 Home agent entity

The first entity is the home agent. An application called *HA_update* runs on each home agent. This application sends the information in its binding cache to the database server. This information can be found in the local `/proc/net/` directory of the home agent, as described in paragraph 5.2.1. The home agent's application contains a function `read_stats()` to read this information. The information is sent to the database server by means of binding add, binding delete and binding update messages. These messages enable the database server to store the home address, home agent, care-of address and binding lifetime of each mobile node present in the location based services network. See paragraph 6.4.1 for details on these messages.

The messages between home agents and the database server can be triggered by network events, for instance a new mobile node (causing a binding add message) or a mobile node's cell change (binding update). Binding updates are also sent at regular intervals without a trigger, to refresh the lifetimes of the bindings stored in the database.

6.1.2 Mobile node entity

The mobile node entity consists of the application that runs on the mobile node (called *MN_application*, see Figure 6-1), together with its interface to the database server. The interface is a mIPv6 connection between the mobile node and the database server. The user is able to access all the information stored at the database server, either by his own direct request or by an automated notification. The application running on the mobile node has two major functions, which will be described in the following paragraphs.

6.1.2.1 Registration and subscription

First of all, the mobile node's application enables the user to register himself and to make clear to the database in which information he is interested. This is called *registration* and *subscription*, respectively. During the registration, a device and user name can be entered. This information is bound to the home address of the mobile node, thus enabling other nodes to use its name and user instead of the home address in their queries. The user's name can be used in queries for a person, instead of a device. The mobile node's user can change or delete the registration information. This method does not require the mobile node to have any knowledge on a domain name server (DNS) and is also more flexible. The exact details on the registration procedure can be found in paragraph 6.4.2.1.

By going through a subscription procedure, the mobile node's user indicates in which information he is interested. In the first stage, this means the user is able to enter a number of cells of interest (or all of them, indicated by a wildcard). From that moment on, he receives notification messages from the database server for every event in a subscribed cell. Such an event can either be network triggered (for instance a new mobile node enters a subscribed cell) or user triggered (name or user change of a mobile node residing in a subscribed cell). Every subscription can be changed or undone afterwards. The subscription procedure is described in more detail in paragraph 6.4.2.2.

6.1.2.2 Information queries

The second function the mobile node's application implements, is a way to let the user request information by sending instant *queries* to the database server. There are four kinds of information available in the first stage of the program. These are:

- Query cells;
- Query home address;
- Query name;
- Query user;

The *query cells* option enables the user to collect information about one or more cells. Either a number of cells, or the wildcard "*", indicating every cell, can be entered. In the first case, the home addresses present in that particular cell are sent back for each queried cell. In the second case, the home address of all known mobile nodes is returned to the inquiring mobile node.

Query home address requires the user to enter one or more home addresses. The database server returns the current name, user and cell prefix information for each home address entered. Cell prefix information is available immediately after a binding add message for this home address has been received by the database; name and user are only available after a successful registration procedure.

The *name* and *user* queries basically have the same functionality as the home address query, but based on mobile node's name and user, instead of its home address. More details on the query functions can be found in paragraph 6.4.2.3.

6.1.3 Database server entity

The database server entity is the most complicated entity, as its application has to be able to interface with both of the other applications. This implies multiple client-server connections at the same time, as well as the possibility to both respond to clients' requests and to send notifications itself. The available requests and notifications are described in the previous paragraphs. How the database server handles the different connections, will be explained in section 6.5.

The database server's application also stores all information it receives from the other two entities in one database. The database is updated as new information arrives from a home agent or mobile node. The mobile node is able to send queries to request certain information stored in this database. A change in the database may lead to the sending of notification messages to subscribed mobile nodes.

6.2 Location information retrieval

There are two ways of retrieving the location information for a mobile node in a LBS network. They can be used simultaneously.

1. Every mobile node announces its current location to the home agent by means of a binding update. This location changes whenever the mobile node moves to another cell and is thus a valid way of determining its current position in the network. This means the home agent's binding cache contains information of every mobile node. The advantage of this method is that any node in a network may contact the home agent to find the current location of a certain mobile node;

2. The mobile node can also derive the subnet part of its primary care-of address itself. This way, the mobile node does not need to contact the home agent to retrieve its own location. However, it still relies on a home agent for the location information of other mobile nodes;

6.3 Message structure

In order to send information between the entities in an efficient and unambiguous way, all sides must use the same format in their messages. This requires a suited protocol, carried by IPv6. Which protocol is used and how its correctness can be verified, will be discussed in this section.

6.3.1 XML language

The structure of the protocol is based on the Extensible Mark-up Language (XML) [15]. XML is a flexible way to create and share common information formats between two computers connected through the World Wide Web (WWW). It is similar to the language of today's web pages, the Hypertext Mark-up Language (HTML). However, XML is "extensible" because, unlike HTML, the mark-up symbols are unlimited and self-defining. Mark-up is defined as everything in a document that is not content. Applications have to use an XML-processor to read XML documents and provide access to their content and structure. XML was formed under the auspices of the World Wide Web Consortium (W3C)[20] in 1996. XML and HTML are actually simpler and easier-to-use subsets of the Standard Generalized Mark-up Language (SGML) [16], the standard for the creation of structured documents. The characteristics of XML make it possible to quickly define messages in a consistent way.

The mark-up indicators are usually called *tags*. A part of a message is called an element and its boundaries are indicated by a begin and end tag. The begin tag is always of the form *<tag>*, while an end tag always looks like *</tag>*. An empty element *<tag/>* is also allowed. This forms an empty element. In these examples, the text "tag" is the (obligatory) name of the tag. Everything between tags is the element's content; empty elements have no content. Elements can also be nested with other elements. This is used to reveal the structure of the protocol.

An element can optionally have one or more attributes to provide extra information. An attribute consists of a name, an equation and an attribute value, for instance: *id="cell1"*. In this case, "id" is the attribute name and "cell1" is the attribute value. Attributes are only allowed in begin and empty tags and are placed just after the tag's name. An example of a start tag with an attribute is *<cell id="fec0:0:c0:a86e">*. In this case, the matching end tag is *</cell>*. This element can be used to query information on a cell denoted by the cell prefix, in this case fec0:0:c0:a86e.

6.3.2 XML validation: DTD

A parser can be used to verify the correctness of the Document Type Definition (DTD) and whether a document follows the rules of DTD. The DTD is the formal definition of the elements, structures and rules for marking up a given type of SGML document [16]. Its purpose is to define the legal building blocks of an XML document by defining each of its

[20] See www.w3.org for details on the W3C.

elements. These building blocks are the tagged elements described in the previous paragraph

The verification by means of a DTD avoids ambiguity in XML documents. For instance, every element can only be declared once. If one or more elements are nested, these sub-elements -called children- have to be defined as well. The children have to present in the same order each time the element they belong to, is used. The declaration of an element with one or more children always looks like this:

```
<!ELEMENT element-name (child-element-name,child-element-name,...)>
```

A child can occur more than one time within an element. The following signs are used to indicate the number of occurrences of a child:

- No sign declares that the child element message can occur only one time inside the element;
- The + sign declares that the child element message must occur one or more times inside the element;
- The * sign declares that the child element message can occur zero or more times inside the element;
- The ? sign declares that the child element message can occur zero or one time inside the element;
- The | sign declares that the child element message must either contain one or the other element.

The declaration of attributes of an element has the following syntax:

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

The *element-name* indicates to which element this attribute belongs. It is followed by the name of the attribute. Next is the *attribute type*, for instance character data (CDATA). The *default-value* field can have the following values:

- #DEFAULT *value*: This declares the attribute's default value, which is *value*;
- #REQUIRED: This declares the attribute's value must be included in the element, but no default value is given;
- #IMPLIED: This declares that the attribute does not have to be included;
- #FIXED *value*: This declares the attribute's value is fixed at *value*.

The DTD for our protocol is listed in Appendix D. It defines the message elements that will be explained in the next section.

6.4 Defined messages

We need the XML-based protocol to enable the applications of the three entities to efficiently exchange their information. All messages that are being sent between the applications start with the *<protocol>* tag and end with the *</protocol>* tag. The protocol's messages are sent over an TCP connection, which is initiated by the client. Paragraph 6.5.1 explains how this connection is being set up in. Three different types of messages are defined:

1. Messages initiated by a home agent and sent to the database server.
   These can be either a binding update, binding add or binding delete message. They are indicated by the term "messageHA2DB" and its corresponding begin and end tags *<messageHA2DB>* and *</messageHA2DB>*.

2. Messages initiated by a mobile node and sent to the database server.
   They consist of registration messages (user register, user change and user delete), subscription messages (subscribe and unsubscribe to notifications) and query messages (query cells, query home address, query name and query user). This group of messages are indicated by "messageMN2DB" and its corresponding begin and end tags are *<messageMN2DB>* and *</messageMN2DB>*.
3. Messages initiated by the database server and sent to mobile nodes.
   This group consists of only one message and is triggered by a change in the database server. It is a notification of a cell change, user change or name change in a certain cell. The notifications are sent by the database server to all mobile nodes subscribed to changes in the concerning cell. These messages are called "messageDB2MN". They have the begin and end tags *<messageDB2MN>* and *</messageDB2MN>*.

Each message is associated with an element that complies with the DTD rules described in the previous section. The remainder of this section describes what these messages and their corresponding elements look like.

6.4.1 MessageHA2DB

As mentioned above, there are three different elements in this category: *binding add*, *binding update* and *binding delete*. An overview of these elements can be found in Appendix E.1. Figure 6-2 shows the message sequence chart (MSC) for this category.

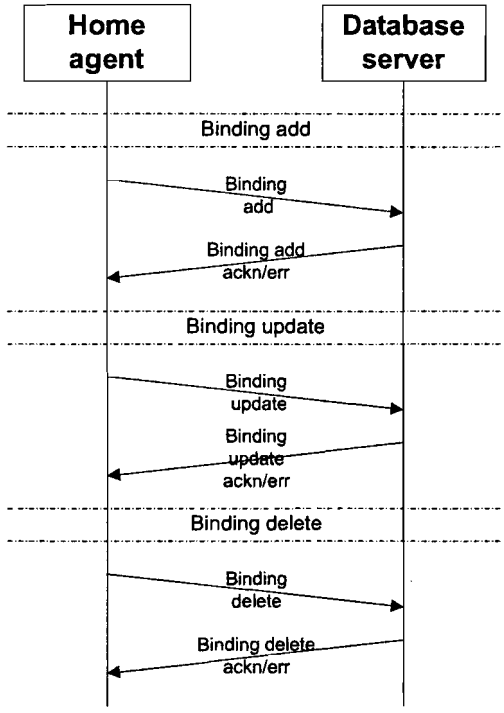


Figure 6-2. MessageHA2DB sequence chart.

Binding add element
The first element indicates the binding add message. This message is sent every time the home agent's application detects a new mobile node in its binding cache. Its begin and

end tags are *<binding_add home_address="...">* and *</binding_add>* respectively. The binding add element has a required attribute, which is the home address of the concerning mobile node. There can be three different elements nested within the binding add element, of which one and just one has to present. This can be binding information, the acknowledgement or the error element.

The binding information is not an element itself, but consists of just the required elements for a binding add message. These are the home agent, care-of address and lifetime of the added mobile node. These terms have been explained in Chapter 4. The corresponding tags are, respectively: *<home_agent>*, *</home_agent>*, *<coa>*, *</coa>*, *<lifetime>*, and *</lifetime>*. The content of these elements is the actual information to be stored in the database.

The acknowledgement element is an empty element and is used to acknowledge a previously sent binding add message. It is sent from the database to a home agent in response to each successful binding add. Its tag is *<ackn/>*. This element is used in each acknowledgement between the different entities.

The error element is used to notify the home agent that the database was unable to process a previously sent binding add element. The error element has no attributes. It's begin and end tag are *<err>* and *</err>*, respectively. The content of the error element is the actual error message. This element is also used by each entity.

Binding update element
This element is sent in two cases. The first case is initiated by a change in care-of address of the mobile node, which implies a cell change. The home agent notices this change in its binding cache and sends a binding update message to the database. The second case is triggered by the home agent application for each mobile node in its binding cache, at a regular interval. Its purpose is to keep the lifetime information stored on the database up to date.

The binding update element's begin and end tags are *<binding_update home_address="...">* and *</binding_update>*, respectively. The binding update element consists of the same possible elements as the binding add element: binding information, an acknowledgement, or an error element. It also has the same attribute as the binding add element to indicate the home address for which it is an update.

The binding information can have the same elements as defined above (home agent, care-of address and lifetime), with the difference that they are all optional. Currently, the home agent's application is configured to send all binding info elements, but this is not required for a correct functioning of the database application.

Binding delete element
The binding delete element is used to delete a binding from the database. This is necessary when a node was present in the home agent's binding cache during a check of the application, but is not present anymore during the next check. This can be due to, for instance, an expired lifetime or a change of home agent. Its tags are *<binding_delete home_address="...">* and *</binding_delete>*. Again, this element has a required attribute indicating the concerning home address.

The binding delete element may include the acknowledgement or error element to acknowledge a previous binding delete sent by a home agent, or indicate its reason of

failure. The actual binding delete element is an empty element, as only the home address is needed to remove the mobile node from the database.

Note that the presence of a mobile node's home address in the database does not necessarily mean it is still present in the listed cell with the listed care-of address. For instance, if a mobile node is suddenly powered off or out of reach of an access point, it is not able to indicate this event. Therefore it is still listed in its home agent's binding cache and no binding delete message will be generated. Only after expiration of its lifetime, a binding delete is sent to the database server. However, this is a problem that should be solved on the IP-level and not on higher layers.

6.4.2 MessageMN2DB

This group consists of nine different messages in three categories: registration, subscription and queries. An overview of these elements can be found in Appendix E.2.

6.4.2.1 Registration

This category contains messages that enable the mobile node to register itself, as well as change and delete this registration. Registration of mobile nodes is useful to be able to use name or user information instead of home address during the interaction with the user. This name and user can then also be used in queries from other mobile nodes. Therefore, this registration is automatically done as the mobile node application is started. This application is described briefly in section 6.7. After registration, the user is able to change or delete the registration by means of the user interface. Figure 6-3 displays the MSC of the possible messages in the user registration process.

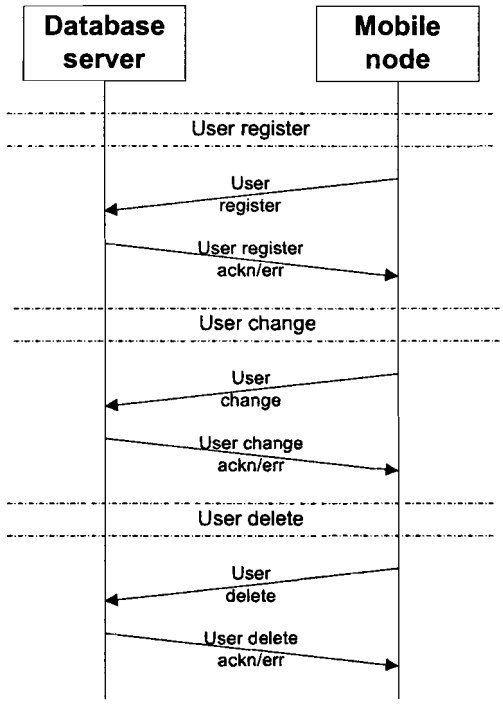


Figure 6-3. User registration messages sequence chart.

User register
The user register element contains one of the following elements: the (optional) user information, the acknowledgement or error element. It can be recognised by its tags *<user_register home_address="...">* and *</user_register>*. Note that this element also includes the sender's home address in the form of a required attribute.

As mentioned before, the registration procedure is automatically started upon launching of the mobile node application by sending the mobile node's user info. This can be its name, user and other relevant information. Currently, only the two first elements (name and user) are defined. The begin and end tags for the name element are *<name>* and *</name>*, respectively. The content is the actual name of the node. The begin and end tags used for the user are *<user>* and *</user>*; the user's name is stored in its contents.

The name element can be used as an alternative for the home address and should therefore be unique when possible, at least within the LBS network. A way to achieve this is to define the mobile node's name as hostname.domainname, for instance ipaq1.isidor.net. The hostname can be found in /etc/hostname on the iPAQ; the domain can be obtained with the standard C-library function `getdomainname()`. However, the user is free to alter this information and the database server currently does not check name uniqueness.

The user element can be used to identify the current user of the mobile node. This name only has to be unique for the mobile node, but is preferably also unique for the LBS network. Uniqueness on node level can be achieved by using the login name as current user. In that case, a user change message can be generated and sent automatically after a login. However, the mobile node's application may not require users to log in. In that case, the current user can only be obtained by means of the user interface of the mobile node application. The added value of the user lies in the fact that one may be interested in a person, regardless of the device he is currently using and his location in the LBS network. This enables the development of ICQ-like applications[21]. The database server only supports one name and user per home address.

The database server sends an acknowledgement upon each successfully processed registration message. If an error occurs during processing, the error element is sent instead. This can be the case for an attempted user registration for a home address that is not known within the database (i.e., a home address for which either no binding add message has been received from a home agent, or whose presence has been deleted by means of a binding delete message). The acknowledgement and error element have been defined in paragraph 6.4.1.

User change
The user change message has the tags *<user_change home_address="...">* and *</user_change>*. The mobile node's user can trigger it to change one or more of the elements sent during the user registration procedure. Like the user register element, it must consist of either user information, or an acknowledgement or error element. In the first case, name and user are optional.

[21] ICQ ("I seek you") is an instant-messaging program, which allows its users to find other users on the Internet and receive notifications as soon as they are connected.

An acknowledgement is sent back to the mobile upon each successful change. If the database server cannot process a user change element, it sends an error element containing the exact error message, instead.

User delete
If, for any reason, the mobile node's user wishes to withdraw the automatic or changed user registration, he may do so by means of a user delete message. This can be done by means of the empty element *<user_delete home_address="..."/>*. A successful user deletion is followed by an acknowledgement from the database server. In all other cases (for instance an attempted user delete for an unknown home address), the error element is sent back. In case of an acknowledgment or error message, the user delete element is not empty, but uses the tags *<user_delete home_address="...">* and *</user_delete>*.

6.4.2.2 Subscription

On of the most important goals of the applications is to enable the mobile nodes to gather information on other mobile nodes present in the same LBS network. This can be done on request (i.e. user triggered), but a user may also subscribe itself to be notified in case certain changes occur. Currently, two subscriptions are possible: to all known cells or to a (limited) number of cells. Any subscription can be undone by a similar unsubscription message. See Figure 6-4 for details.

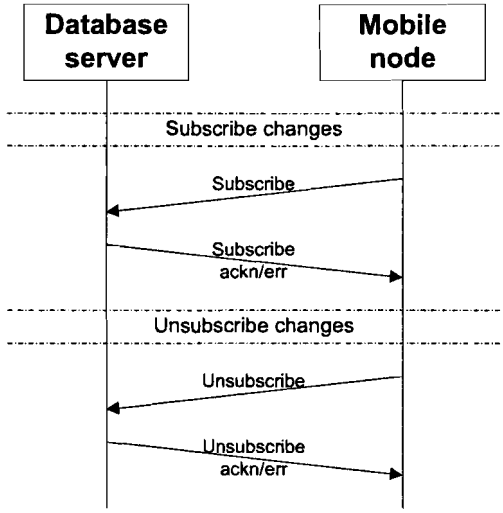


Figure 6-4. Subscription messages sequence chart.

Subscribe
This element uses the *<subscribe home_address="...">* and *</subscribe>* tags. It must contain one and only one of these elements: one or more cell prefixes, an acknowledgement or error element. The acknowledgement element is used to notify the mobile node of a successful subscription. If there was an error processing the subscription request, an error element is returned by the database. At the moment, this can only happen in case of a subscription attempt for an unknown mobile node.

In case of a subscription, one or more cells have to be included in the subscription element. It may contain two things: a query for all known cells or a query for a number of cells. The cell element has the tags *<cell_id>* and *</cell_id>*. Its content is the actual cell prefix or the wildcard "*". The last case indicates the user is interested in all cells, without

have to enter each cell's prefix. In the first case, a list of similar cell elements may follow. Currently, there is no theoretical limit on the number of cells to enter in this query, but there are practical limitations, however. This is because of the fixed memory allocations in the database server and mobile node applications. The exact limitations are yet to be researched, but allocations can easily be adjusted to solve possible storage problems.

Unsubscribe
The unsubscribe element has the tags *<unsubscribe home_address="...">* and *</unsubscribe>*. The elements that can be nested in the unsubscribe element are exactly equal to the ones used in the subscribe element defined above.

6.4.2.3 Queries

As mentioned before, there are two ways for a mobile node to receive information on the LBS network. The first one is by means of notifications of events in cells to which the mobile node is subscribed. The subscription method is discussed in paragraph 6.4.2.2. The other method is to submit a query directly to the database server. The database processes this query and sends the required information to the mobile node in a query response message. Currently, there are four queries specified: query cells, query home address, query name and query user. The involved messages can be seen in Figure 6-5.

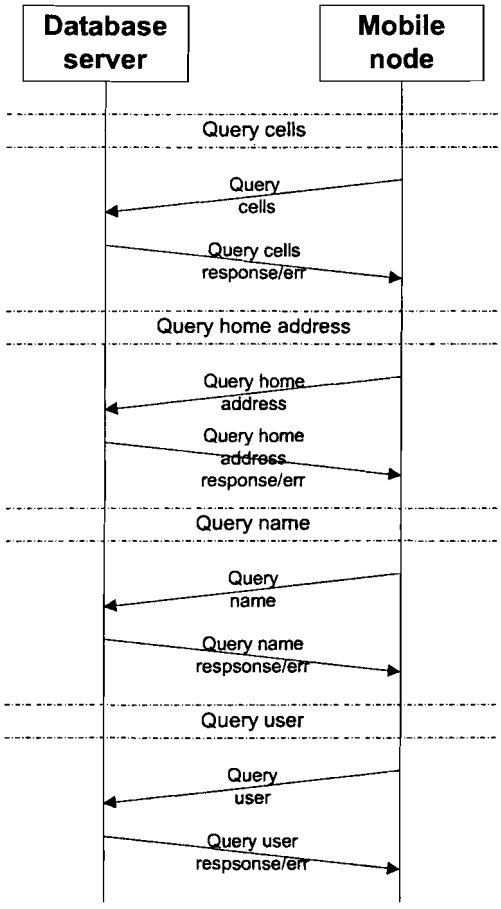


Figure 6-5. Query messages sequence chart.

Query cells
This query message is used to retrieve a list of all mobile nodes present in one or all known cells. The corresponding (empty) tag is *<query_cells cell_id="..."/>*. The required attribute has a default value of "*", indicating all cells. Alternatively, it may contain the cell prefix of a cell the user wishes to retrieve all list of all the present nodes of.

The response to this query cells message uses the *<query_cells cell_id="...">* and *</query_cells>* tags. Its contents depends on the query message it is a response to. If one cell was queried, it has just one element with the tags *<cell id="...">* and *</cell>*. In the case all the cells are queried, the response may contain more than one cell elements. The contents of a cell element is a list of home address elements, one for each mobile node in that cell. A mobile node element uses the tags *<home_address>* and *</home_address>*.

Query home address
This query is used to retrieve additional information on a known home address. The information available for a mobile node is its name and user and the cell it is currently in. Name and/or user are only available after a successful user registration as described in 6.4.2.1. Its current cell is derived from the mobile node's care-of address (and is therefore available immediately after a binding add) and is updated with each successful binding update. Home address queries use the tag *<query_home_address home_address="..."/>*.

The response message uses the corresponding begin and end tags: *<query_home_address home_address="...">* and *</query_home_address>*. The response contains the name, user and cell elements belonging to the queried home address (if present in the database), as described in 6.4.2.1. In case of a failure in the processing of the query, the well-known error element is returned instead.

Query name
This query uses the empty tag *<query_name name="..."/>*. It is can be used by a mobile node to gather additional information for a known name. Currently, this is information is limited to home address, user and current cell. It is composed analogously to the query home address with the terms "home address" and "name" switched.

Query user
This query is composed completely analogously to the query home address and query name element. It sends a query for a certain user by means of a required attribute. This yields the empty tag *<query_user user="..."/>*. The response included the corresponding home address, name (if known) and current cell. It uses the tags *<query_user user="...">* and *</query_user>* in a response.

6.4.3 MessageDB2MN

As mentioned before, there is just one message type in this category: the notification message. It is sent by the database server to subscribed mobile nodes upon an event in a certain cell. This can be a new mobile node entering the cell (triggered by a binding add or a binding update with a changed care-of address), a node leaving the cell (triggered by a binding delete message element or a binding update with a changed care-of address), or a change in the node's registration (triggered by a user register, user change or user delete message element). The notification message's tags are *<user_notify home_address="...">* and *</user_notify>*. See Appendix E.3 for an overview and Figure 6-6 for a MSC of the possible messages.

The notification must contain either information on the home address that caused the notification or an acknowledgement or error element, sent by the mobile node as a response. The information that is sent depends on the trigger. In case of a cell change, the cell element is included. In the case of user or name change, the corresponding element(s) are included.

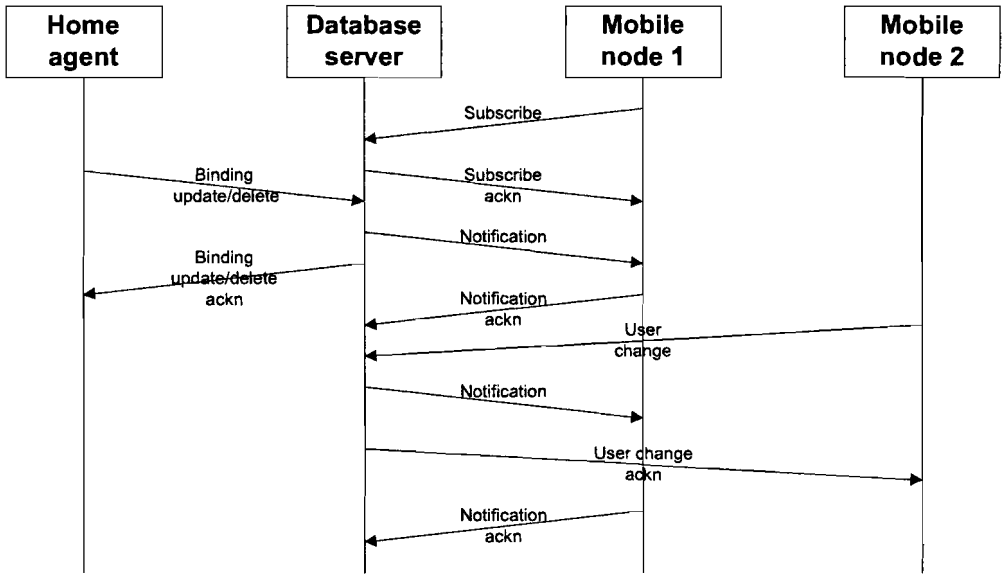


Figure 6-6. Notification messages sequence chart.

The mobile node acknowledges a successful notification by means of an acknowledgment element as described in 6.4.1. An error element is returned in case of an erroneous notification, as has been described in 6.4.1 as well.

6.5 Client-server communication

If the different entities in the network (home agent, database server and mobile node) want to be able to exchange data, there has to be a connection on the TCP[22] layer first. This involves several compulsory steps, which will be described in this section. More information on TCP/IP can be found in [6] and [17].

6.5.1 Set-up of a TCP connection

TCP uses connection-oriented sockets to allow data to flow back and forth. Each end specifies the so-called socket pair for the connection. This consists of the local IP address, local port, foreign IP address and foreign port. The client must specify the foreign IP address and port in the call in order to connect.

TCP uses a set of elementary functions for client-server communication. These functions can be seen in Figure 6-7. The server has to be started first; otherwise a client may not be able to connect to it. We assume communication takes place with a request of the client to the server. The server processes this request and sends back a reply. This continues until

[22] The transmission control protocol (TCP) is a widely used transport layer protocol to release reliable transmission of network layer (IP) packets. With respect to the ISIDOR requirements (see 2.1.2), the choice for this protocol is obvious.

the client closes its end of the connection, after which the server does the same. When the server has closed the connection, it is again able to accept incoming connections.

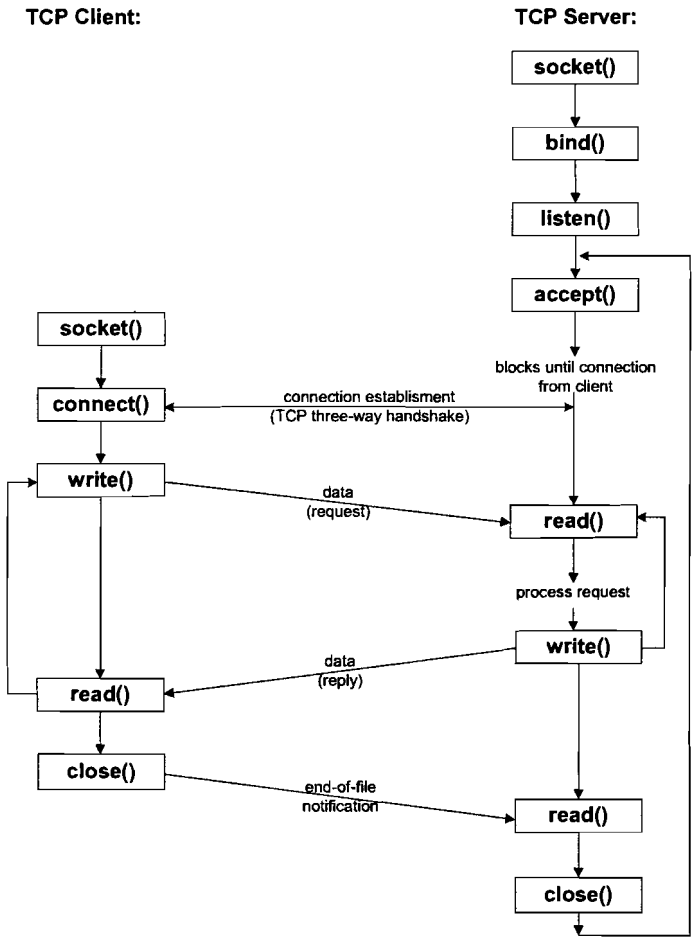


Figure 6-7. Socket functions for elementary TCP client-server connection.

The following functions are defined in Figure 6-7:

`Socket();` this specifies the type of communication protocol desired, the corresponding family and the socket type description;

`Connect();` the TCP client uses this function to establish the connection with the TCP server;

`Bind();` *bind()* assigns a local protocol address to a socket. In IPv6 the protocol address is the combination of a 128-bit IPv6 address, along with a 16-bit TCP port number;

`Listen();` this function is called by the TCP server and performs two actions:
1. It converts an unconnected socket into a passive socket, indicating that the host should accept incoming connection requests directed to this socket;
2. It specifies the maximum number of connections that the kernel should queue for this socket (the so-called *backlog*);

`Accept();` *accept()* is called by the TCP server to return the next completed connection from the front of the completed connection queue. If this

queue is empty, the process is put to sleep (assuming the default of blocking a socket once the server is connected to a client);

`Write();` this function sends a message to the peer connection;

`Read();` this reads the message sent by the peer's *write()* function;

`Close();` used by the client to close its end of the connection and send an end-of-file notification to the server. The server uses the *close()* function to close its end of the connection and either terminates or returns to the accept state;

These functions have an equivalent in the C programming language, so the same set-up procedure can be used in the database server application. Their exact definitions can be found in the corresponding manual pages.

6.5.2 Multiple user connections

As there may be more than one home agents and mobile nodes present in the LBS network, the database server has to be able to maintain multiple client connections at the same time, especially because the home agent's application has a continuous connection with the database server to send its updates[23] to. So, if a mobile node wants to send to or receive anything from the database, multiple simultaneous connections have to be possible. Figure 6-8 gives a graphical representation of a server connected to two clients. The dots indicate the values needed for a socket pair, as described in the previous paragraph.

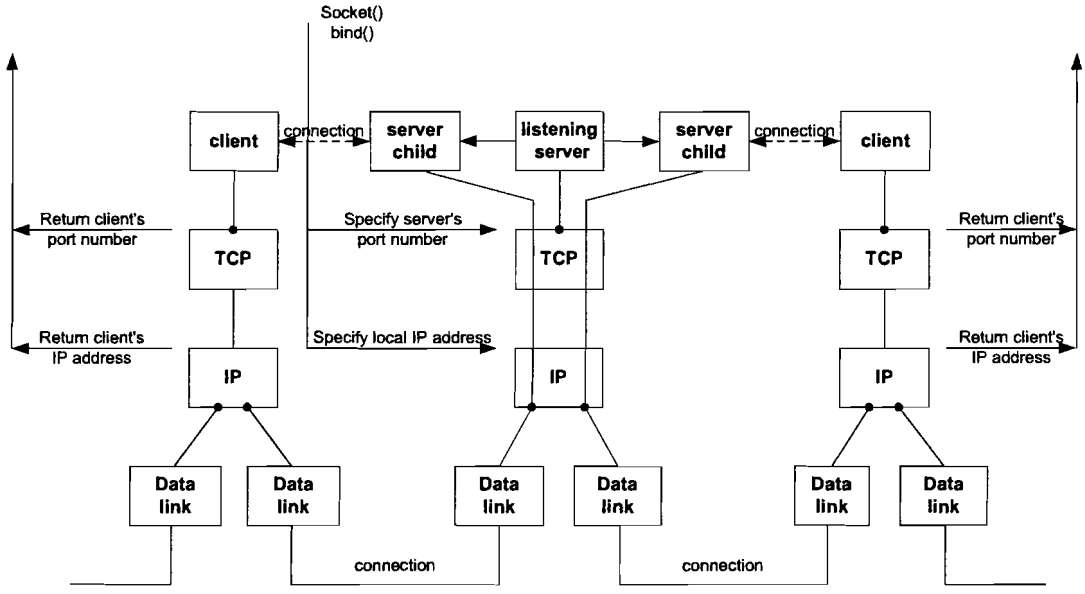


Figure 6-8. TCP client-server model with two clients (from client's perspective).

6.5.2.1 Forks

The handling of multiple clients can be done with so-called *forks*: the server's parent function accepts an incoming connection and forks a child. From here, the child handles

[23] See section 6.6.3 for a more detailed evaluation of this choice.

the client. Now there are two connected sockets which both have a socket receive buffer on the server. However, using forks to handle multiple connections has two major disadvantages:

- Forks are expensive with respect to memory and CPU usage. Memory is copied from the parent to the child and all descriptors are duplicated in the child. Memory usage would be way too high in case of multiple mobile nodes and home agent connections.
- We need inter-process communication (IPC) to pass information between the parent and child functions after the fork process. Sending information from parent to child is easy, since the child starts with a copy of the parent's data space and all its descriptors. However, this can only be done as the fork is formed. Furthermore, returning information from the child to its parent takes far more work.

The next paragraph presents a solution for these problems in the form of so-called *threads.*

6.5.2.2 Threads

Basically, a *thread* is the information needed to serve one individual user or a particular service request. Threads are sometimes called lightweight processes since a thread is "lighter weight" than a process. That is, a thread creation can be 10-100 times faster than the process creation [18].

All threads within a process share the same global memory. This makes sharing of information between threads easy. Not only the global variables are shared, all threads within a process share:

- Process instructions;
- Most data;
- Open files (e.g. descriptors);
- Signal handlers and dispositions;
- Current working directory;
- User and group Ids;

But each thread has its own:

- Thread ID;
- Set of registers, including program counter and stack pointer;
- Stack (for local variables and return address);
- Error number for creation;
- Signal mask;
- Priority;

By using multi-threading we are able to save an enormous amount of time in the handling of a large number of connections because we are able to handle them parallel. With this solution it is possible for a large number of mobile nodes to connect to the database simultaneously, without slowing down the processing of their individual requests. This effect is shown in Figure 6-9.

As all of the development is done in a Linux environment, we use the so-called POSIX (Portable operating system Unix) threads, also called *pthreads.* They are similar to the non-POSIX *cthreads.* A pthread can be created with the pthread_create function, which requires four arguments [19]: a thread variable or holder for the thread, a thread attribute,

the function for the thread to call when it starts execution and finally an argument to that function. An example of this creation is

```
pthread_create(&a_thread, a_thread_attribute, (void *)
&thread_function, (void *) &some_argument);
```

After creation, it has to be disassociated from its parent with the function:

```
pthread_detach(a_thread);
```

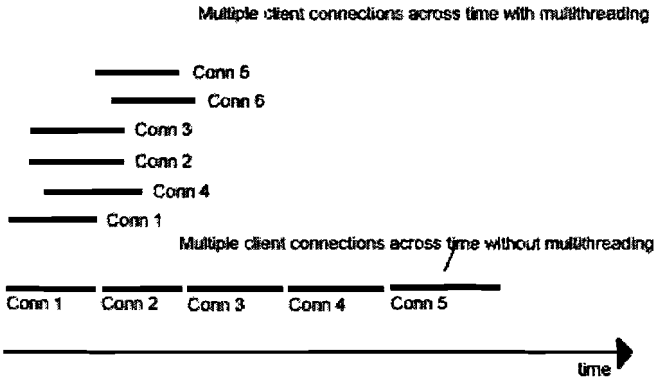


Figure 6-9. Multiple threads vs. single threads across time.

A thread can be exited with the following command:

```
pthread_exit(void *retvalue);
```

Note that a thread can start with any function we like, in contrast to the fork's child, which begins execution at the same point as its parent.

The creating of new threads in two steps (create and detach) works fine if the number of new connections per second is low. However, if multiple clients attempt to connect to the server in a short amount of time, the server may accept new incoming connection before the previous one's detaching has completed. This results in the blocking of all threads except for the last one. Detaching the thread automatically after its creation can solve this problem. This can be achieved by setting an appropriate attribute:

```
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
```

With this attribute, the pthread-detach statement is not necessary anymore, as the new thread is automatically detached after its creation.

6.5.2.3 Mutexes

However, the big advantage of being able to run threads parallel yields a new problem: that of synchronization. When all threads run concurrently and have access to shared resources, this access must be controlled. Otherwise, all threads would be able to read and write to these resources at the same time. This may cause problems. For instance, if a mobile node sends a query message for a certain home address at the same time the home agent sends a binding update for this home address, the database server does not have a well-defined state anymore. So, only one thread at a time may be able to access shared resources.

The simplest way to solve this problem is with the use of mutexes. Mutexes are simple lock primitives that can be used to control access to shared resources. A mutex has only two states: locked and unlocked. Whenever a thread wants to access a shared resource, it locks the mutex first. When it is finished, the mutex is unlocked again. During a lock, none of the other threads is able to lock or access the shared resources. They are blocked until the mutex is unlocked again by the tread that originally placed the lock. A mutex with name *mutex* can be declared in the same way a variable is declared:

```
pthread_mutex_t mutex;
```

Locking and unlocking of the mutex is done by the following functions:

```
pthread_mutex_lock(&mutex);
pthread_mutex_unlock(&mutex);
```

In the database server application, we want to use two different mutexes: one to control the mobile node part of the database (filled and updated by the connected home agents) and one to control the database's list of connected clients. The database server maintains the client list to store which clients are connected on which ports. It needs this information to send notifications to the right nodes. As these resources are independent of each other, they can be blocked individually. Using one mutex for both resources would cause unnecessary delays in the application. After all, there is no need for a thread wishing to update a mobile node's lifetime to wait for another thread that is updating the client list, because a new mobile node has connected. The exact structure of the database can be found in paragraph 6.6.2.

6.6 Database server application functioning

6.6.1 Parent/child functionality

All the concepts and functions explained in this Chapter have to be implemented in an application. Considering the required functionality (speed is most important, no user interface is required), the C-programming language has been chosen for this task. The application can be divided into two parts: the parent (the main loop) and the child (all the other functions). The main loop of the database application functions as can be seen in Figure 6-10. It starts by preparing a new socket for an incoming connection. After this has been done, it waits for an incoming connection. For every incoming connection, a new threat is created. A successfully created thread is detached and the loop starts all over again.

Every detached thread starts with the child function. See Appendix F for an overview of the functioning of the child. The database server waits for data from the client and buffers it until a complete message has been received. A complete message starts with the *<protocol>* tag and ends with the *</protocol>* tag. In case more than one complete message has been received, they are processed in succession. The *handle_buffer* function first determines what type of message has been received (HA2DB, MN2DB or DM2MN).

In case of a HA2DB message, the next step is to determine the actual binding message. Binding messages are processed and an according response (either an acknowledgment or error) is generated. For a binding update message, this step is followed by a check to see if a cell change has occurred. If so, a notification message is generated and sent to all

subscribed clients. In case of a binding add or binding delete message, a notification is always generated. Only after all notifications have been sent, the response to the client will be sent.

The child function does not wait for notification acknowledgement messages from the mobile nodes. The reason for this choice is that the generation of such acknowledgements by the mobile nodes is thought to be too slow compared to the speed of the database server's application. This assumption will be verified in Chapter 7.

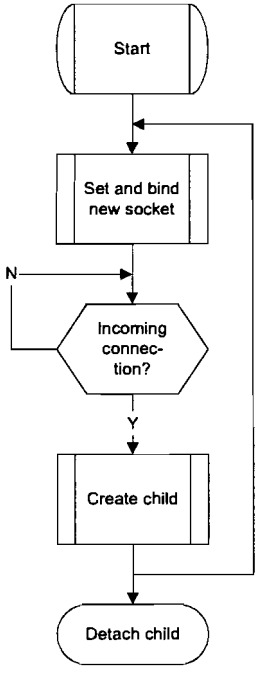


Figure 6-10. Schematical functioning of main loop.

In case of a MN2DB message, there are three possibilities: a registration, an (un)subscribe or a query message. The messages are processed and a proper response is generated. In case of an (un)subscription, this is either an acknowledgement or an error message. In case of a query message, a query response or error message is generated. The registration message also triggers the generation of a response message (acknowledgment or error), but a notification message is generated and sent to subscribed clients as well.

The last possible received message by the database server (messageDB2MN) can only be the response to a previously sent notification. As this is either an error or an acknowledgement, it does not require the generation of a response or new notification message.

Finally, the response is sent to the client, if one is waiting in a buffer. After sending, the child checks whether there are other received messages waiting to be processed. If not, it starts all over again by waiting for new data from the client.

6.6.2 Information storage and retrieval

The information stored within the database consists of four parts:

1. Known home addresses;
2. Subscriptions for a certain home address;
3. Known cells in the LBS network;
4. Clients currently connected to the database server;

The first part -which is also the largest- contains all information on a certain mobile node. This information is stored in a linked list, which is sorted by home address. The structure contains the information retrieved from the home agent, as well as the information retrieved from the mobile node itself. Its declaration looks like as follows:

```
struct address {
  char home_addr[36];     // MN home address
  char h_a[64];           // MN home agent
  char co_addr[36];       // MN care-of address
  char cell[18];          // MN cell prefix
  char name[64];          // MN name
  char user[64];          // MN current user
  int lifetime;           // MN lifetime
  int flags;              // MN flag (entry_valid or entry_remove)
  int socket;             // MN home address to sd link
  struct subscription *subs;  // Pointer to subscription struct
  struct address *next;   // Pointer to next struct address
};
```

The home address, home agent, care-of address and lifetime can be derived directly from binding add and binding update messages received from the mobile node's home agent. The cell prefix can be computed by simply taking the first 16 (= prefix size) characters of the unformatted care-of address. The name and user fields are obtained from user register and user change messages. The MN flag is used to indicate whether an entry is valid (should be kept in the database) or invalid (should be removed). The socket field contains the socket descriptor of the connection with this mobile node. This is the descriptor that is used for sending notifications for cells listed in the linked list *subscription*. Finally, there is a pointer to the next struct (or to NULL if no next struct is present).

The second part of the database is also part of the first structure. It is the linked list subscription, which contains for each mobile node the cells it is currently subscribed to. If the mobile node is subscribed to all notifications, it only contains a struct for the wildcard "*". It is declared as follows:

```
struct subscription {
  char cell[18];          // Subscribed cell's prefix
  int flags;              // Subscription flag (valid or remove)
  struct subscription *next;   // Pointer to next struct
};                             // subscription
```

The third part of the database is a linked list of all the cells that are currently known within the network. Its main purpose is to have a sorted list of all present cells. This list is used to loop through in case a response to a <cell_id id="*"> tag is being generated. It is declared like:

```
struct cells {
  char cell[18];          // Cell prefix
  int flags;              // Cell flag (valid or remove)
```

```
  struct cells *next;     // Pointer to next struct cells
};
```

The first field is the actual information to be stored: the cell's prefix. Next is a flag indicating the entry is valid or should be removed from the list. Finally, there is a pointer to the next struct. The list is sorted on cell prefix and is being updated after every received binding add and binding update message. As mentioned in 6.5.2.3, this struct is controlled by the same mutex as the address struct. This is because a different care-of address may indicate another cell prefix. Therefore, both structs are linked directly, and only one at a time should be available for reading or writing.

The last part of the information database is a linked list containing the socket descriptors of all currently connected clients. This list is used to loop through to make future broadcasting functions easier. It can also be used to retrieve information on the current number of connections. Its declaration looks as follows:

```
struct clients {
  int socket;         // Client's socket descriptor
  char ip[40];        // Client's IP address (either IPv4 of IPv6)
  int flags;          // Client's flag (entry_valid or entry_remove)
  struct clients *next;   // Pointer to next struct clients
};
```

This struct is analogue to the cells struct but with an extra field. Instead of cell prefix, the client's socket descriptor is stored in the first field. This is the same socket as is stored in the address struct. The second field stores the IP-address of the connected client. This can be either an IPv4 or an IPv6 address. Finally, there are a flag and a pointer to the next struct. The list is being updated after each opened or closed connection. As this list has no direct connection with the other information, it can use its own mutex to block threads.

6.6.3 Update considerations

The lifetime of each mobile node's binding is the absolute time to expiration of this binding. It is stored statically in the database. The database does not compute new values of the lifetime itself. Instead, it depends on the binding update messages from the mobile node's home agent application for lifetime synchronization. There are several reasons for this choice:

- The load on the database may influence its calculation speed, which may lead to an incorrect lifetime calculation;
- In case of the expiration of a lifetime in the home agent's binding cache, its application immediately sends a binding delete message to the database. This means the validity of the mobile node's presence in the database is always secured, though its lifetime may be stored with a small deviation;
- There is no advantage for a mobile node to be able to retrieve the exact value of the lifetime of a binding, as long as the validity of this binding is guaranteed;

The updating of the lifetime values can be achieved in two ways: with a continuous connection or with a connection that will be set-up for each message. Initially, the first option is chosen because of the frequency of binding updates (every 3 seconds for every mobile node) and the relative easy implementation. Drawback of this choice is that unnecessary connections waste processing power and memory. However, considering the relative low amount of home agents in an arbitrary LBS network (compared to mobile nodes), this will not easily become significant. If so, the home agent's application can be

adjusted to disconnect after each message. This leads to a recommendation that will be described in section 8.2.

6.6.4 Robustness

In case the database is not able to properly process a received message, it returns an error message containing the reason for the occurrence of the error. The more errors the database server's application can handle, the more robust it functions. However, there are other aspects that influence the robustness of the database application. For instance the handling of the following events:

1. Disconnection from a client;
2. Not receiving a required response message;
3. Receiving an invalid message;

6.6.4.1 Client disconnection

Both sides can close the connection between the client and database server. In case the database server closes the connection to a mobile node, the mobile node is removed from the client list, its subscriptions are deleted and the corresponding thread is exited. The database server closes the connection to a client in the following situations:

- The sending of a notification message to a mobile node fails. This is the case when a mobile node disconnects unannounced;
- The database server detects an disconnection by receiving a trigger from the TCP layer;

Currently, no measures are taken when the sending of a response message to a client fails. However, as will be stated in 8.2, it is recommended to handle the occurrence of this event more properly.

If the home agent's application initiates the disconnection, it sends a binding delete message for all mobile nodes in its binding cache, prior to the disconnection itself. This ensures there are no mobile nodes in the database, for which no binding update or binding delete message will be received anymore. The implication of this measure is that a possible registration and subscription for this mobile node are lost. Furthermore, no information on this node can be queried anymore. The actual disconnection is detected again by receiving a trigger from the TCP layer for the corresponding socket pair.

If the connection between home agent and database server gets disconnected unintended, the home agent's application detects this event and automatically re-connects to the database server. In the meantime, the database server does not update or delete the mobile node bindings of this home agent. This means that during the reconnection time the information in the database may be outdated. See section 8.2 for recommendations regarding this topic.

6.6.4.2 No response message

As explained in 6.4.3, the notification message is the only message that is initiated by the database server and responded to by a client. The notification acknowledgement message indicates a proper delivery and processing of the notification. In case an error message is returned, processing at the client yielded a problem. Currently, this error information is only displayed on the screen. Depending on the error, the application either continues without taking any action, or the connection causing the error is closed.

A better way to handle the mobile node acknowledgement messages is to wait for them before sending another message. In case there was an error, the entire message can be sent again. But as will be shown in Chapter 7, this would cause a too large delay in the database server's application.

6.6.4.3 Invalid message reception

Because it is very unlikely an invalid message will be generated by an application once they all function well, handling of such events is quite straightforward. The mobile node's application, which will be discussed in 6.7, does not allow its user to enter any text, thereby reducing the change of protocol violations. The mobile node's user can only use predefined options or information retrieved from the database server to contact the database server.

However, if the database application detects a violation of the messages as defined in section 6.4, the corresponding thread is immediately exited and the corresponding connection is closed. For debugging purposes, the tag that caused the violation and the database's current contents are printed to the screen.

6.7 Mobile node application functioning

As stated in section 2.1, the mobile node's application will be implemented in Java. Other participants in the ISIDOR project have performed this task. The application consists of two parts. The first part is based on the application described in paragraph 5.3.5. It can be used to inform the user of his current location in the LBS network and to show some information with respect to that location. Currently, this is a Bluetooth image in cell 2 and an image of the iPAQ in cell 1. In case no cell is found and the mobile node is not able to retrieve a care-of address, the ISIDOR logo is shown. See Figure 6-11 for a screenshot.

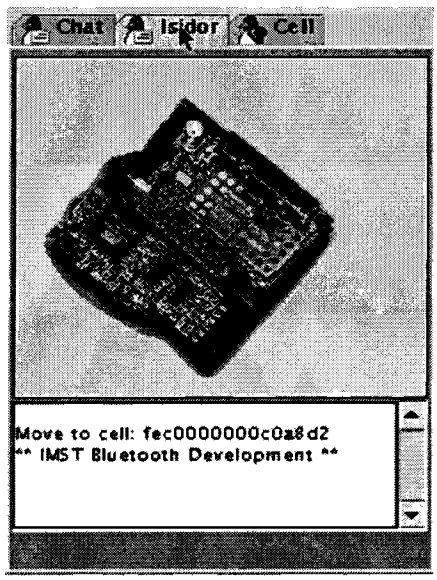


Figure 6-11. Mobile node application showing current cell information.

The second part displays the information obtained from the database server. After starting the application, the mobile node first determines its current location as described in the previous application. Next, it looks for a configuration file for to obtain its name and user to register with. A successful registration is followed by a subscription for the cell the mobile node is currently residing in.

Next, it sends a query cells message for the cell corresponding to this location. This yields information on all other mobile nodes in this cell. The information is used to compose the layout as can be seen in Figure 6-12.

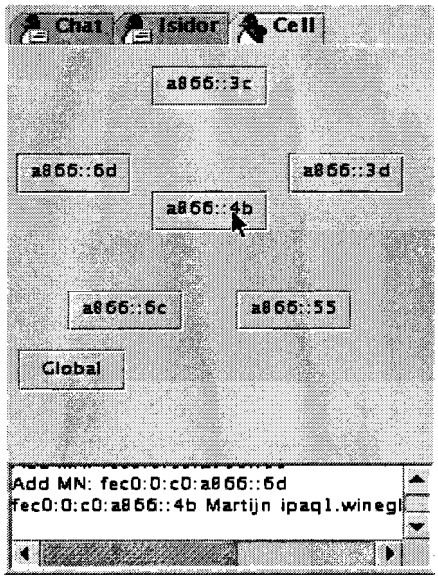


Figure 6-12. Mobile node application showing network information.

The mobile node's home address is displayed in the centre, surrounded by the mobile nodes listed in the query cells response message. In case of change in the cell, a notification is received and the display is updated according to the changes. Clicking on a home address triggers the sending of a query home address message for that address. The returned information (name, user, cell and any other information available) is shown in the bottom window. The button "Global" is meant to change the subscription type to, for instance, all cells.

Chapter 7. Measurements

7.1 Introduction

The work described in this Chapter serves two functions:

1. Verify the functionality of the database server as it has been described in Chapter 6;
2. Retrieve information on the reliability and the speed of the database server application and determine whether it meets the requirements given in paragraph 2.1.2;

All communication between the mobile nodes, home agents and database server takes place by means of the messages defined in the previous Chapter. Each of these messages is answered by an acknowledgement, an error or a (query) response message. The faster an answer arrives at its destination, the more up to date the information within the LBS network will be. There are several factors that influence the delay between the sending of a message and the delivery of its answer. These factors can be divided into two categories. The first category contains factors that are network related, for instance:

- Configuration of the network: this includes many factors, such as routing and the speed of the connections between the nodes;
- Load on the network: congestion, for instance, increases the time to deliver a packet at its destination;
- Size of the network (number of hops): generally, the more hops a packet has to travel, the more time it takes;
- Load on the database: the more home agents and mobile nodes communicate with the database simultaneously, the slower a response will be generated;

The second category contains the factors that depend on the implementation as described in Chapter 6. These are, for instance:

- Speed of the database server's implementation: the more time it takes for the application to generate a single response or notification, the longer the total delay will be;
- Efficiency of the database: the more efficient the parallel threads and the locking of mutexes are handled, the faster the database server works;

As many factors in the first category are hard to measure in the relatively small network of Figure 3-3 and are beyond the scope of this thesis, only some quantitative results for a fixed configuration will be presented. The focus lies on the aspects of the second category, thus trying to determine the quality of the implementation as described in the previous Chapter. Section 7.2 presents different scenarios to measure the most important delays caused by the database server's application. The results of these measurements will be presented in section 7.3. The last section contains the conclusions that can be drawn from the results.

7.2 Scenarios and measurements

7.2.1 Scenarios

Generally, there are two different types of delay, caused by the database server, to be measured:

1. Delay between the reception of a message at its interface and the sending of a corresponding response message. Each type of response has its own delay. The response is unicasted back to the home agent or mobile node that sent the initial message;
2. Delay between the reception of a message causing a notification and the sending of this notification to the mobile nodes subscribed to the particular cell. The more mobile nodes are subscribed to a cell, the more notifications will be sent. The set of recipients does not necessarily have to include the client that triggered the notification;

The three scenarios to measure the most important delays are:

1. A new mobile node enters the LBS network. The delay between a binding add and response message will be measured (type 1 delay);
2. A handover for a mobile node, without notifications. The delay between a binding update indicating a cell change and its response will be measured in the case no mobile nodes are subscribed to the particular cell (type 1 delay);
3. A handover for a mobile node, with notifications. This scenario enables us to measure the additional delay caused by the sending of notifications (type 2 delay);

Note that, strictly speaking, the term 'message' is not correct in this context, as it belongs to the application layer, while we will be measuring delays at a lower layer. The messages, as defined in the previous Chapter, are actually stored in the data part of a TCP-segment. But as the TCP-term 'segment' is at least confusing and hardly used outside the TCP-world in this context[24], we will use the term 'message' in this Chapter for any protocol data unit (PDU) sent by a transport layer entity.

As stated in 7.1, there are multiple factors that influence the delays. By using different scenarios, we will try to determine what factor is the bottleneck and how we can reduce the delay it causes. See Figure 7-1 for a fictive example of a binding add message and its response. At t = 0 ms. a binding add message is sent from the home agent to the database server. It takes 0.05 ms. for the message to arrive at the database. A 0.10 ms. later an acknowledgement message has been generated by the database. This response arrives at the home agent after an additional 0.06 ms. From this example we can conclude that the total delay is 0.21 ms., of which the network causes 0.11 ms. and the database server causes the remaining 0.10 ms.

[24] According to [6], section 6.6, page 588.

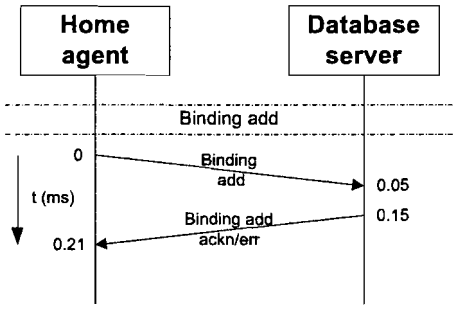


Figure 7-1. Example of delay measurement.

7.2.2 Measurement method

To measure the delay between the deliveries of two corresponding messages, we need a tool that can both visualise the messages sent and is, at the same time, able to measure the elapsed time in a sufficient accurate way. The Ethereal tool, which has been described in paragraph 5.2.2.4, satisfies these conditions. With Ethereal, we are able to see each message and its arrival time at a node's interface. By measuring at multiple nodes at the same time, the total delay can be divided in hop-by-hop delays. However, there is a problem when using Ethereal on multiple nodes in the network simultaneously. Though the accuracy of time measurement at each node is sufficient, the offset between the clocks of the various nodes is in the same order of magnitude as the time it takes for a message to arrive at the next node. Moreover, this offset has a drift (a so-called jitter), so it cannot be accounted for by adding a constant factor. This means it is very hard to measure the interval between the sending of a message at a hop and its arrival at the next.

To solve this problem, we will estimate this interval, instead of measuring it directly. To be more specific, the delay for sending a message between the home agent and the database server, as well as between the database server and a mobile node are to be estimated. These delays depend on two factors:

1. The chosen path between source and destination. Different paths can have different connection speeds and a different number of hops. This aspect is only network-related and therefore constant for a fixed source and destination[25];
2. The size of the message sent. The larger the amount of bytes to process, the more time it takes for a node to store and forward the message, especially when it is a router. Of course, it also takes more time to send the message over the line;

To estimate these intervals, the following measurements have been done. First, 100 *ping6*[26] messages are sent from source to destination and the average round trip time is calculated. To indicate the reliability of the average value, the standard deviation can be calculated. The standard deviation σ is defined as:

$$\sigma = \sqrt{\frac{n\sum x^2 - \left(\sum x\right)^2}{n^2}} \qquad (1)$$

[25] Additional delays due to congestion or other unforeseen circumstances are not accounted for in these measurements.

[26] This is a regular *ping* command (which sends ICMP ECHO_REQUEST packets to network hosts), but with an ICMPv6 header instead of ICMPv4. Both ICMP headers have a fixed size of 8 bytes.

Where x is the measured value and n is the total number of measured values. We assume a normal distribution of the measured values, which means that about 68 % of the values lie within one standard deviation about the mean, and about 95 % of the values lie within two standard deviations about the mean.

7.2.3 Message sizes

Next, this measurement is repeated for different ICMP data sizes to investigate the influence of the message's size. The ICMP data size is the same as the packet size that can be entered after a ping6 command. Measurements for packet sizes of 100, 200, 300, 400 and 500 bytes will be done. To be able to apply the results on the defined messages of Appendix E, we have to account for the 8 byte ICMP header to obtain an expression that is a function of the IPv6 payload size.

The defined messages are carried in TCP data instead of ICMP data. To compare both forms of data, we will add the 20 bytes of the TCP header to the TCP data size of a message to obtain the IPv6 payload size. This size can be used in the estimated round trip value for the ping6 messages. See Figure 7-2 for details. The only possible inaccuracy in this method is the implicit assumption that the processing of an ICMPv6 header of 8 bytes takes as much time as the processing of a 20-byte TCP header.

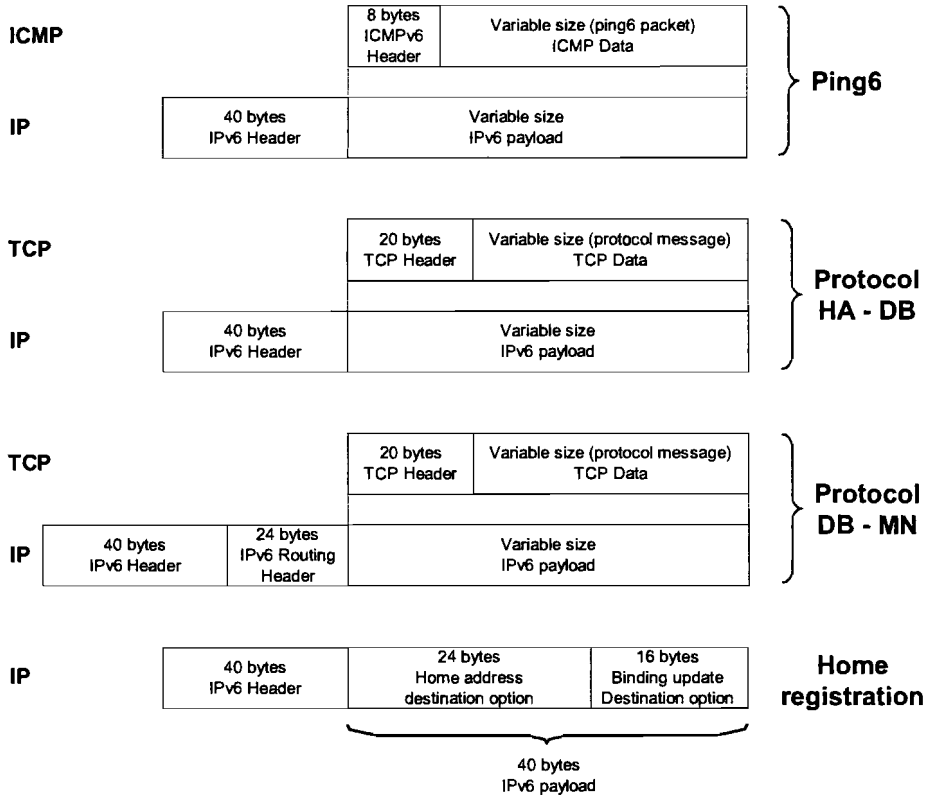


Figure 7-2. Message size differences.

In case of the binding update destination option (a home registration), there is no TCP, nor ICMP data present. This means we can use the IPv6 payload size of 40 bytes without additional calculations. This yields the IPv6 payload sizes as listed in Table 7-1

Table 7-1. IPv6 payload size of defined messages.

Message	Size (byte)
Ping6 with packet size *x*	*x* + 8
Binding update destination option (home registration)	40
Binding update destination option acknowledgement	40
Binding add	255
Binding add acknowledgement	139
Binding update	169
Binding update with cell change	261
Binding update acknowledgement	145
Cell change notification	167
Cell change notification acknowledgement	139

We expect the round trip to increase linearly with increasing payload size, as processing the double amount of data takes the double amount of time. As can be seen in paragraph 7.3.1, the measurements confirm this assumption. Therefore, a first order polynomial (a so-called trend line) will be fitted through the five averaged values. This yields an expression for the expected round trip delay as function of its size (in a multiple of 100 bytes). With this expression we are able to calculate the expected delay for a message if its data size and route are known.

The R-squared value of the fitted polynomial is also computed. The R-squared value is an indicator between 0 and 1, which reveals how well the estimated values of the trend line correspond to the actual data. It is also known as the coefficient of determination and is defined as:

$$R^2 = 1 - \frac{Mean\ square\ error}{Variance} = 1 - \frac{\sum\left(Y_j - \overline{Y}_j\right)^2}{\left(\sum Y_j^2\right) - \frac{\left(\sum Y_j\right)^2}{n}} \qquad (2)$$

Where Y_j is the measured value, $\overline{Y}_j$ the approximated value and n the total number of values. The closer R-squared is to 1, the better the trend line fits the averaged values. Note that the variance in equation (2) is not equal to σ^2 of equation (1), as it calculated for the fitted curve instead of an averaged ping6 value.

7.2.4 Measurement set-up

The measurements will be done with the network configuration of Figure 3-3. In this configuration, the Wine220 computer serves as home agent for all mobile nodes in the LBS network. Router 5 is used as database server. Furthermore, cell 1 corresponds to the area covered by access point 110 (AP110) and cell 2 corresponds to the area covered by access point 210 (AP210). The areas overlap to be sure a cell change is possible.

In this configuration, messages sent from the home agent to the database server have to travel only one hop, where messages from the database server to a mobile node have to travel six hops. As messages for a mobile node in cell 2 follow a different route as messages destined for a mobile node in cell 1, both values have to be estimated

separately. The response messages travel the same route with the hops in reverse order and are considered to have the same delay. The exact routes are as follows (see Figure 3-3 for details):

- Home agent to database server: Wine220 – Router 5;
- Database server to mobile node in cell 1: Router 5 – RTC – RTD – Router 2 – Switch 110 – AP110 – MN;
- Database server to mobile node in cell 2: Router 5 – RTC – RTB – Router 3 – Switch 210 – AP 210 – MN;

In this configuration, all messages between the home agent, database server and the mobile nodes go either to or through router 5. This allows us to run just one instance of Ethereal on router 5 to monitor all the traffic we are interested in. Next, we use the estimated round trip values to calculate the other delays in the scenario.

7.3 Measurement results

In this section the results of the scenarios and measurements described in 7.2 are presented. First, the three round trips are estimated in paragraph 7.3.1. With these values, the delays in the different scenarios are calculated in paragraph 7.3.3.

7.3.1 Round trip estimations

As explained in paragraph 7.2.4, three intervals have to be estimated:

1. Home agent to the database server, see 7.3.1.1;
2. Database server to a mobile node in cell 1, see 7.3.1.2;
3. Database server to a mobile node in cell 2, see 7.3.1.2;

7.3.1.1 Home agent to database server

The measured values and their fitted curve for messages between the home agent and the database server can be seen in Figure 7-3. The standard deviation of each averaged ping6 value is also plotted by means of five confidential intervals.

The trend line equation in this figure is a function of the ICMP data size in a multiple of 100 bytes. Note that to derive an expression for the round trip value as a function of the IPv6 payload size in bytes, we will also have to account for the ICMP header size of 8 bytes. This yields the following expression:

$$t = 3{,}29 \cdot 10^{-4} \cdot (x - 8) + 4{,}157 \cdot 10^{-1} \quad [ms] \tag{3}$$

With x the IPv6 payload size in bytes.

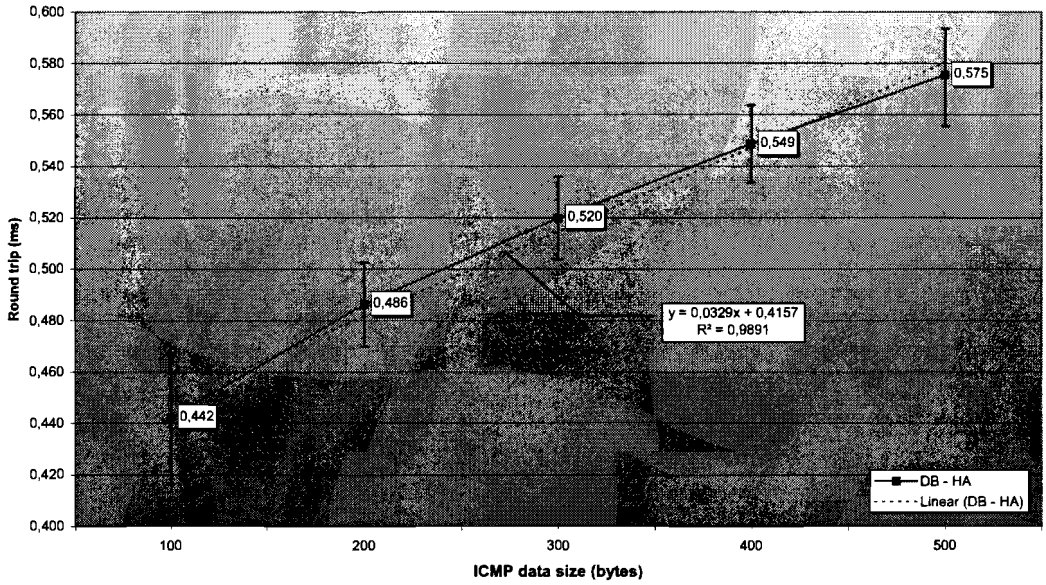


Figure 7-3. Round trip time between the home agent and the database server.

7.3.1.2 Database server to mobile nodes

The measured values for the round trip time between the database server and a mobile node in cell 1 can be found in Figure 7-4.

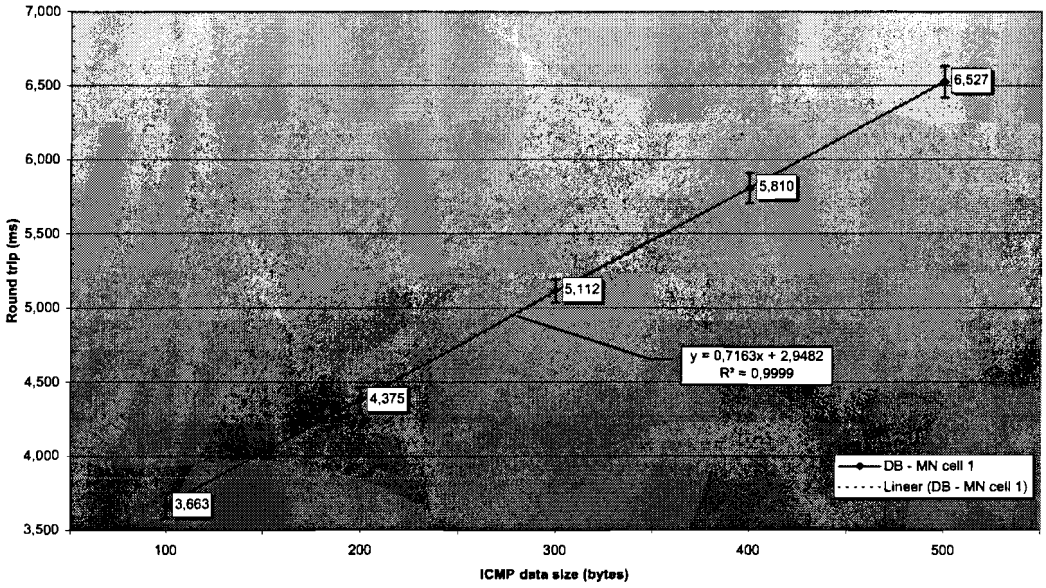


Figure 7-4. Round trip time between the database server and a mobile node in cell 1.

In this figure we can see that the linear approximation for the round trip between the database server and a mobile node in cell 1 is given by:

$$t = 7{,}163 \cdot 10^{-3} \cdot (x - 8) + 2{,}9482 \quad [ms] \tag{4}$$

With *x* again the IPv6 payload size in bytes. As the number of hops is greater, equation (4) yields a higher value than (3) for the same packet size. Probably due to this longer delay, the standard deviations of the averaged values are smaller than the ones in Figure 7-4. This also results in a higher R-squared value (0,9999) for the curve of equation (4). The same measurement results for a mobile node residing in cell 2 can be found in Figure 7-5.

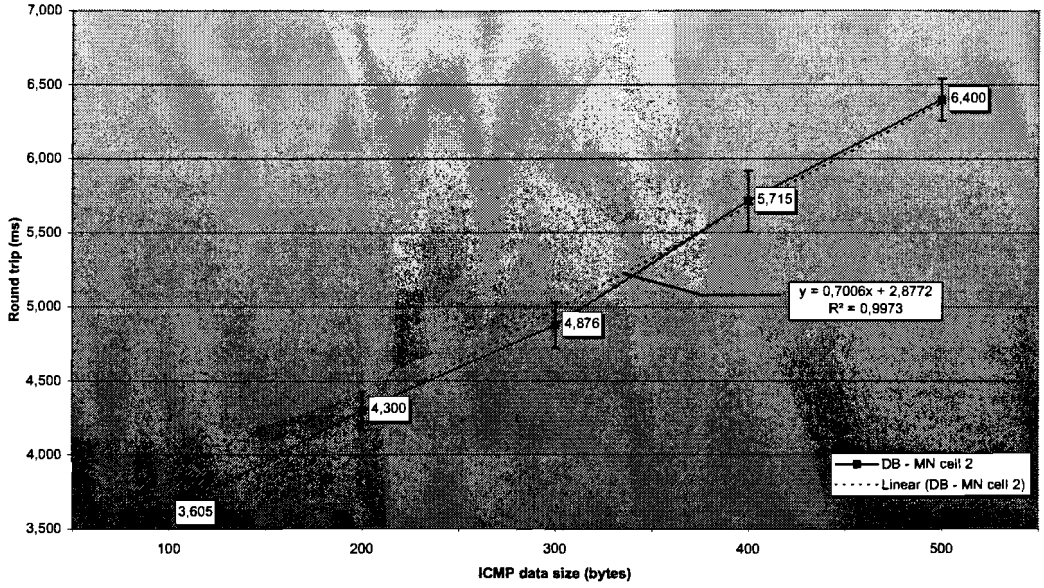


Figure 7-5. Round trip time between the database server and a mobile node in cell 2.

This yields the following expression for round trip time from the database server to a mobile node in cell 2:

$$t = 7{,}006 \cdot 10^{-3} \cdot (x - 8) + 2{,}8772 \quad [ms] \tag{5}$$

The R^2 value of this measurement is 0,9973. This result is comparable to the one of equation (4). Mainly due to the averaged value for an ICMP data size of 300 bytes, the R-squared value is somewhat lower.

7.3.2 Round trip conclusions

With equations (3) through (5), the delays of the messages of Table 7-1 can be calculated. For example: to calculate the delay to send a binding update message of 169 bytes from home agent to the database server, we use the following expression:

$$t = \left(3{,}29 \cdot 10^{-4} \cdot (169 - 8) + 4{,}157 \cdot 10^{-1}\right) / 2 = 0{,}234\ ms. \tag{6}$$

Where *169* is the IPv6 payload size of a binding update message, as can be seen in Table 7-1. Note that we have to divide the entire delay by 2 because equation (5) is a round trip value.

The home registration message causes some difficulties. In paragraph 7.3.1, only the round trip times between the database server and the home agent, and between the database server and a mobile have been measured. However, the home registration

message is being sent directly from mobile node to his home agent, but goes through router 5. We can solve this problem by adding the delays from mobile node to database server and database server to home agent, as well as the delay caused by passing router 5. The first two delays are known, the last can be calculated from an Ethereal trace on all interfaces of router 5, while a home registration is monitored. We can simply subtract the moment the home registration message is received on the eth0 interface from the moment it is seen as an outgoing message on eth1. The same procedure can be followed for the corresponding acknowledgement message. The results of these calculations are the values that can be found in Table 7-2.

Table 7-2. Calculated message delays.

Message	Size (byte)	Delay (ms)
Binding update destination option (home reg.) from cell 1	40	1,878
Binding update destination option (home reg.) from cell 2	40	1,839
Binding update destination option from cell 1 ackn.	40	1,878
Binding update destination option from cell 2 ackn.	40	1,839
Binding add	255	0,245
Binding add acknowledgement	139	0,228
Binding update	169	0,232
Binding update with cell change	261	0,246
Binding update acknowledgement	145	0,229
Cell change notification to MN in cell 1	167	2,001
Cell change notification to MN in cell 2	167	1,954
Cell change notification acknowledge from MN in cell 1	139	1,909
Cell change notification acknowledge from MN in cell 2	139	1,864

7.3.3 Scenario measurements

This paragraph presents the results of the measurements for the scenarios as described in 7.2.1. Together with the results of paragraph 7.3.2, all times within the different scenarios are known.

7.3.3.1 New mobile node enters LBS network

Whenever a new mobile node enters the LBS network, it sends a MIPv6 binding update destination option to its home agent first. This update is triggered by a router advertisement. The broadcasting of these advertisements can be controlled with the router advertisement daemon, as discussed in paragraph 5.2.2.2. The binding update destination option is also known as a home registration and has been described in paragraph 4.4.1. It serves as notification for the home agent of the mobile node's acquired care-of address. In a home registration, the "acknowledge" bit is always set, indicating a binding acknowledgement from the home agent is requested. Before sending this acknowledgement, the home agent MUST multicast a "gratuitous" neighbour advertisement with the new mobile node's home address as source address [10]. Any node receiving this multicast will update its binding cache with the new information.

When the home registration has completed, the home agent's application generates and sends a binding add message to the database server. The sequence chart of the complete scenario can be seen in Figure 7-6.

The measurements have been done with just one mobile node that is powered on in cell 2, after the home agent and database server's applications have been started. The measurements have been repeated 20 times after which they are averaged to increase accuracy. The intervals ($t_1 - t_0$), ($t_4 - t_3$), ($t_6 - t_5$) and ($t_8 - t_7$) can be derived from Table 7-2. The other intervals are determined by the measurements and can be found in Table 7-3.

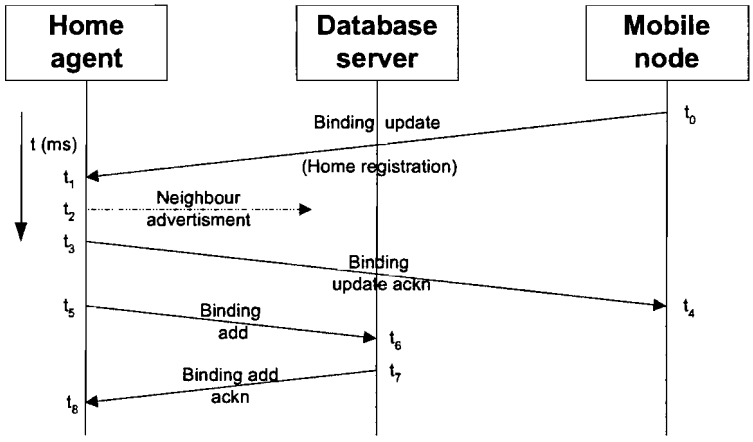


Figure 7-6. New mobile node startup scenario.

Table 7-3. Home agent and database delays for new mobile node startup scenario.

Message generation	Interval	Delay (ms)
Binding update – Neighbour advertisement	$t_2 - t_1$	0,151
Neighbour advertisement – Binding update ackn.	$t_3 - t_2$	0,120
Binding update acknowledgement – Binding add	$t_5 - t_3$	53,79
Binding add – Binding add acknowledgement	$t_7 - t_6$	2,768

This yields the scenario of Figure 7-7. Note that the interval ($t_7 - t_6$) can be significantly larger in case there are mobile nodes subscribed to cell 2, as the binding add acknowledgement is only sent after the last notification. This effect will be examined in more detail in paragraph 7.3.3.3.

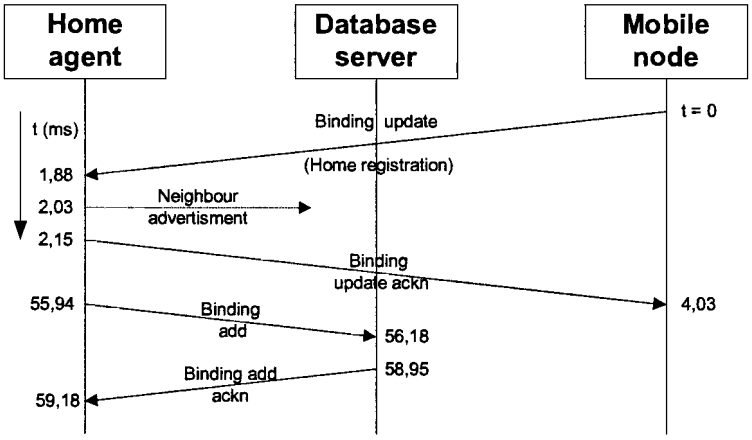


Figure 7-7. Results of the new mobile node startup scenario.

The longest delay in this process is caused by the generation of a binding add message by the home agent's application. This is due to the fact the application sleeps for 100 ms between checks of its binding cache. In case of a change other than the lifetime, a message for the database server is generated and sent. If there have not been any changes for 3 seconds, binding update messages for all mobile nodes in the binding cache are generated. So, on the average, it takes the home agent's application 50 ms to detect a new mobile node.

7.3.3.2 Mobile node handover without notifications

The mobile node handover time is an important aspect in the performance of the MIPv6 network, as well as in the performance of the database server's application. A cell change is, on IP level, handled in the same way as described in the previous paragraph. Only the neighbour advertisement multicast is left out, as the mobile node's home address is already known within the network. This does not change the delays significantly, however. Therefore, we limit ourselves to the delay caused by the database server's application. This will be different from the previous scenario, as the mobile node does not have to be added to the database anymore. Again there are no mobile nodes subscribed to the concerning cells, so no notifications are sent. See Figure 7-8 for the scenario.

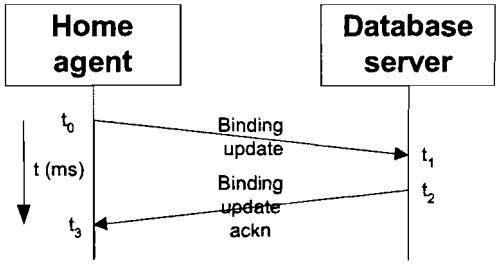


Figure 7-8. Mobile node handover scenario without notifications.

The measurements are done with the same set-up as in the previous scenario, but instead of restarting the mobile node, it is repeatedly moved from one cell to cell the other. The duration of the intervals ($t_1 - t_0$) and ($t_3 - t_2$) can again be found in Table 7-2. The delay the database causes is equal to ($t_2 - t_1$). The result of the measurements can be found in Table 7-4. This result leads to the scenario of Figure 7-9.

Table 7-4. Database delay for mobile node handover scenario.

Message generation	Interval	Delay (ms)
Binding update – Binding update acknowledgement	$t_2 - t_1$	2,131

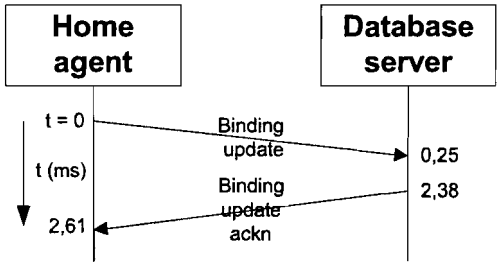


Figure 7-9. Results of the mobile node handover scenario without notifications.

7.3.3.3 Mobile node handover with a single notification

The next scenario is the one of Figure 7-10. A mobile node changes its cell, which will be detected by the database server by means of a received binding update message. This event triggers the sending of a notification to all mobile nodes subscribed to the concerning cells (one cell for the departure of the mobile node and the other cell for the new mobile node).

The first measurements have been done with one mobile node subscribed to its own cell, which is cell 2. Another mobile node changes repeatedly from this cell to cell 1, causing notifications to be sent for each cell change. The mobile node's application, as described in section 6.7, returns an acknowledgement message for each successfully received notification.

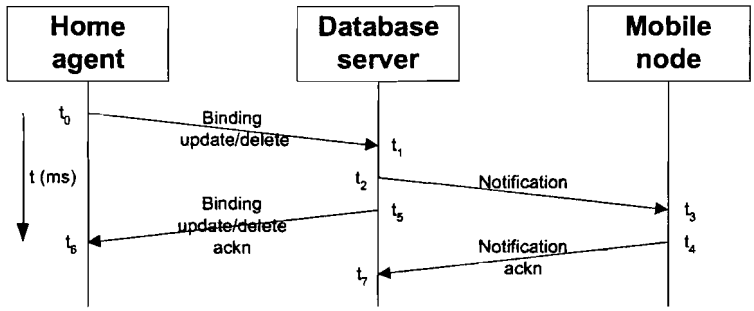


Figure 7-10. Mobile node handover scenario with a single notification.

From the measurements, the intervals between receiving and sending a message at the database server can be derived. These are the intervals between a binding update and notification message ($t_2 - t_1$), as well as the interval between a notification and binding update acknowledgement message ($t_5 - t_2$). The other intervals have already been calculated in Table 7-2. With these values, we are also able to compute the time the mobile node requires to generate a notification acknowledgement. This period corresponds to ($t_4 - t_3$) in Figure 7-10. This yields the database delays of Table 7-5.

Table 7-5. Database delays for mobile node handover scenario with a single notification.

Message generation	Interval	Delay (ms)
Binding update – Notification	$t_2 - t_1$	1,344
Notification – Binding update acknowledgment	$t_5 - t_2$	1,489
Binding update – Binding update acknowledgement	$t_5 - t_1$	2,833

In addition to the database delay values of Table 7-5, there is the delay between the receiving of a notification and the sending of a notification acknowledgement by the mobile node's application. This turns out to be, on the average, more than 500 ms. Together with the results of Table 7-2, this yields the scenario that can be found in Figure 7-11.

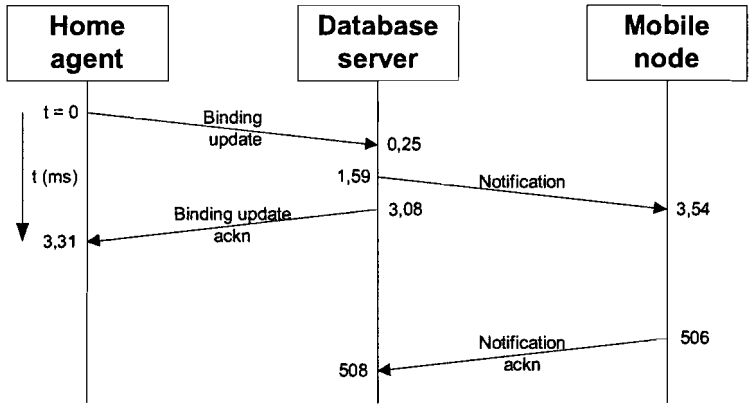


Figure 7-11. Results of the mobile node handover scenario with a single notification.

As the database server only returns a binding update acknowledgement message after all notifications have been sent, we are also interested in the effect of the number of subscribed mobile nodes. Therefore, the next scenario consists of 4 extra mobile nodes, subscribed to their current cells (cell 2). So, in total five notifications will be sent for each binding update that contains a cell change. See Figure 7-12 for the scenario.

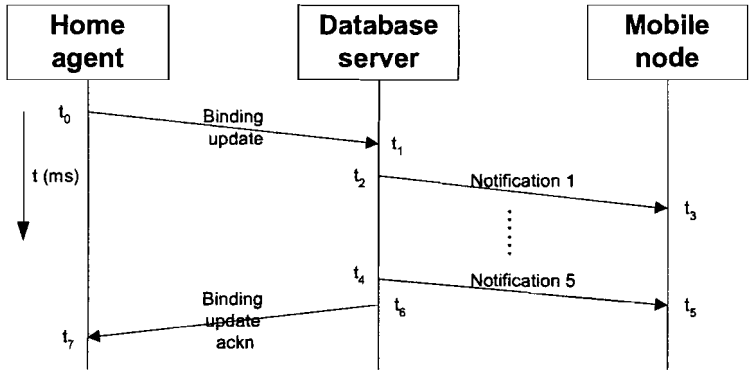


Figure 7-12. Mobile node handover scenario with 5 notifications.

The notification acknowledgements sent by the mobile nodes are left out in this scenario, as they do not influence the other results and it is already clear a mobile node's application generates an answer much slower than the database server's application.

The intervals $(t_1 - t_0)$, $(t_3 - t_2)$, $(t_5 - t_4)$ and $(t_7 - t_6)$ can again be derived from Table 7-2. The others are derived from the measurements and can be found in Table 7-6.

Table 7-6. Database delays for mobile node handover scenario with 5 notifications.

Message generation	Interval	Delay (ms)
Binding update – First notification	$t_2 - t_1$	1,310
First notification – Last notification	$t_4 - t_2$	0,358
Last notification – Binding update acknowledgment	$t_6 - t_4$	2,295
Binding update – Binding update acknowledgement	$t_6 - t_1$	3,963

The resulting scenario is the one of Figure 7-13.

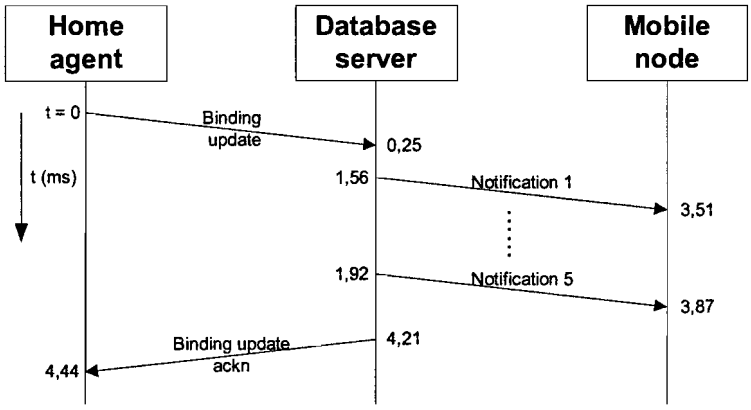


Figure 7-13. Results of the mobile node handover scenario with 5 notifications.

7.4 Measurement conclusions

The conclusions that can be drawn from the results presented in this Chapter are divided in the round trip estimations and the scenario measurements.

7.4.1 Round trip measurement conclusions

7.4.1.1 Linear round trip approximations

The assumption that there is a linear relation between packet size and round trip delay seems to be justified by the different measurements presented in paragraph 7.3.1. The trend line lies between the confidential intervals in every measurement. Only the measurements for the delay between the home agent and the database server do not fully satisfy the linear assumption. However, after two additional measurements and a comparison with the other results, a linear approximation is still considered to be the most applicable one. The deviation of this measurement is probably due to the fact that a low number of hops decreases the accuracy of the measurements. This can also be seen in the relatively large confidential intervals shown in Figure 7-3, compared to the other measurements.

7.4.1.2 Inaccuracies

The only inaccuracies lie in the difference between the ICMPv6 header of a ping6 message and the TCP header of a message sent between the different entities, and in the assumption that the routes will cause the same amount of delay in the reverse direction. The first inaccuracy can –if required- be accounted for afterwards, by means of a constant factor. The second inaccuracy is harder to compensate afterwards, but according to the measurements it appears to be insignificant.

7.4.1.3 Differences between cell 1 and cell 2

Finally can be concluded that the delays for sending a message to a mobile node in cell 1 and cell 2 are almost equal. This is not surprising, as both routes completely consist of comparable hops.

7.4.2 Scenario measurement conclusions

7.4.2.1 New mobile node scenario

From Figure 7-7 we can conclude that a new mobile node knows in approximately 4 ms. whether his home registration is successful. The bottleneck in this scenario is formed by the detection mechanism of the home agent's application. Once a new mobile node is detected and a binding add message is sent (after approximately 50 ms.), the processing time of the database server is in the same order of magnitude as a home registration message, received by the home agent.

Looking at Table 7-3 and Table 7-4, we can conclude that the delay caused by the database server in case of a binding update message indicating a cell change, is smaller than in case of binding add message, as expected: 2,131 vs. 2,768ms, a decrease of about 23%. This value is still limited by the implementation of the database server. The database server application always generates a notification message for each binding update message where a cell change is involved. Only after this generation it looks in its database to see whether there are nodes subscribed to the concerning cells. As binding updates are likely to be received in greater numbers than binding add messages, it may be worth the trouble to increase the speed of the database in this area. This can be done by only generating notification messages in case there are mobile nodes subscribed to the concerning cells.

7.4.2.2 Mobile node handover scenarios

From Figure 7-11 we can conclude it is a good decision that the database server does not wait for a notification acknowledgement message, before returning a binding update acknowledgement message to the home agent. After all, generation of the mobile node's response takes more than 370 times as much time as the generation of the notification by the database server. The effect of sending an acknowledgement before returning a binding update acknowledgement to the home agent is also visible in this Figure. The delay caused by the database before returning this binding acknowledgement has increased by almost 33%. When 5 notifications are sent, the increase is about 85% (see Figure 7-13), which is 0,79 ms. per notification. This means the delay per notification decreases, as the total number of notifications sent gets larger. This is because of the fact that the sending of a notification takes a relatively low amount of time compared to its generation.

Chapter 8. General conclusions and recommendations

8.1 General conclusions

Looking back at the assignment and the corresponding work, several conclusions can be drawn. They are divided into two groups: functionality of the concept of the entire project and the performance of its implementation.

8.1.1 Functionality

General
From the previous Chapters can be concluded that the combination of mobile IPv6 with Linux and wireless LAN meets the requirements of the ISIDOR project. The implementation of the database server makes it possible for every mobile node in the LBS network to retrieve information on both the network topology and its mobile nodes. The information is graphically presented on the PDA and can be updated automatically. For the user, no knowledge on MIPv6 is required for a correct functioning of the mobile node's application.

Wireless access technology
However, the choice for wireless LAN also has some drawbacks. A mobile node cannot communicate in two cells simultaneously, disabling the opportunities of a service proxy. Furthermore, the size of the cells is quite large, which limits the accuracy of the location determination. Only by reducing the transmitting power of the access points this problem can be solved. Finally, the power consumption of the wireless LAN PCMCIA cards is still too high to ensure long use of the mobile node. All the problems with respect to wireless LAN can be solved or reduced by using Bluetooth as wireless access technology. Unfortunately, this was not yet possible at the start of the project due to insufficient support for usage on a PDA, combined mobile IPv6.

Linux distribution choice
The choice for the Familiar distribution has turned out to be a good one. Familiar has proven to be the best documented iPAQ distribution available, with a lot of active developers and debuggers. Frequent kernel updates implement new possibilities and reduce the number of bugs. The number of Familiar users is still growing, in contradiction to the other distributions considered in Section 5.3. For instance, the open source development of PocketLinux has completely stopped as it is going to be a commercial product.

Assignment
Looking back at the assignment, not all tasks have been fulfilled. There are two main reasons for this. First of all, some elements of the assignment took more time than

previously thought. Especially a reliable software implementation of the database server took a large amount of time. This was mainly due to my limited experience in writing large programs in the C-programming language, resulting a long debugging period. The measurements described in Chapter 7 are only reliable in case the software implementation is reliable as well.

Secondly, the scope of the ISIDOR project was slightly altered after the composition of the assignment. For instance, less attention than expected was given to security and QoS aspects of the IPv6 network. Furthermore, the development of the actual content for location based services in Java was considered to be a too much time consuming task, compared to the size of the project and its participants. For this reason the developed demonstration applications are rather straightforward. However, this does not reduce their usefulness, as the technological background and requirements are still unaltered.

8.1.2 Performance

The use of mobile IPv6 ensures quick and reliable hand-overs. Hand-overs are completed well before a mobile user is likely to leave the cell again. The database server functions properly and is able to handle simultaneous client connections. Generation of response messages is fast, compared to the other two entities. Updating of its information roughly takes the same amount of time as the change takes on mobile IP-level.

It is hard to predict its performance in a much larger LBS networks, as no simulations on this subject have been done yet. However, once the recommendations of section 8.2 are implemented, it looks like scaling up to the size of the intended ISIDOR scenarios is possible.

8.2 Recommendations

Based on the conclusions of section 8.1 and the report in general, several recommendations for future work can be done. They are divided into two categories: recommendations with respect to LBS network and recommendations with respect to the implementation of the entities in the LBS network.

8.2.1 LBS and network related

The following recommendations can be made with respect to location based services and the mobile Ipv6 network:

Development of applications using LBS concepts
As stated in 8.1, the task of implementing a location based service in Java has not been completely fulfilled. Main reason for this is the fact it is a time-consuming task. However, without the appropriate services implemented on the mobile nodes, the LBS network is of no true use for its end users. The more services are implemented, the more valuable the ISIDOR project will be for both IMST and potential costumers.

Perform research on QoS
As more location based services are going to be developed and used, the quality of these services becomes more important. Available bandwidth becomes smaller and congestion or latency may increase. This is undesirable for real-time applications as VoIP and unpleasant for all other applications, as a bigger much latency is generally allowed for

data transfer. Defining and implementing QoS-parameters for each service may avoid or reduce these problems. A good starting point may be found in [20].

Implementation of the fast hand-over algorithm
If services that require real-time data are implemented, the performance of the network is important. One way to achieve this is by implementing so-call fast hand-over algorithm [21]. During a hand-over, the mobile node may not be able to send or receive any IPv6 packets. This period is known as the handoff latency. The hand-off latency may be unacceptable for certain types of real-time data. Implementing fast hand-over in the LBS network may solve this problem.

Implementation of Bluetooth as wireless access technology
As concluded in paragraph 2.1.3, the use of Bluetooth as wireless access technology was not feasible at the start of the ISIDOR project. But because of the facts that Bluetooth allows a better location determination, uses less power, and allows the forming of ad-hoc networks, it is still desired to implement this technology in the LBS network. This can be either supplementary or as substitute for the wireless LAN access technology.

8.2.2 Implementation related

Extension of the database
The functionality of the available location based services largely depends on the amount of information that is available for the mobile nodes. Because of the structures described in 6.6.2, it is relatively easy to add information to the database. New messages can be defined to store and retrieve this additional information. Note that for each new message the DTD (see Appendix D) must be adapted. An example of this information could be the services a mobile node offers. For instance, if a mobile node supports VoIP, it may be interesting for the user to know which other mobile nodes also support this protocol.

Better error handling
As can be derived from sections 6.4 and 6.6.4, not all possible errors are properly handled yet by the applications. Two aspects can be further improved in this area:

1. Provide feedback for the user with respect to the occurred error;
2. Define and process more error messages;

Ad 1. In case the mobile node receives an error message, the user should be notified of this event through the GUI. This is important, as the user may be the cause of the error, instead of the application. However, errors are currently only handled within the mobile node's application itself. In case the home agent or database server's application receives an error message, the situation is different as there is no direct user of these applications. This means the application should be able to handle all possible errors internally.

Ad 2. Currently, just the errors that are most likely to occur return an error message with an adequate description of the problem. The more different errors can be defined, the better they can be handled. In case of the mobile node's application, it is also important to present a sufficiently detailed error description to the user, so he can take appropriate action, if necessary.

Better connection handling
In paragraph 6.6.3, the (initial) choice for continuous connections between home agents and the database server has been argued. Drawback of this decision was the processing power and memory usage wasted for unnecessary connections. This applies even more

to the connections between the database server and the mobile nodes, as their number will generally be greater and the amount of traffic per connection smaller. It would be best if each connection were closed after a message or after a certain period of inactivity on the connection. However, this means that the database server initiates one or more connections for each notification. By doing this, the roles of the client and the server, as described in paragraph 6.5.1, are changed. This is an undesired situation, as the difference between the two roles has disappeared.

The behaviour of the entities when a connection is involuntarily lost can also be improved. This behaviour has been described in paragraph 6.6.4.1. An improvement may be a client's application that tries to re-connect whenever it detects a disconnection. This can, for instance, be detected by using time-outs for responses to sent messages. Currently, only the home agent's application is capable of reconnecting to the database server automatically.

Peer-to-peer communication for mobile nodes
As indicated in paragraph 2.2.2, peer-to-peer communication between mobile nodes is a desirable extension of the ISIDOR project. But as concluded in section 3.4, the peer-to-peer architecture is currently not possible due to hardware restrictions. However, as soon as Bluetooth is implemented as wireless technology, these restrictions will no longer exist.

With the wireless LAN technology, it is also possible to implement peer-to-peer communication for mobile nodes *within* the client-server architecture. In this architecture, a mobile node can query the database server for a certain device of his interest. Depending on the returned information on this device, the mobile node may decide to communicate directly (peer-to-peer) with this device.

Increase database efficiency
As stated in Chapter 7, the efficiency of the database influences the amount of time needed to generate response and notification messages. It should be possible to increase the overall efficiency of the database application without too much effort, especially with respect to the storage of and searching through the information in linked lists.

Better message response handling
All messages sent by on of the entities are replied to. This can either be a query response, an error or acknowledgment message. Normally, a protocol would wait for such a response before sending another message to the peer entity. However, as can be seen in paragraph 7.3.3, the generation of a response message by the mobile node takes so much time, it would drastically reduce the database performance if the latter were to wait for a response to each notification sent. Instead, a binding update response is sent to the home agent.

This raises the question whether the notification acknowledgement message has any meaning. It is therefore recommended to review the protocol with respect to the client's responses with the results of Chapter 7 in mind.

IP triggered updates for home agent's application
As stated in paragraph 7.3.3, on the average it takes about 50 ms before the database server receives a binding add, binding delete of binding update message. This is due to the fact that it is not possible to trigger the home agent's application in case of an event on the IP-level that result in a change in the home agent's binding cache. Therefore, the application has to check for changes on regular intervals. The interval may be reduced to

detect a change quicker, but this is not an efficient solution. To reduce the delay and to make the home agent's application more efficient, the MIPv6 stack may be altered, so that it is able to send a trigger in case of certain events, for instance a cell change.

Appendix A. Abbreviations and definitions

A.1 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AP	access point;
API	application program interface;
CN	correspondent node;
COA	care-of address;
CPU	central processing unit;
DB	database;
DB2MN	message from database to mobile node;
DHAAD	dynamic home agent address discovery;
DHCP	dynamic host configuration protocol;
DNS	domain name system;
DTD	document type definition;
GPRS	general packet radio services;
GSM	global system for mobile communication;
GUI	graphical user interface;
HA	home agent;
HA2DB	message from home agent to database;
HSCSD	high-speed circuit switched data;
HTML	hypertext mark-up language;
HTTP	hypertext transfer protocol;
ICMP	Internet control message protocol;
ID	identification;
IETF	Internet engineering task force;
IP	Internet protocol;
IPC	inter-process communication;
ISIDOR	information system improved by location dependant services over radio;
JRE	Java runtime environment;
JVM	Java virtual machine;
LAN	local area network;
LAS	location aware services;
LBS	location based services;
LDS	location dependant services;
MN	mobile node;
MN2DB	message from mobile node to database;
MPLS	multiprotocol label switching;
MSC	message sequence chart;
NFS	network file system;
NTP	network time protocol;

OS	operating system;
OSPF	open shortest path first;
PCMCIA	personal computer memory card international association;
PDA	personal digital assistant;
PDU	protocol data unit;
POSIX	portable operating system Unix;
QoS	quality of service;
QPE	Qt palmtop environment;
RFC	request for comment;
RIP	routing information protocol;
SGML	standard generalized mark-up language;
SMS	short message service;
TCP	transmission control protocol;
UMTS	universal mobile telecommunication system;
VoIP	voice over IP;
W3C	world wide web consortium;
WAN	wide-area network;
WINEGLASS	wireless IP network as generic platform for location aware service support;
WLAN	wireless local area network;
WWW	world wide web;
XML	extensible mark-up language;

A.2 General terms

Host	Any node that is not a router;
Interface	A node's attachment to a link;
Interface identifier	A number used to identify a node's interface on a link. The interface identifier is the remaining low-order bits in the node's IP address after the subnet prefix;
IPvx	Internet Protocol Version x;
Link	A communication facility or medium over which nodes can communicate at the link layer, such as an Ethernet (simple or bridged). A link is the layer immediately below IP;
Link-layer address	A link-layer identifier for an interface, such as IEEE 802 addresses on Ethernet links;
Node	A device that implements IP;
Packet	An IP header plus payload;
Router	A node that forwards IP packets not explicitly addressed to itself;
Security Association	A security object shared between two nodes that includes the data mutually agreed on for operation of some cryptographic algorithm (typically including a key, as defined below);
Security Policy Database (SPD)	A database of security associations selectable by rule sets (policies) that determine the packets for which each security association is to be applied;
Subnet prefix	A bit string that consists of some number of initial bits of an IP address;

A.3 Mobile IPv6 terms

Binding	The association of the home address of a mobile node with a care-of address for that mobile node, along with the remaining lifetime of that association;
Binding key	A key used for authenticating binding update messages;
Binding Security Association (BSA)	A security association established specifically for the purpose of producing and verifying authentication data passed with a binding update destination option;
Care-of address	An IP address associated with a mobile node while visiting a foreign link; the subnet prefix of this IP address is a foreign subnet prefix. Among the multiple care-of addresses that a mobile node may have at a time (e.g., with different subnet prefixes), the one registered with the mobile node's home agent is called its primary care-of address;
Correspondent node	A peer node with which a mobile node is communicating. The correspondent node may be either mobile or stationary.
Foreign link	Any link other than the mobile node's home link;
Foreign subnet prefix	Any IP subnet prefix other than the mobile node's home subnet prefix;
Home address	An IP address assigned to a mobile node within its home link;
Home agent	A router on a mobile node's home link with which the mobile node has registered its current care-of address. While the mobile node is away from home, the home agent intercepts packets on the home link destined to the mobile node's home address, encapsulates them, and tunnels them to the mobile node's registered care-of address;
Home link	The link on which a mobile node's home subnet prefix is defined. Standard IP routing mechanisms deliver packets destined for a mobile node's home address to its home link;
Home registration	The binding update sent to the mobile node's home agent to register its primary care-of address;
Home subnet prefix	The IP subnet prefix corresponding to a mobile node's home address;
Mobile node	A node that can change its point of attachment from one link to another, while still being reachable via its home address;
Movement	A change in a mobile node's point of attachment to the Internet such that it is no longer connected to the same link as it was previously. If a mobile node is not currently attached to its home link, the mobile node is said to be away from home;

Appendix B. MIPv4 implementation information

B.1 Windows implementation

National University of Singapore (NUS)
Abstract of the documentation[27]:

> *The Centre for Wireless Communications, Singapore and the National University of Singapore is pleased to announce the release of NUS WinMIP. NUS WinMIP is an implementation of the mobile IP protocol on the Windows operating system. The MN currently works on Windows 95, while the HA and the FA works on Windows NT 4.0 Workstation. NUS WinMIP is released under the GNU General Public License, the whole package consisting of complete source code, binaries and documentation can be obtained from our web site at http://mip.ee.nus.edu.sg/winmip/*

B.2 Linux implementation

Hewlett Packard
See http://www.hpl.hp.com/personal/Jean_Tourrilhes/MobileIP/index.html.

National University of Singapore
Abstract of the documentation[28]:

> *Version 3.0 beta release for our implementation of IETF mobile IPv4 is implemented in Linux kernel version 2.0.34. (February 1999). Supported features of this version are:*
>
> - *Mobile IP base protocol,*
> - *Route Optimisation,*
> - *Bi-tunnelling,*
> - *Multiple simultaneous mobility bindings,*
> - *Regional Aware Foreign Agent (RAFA),*
> - *Fast Handoff,*
> - *Multicast support for MIP (bi-tunnel scheme),*
> - *Providing mobile QoS support by interworking with RSVP,*
> - *Mobile Middleware layer.*

[27] See http://opensource.nus.edu.sg/projects/mobileip/winmip/.
[28] See http://opensource.nus.edu.sg/projects/mobileip/v3.0beta/.

Dynamics
Abstract of the documentation[29]:

> *The Dynamics solution runs entirely on user space, so no implementation specific kernel patches are needed. In addition to the source package we provide compiled versions of the software as Red Hat RPM packages and Debian deb packages. Linux kernel version 2.2.x is required with following networking options in addition to the default selection:*
>
> - *Packet socket (CONFIG_PACKET)*
> - *Kernel/User netlink socket (CONFIG_NETLINK)*
> - *Routing messages (CONFIG_RTNETLINK)*
> - *IP: Socket Filtering (CONFIG_FILTER) (for MNs; optional, but recommended)*
> - *IP: tunnelling (CONFIG_NET_IPIP)*
> - *IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) (for FAs)*
> - *IP: policy routing (CONFIG_IP_MULTIPLE_TABLES) (for FAs)*

MosquitoNet
Abstract of the documentation[30]:

> *The MosquitoNet mobile IP is an implementation of the mobile IP protocol as specified in RFC 2002. It serves two purposes. One is to use it as a building block for our mobile computing test bed, which supports transparent mobility of portable computers while moving between networks either wired or wireless (for example, between Ethernet and Metricom radio). Two, and possibly more importantly, is to explore the ways in which the mobile IP protocol can best be used. Toward this, we have added a number of enhancements that we find are desirable for mobile hosts from our daily experience. Some of the features are listed in the next section. We have ported our mobile IP implementation to Linux 2.2.x kernels. MIP 2.0.2beta is now available.*

[29] See www.cs.hut.fi/Research/Dynamics/.
[30] See http://gunpowder.stanford.edu/mip/.

Appendix C. iPAQ Linux installation

The following steps have to be followed to install Familiar 0.4 on the iPAQ H3600 series:

1. Install and configure the bootloader according to the document *iPAQ H3600 Handhelds_org bootloader install instructions* (The first part of *Familiar v0_4 installation instructions* can also be used.).
2. Load kernel 2.4.7-rmk3-np1-devfs;
3. Flash root file system bootstrap-2.4.7-rmk3-np1-devfs-hh2.jffs2. See *Familiar v0_4 installation instructions*";
4. Boot and login with user *root* and password *rootme*;
5. Install *bash*, *libreadline*, *ftp* and *less* with IPKG. Until there is a network connection, you have to use 'rz –X filename.ipk' to get these;
6. Edit '/etc/pcmcia/network.opts' for current network settings;
7. Create '/etc/resolv.conf' with DNS, if not yet present;
8. Install *ntpdate* and *mipv6bits* in order to use mipdiag;
9. Edit '/etc/pcmcia/wireless.opts' for current wireless LAN card;
10. Edit '/etc/sysconfig/network-mip6.conf' for current network settings;
11. Copy file 'isidor.start'[31] to /etc/init.d/ and type 'ln –s /etc/init.d/isidor.start /etc/rc2.d/S90inet6'. Check the permissions for this file, it should be executable for all;
12. Reboot;
13. Install *ssh*, *procps* and *wireless tools* (for *iwconfig*). If the network is up, 'ipkg install 'package.ipk'' can be used;

Additional steps to install PocketLinux GUI for Java support (see comment for just Java/kaffe support without GUI):

1. Add the following line to '/etc/ipkg.conf': 'src pocketlinux http://www.pocketlinux.com/ftp/dists/testing/ipkg-arm';
2. Run 'ipkg update' and 'ipkg install task-pockelinux';
3. Calibrate the screen with '/opt/tvt/bin/calibrate &'. If calibration still complains about fonts after install: 'export FGL_FONTS=/opt/tvt/lib/fgl/fonts' and recalibrate;
4. Start the PocketLinux GUI with '/etc/init.d/pocketlinux start &' or add this line to the 'isidor.start' file;

[31] Contents of the isidor.start file:

```
echo Loading module ipv6
/sbin/modprobe ipv6
sleep 3 # enable module to load before configuring address
echo configuring ipv6 address
/sbin/ifconfig eth0 inet6 add fec0:0:c0:a866::4b/64
echo starting mobile ipv6
/etc/init.d/mobile-ip6 start
echo isidor.start done
```

Appendix D. XML protocol DTD

```
<!ELEMENT protocol (messageHA2DB | messageMN2DB | messageDB2MN)>

<!ELEMENT messageHA2DB (binding_update | binding_add | binding_delete)>
<!ELEMENT binding_add ((home_agent, coa, lifetime) | ackn | err)>
    <!ATTLIST binding_add home_address CDATA #REQUIRED>
<!ELEMENT binding_update ((home_agent?, coa?, lifetime?) | ackn | err)>
    <!ATTLIST binding_update home_address CDATA #REQUIRED>
<!ELEMENT binding_delete (EMPTY | (ackn | err)?)>
    <!ATTLIST binding_delete home_address CDATA #REQUIRED>

<!ELEMENT messageMN2DB (user_register | user_change | user_delete | subscribe |
     unsubscribe | query_cells | query_home_address | query_name | query_user)>
<!ELEMENT user_register ((name?, user?) | ackn | err)>
    <!ATTLIST user_register home_address CDATA #REQUIRED>
<!ELEMENT user_change ((name?, user?) | ackn | err)>
    <!ATTLIST user_change home_address CDATA #REQUIRED>
<!ELEMENT user_delete (EMPTY | (ackn | err)?)>
    <!ATTLIST user_delete home_address CDATA #REQUIRED>
<!ELEMENT subscribe (cell_id+ | ackn | err)>
    <!ATTLIST subscribe home_address CDATA #REQUIRED>
<!ELEMENT unsubscribe (cell_id+ | ackn | err)>
    <!ATTLIST unsubscribe home_address CDATA #REQUIRED>
<!ELEMENT query_cells (EMPTY | cell_id* | err)>
    <!ATTLIST query_cells cell CDATA "*">
<!ELEMENT query_home_address (EMPTY | (name?, user?, cell_id?) | err)>
    <!ATTLIST query_home_address home_address CDATA #REQUIRED>
<!ELEMENT query_name (EMPTY | (home_address?, user?, cell_id?) | err)>
    <!ATTLIST query_name name CDATA #REQUIRED>
<!ELEMENT query_user (EMPTY | (home_address?, name?, cell_id?) | err)>
    <!ATTLIST query_user user CDATA #REQUIRED>

<!ELEMENT messageDB2MN (user_notify)>
<!ELEMENT user_notify ((cell_id?, name?, user?) | ackn | err)>
    <!ATTLIST notify home_address CDATA #REQUIRED>

<!ELEMENT lifetime (#PCDATA)>
<!ELEMENT coa (#PCDATA)>
<!ELEMENT home_address (#PCDATA)>
<!ELEMENT home_agent (#PCDATA)>

<!ELEMENT cell_id (#PCDATA | EMPTY)>
<!ELEMENT cell (home_address+)>

<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT user (#PCDATA)>

<!ELEMENT ackn EMPTY>
<!ELEMENT err (#PCDATA)>
```

Appendix E. Defined XML messages

E.1 MessageHA2DB

```
<protocol>
   <messageHA2DB>
         <binding_add home_address="...">
               <home_agent>...</home_agent>
               <coa>...</coa>                          } Required
               <lifetime>...</lifetime>
         </binding_add>
   </messageHA2DB>
</protocol>

<protocol>
   <messageHA2DB>
         <binding_add home_address="...">
               <ackn/>
         </binding_add>
   </messageHA2DB>
</protocol>

<protocol>
   <messageHA2DB>
         <binding_add home_address="...">
               <err>...</err>
         </binding_add>
   </messageHA2DB>
</protocol>


<protocol>
   <messageHA2DB>
         <binding_update home_address="...">
               <home_agent>...</home_agent>
               <coa>...</coa>                          } Optional
               <lifetime>...</lifetime>
         </binding_update>
   </messageHA2DB>
</protocol>
```

```
<protocol>
   <messageHA2DB>
         <binding_update home_address="...">
               <ackn/>
         </binding_update>
   </messageHA2DB>
</protocol>

<protocol>
   <messageHA2DB>
         <binding_update home_address="...">
               <err>...</err>
         </binding_update>
   </messageHA2DB>
</protocol>


<protocol>
   <messageHA2DB>
         <binding_delete home_address="..."/>
   </messageHA2DB>
</protocol>

<protocol>
   <messageHA2DB>
         <binding_delete home_address="...">
               <ackn/>
         </binding_delete>
   </messageHA2DB>
</protocol>

<protocol>
   <messageHA2DB>
         <binding_delete home_address="...">
               <err>...</err>
         </binding_delete>
   </messageHA2DB>
</protocol>
```

E.2 MessageMN2DB

Register:

```
<protocol>
   <messageMN2DB>
         <user_register home_address="...">
               <name>...</name>                          }
               <user>...</user>                          } Optional
               <other_user_info>...</other_user_info>    }
         </user_register>
   </messageMN2DB>
```

```
</protocol>

<protocol>
   <messageMN2DB>
         <user_register home_address="...">
               <ackn/>
         </user_register>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <user_register home_address="...">
               <err>...</err>
         </user_register>
   </messageMN2DB>
</protocol>


<protocol>
   <messageMN2DB>
         <user_change home_address="...">
               <name>...</name>                          }
               <user>...</user>                          } Optional
               <other_user_info>...</other_user_info>    }
         </user_change>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <user_change home_address="...">
               <ackn/>
         </user_change>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <user_change home_address="...">
               <err>...</err>
         </user_change>
   </messageMN2DB>
</protocol>


<protocol>
   <messageMN2DB>
         <user_delete home_address="..."/>
   </messageMN2DB>
</protocol>
```

```
<protocol>
   <messageMN2DB>
         <user_delete home_address="...">
               <ackn/>
         </user_delete>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <user_delete home_address="...">
               <err>...</err>
         </user_delete>
   </messageMN2DB>
</protocol>
```

Subscribe:

```
<protocol>
   <messageMN2DB>
         <subscribe home_address="...">
               <cell_id>*</cell_id>          <!—default:*. Or: -->
               <cell_id>...</cell_id>
                     ⋮
               <cell_id>...</cell_id>
         </subscribe>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <subscribe home_address="...">
               <ackn/>
         </subscribe>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <subscribe home_address="...">
               <err>...</err>
         </subscribe>
   </messageMN2DB>
</protocol>
```

```
<protocol>
   <messageMN2DB>
         <unsubscribe home_address="...">
               <cell_id>*</cell_id>          <!-- default: *. Or: -->
               <cell_id>...</cell_id>
                     ⋮
               <cell_id>...</cell_id>
         </unsubscribe>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <unsubscribe home_address="...">
               <ackn/>
         </unsubscribe>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <unsubscribe home_address="...">
               <err>...</err>
         </unsubscribe>
   </messageMN2DB>
</protocol>
```

Query:

```
<protocol>
   <messageMN2DB>
         <query_cells cell_id="*"/>          <!-- default :* -->
   </messageMN2DB>
</protocol>
```

```
<protocol>
   <messageMN2DB>
         <query_cells cell_id="*">
               <cell id="...">
                     <home_address>...</home_address>   <!-- list all MN in cell -->
                     <home_address>...</home_address>
                                 ⋮
                     <home_address>...</home_address>
               </cell>
                  ⋮
               <cell id="...">
                     <home_address>...</home_address>
                     <home_address>...</home_address>
                                 ⋮
                     <home_address>...</home_address>
               </cell>
         </query_cells>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <query_cells cell_id="*">
               <err>...</err>
         </query_cells>
   </messageMN2DB>
</protocol>


<protocol>
   <messageMN2DB>
         <query_home_address home_address="..."/>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <query_home_address home_address="...">
               <name>...</name>
               <user>...</user>
               <cell_id>...</cell_id>
               <other_info>...</other_info>
         </query_home_address>
   </messageMN2DB>
</protocol>
```

Optional

```
<protocol>
   <messageMN2DB>
         <query_home_address home_address="...">
               <err>...</err>
         </query_home_address>
   </messageMN2DB>
</protocol>


<protocol>
   <messageMN2DB>
         <query_name name="..."/>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <query_name name="...">
               <home_address>...</home_address>
               <user>...</user>
               <cell>...</cell_id>
               <other_info>...</other_info>
         </query_name>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <query_name name="...">
               <err>...</err>
         </query_name>
   </messageMN2DB>
</protocol>


<protocol>
   <messageMN2DB>
         <query_user user="..."/>
   </messageMN2DB>
</protocol>

<protocol>
   <messageMN2DB>
         <query_user user="...">
               <home_address>...</home_address>
               <name>...</name>
               <cell>...</cell_id>
               <other_info>...</other_info>
         </query_user>
   </messageMN2DB>
</protocol>
```

Optional

Optional

```
<protocol>
   <messageMN2DB>
         <query_user user="...">
               <err>...</err>
         </query_user>
   </messageMN2DB>
</protocol>
```

E.3 MessageDB2MN

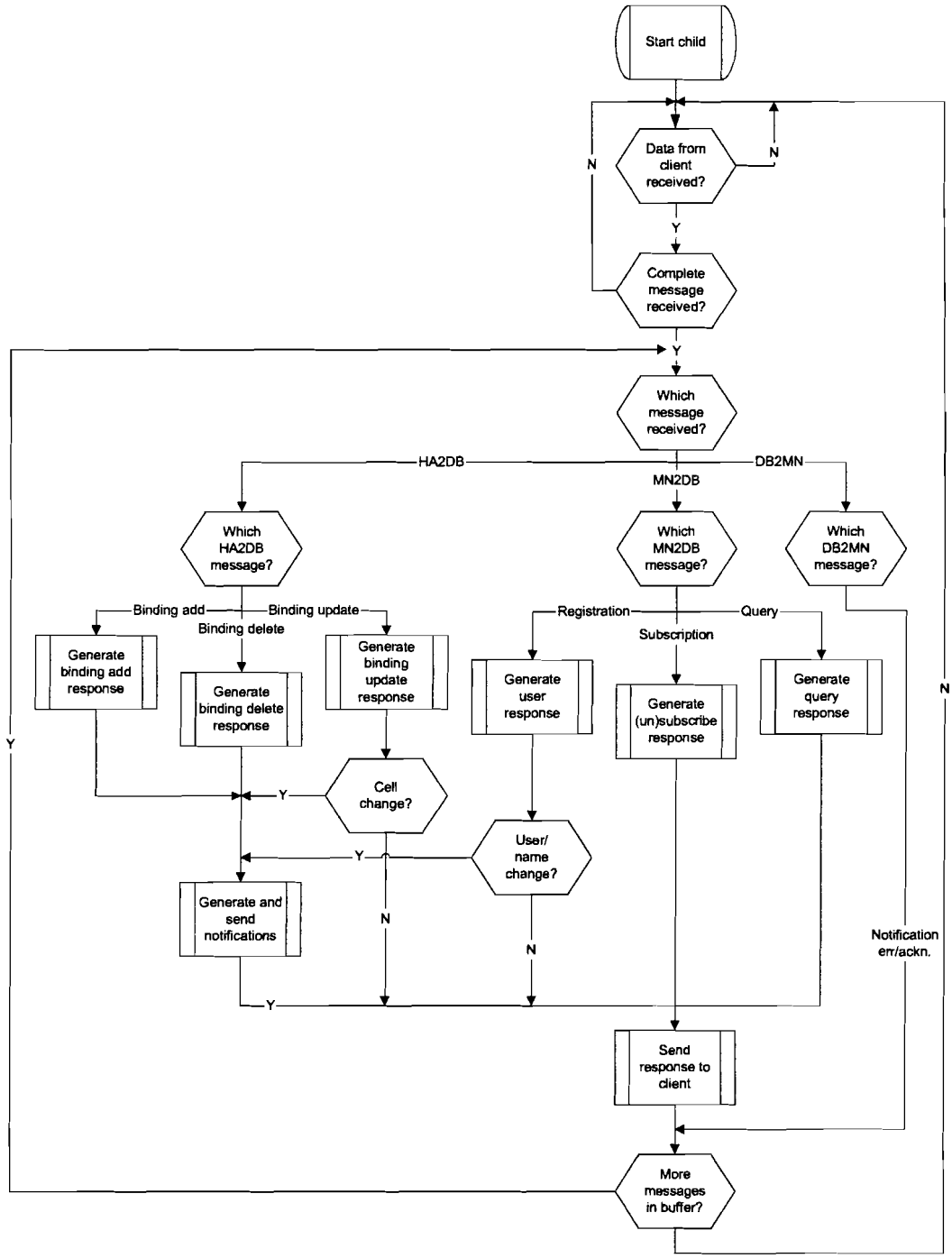
```
<protocol>
   <messageDB2MN>
         <user_notify home_address="...">
               <cell_id>...</cell_id>
               <name>...</name>
               <user>...</user>
               <other_info>...</other_info>
         </user_notify>
   </messageDB2MN>
</protocol>
```

Optional

```
<protocol>
   <messageDB2MN>
         <user_notify home_address="...">
               <ackn/>
         </user_notify>
   </messageDB2MN>
</protocol>

<protocol>
   <messageDB2MN>
         <user_notify home_address="...">
               <err>...</err>
         </user_notify>
   </messageDB2MN>
</protocol>
```

Appendix F. Flowchart of child function



Appendix G. References

[1] *The Book of Visions 2000 - Visions of the Wireless World*, Version 1.0, Wireless World Research Forum, November 2000.

[2] Johnson, D.B. and C. Perkins.
Mobility support in IPv6, www.ietf.org/internet-drafts/draft-ietf-mobileip-ipv6-16.txt, IETF Mobile IP Working Group Internet-Draft, March 2002.

[3] Moy, J.
The OSPF Specification Version 2, RFC 1247, www.ietf.org/rfc/rfc1247.txt, Proteon Inc, January 1991.

[4] Hedrick, C.
Routing Information Protocol, RFC 1058, www.ietf.org/rfc/rfc1058.txt, Rutgers University, June 1988.

[5] Hinden, R. and S. Deering.
IP Version 6 Addressing Architecture, RFC 2373, www.ietf.org/rfc/rfc2373.txt, IETF Network Working Group, July 1998.

[6] Tanenbaum, A.S.
Computer Netwerken, tweede herziene uitgave, Schoonhoven: Academic Service, March 2000.

[7] Deering, S. and R. Hinden.
Internet Protocol, Version 6 (IPv6) Specification, RFC 2460, www.ietf.org/rfc/rfc2460.txt, IETF Network Working Group, December 1998.

[8] Perkins, C.
IP mobility support, RFC 2002, www.ietf.org/rfc/rfc2002.txt, IETF Network Working Group, October 1996.

[9] Thomson, S. and T. Narten.
IPv6 Stateless Address Autoconfiguration, RFC 2462, www.ietf.org/rfc/rfc2462.txt, IETF Network Working Group, December 1998.

[10] Narten, T. e.a.
Neighbor Discovery for IP Version 6 (IPv6), RFC 2461, www.ietf.org/rfc/rfc2461.txt, IETF Network Working Group, December 1998.

[11] Conta, A. and S. Deering.
Generic Packet Tunneling in IPv6 - Specification, RFC 2473, www.ietf.org/rfc/rfc2473.txt, IETF Network Working Group, December 1998.

[12] Conta, A. and S. Deering.
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) - Specification, RFC 2463, www.ietf.org/rfc/rfc2463.txt, IETF Network Working Group, December 1998.

[13] Johnson, D. and S. Deering.
Reserved IPv6 Subnet Anycast Addresses, RFC 2526, www.ietf.org/rfc/rfc2526.txt, IETF Network Working Group, March 1999.

[14] Bradner, S.
Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, www.ietf.org/rfc/rfc2119.txt, IETF Network Working Group, March 1997.

[15] Bray, T, Paoli, J, Sperberg-McQueen, C.M., Maler, C.
Extensible Markup Language (XML) 1.0 (Second Edition), www.w3.org/TR/REC-xml, W3C Recommendation, October 2000.

[16] Arbortext, Inc.
SGML: Getting Started. A Guide to SGML (Standard Generalized Markup Language) and Its Role in Information Management, Arbortext SGML White Paper, http://www.arbortext.com/data/getting_started_with_SGML/getting_started_with_sgml.html, 1995.

[17] Stevens, W. Richard,
TCP/IP Illustrated, Volume 1 – The Protocols, Addison-Wesley, Reading, Massachusetts, September 1998.

[18] Fang, X, Gu, Y, Tang, Y.
Distributed Object Systems and Technology. Socket Programming Overview, University of New South Wales,
www.cse.unsw.edu.au/~cs4111/00s2/SocketProgram.html.

[19] Wagner, T. and D. Towsley.
Getting Started With POSIX Threads, Department of Computer Science, University of Massachusetts at Amherst, July 1995.

[20] K. Zhigang e.a.
Mobile IPv6 and some issues for QoS, INET 2001 conference proceeding, Stockholm, June 2001.

[21] G. Dommety e.a.
Fast Handovers for Mobile IPv6, IETF Internet-draft, www.ietf.org/internet-drafts/draft-ietf-mobileip-fast-mipv6-04.txt, March 2002.