Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Samuel Korpi

Setting up an Ad Hoc Testbed Network

Student Thesis: SA-2005-32
April 2005 to July 2005

Supervisor: Károly Farkas
Professor: Bernhard Plattner

Abstract

In this project, the goal was to build a testbed mobile ad hoc network to be used as an experimental platform for the SIRAMON framework [1]. The test application is a multi-player game running on this testbed network. The existing testbed network built by Rolf Grüninger during his master's thesis [2] served as a basis for this project. The network consisted of two desktop PCs and two laptops, all communicating via WLAN. During this project, the network was extended with two Compaq iPAQ PDAs. A lot of work went into finding suitable software for the iPAQs to establish a working testbed network. Also, porting the SIRAMON framework to the new architecture was a major task.

Contents

List of Figures

List of Tables

1 Introduction

In the modern world mobility has become increasingly important. PDAs[1] and especially mobile phones follow us almost everywhere in our nowaday lives.

Mobility poses a great challenge in the field of communications and networking. Traditional networks which rely heavily on a fixed infrastructure simply do not scale well to mobile networks. So, a new methodology was created. Ad Hoc networking deals with mobile networks where no fixed infrastructure can be assumed.

SIRAMON[2] [1] is a framework extending the idea of ad hoc networking. It serves as a middle-layer between networking and applications by allowing dynamic creation and management of services. Interested readers are referred to *Service Provisioning in Mobile Ad Hoc Networks* [2], a Master's Thesis written by Rolf Grüninger in 2004.

Mobility also affects the hardware. In traditional networks, client devices are fixed and thus not generally restricted when it comes to the power consumption or the size of devices, among others. Mobile devices, in contradiction, have to meet very strict requirements. They should be easy to carry around and have a long battery life time etc. These requirements heavily affect on the size of the devices. This, in turn, limits the abilities of the devices. The size is one of the most limiting factors when it comes to designing a suitable user interface. The limitations have to be taken into account also when developing applications for these devices.

Testing is an essential part of every design process, thus also the SIRAMON framework needs to be tested. Some amount of testing can be done locally, with simulations etc. But at some point a real test environment becomes invaluable, and this is the main motivation for the project described in this document. There was already an existing testbed network in place, but it only included two PCs and two laptops, which is really not sufficient to emulate a real ad hoc network environment. Introducing a couple of small mobile devices (i.e. PDAs) would have a strong positive effect on the usability of the testbed in real-world simulations.

[1] Personal Digital Assistant
[2] Service provIsioning fRAMework for self-Organized Networks

2 Design/Architecture

This section describes the project at a greater length. I also discuss some of the problems I was faced with during the project.

2.1 The task and the existing environment

The task was to create an ad hoc testbed network by extending an existing testbed with a couple of PDA devices.

The existing network was built by Rolf Grüninger during his master's thesis. It consists of two laptops and two PCs all equipped with WLAN cards for communication. Figure 1 shows a greatly simplified picture of the testbed. In the picture, every device (i.e. node) is connected to all the others with a dashed line. This represents a situation where all the devices are close enough to each other so that direct communication is possible.

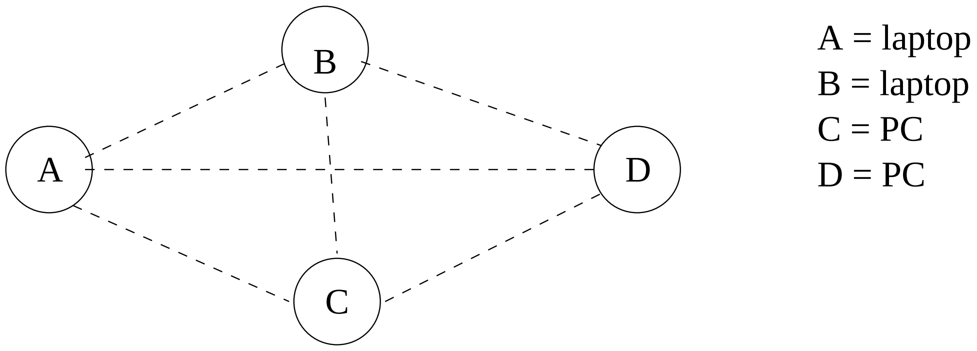


Figure 1: The existing testbed network, circles representing the devices.

Each of the testbed devices is running the SIRAMON, or ROSAMON[3], framework. Routing is left to a routing protocol called *Mobile Mesh* [3]. The framework takes care of dynamic changes in the network (i.e. a device entering/leaving the network) in conjunction with the routing protocol. Routing is needed to enable multi-hop connections[4]. An example might be as follows: Let's take the network as shown in figure 1. Now let's assume that device A is too far from device D to communicate with it directly. In this case either device B or device C could forward messages from A to D and vice versa.

2.2 Idea of what was wanted

The existing network was working just fine, but it was too simple to effectively emulate any real environment where SIRAMON might be used. Adding PDAs to the network would better test the abilities of SIRAMON. A simple diagram showing the structure of the extended testbed network is presented in figure 2. As can be seen from the picture, the network is not that much different with the new devices included. Notice, however, that between the PDAs the communication line is marked differently. This is intentional, marking the possibility of Bluetooth communication between the PDAs.

Linux was selected as the operating system on the PDAs. So it is the platform on which SIRAMON should be working. Linux was already used in the old testbed, and SIRAMON was tested to be working in that environment. The PDAs, however, have limitations which affect also the Linux environment. In this project, Compaq iPAQs (model H3870) were used. These devices have integrated Bluetooth, but WLAN required an extension pack[5]. As the iPAQs came pre-installed with Windows, the old system had to be replaced. See section 2.3.1 for more information.

[3]ROSAMON was the original name for the service provisioning framework. The acronym comes from *ROlf's Service frAmework for Mobile ad hOc Networks*

[4]If two devices cannot communicate directly, a connection can be established by routing messages through a third party which is able to communicate directly with both. This is called a multi-hop connection.

[5]The original idea was to use a bit newer HP iPAQs which would have included both integrated WLAN and Bluetooth. At the time, however, there were no Linux distributions available for those devices.

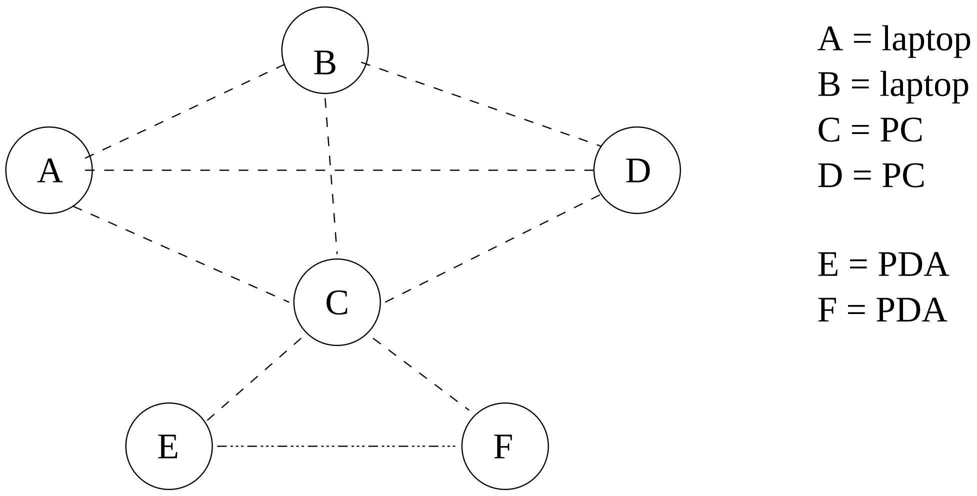


Figure 2: The extended testbed network, circles representing the devices.

2.3 Project steps

The following list covering the main steps of the project is a modified version of the task list from the task description I wrote in the beginning of the project (see appendix A.2).

1. Installing the basic environment (Linux, etc.) on the iPAQs. (2.3.1)
2. Getting the PDAs to work as a part of the network. (2.3.2)
3. Porting the Java code of the SIRAMON framework to the iPAQs. (2.3.3)
4. Testing. (2.3.4)

In the following, I will shortly cover each of the process steps in turn.

2.3.1 Installing the basic environment (Linux, etc.) on the iPAQs

The iPAQs used in this project were pre-installed with a Windows CE (PocketPC) operating system. This was replaced with a Familiar Linux distribution [4]. The installation procedure is covered in appendix C.

After installation, some general configurations had to be made, in order to enable communications between the iPAQs and other devices. These are listed in appendix D, along with information about the user accounts etc.

2.3.2 Getting the PDAs to work as a part of the network

The WLAN and Bluetooth capabilities of the iPAQs where recognized on-the-fly by the operating system (i.e. no external device drivers were needed). The configurations to get the network working, however, were another matter. The necessary settings can be found in appendix D. Furthermore, I wrote a couple of scripts to help with changing the configurations between ad hoc and infrastructure use. Connection to the Internet (i.e. using the infrastructure mode) is required when the Linux environment on the iPAQs needs updating or some additional software. Of course, all the necessary packages could be transmitted via serial/USB connection, but a direct connection to the Internet makes things easier. The scripts are listed in appendix E.1.

By default, no routing was provided by the operating system. So, in order to make multi-hop connections work, one had to be installed. As mentioned previously, the original testbed used a routing protocol called *Mobile Mesh*. However, this routing protocol is already five years old and as such there is no use in trying to get it working on the iPAQs. Had there been precompiled binaries for the iPAQs, I would have considered using them. But as it seemed necessary to cross-compile the protocol, why not try to find a newer and perhaps even better alternative to Mobile Mesh.

AODV, Ad hoc On-Demand Distance Vector [5], seems the best suited routing protocol currently available for ad hoc networks. For this project, though, the most important thing was to get a testbed running, so the efficiency and other characteristics of the protocol were of lesser importance. There are many AODV implementations available to the public. I concentrated my efforts on two of those, namely *Kernel AODV* from NIST [6] and

AODV-UU from Uppsala University [7]. They were recommended on a kind of an “AODV portal” [8] containing links to various publications, implementations etc. I found the site quite useful source of information concerning AODV.

Of the two possible AODV implementations I finally selected AODV-UU, mainly because it was the one I could get working on the iPAQs. The instructions on how to cross-compile the sources for ARM (iPAQs) are included in section C.2.

Bluetooth communications were only marginally tested. Connections between devices could be established using Bluetooth, but only manually. The reason why I didn’t further test the Bluetooth capabilities was the lack of time and the fact that without a gateway between the WLAN and Bluetooth networks the result would have been two separate networks with no cross-communication possibilities. The subject is further discussed in section 4.2.1.

2.3.3 Porting the Java code of the SIRAMON framework to the iPAQs

The porting of the SIRAMON software to the iPAQs was most likely the biggest, or at least the most time-consuming task in the whole project. The original version was programmed using SWT[6] [9] as the GUI[7] toolkit. SWT is not part of the official Sun Java environment, but an extension provided as a part of the Eclipse Project [10]. It is quite wide-spread in normal PC/laptop environments, but for Linux on iPAQ it seemed quite an impossible task to find working versions of the needed libraries. So, finally I decided to rewrite the GUI using AWT[8] and Swing components, which were supported also on the iPAQs. More information concerning the Java Runtime environment I used on the iPAQs can be found in section D.2.

2.3.4 Testing

The final results – the testbed network in the state I managed to get it working during the project – were tested using an example application running on the SIRAMON framework. It is a multi-player game programmed originally by Rolf Grüninger. There are two versions of the game, a peer-to-peer and a client-server version. However, I only had the working code of the peer-to-peer version of the game to port on the iPAQs[9]. Figure 3 shows a couple of pictures of the running game.

Due to some problems with the Java key mappings on the iPAQs, I had to insert control buttons directly on the game window. This is not a very efficient solution, as the player has to click the buttons with a pen – not a very fast way of controlling the game which means the players using laptops/PCs have a great advantage. However, the main function of the game, to show that the testbed is working, is fulfilled.

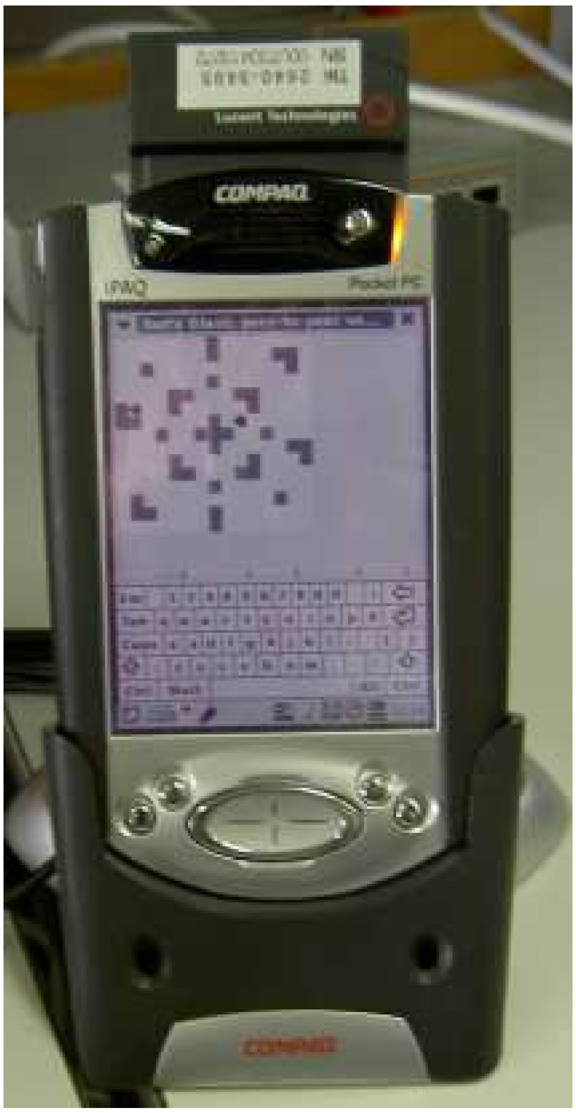
2.4 Difficulties I was faced with during the project

Probably the biggest difficulty was trying to find suitable software to do the things I wanted. Especially binaries were difficult to find, and in many cases cross-compilation was the only feasible solution. The problem with this approach was that I was not experienced in cross-compilation, so a lot of time went to just figuring out what was needed for the cross-compilation process etc.

Lack of information was also at times annoying, and mostly related to the aforementioned difficulties of finding precompiled software. Part of the problem, of course, was that a lot of the software used in this project is under development. This often causes inconsistencies in the documentation, which might not be as frequently updated as the software itself. For some older software, the documentation is sometimes outdated because the environment it was written to has changed even if the software itself has stayed the same.

Also related to previous problems is the lack of standard implementations of Java, for example. In a PC environment you can always download Sun’s Java environment, no matter if you are using Windows or Linux. But for Linux on iPAQs there were no Java implementation provided by Sun Microsystems. So I had to rely on Open Source alternatives, which have the disadvantage of still being under development and as such not completely Sun-Java compatible. This made the porting process that much more difficult.

[6]The Standard Widget Toolkit
[7]Graphical User Interface
[8]Abstract Windowing Toolkit
[9]Partly parallel to my project the code of SIRAMON was reorganized to be more modular. As a result of this reorganization the code became incompatible with the old version and as the client-server version of the game was not updated, it wouldn’t work on the new framework.



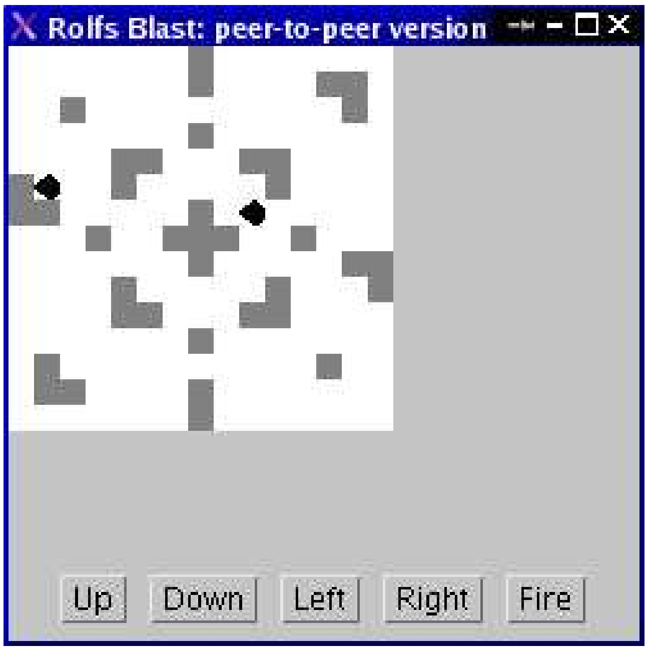


Figure 3: The multi-player game, Rolf’s Blast, running on an iPAQ (left) and on a laptop (right).

3 Results

This section lists the results achieved during the project.

3.1 Environment

The selected operating system, Familiar Linux v0.8.2 [4], was successfully installed and configured. A Java Virtual Machine, *JamVM* [11], was also installed, as a platform for the SIRAMON software.

3.2 Networking

Networking over WLAN was configured and tested using both ad hoc and infrastructure modes. The testbed environment does not use the infrastructure mode, but it is needed when connecting to an outside network (i.e. the Internet), in order to update the Linux environment on the iPAQs, for example. The network settings I used can be found in appendix D.

Bluetooth networking was only tested to the extent of allowing further research without having to deal with problems in the basic configuration. I checked that the Bluetooth was correctly recognized by the operating system, and that connections to another Bluetooth devices could be established manually.

3.2.1 Routing

AODV [5] was selected as the routing protocol and AODV-UU [7] as the implementation for that protocol. AODV-UU was successfully cross-compiled for the iPAQs (C.2) and tested with simple pings[10] and also with the SIRAMON test application. True multi-hop functionality still needs to be tested in an environment where some of the devices (i.e. nodes) in the network are not directly reachable from the whole network.

3.3 Porting of the SIRAMON code

The minimum functionality of the SIRAMON software was ported to the iPAQs. The GUI was rewritten using AWT/Swing instead of SWT. A test application, a multi-player game called Rolf's Blast[11], was also ported and tested to be working. The XML[12] editing capabilities, meant to be used to modify/create the service descriptions used within the SIRAMON framework [2], were left out. The problem was that the SWT widgets used to accomplish this functionality did not have good counterparts in AWT/Swing. A simple workaround to this would be to just have a simple text-editing feature to edit the XML code directly. This, of course, requires some knowledge of XML from the user, but XML is designed to be human-readable and should not require too much effort on the part of the user.

[10]The simplest way of testing a network functionality is to send ping packets to the devices in the network using the IP addresses of the devices. Most operating systems include a simple program (usually called *ping*) for sending these packets.

[11]The peer-to-peer version

[12]eXtensible Markup Language

4 Summary

In this section, I provide a short summary of the whole project, starting with conclusions (4.1) and ending with a list of future work (4.2).

4.1 Conclusions

This project was definitely an interesting one, and a challenge. I learned a lot over the course of this project, although it is apparent to me that there still is a lot to learn.

Maybe the biggest obstacle on my way was the time. I admit that I could have done a lot better job in planning the time I required. On the other hand, in this kind of a project which includes a lot of “research stuff” (i.e. usage of software, protocols, etc. that still are under heavy research), you never know how much time you will need to complete the tasks you set out to do, or even whether your goals are achievable or not.

In this project starting to build everything on top of a Linux environment had its’ pros and cons. Linux definitely gives more freedom to the developer than Windows. On the other hand, what I discovered is that it is quite difficult to find precompiled software and libraries for the Linux environment on iPAQs. Whether the situation is any better when Windows is used, I do not know. At least the SWT libraries[13] are available for the Windows environment.

4.2 Future work

4.2.1 Gateway functionality between Bluetooth and WLAN networks

Actually, programming the gateway functionality into the testbed was originally a part of this project. Other parts of the project, however, took more time than anticipated, so this feature had to be dropped. Still, this kind of a gateway would make the network a lot more closer to a real environment.

The idea of a gateway is to pass information between different types of networks[14]. A gateway device must be able to communicate with both networks, i.e. it must have interfaces for both networks and it has to be in communication range from both networks. The gateway device then has to be able to pass information from one network to the other, modifying the network packets if necessary.

In this project, the networking standards we are interested in are WLAN and Bluetooth. Most of the newer mobile phones, for example, are Bluetooth-enabled by default. But they rarely have WLAN interfaces, so a gateway is needed to make it possible for these devices to communicate with a WLAN network.

Even though I was not able to include the gateway functionality into the iPAQs within this project, I did some research in the area. The problem in bridging Bluetooth and WLAN networks lays in some basic differences in their architecture. Bluetooth does not use IP packets and is thus, by default, incompatible with WLAN or any other TCP/IP[15] network and also with the current SIRAMON implementation.

However, there is a feature in Bluetooth called PAN[16], which offers TCP/IP support. A HowTo document introducing different PAN scenarios is available online [12]. One gateway scenario is also covered, but it bases on a fixed NAP[17] (i.e. gateway) and is thus not readily usable in our case. In a truly mobile environment each device capable of acting as a gateway should automatically do so when necessary.

I found a general “Bluetooth portal” containing links to information about *Bluetooth and Linux* online [13]. Might be of use in further research.

4.2.2 Improvements to the SIRAMON code

The SIRAMON code is far from complete, when we look at the functionality and usability on the iPAQs. I had to make quite a lot of changes in the code (see appendix E.2) to get it working on the iPAQs. And still I only managed to port the very basic functionality – enough to get the test application running. Even some pretty important features, for example XML editing capability, were left out. Furthermore, the code is partly pretty

[13]The original SIRAMON software was programmed using SWT to build the GUI.
[14]Possibly even incompatible networks
[15]TCP = Transmission Control Protocol, IP = Internet Protocol
[16]Personal Area Networking
[17]Network Access Point

difficult to read and to follow – cleaning the code and unifying the used commenting styles is an important task to keep the code easy to manage.

It might even be reasonable to try another port of the original code, if the required SWT JNI libraries[18] for the iPAQs can be found. For Windows environment they are readily available, not so for Linux. It may be possible to cross-compile the libraries [14]. The sources are included in the Eclipse SDK or available for download via public CVS.

[18]The original version of SIRAMON was programmed using SWT as the GUI toolkit. The toolkit uses JNI (Java Native Interface) to interact with the native widgets in the operating system. The SWT JNI libraries need to be compiled to match the environment they are used in. Pre-compiled libraries are available for certain environments.

A Appendix: Original task description

A.1 Original project announcement

Setting up an Ad Hoc Testbed Network

Ad-hoc networking is an emerging and interesting communication paradigm nowadays. The infrastructureless, self-organized behaviour of mobile devices forming a communication network makes this research field challenging.

In this project our goal is to setup a wireless ad-hoc testbed network from mobile devices integrating different wireless technologies (e.g., WLAN, bluetooth) into it. These devices can vary from traditional computers (laptops) to small computers or handheld devices (palmtops, cellular phones, etc.). In the testbed the involved devices should be able to communicate with each other even in a multihop manner using mobile ad-hoc routing protocols. This testbed intends to form the experimental platform of SIRAMON, a service provisioning framework for self-organized networks.

Kind of Work:	Setup, implementation, test
Requirements:	Linux configuration experiments
Contact Person:	Károly Farkas, ETZ G60.1, +41 1 63 25447
Tutors:	Károly Farkas
Professor:	Prof. Bernhard Plattner
Keywords:	Ad-hoc networks, wireless, testbed

A.2 Revised task description

Introduction

In this project, the goal is to build a testbed mobile ad hoc network to be used as an experimental platform for the SIRAMON framework [1]. The test application will be a multi-player game running on this testbed network.

The existing testbed network consists of two desktop computers (PCs) and two laptops. I will refer to these computers and any further devices in the network as the nodes of the network. The existing network is a fully functional ad hoc network, in the sense that the nodes are connected via WLAN and any of the nodes can be removed or added to the network preserving the functionality of the whole network.

The problem with the current testbed network is that it isn't completely mobile. You cannot just put a laptop in your pocket. So the idea is to extend the network with some PDAs (Personal Digital Assistants). The PDA device we will be using is HP iPAQ [15]. That particular PDA contains both WLAN and Bluetooth interfaces. To make it easy to add also plain Bluetooth devices to the network (e.g. Bluetooth enabled mobile phones), these PDAs with both communication interfaces should act as gateways between Bluetooth and WLAN devices, where appropriate.

The nodes in the network will use a Linux Operating System and on top of that the SIRAMON framework and the test application will run. The necessary framework/application code (programmed in Java) will be ported to the Linux platform on the iPAQs.

Tasks and Working Plan

1. Planning and collecting information. Getting to know the existing network. Possibly adding one or two laptops to the network (just copying the configurations from the existing nodes). Gathering necessary information from various sources (mainly the Internet).

2. Getting to know the PDAs which will be appended to the network. Information from the Internet (manuals, specifications, ...). Especially how the WLAN/Bluetooth interfaces work. Also getting information about the Linux platform which will be installed on the PDAs.

3. Installing the basic platform (Linux, etc.) on the PDAs. Basic configurations.

4. Getting the PDAs to work as a part of the network (only with WLAN at this point).

5. Getting the PDAs to communicate with each other via Bluetooth.

6. Implementing the Gateway functionality.

7. Porting the Java code of the SIRAMON framework to the iPAQs.
8. Building the testbed network.
9. Testing and evaluation.
10. Writing the thesis report.
11. Finalizing the project and a final presentation.

The original working plan (timetable) can be found in section B.1.

B Appendix: Timetable

B.1 Originally planned timetable

The following table (table 1) shows the working plan of the project. The week number is the calendar week, with the project week displayed in parentheses. The Date column shows the actual working days (weekends/holidays excluded). The task number refers to the task listing in the revised task description (section A.1).

Week	*Date*	*Task*	*Comments*
16 (1)	19.04.-22.04.2005	1,10	General planning.
17 (2)	25.04.-29.05.2005	1,10	
18 (3)	02.05.-04.05.2005	2,10	
19 (4)	09.05.-13.05.2005	3,10	Possible changes to the schedule should be done this week, at the latest.
20 (5)	17.05.-20.05.2005	4,9,10	
21 (6)	23.05.-27.05.2005	5,9,10	
22 (7)	30.05.-03.06.2005	6,9,10	Intermediate presentation.
23 (8)	06.06.-10.06.2005	6,9,10	
24 (9)	13.06.-17.06.2005	7,9,10	The porting of the code should begin even earlier, if possible.
25 (10)	20.06.-24.06.2005	7,9,10	
26 (11)	27.06.-01.07.2005	7,9,10	
27 (12)	04.07.-10.07.2005	8,9,10	
28 (13)	11.07.-15.07.2005	11	Final presentation.

Table 1: Originally planned timetable for the project.

B.2 Realized timetable

The following table (table 2) shows the actual progress I did in the project on a time-scale. As in previous subsection, the task number refers to the task listing in the revised task description (section A.1).

Week	*Date*	*Task*	*Comments*
16 (1)	19.04.-22.04.2005	1	General planning.
17 (2)	25.04.-29.05.2005	1,2	
18 (3)	02.05.-04.05.2005	1,2	
19 (4)	09.05.-13.05.2005	2,3	
20 (5)	17.05.-20.05.2005	3,4,9	The extension to the project by two weeks was finalized.
21 (6)	23.05.-27.05.2005	4,5,9	Only basic Bluetooth communications.
22 (7)	30.05.-03.06.2005	5,9	Only basic Bluetooth communications.
23 (8)	06.06.-10.06.2005	9	
24 (9)	13.06.-17.06.2005	7,9	No real porting yet, just getting to know the code.
25 (10)	20.06.-24.06.2005	7,9	
26 (11)	27.06.-01.07.2005	7,9	
27 (12)	04.07.-10.07.2005	7,8,9	
28 (13)	11.07.-15.07.2005	7,8,9	
29 (14)	18.07.-22.07.2005	7,8,9,10	
30 (15)	25.07.-29.07.2005	7,8,9,10,11	Final presentation on 28th.

Table 2: Realized timetable for the project.

The report was mostly written after the final presentation, though I did collect notes throughout the project. I finally finished the report in October.

C Appendix: Installation

C.1 Installation of the Operating System (Linux)

For the installation of Linux on the iPAQs, there are pretty good instructions available on the Handhelds.org website [16]. In this section I am pointing out the main steps and giving pointers to external information sources where appropriate.

1. Get the latest Familiar distribution from the Familiar Project homepage [4]. There are two graphical user interfaces to choose from, GPE [17] and Opie [18]. I used GPE.

2. Follow the installation instructions [19] carefully. They show how to back up PocketPC[19], install the Boot-Loader and finally the Familiar distribution itself.[20]

3. After Familiar is up and running, set up the network connections. The network devices (USB, WLAN, Bluetooth) should be recognized on the fly. Just check the IP settings for the USB interface (see tables 4 and 5 in section D), set up USB networking on your computer (section D.1.2) and you should be ready to start working.

Note: To enable normal Internet connection through ETH WLAN network, I had to install OpenSSH as the default ssh client, Dropbear [20], does not support SSH1 connections. In order to save space it may be necessary to remove this software at some point. This can be done by removing packages *libcrypto0.9.7* and *openssh-ssh* using `ipkg`[21] and finally linking `/usr/bin/ssh` back to `/usr/sbin/dropbearmulti`.

C.2 Compiling AODV-UU

The routing protocol I finally ended up using is AODV-UU [7]. This section describes the compilation process of the routing protocol for the iPAQs.

1. Get the cross-compiler toolkit from Handhelds.org[22]

2. Get the kernel source from public CVS at Handhelds.org.

3. Set the environment variables necessary for cross-compilation.

4. Set the kernel source ready for compilation. In directory `include/asm/` link arch-sa1100 -> arch and proc-armv -> proc

5. Configure the kernel. For iPAQs I used (model H3870), the default configuration (which was enough war my purposes) could be loaded with `make ipaqsa_config`). Finalize the configuration by running `make oldconfig`[23]

6. Compile AODV-UU with `make KERNEL_DIR=/ipaq_kernel_src/ arm`

[19]the Windows operating system that most likely came pre-installed with the iPAQ

[20]During the installation process, you will need to use serial connection between the iPAQ and your computer. More information about this can be found in section D.1.1, but also through the installation instructions.

[22]I found two toolkit versions, 3.3.2 and 3.4.1, tried both, and fi nally decided to stick with 3.3.2 - seems to be working well enough.

[23]There might be some unset confi g options that you will be asked for. Hitting enter to select the defaults should be suffi cient.

D Appendix: Configurations

The testbed network was extended with two iPAQs. They were shipped with a pre-installed Windows Pocket PC (Windows CE) Operating System, which was backed up and replaced with a Familiar Linux 0.8.2 distribution [4]. The user accounts for the Linux installations are listed in table 3.

User name	Password	Comments
user	ipaq	Normal account.
root	smon	Root account. Only for administrative tasks.

Table 3: User accounts for the iPAQs.

The basic network configurations for the iPAQs are listed below in tables 4-5. I also wrote a couple of simple scripts (see section E.1) to allow the configuration to be quickly changed[24].

In addition to WLAN, Bluetooth, USB or plain serial connection (RS232/USB) can be used to connect the iPAQs to the outside world. The serial connection is normally used during the installation process, and for debugging purposes. Necessary configurations for different communication methods are discussed in section D.1. For more information about the installation procedure, see section C.1.

The Siramon framework is implemented in Java. The Java environment and necessary configurations for the iPAQs are discussed in section D.2.

IP (wlan)	192.168.0.10
IP (usb)	192.168.1.201
ESSID	siramon
NICK	iPAQ I

Table 4: iPAQ I - Network configurations

IP (wlan)	192.168.0.11
IP (usb)	192.168.2.201
ESSID	siramon
NICK	iPAQ II

Table 5: iPAQ II - Network configurations

D.1 Configurations necessary for communication over cable (Linux $\leftrightarrow$ iPAQ)

Note: These configurations apply directly for Fedora Core 4 Linux distribution. For other distributions, and for communication possibilities with computers running Windows, information can be found on the HandHelds.org homepage [21].

D.1.1 Serial Connection (RS232/USB)

Plain serial connection (RS232) is nice in that it doesn't really require much – only a serial (COM) port, which unfortunately is missing from many newer computers[25]. When the hardware issues are in order, feel free to use your favorite terminal emulator (e.g. *minicom*) for communication. Just be sure to set the serial port settings to match the ones in table 6[26].

[24] During the project I frequently connected to the Internet with the iPAQs, updating the network and downloading necessary software.

[25] Speed is a very limiting factor for RS232 and thus its usefulness has quickly vanished over the years. Nowadays, it is being replaced by other (faster) communication methods, like USB, for example.

[26] Check the Serial Device to match the COM port you are connecting your iPAQ. In Linux COM1 maps to /dev/ttyS0, COM2 to /dev/ttyS1 and so on.

Serial Device	/dev/ttyS0
Bps/Par/Bits	115200 8N1
Hardware Flow Control	No
Software Flow Control	No

Table 6: Serial Port Settings.

The serial connection can be established through the ARM Bootloader. Although, after the installation procedure, the serial connection should always be up. That is, it should be possible to just plug in the serial cable, start the terminal program, boot up the iPAQ and a login screen should appear.

USB console has reportedly been successfully used[27], but I couldn't get it working.

D.1.2 USB Networking

First of all, you need to have USB networking support enabled in your kernel. Here it is assumed that you are running a 2.6 kernel. For older kernel versions, please refer to documentation at handhelds.org [22]. The following options need to be checked:

- Multi-purpose USB Networking Framework (can be compiled as a module)
- Embedded ARM Linux links (iPAQ, ...)

When using `make menuconfig` these can be found through *Device Drivers* ⟶ *USB support* ⟶ *USB Network Adapters*.

The next step is to configure the usb interfaces. The files listed in tables 7 and 8 should do the trick.

```
ONBOOT=yes
IPADDR=192.168.1.200
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
DEVICE=usb0
SERVICE=static
MII_NOT_SUPPORTED=no
TYPE=Tuntematon
```

Table 7: /etc/sysconfig/network-scripts/ifcfg-usb0 (for iPAQ I)

```
ONBOOT=yes
IPADDR=192.168.2.200
NETMASK=255.255.255.0
NETWORK=192.168.2.0
BROADCAST=192.168.2.255
DEVICE=usb1
SERVICE=static
MII_NOT_SUPPORTED=no
TYPE=Tuntematon
```

Table 8: /etc/sysconfig/network-scripts/ifcfg-usb1 (for iPAQ II)

Notice, that in order for the devices to be correctly identified, you have to first plug in iPAQ I. Only after this device has been successfully recognized can you plug in the second device. The reason for this is that the network settings between the communication partners have to match. It may be possible to define both iPAQs to be part of the same local network thus getting rid of the problem, though I haven't tested it.

[27] In the Familiar Linux installation instructions [19] only installing via serial port is covered, but I found some references to successful USB console experiments in [23] and [24]. Furthermore, the ARM Bootloader has *USB console* listed as a possibility, so it should be possible to get it working.

Furthermore, the first iPAQ you connect might hang. Rebooting the device should help, and usually after that the connection is made without any further problems.

Plugging the device should activate the Linux hotplug subsystem and automatically make the connection (that is, a new network device will be activated and after that SSH connection between your computer and the iPAQs should be possible). If you cannot connect, be sure that your computer has no active VPN connections. VPN might, for security reasons, deactivate all local network connections which, of course, affects also the newly created network connections to the iPAQs.

D.2 Java on iPAQs

The Java VM (Virtual Machine) I used in the project is called *JamVM* [11]. It uses GNU Classpath [25] to provide compatibility with Sun's Java API[28]. Currently, Classpath is fairly compatible[29] with standard APIs up to version 1.4[30]. JamVM is available for the Familiar Linux as a binary package [26].

[28]Application Program Interface

[29]85.45% compatibility reported on 25th July, 2005; although I am using an older version

[30]GNU Classpath is an ongoing project so the compatibility is subject to change. Current comparison tables (for API v. 1.4) can be found at `http://www.kaffe.org/~stuart/japi/htmlout/h-jdk14-classpath.html`

E Appendix: Code

In this section I have included some of the actual code that I have written during the project. I start by listing the few shell scripts I wrote in subsection E.1. The scripts are configured with the settings for “iPAQ I”[31] but simple modifications[32] suffice to use the scripts on any iPAQ. Following the scripts, in subsection E.2, I discuss the changes I made to the actual SIRAMON code in order to get it working on the iPAQs.

E.1 Scripts

E.1.1 aodv

```
#!/bin/sh
#
# This script prepares the network (WLAN) to be
# used with aodv routing in Ad Hoc mode.
#
default_nick="iPAQ II"
default_essid="siramon"
default_ip="192.168.0.11"
nick=""
essid=$default_essid
ip=""

if [ $# -eq 0 ]
then
    echo "Using default values..."
    nick=$default_nick
    ip=$default_ip
elif [ $# -eq 2 ]
then
    if [ $1 = "--ip" ]
    then
        nick=$default_nick
        ip=$2
    elif [ $1 = "--nick" ]
    then
        nick=$2
        ip=$default_ip
    fi
elif [ $# -eq 4 ]
then
    if [ $1 = "--ip" && $3 = "--nick" ]
    then
        nick=$4
        ip=$2
    fi
fi

if [ "X$nick" != "X" ]
then
    if [ "X$ip" != "X" ]
    then
        echo "Setting MODE to ’Ad-Hoc’..."
        iwconfig eth0 mode Ad-Hoc
        sleep 1
        echo "Setting ESSID to ’$essid’..."
        iwconfig eth0 essid $essid
```

[31]Here, as also in other parts of the report, “iPAQ I” refers to the device confi gured with IP address 192.168.0.11 for Ad Hoc networking.

[32]Actually, the only script that modifi es the IP address is *aodv*. The default values of the script should be modifi ed depending on the device it is installed on so overlapping addresses can be avoided.

```
        sleep 1
        echo "Setting NICKNAME to ’$nick’..."
        iwconfig eth0 nick "$nick"
        sleep 1
        echo "Setting IP address to $ip..."
        ifconfig eth0 $ip
        echo "Refreshing the network device..."
        ifconfig eth0 down
        ifconfig eth0 up
        echo "Killing all running DHCP clients..."
        killall udhcpc
        echo "All done."
        echo "The aodv server can be located in /usr/local/aodv-uu/"
    else
        echo "Usage: {--help} [--ip ip_address] [--nick nick_name]"
    fi
else
echo "Usage: {--help} [--ip ip_address] [--nick nick_name]"
fi
#
# Author: Samuel Korpi (samuel.korpi@hut.fi)
#
```

E.1.2 ethz

```
#!/bin/sh
#
# This script restores the network settings
# so that it is possible to connect to the
# Internet and use the ’update’ script to
# check that the environment is up-to-date.
#
echo "Restoring MODE to ’Managed’..."
iwconfig eth0 mode Managed
echo "Restoring ESSID to ’public’ (for ETH network)..."
iwconfig eth0 essid public
echo "Refreshing the network device..."
ifconfig eth0 down
ifconfig eth0 up
echo "Starting the DHCP client..."
udhcpc -b -p /var/run/udhcpc.eth0.pid -i eth0
echo "All done."
echo "You may now run ’./update username’ to"
echo "update the environment. Replace ’username’"
echo "with your N-ETHZ username."
#
# Author: Samuel Korpi (samuel.korpi@hut.fi)
#
```

E.1.3 login_ethz

```
#!/bin/sh
#
# This script restores the network settings
# so that it is possible to connect to the
# Internet and use the ’update’ script to
# check that the environment is up-to-date.
#
echo "Restoring MODE to ’Managed’..."
iwconfig eth0 mode Managed
```

```
echo "Restoring ESSID to ’public’ (for ETH network)..."
iwconfig eth0 essid public
echo "Refreshing the network device..."
ifconfig eth0 down
ifconfig eth0 up
echo "Starting the DHCP client..."
udhcpc -b -p /var/run/udhcpc.eth0.pid -i eth0
echo "All done."
echo "You may now run ’./update username’ to"
echo "update the environment. Replace ’username’"
echo "with your N-ETHZ username."
#
# Author: Samuel Korpi (samuel.korpi@hut.fi)
#
```

E.1.4 update

```
#!/bin/sh
#
# This script tries to update the clock settings from a time
# server, upgrade the system (software packages), or both,
# depending on wthe parameters you use to call the script.
#
# Note: This script requires a working Internet connection.
#       Use the scripts ’ethz’ and ’login_ethz’ to accomplish
#       this
#
if  [ $# -ne 1 ]
then
    echo "Usage: update time|system|all"
else
    if [ $1 = "time" ]
    then
        echo "Checking the time and adjusting when necessary..."
            ntpdate -b swisstime.ethz.ch
    elif [ $1 = "system" ]
    then
        echo "Updating the local ipkg packet lists..."
        ipkg update
        echo "Updating the environment if necessary..."
ipkg upgrade
    elif [ $1 = "all" ]
    then
        echo "Checking the time and adjusting when necessary..."
        ntpdate -b swisstime.ethz.ch
        echo "Updating the local ipkg packet lists..."
        ipkg update
        echo "Updating the environment if necessary..."
        ipkg upgrade
    else
        echo "Usage: update time|system|all"
    fi
fi
#
# Author: Samuel Korpi (samuel.korpi@hut.fi)
#
```

E.2 Changes to the SIRAMON code

```
CHANGELOG (29.7.2005)

Project: Porting of Siramon framework to iPAQs

Status: Partially completed, main problems with XML showing/editing
        support.

Changes made to the original version:

- every class having anything to do with GUI (i.e. classes importing
  any SWT classes) were modified to use AWT/Swing instead of SWT

- the event fetching loop of the original GUI had to be replaced
  by a loop that checks whether the related GUI window is disposed
  or not. For this, I also introduced further code to set this
  isDisposed-tag when the window has been closed. On the iPAQs, however,
  this causes a segmentation fault when closing a window. After that,
  the still running java engine has to be closed by ’killall -9 java’.

- the code fetching the local ip addresses in
  Siramon.Network.Sender.retrieveLocalAddresses() was incompatible
  with the Java version on iPAQs. Now the code shows some IPs manually
  entered, but actually  other changes in the code since enable the user
  to enter the IPs when starting the Siramon (see Code/Setup/RunSiramon
  and RunSiramon-ipaq for information on how this is done in practise).

- the Siramon.Gui.XmlEditor(Content).java and
  Siramon.Gui.XmlViewer(Content).java still need some work to get them
  running nicely on the iPAQs. The ’tree view’ design used in the original
  Siramon might not be the best solution on the iPAQs - programming it
  using only AWT/Swing components may prove to be too difficult. The easiest
  solution (if you want to get it working fast) might be just to use a
  TextArea where the whole xml file is loaded and force the user to modify
  the xml code directly. This could then be checked using an xml parser and
  then used as the output.

- I didn’t really have the chance to clean the code and provide good
  comments for my changes, so if you have any guestions, don’t hesitate
  to contact me.

- RolfsBlast: Here, in addition to the already mentioned changes, I run
  into problems with Java’s KeyEvent recognition on the iPAQs. So I added
  some buttons for simple movement of one player just by clicking the
  buttons. The KeyEvent checking is still in the code and works when the
  code is run on a PC but not on the iPAQs.

--
Samuel Korpi (samuel.korpi@hut.fi)
```

References

[1] Farkas, K. (2005). *SIRAMON - Service provIsioning fRAMework for self-Organized Networks*.
`http://www.csg.ethz.ch/research/projects/siramon`
(Referenced on 17th July, 2005)

[2] Grüninger, R. (2004). *Service Provisioning in Mobile Ad Hoc Networks*.
`http://www.tik.ee.ethz.ch/~farkas/publications/Rosamon-master_thesis.pdf`
(Referenced on 24th July, 2005)

[3] Grace, K. (2000). *Mobile Mesh: Providing Solutions For Mobile Adhoc Networking*.
`http://www.mitre.org/working/tech_transfer/mobilemesh/`
(Referenced on 3rd October, 2005)

[4] Hicks, J. and others (2005). *The Familiar Project*.
`http://familiar.handhelds.org/`
(Referenced on 24th July, 2005)

[5] Perkins, C. and others (2003). *Ad hoc On-Demand Distance Vector (AODV) Routing*.
`ftp://ftp.rfc-editor.org/in-notes/rfc3561.txt`
(Referenced on 7th October, 2005)

[6] Grace, K. (2000). *Kernel AODV*.
`http://www.mitre.org/working/tech_transfer/mobilemesh/`
(Referenced on 7th October, 2005)

[7] Nordström, E. and others (2005). *AODV-UU – Ad-hoc On-demand Distance Vector Routing*.
`http://core.it.uu.se/AdHoc/AodvUUImpl`
(Referenced on 26th July, 2005)

[8] MOMENT Lab @ UCSB (2004). *AODV*.
`http://moment.cs.ucsb.edu/AODV/aodv.html`
(Referenced on 7th October, 2005)

[9] Eclipse.org (2005). *SWT: The Standard Widget Toolkit*.
`http://www.eclipse.org/swt/`
(Referenced on 5th October, 2005)

[10] Eclipse.org (2005). *Eclipse Project*.
`http://www.eclipse.org/eclipse/`
(Referenced on 7th October, 2005)

[11] Lougher, R. (2005). *JamVM*.
`http://jamvm.sourceforge.net/`
(Referenced on 24th July, 2005)

[12] Schmidt, M. (2002). *HowTo set up common PAN scenarios with Bluez's integrated PAN support*.
`http://bluez.sourgeforge.net/contrib/HOWTO-PAN`
(Referenced on 8th September, 2005)

[13] Holtmann, M. (2004). *Bluetooth and Linux*.
`http://www.holtmann.org/linux/bluetooth/`
(Referenced on 8th September, 2005)

[14] Eclipse.org (2005). *The SWT FAQ*.
`http://www.eclipse.org/swt/faq.php`
(Referenced on 5th October, 2005)

[15] Hewlett-Packard Development Company, L.P. (2005). *HP Handheld Devices*.
`http://welcome.hp.com/country/us/en/prodserv/handheld.html`
(Referenced on 26th July, 2005)

[16] Handhelds.org (2005). *Handhelds.org*.
`http://www.handhelds.org/`
(Referenced on 24th July, 2005)

[17] Handhelds.org (2004). *GPE: The GPE Palmtop Environment*.
`http://gpe.handhelds.org/`
(Referenced on 26th July, 2005)

[18] Handhelds.org (2005). *Opie: Open Palmtop Integrated Environment*.
`http://familiar.handhelds.org/`
(Referenced on 26th July, 2005)

[19] Handhelds.org (2005). *Familiar v0.8.2 Installation*.
`http://familiar.handhelds.org/releases/v0.8.2/install/`
(Referenced on 25th July, 2005)

[20] Johnston, M. (2005). *Dropbear SSH server and client*.
`http://matt.ucc.asn.au/dropbear/dropbear.html`
(Referenced on 26th July, 2005)

[21] Handhelds.org (2005). *Handhelds.org – User Documents*.
`http://www.handhelds.org/moin/moin.cgi/UserDocuments`
(Referenced on 24th July, 2005)

[22] Handhelds.org (2005). *Handhelds.org – UsbNet (Connecting Your iPAQ to a Linux Desktop via USB)*.
`http://www.handhelds.org/moin/moin.cgi/UsbNet`
(Referenced on 25th July, 2005)

[23] Gray, T. (Thu Jul 24 2003). *Re: Help, I can't restore my pda to poclet pc.*.
The Familiar Archives [`http://handhelds.org/hypermail/familiar/0307/15779.html`]
(Referenced on 25th July, 2005)

[24] Campbell, B. (Wed Aug 06 2003). *Re: usb seriall*.
The Bootldr Archives [`http://handhelds.org/hypermail/bootldr/0308/2301.html`]
(Referenced on 25th July, 2005)

[25] Free Software Foundation, Inc. (2005). *GNU Classpath*.
`http://www.gnu.org/software/classpath/classpath.html`
(Referenced on 24th July, 2005)

[26] Korsgaard, P. (2003). *IpkgFind: Your familiar search engine*.
`http://ipkgfind.handhelds.org/`
(Referenced on 7th October, 2005)