

HELSINKI UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INSTITUT EURECOM

Anna-Kaisa Pietiläinen

Measuring Human Mobility

Master's thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Technology

Espoo, 15th October 2007

Supervisor: Prof. Olli Simula

Instructors: Christophe Diot, D.Sc.(Tech.) and Pietro Michiardi, D.Sc.(Tech.)

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Computer Science and Engineering		ABSTRACT OF MASTER'S THESIS	
Author	Anna-Kaisa Pietiläinen	Date	15th October 2007
		Pages	vii + 72
Title of thesis	Measuring Human Mobility		
Professorship	Communications Engineering at Eurecom	Code	T-115
Supervisor	Prof. Olli Simula		
Instructors	Christophe Diot, D.Sc.(Tech.) and Pietro Michiardi, D.Sc.(Tech.)		
<p>In this thesis we present the design and the implementation of an opportunistic communications library and a testbed for measuring human mobility together with a first mobility measurement experiment performed with the testbed. Opportunistic communications or Pocket Switched Networking (PSN) is an emerging communication paradigm that aims to exploit intermittent contacts between mobile devices to exchange data without a contemporaneous end-to-end path. One of the main characteristics of PSN is the human mobility. Measuring and understanding the mobility patterns and social interactions will help in designing efficient forwarding algorithms for PSN.</p> <p>While the mobile devices already have several wireless networking interfaces, taking an advantage of them in an ad hoc way requires still an access to the low-level programming APIs. In this work we initially choose to work with Windows Mobile based devices and we present the relevant development platforms for them: the Java development platform and the native Windows Mobile APIs. We use the latter to implement a networking library to support opportunistic communications and a testbed upon it to measure human mobility. The human mobility traces collected by the testbed contain information about the nearby Bluetooth devices, 802.11 (Wifi) access points and ad hoc nodes, and the available operators and the current cell id. We give a detailed description of the functionality and the design of the software together with our experiences in working with the Windows Mobile native APIs and devices.</p> <p>Finally, we present the first human mobility experiment performed at Thomson Paris research laboratory during which we collect over a 1-month long mobility trace using 15 Smartphones that are carried by the lab personnel. We evaluate the testbed performance based on the experiment results and provide statistics and an analysis of the initial results.</p>			
Keywords	Pocket Switched Networks (PSN), Opportunistic communications, Human mobility, Measurement, Windows Mobile, Bluetooth, 802.11, GSM		

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto		DIPLOMITYÖN TIIVISTELMÄ	
Tekijä	Anna-Kaisa Pietiläinen	Päiväys	15. Lokakuuta 2007
		Sivumäärä	vii + 72
Työn nimi	Ihmisten liikkuvuuden mittaaminen		
Professuuri	Communications Engineering at Eu-recom	Koodi	T-115
Työn valvoja	Prof. Olli Simula		
Työn ohjaajat	Tekn.tri. Christophe Diot ja Tekn.tri. Pietro Michiardi		
<p>Tässä diplomityössä esittelemme opportunistiseen tiedonvälitykseen mobiililaitteiden välillä tarkoitetun ohjelmistokirjaston sekä kirjastoa hyödyntävän, ihmisten liikkuvuutta mittaavan testiohjelmiston. Työssä kuvataan myös ensimmäinen laajempi ihmisten liikkuvuutta mittaava koe, joka toteutettiin testiohjelmistomme avulla.</p> <p>Opportunistinen tiedonvälitys (myös "taskuverkot", Pocket Switched Networks) on uudenlainen tiedonvälitys tapa, joka hyödyntää mobiililaitteiden välillä ajoittain ilmaantuvia kontakteja ilman vaatimusta samanaikaisista päästä-päähän yhteyksistä. Ihmisten liikkuvuus on yksi tämän kaltaisten laitteiden välisten ad-hoc verkkojen pääominaisuuksista, jota voidaan hyödyntää tiedonvälityksessä opportunististen kontaktien sekä infrastruktuurin lisäksi. Ihmisten liikkuvuuden mittaaminen, liikkuvuusmallien sekä sosiaalisten vuorovaikutusten ymmärtäminen on siten tärkeää mm. tehokkaiden välitysalgoritmien suunnittelussa.</p> <p>Mobiililaitteissa on nykyään tyypillisesti käytettävissä useampi langaton radioteknologia tiedonvälitykseen. Näiden täydellinen hyödyntäminen ohjelmallisesti vaatii matalan tason ohjelmointirajapintojen käyttöä. Tämä työ on toteutettu Windows Mobile mobiililaitteilla. Esittelemme kaksi Windows Mobile -käyttöjärjestelmän tukemaa ohjelmistokehitysympäristöä, jotka ovat Java -alusta sekä natiivit Windows Mobile -ohjelmointirajapinnat. Valitsimme jälkimmäisen ohjelmistokirjastomme sekä testiohjelmistomme implementointiin. Kuvaamme yksityiskohtaisesti ohjelmistokirjaston ja testiohjelman toiminnallisuuden, suunnitelman sekä toteutuksen. Lisäksi käsittelemme käytännön kokemuksiamme Windows Mobile -rajapinnoista ja -laitteista.</p> <p>Lopuksi esittelemme tekemämme ihmisten liikkuvuutta mittaamisen kokeen, joka toteutettiin Thomson Paris tutkimuslaboratoriossa. Koe suoritettiin 15 Windows Mobile -älypuhelimella, jotka annettiin laboratorion henkilökunnan käyttöön kuukauden ajaksi. Testiohjelmamme mittasi liikkuvuutta opportunististen kontaktien kautta keräämällä tietoa lähettyvillä olevista Bluetooth -laitteista, 802.11 -tukiasemista ja ad-hoc laitteista sekä GSM/3G verkkoympäristöstä. Arvioimme testiohjelmistomme tehokkuutta ja luotettavuutta koetulosten perusteella sekä esittelemme ja analysoimme kerättyjä tietoja.</p>			
Avainsanat	Pocket Switched Networks (PSN), Opportunistinen tiedonvälitys, Ihmisten liikkuvuus, Mittaaminen, Windows Mobile, Bluetooth, 802.11, GSM		

Acknowledgements

This Master of Science thesis was done as a part of the European Hagggle project at the Thomson Paris Research Lab under the supervision of Dr. Christophe Diot and Dr. Pietro Michiardi from Institut Eurecom.

I would like to thank Dr. Diot for the invaluable opportunity to work under his supervision at the Thomson Paris Research Lab. He has provided me with his time, guidance and an encouraging and interesting working environment for this thesis work. Special thanks for help and collaboration go to the Hagggle project people in the lab, namely Augustin Chaintreau, Abderrahmen Mtibaa and Erik Nordström. I would also like to thank the whole personnel and visitors of the Thomson Paris Research Lab for their support, for interesting discussions and for being such a good experimentalists. Lastly, many thanks also to all the European Hagggle project partners that I had the possibility to meet and to work with during my thesis.

I also appreciate all the help and support from Institut Eurecom staff and professors during the final year of my studies and in finding an interesting thesis work. Similarly, I thank Dr. Olli Simula, the official thesis supervisor at my home university, Helsinki University of Technology (HUT) for his feedback, and both Dr. Simula and Ms Eija Kujanpää, the international study programme coordinator of HUT for encouraging me to finish my studies at Insitut Eurecom and providing assistance on the practical matters.

Finally, I express my gratitude to my family and friends for all the support.

Paris, 9th October 2007

Anna-Kaisa Pietiläinen

Contents

Acknowledgements	iii
Table of Contents	iv
List of Figures	v
List of Tables	vi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Objectives and Scope	2
1.3 Outline	4
2 Background and Related Work	5
2.1 Opportunistic Communications in Mobile Environments	5
2.1.1 Architectures and Protocols	5
2.1.2 Forwarding Algorithms	6
2.2 Measuring Human Mobility	7
2.3 Wireless Networking Technologies	7
2.3.1 Bluetooth	7
2.3.2 IEEE 802.11b/g	10
2.3.3 GSM and GPRS/EDGE	13
3 Java for Mobile Devices	17
3.1 Java Platform Micro Edition	17
3.2 General Issues with Java ME	19
3.3 Java Virtual Machines for Windows Mobile	20
3.3.1 Esmertec Jbed	20
3.3.2 NSIcom CrEme	21
3.3.3 IBM WebSphere Everyplace Micro Environment J9	21
3.3.4 PhoneME	22
3.4 Discussion	22

4	Windows Mobile Platform	24
4.1	Platform Overview	24
4.2	Development Environment	25
4.2.1	Languages and Tools	25
4.2.2	Platform Security Model	25
4.3	Native Networking APIs	26
4.3.1	Bluetooth	27
4.3.2	802.11 and NDIS	28
4.3.3	Radio Interface Layer	30
4.4	System and Device Power Management	31
4.4.1	Smartphone	31
4.4.2	Pocket PC	32
5	Testbed	34
5.1	Requirements and Functionality	34
5.2	Design	36
5.3	Implementation	36
5.3.1	General Issues	36
5.3.2	Bluetooth	38
5.3.3	802.11	38
5.3.4	Radio Interface Layer	39
5.3.5	Power Management	39
5.4	Discussion	40
6	Device and Testbed Evaluation	42
6.1	Devices Performance Evaluation	42
6.1.1	Simple Battery Life Experiment	42
6.1.2	First Real Life Usage Experiment	45
6.1.3	Controlled Battery Life and Bluetooth Sampling Effect Experiment	46
6.1.4	Discussion	48
6.2	Testbed Deployment	48
6.2.1	Experiment Setup	48
6.2.2	Initial Results	49
6.2.3	Discussion	58
7	Conclusion	62
7.1	Summary	62
7.2	Future Work	63
	Appendix	64
A	Test Devices	64

List of Figures

2.1	Bluetooth device state diagram	8
2.2	Bluetooth protocol layers	9
2.3	802.11 MAC frame format [60]	11
2.4	802.11 STA state diagram	12
2.5	GSM network structure [28]	14
2.6	GSM identifiers for location areas and base stations [42]	15
3.1	Java platform architecture [16]	18
3.2	Java ME configurations [16]	18
4.1	Windows CE 5.0 communications architecture [34]	26
4.2	Windows CE 5.0 NDIS architecture [30]	29
4.3	Windows CE 5.0 RIL architecture [32]	31
5.1	Testbed control GUI	35
5.2	Testbed components and architecture	37
6.1	Simple battery life experiment scan results	43
6.2	Controlled battery life experiment setup	46
6.3	Controlled battery life experiment scan results	47
6.4	Lab experiment trace and offline durations per device	50
6.5	Lab experiment unique Bluetooth devices per device	51
6.6	Lab experiment Bluetooth contact and inter-contact time distributions	52
6.7	Lab experiment discovered Bluetooth device trace from device 12	54
6.8	Lab experiment unique Wifi nodes per device	55
6.9	Lab experiment Wifi contact and inter-contact time distributions	56
6.10	Lab experiment Wifi contacts trace from device 12	57
6.11	Lab experiment unique operators and cells per device	59
6.12	Lab experiment cell contact and inter-contact time distributions	60
A.1	Test devices	66

List of Tables

2.1	802.11 Beacon frame body format	11
2.2	802.11 Probe Response frame body format	13
3.1	Java ME APIs, versions and specifications	19
3.2	Java Virtual Machines for Windows Mobile 5.0	20
4.1	WM5.0 SDK Bluetooth libraries	27
4.2	WM5.0 SDK Bluetooth header files	28
4.3	WM5.0 SDK NDIS header files	29
6.1	Simple battery life experiment statistics	44
6.2	4-day real life usage experiment statistics	45
6.3	Controlled battery life and sampling experiment statistics	46
6.4	Imote discovery rate by distance	48
6.5	Lab experiment Bluetooth contacts and unique devices	50
6.6	Lab experiment Wifi contacts and unique nodes	53
A.1	Technical details of the test devices	65

Chapter 1

Introduction

This chapter gives an introduction and the motivation for this thesis work. We also present the objectives and the scope of the thesis and an outline of the rest of the report.

1.1 Background and Motivation

Mobile phone is one of the fastest spreading and most pervasive communication technology of the human history. Today worldwide we already have more mobile phones than PCs and the markets are still growing. One recent study reports that the number of mobile phone users has passed the record of 3 billion users which is equivalent to half the world's population [39]. While Europe for example has already attained a 100% penetration, the device renewal rate is very high. In addition there are emerging markets especially in the developing countries where mobile phones can offer an affordable communications media and services compared to PCs. The second important aspect of the mobile revolution is that the devices start to have fairly good computing power, memory and battery life. The storage space with the advent of removable memory cards is no more an issue. In addition, most of the devices today include not only the simple voice service (GSM/UMTS/CDMA) but a number of wireless data transmission interfaces such as the GSM data services (GPRS/EDGE), 3G packet data services, Bluetooth or infrared for short-range radio communications, or even 802.11 (Wifi) interface on more advanced devices like Smartphones and PDAs.

Pocket Switched Networking (PSN) [72] is an emerging communication paradigm defining the communications between mobile devices carried by humans, such as cell phones, Smartphones, laptops and PDAs. In PSN the nodes take advantage of the local, usually wireless, connection opportunities between the devices and the inherent user mobility in addition to the (most likely intermittent) infrastructure connectivity. The architecture and the protocols of today's Internet operate poorly in such environments. For example routing based on fairly stable end-to-end paths and TCP/IP end-to-end connections break as soon as we face frequent disruptions, partial and intermittent connectivity, asymmetric data rates or long delays that characterise PSN. To overcome these challenges, a novel communication architecture and new forward-

ing mechanisms are needed. This is currently an active research area and falls under the more general concept of Delay Tolerant Networking (DTN) [9].

Haggle [11] is a new autonomic networking architecture designed to enable communications in the presence of intermittent network connectivity by using opportunistic communications when the end-to-end communication infrastructures are not available, i.e. essentially in PSN. Haggle is also the name of the Future and Emerging Technologies (FET) Integrated Project funded by the European Union's Framework Programme 6 (FP6). Work under Haggle has started in January 2006 and the project will last for 4 years. This thesis work is a part of the Haggle project. Previous work within the project include measurements on human mobility with Bluetooth devices and analysis of the mobility patterns for the forwarding algorithms in PSN, architecture prototype for desktop PCs and various studies and proposals for PSN forwarding algorithms.

Human mobility is a key characteristic of PSN. In order to design, test and analyse efficient forwarding algorithms, and systems and architectures in general, for opportunistic communications, we need detailed knowledge about the human mobility patterns, social networks and communities. This is also the main motivation for this thesis work. We want to continue the work already started on human mobility trace collection, this time using real mobile devices and experimentalists.

1.2 Objectives and Scope

The main objective of this thesis is to study, design and implement a software library for PSN-like networking for mobile devices. The core functionality of the library consists of the network neighborhood detection, the network interfaces management and data communications. The network neighborhood includes information such as nearby Bluetooth devices, available Wifi (802.11) access points and ad hoc nodes, the available cellular operators and the current cell id. The interfaces management means device power control and configuration (association, link establishment and address configuration) to enable autonomous communications. Finally, possible data transmission media include RFCOMM over Bluetooth, and TCP/IP over Wifi or GPRS.

We use the core library to build a testbed on mobile devices to collect traces about human mobility and opportunistic contacts. The testbed will continue the previous human mobility measurements performed within the Haggle project. The previous experiments were done using Intel Motes (iMotes) in research lab, university and conference environments [49, 58]. The iMote is a simple device with an ARM processor, Flash memory and a Bluetooth radio. The iMote platform has several limitations: short battery life (non-rechargeable), small storage (512kB), it is just a simple radio device without any useful functionality (from the carrier point of view), and developing on it is quite hard as reported in [69].

The new testbed aims to overcome these limitations by using real mobile devices to collect the data. The mobile devices are not limited by the storage (MicroSD

or MiniSD being almost a default option) nor CPU. As the devices of today have several networking interfaces, we can collect more detailed and varying information about human mobility and opportunistic contacts by recording not only Bluetooth contacts, but also Wifi and cellular interfaces information. Using real devices will also provide us with a more realistic scenario since the devices will be carried by the experimentalists in a “natural” way. Building a testbed on real (and popular) device platform makes it also possible to run larger-scale experiments as the software can be installed on any compatible device. Finally, and related to the last point, the mobile devices offer common and usually well documented development environment(s) which should provide us the required functionality for implementing the testbed and library functions in an easy way.

The secondary objective (and the preliminary work required to build the library and to run the experiments) is to choose a suitable development platform and a device for the first large-scale experiment we envisage to execute with the testbed. The Huggle project is also moving on a mobile platform with their prototype, and this work is used as an input for their device and platform selection. The device and platform choice is limited initially on purpose to Windows Mobile based devices due to the Huggle project requirements. At the time of writing it is also the only common mobile operating system having the native support for Bluetooth, Wifi (802.11) and GSM/GPRS (and/or UMTS/HSDPA). Symbian, for example, contains the native Wifi support only in its latest versions (9.3 and onwards, some OEM specific extensions available for previous versions) and Linux devices are not yet widely available. One of the advantages of Windows Mobile over the other options is exactly its popularity. The Huggle project wants to build its prototype to be available to a largest possible community, so marginal platforms are ruled out. Of the actual available development platforms for Windows Mobile operating system, we evaluate the Java Micro Edition and the native Windows Mobile APIs.

In the device evaluation we compare five mobile devices from different manufactures running Windows Mobile 5.0 or 6.0. We use the testbed software to get the basic benchmarks. We are mainly interested in the mobility tracing functionality and battery life. Additional evaluation including efficient device and service discovery and data transmission experiments is left for future work even though we mention shortly some of those aspects already in this work.

To validate the testbed functionality and to collect a first human mobility data set with the new testbed, we acquire 20 Smartphones based on the device evaluation and perform a larger-scale mobility measurement experiment at the Thomson Paris Research Lab during August 2007. The initial mobility traces collected during the experiment are presented in this report but detailed analysis of the traces (mobility patterns etc.) is out of the scope of this work.

1.3 Outline

This paper is organised as follows. In Chapter 2 we give an overview with the relevant literature references to the opportunistic communications research, the previous work on human mobility measurements and the networking technologies and standards related to this work. In Chapter 3 we describe the Java platform for mobile devices and our experiences with it. Then we give an overview to Windows Mobile Operating System and the supported networking technologies and APIs in Chapter 4. Chapter 5 details the core library and testbed functionality, design and implementation. We present the devices performance evaluation and the testbed deployment results and analysis in Chapter 6. Finally, Chapter 7 concludes the work and discusses the future work. The mobile devices referenced throughout this paper are presented in Appendix A.

Chapter 2

Background and Related Work

In this chapter we give an overview to the opportunistic mobile communications research including proposed architectures, communication protocols and forwarding algorithms. We present the previous work on measuring human mobility, and finally, we take a look into the networking technologies and standards related to this thesis work.

2.1 Opportunistic Communications in Mobile Environments

Pocket Switched Networking (PSN) was already briefly presented in the introduction. The term comes from the Huggle project and was initially introduced in [72]. It is used to define the opportunistic communications between mobile devices carried by humans, such as cell phones, Smartphones, laptops and PDAs. In PSN there are three methods to transfer data: 1) the device to device local connections, 2) the infrastructure connectivity, and 3) the user mobility. Below we present the related work on PSN-like architectures and protocols including the Huggle architecture. Then we overview some forwarding algorithms suitable for PSN.

2.1.1 Architectures and Protocols

PSN is a special case of Delay Tolerant Networking (DTN). The DTN research group of Internet Research Task Force [9] addresses the architectural and protocol design principles arising from the need to provide interoperable communications with and among extreme and performance-challenged environments where continuous end-to-end connectivity cannot be assumed. Examples of such challenged networks are spacecrafts, military, disaster scenarios and sensor networks. Those environments are characterised by long delays and frequent network partitions. Additionally, often the communicating nodes have limited battery power, memory or other resources. The proposed architecture [54] is based on asynchronous message switching. The architecture forms an overlay over different technology regions (such as TCP/IP or a sensor network).

The overlay is composed of DTN gateways between the regions and DTN nodes within the region. The messages, or "bundles" as they are called in the DTN literature, are forwarded by the gateways in a store-and-forward manner. The DTN group has also published a multi-platform reference implementation of the architecture available for download at [9]. The reference implementation has also been ported for mobile devices including the Nokia 770 and 800 Internet Tablet and Symbian [8].

The architecture proposed by the DTN group shares similar ideas with the Haggle architecture, but Haggle goes even deeper in several aspects by taking advantage of multiple network interfaces present on the device, by leveraging human mobility as a means of transport, by exposing the user data in to the network and by building resource management tightly into the architecture. [72]

Another related architecture for opportunistic mobile communications is the Tetherless Computing architecture: "Tetherless computing is a style of computing where smart mobile devices, such as cell phones and PDAs opportunistically communicate with centralised server clusters over heterogeneously administered wireless networks" [24]. The architecture relies on the DTN bundle forwarding protocols, and adds support for node mobility and persistent sessions over multiple connections. [73]

2.1.2 Forwarding Algorithms

Many algorithms have been proposed for PSNs and DTNs in general. We can categorise the existing algorithms loosely to ones that are based on the message replication, and to ones that try to forward a single copy of a message towards the destination based on some temporal knowledge about the network or the node's contacts and contact opportunities. The classification is similar to one proposed in [43].

The replication based schemes are also called flooding. The simplest example is epidemic flooding [78] in which a message is replicated to each opportunistic contact who yet did not have it. The other extreme is the direct forwarding in which the source stores the message until it meets the destination. Between these two there is a range of algorithms which propose to limit the message replication to provide a reasonable cost-performance trade-off. The more we have messages in transit, the greater is the probability that it gets eventually delivered and with less delay. Whereas having too many replicas consumes resources on the (intermediate) nodes and the overall available bandwidth which can again lower the performance. In [55], the authors propose a simple two-hop relay scheme which floods the message to first n contacts and then uses direct delivery to reach the destination. Several proposals exist that limit the maximum number of hops the message can travel from the source, like the original epidemic flooding proposal cited above, or the number of copies of the message in the network in general, for example [74, 65, 63, 48, 51].

The forwarding schemes, in contrast, need some network topology information to select the best next hop for a message. A general description of so called "oracles" with varying degree of future knowledge and how they are used in message forwarding can be found in [64].

There exists also some work on community based forwarding for PSN. The prob-

abilistic model in [65] already considered communities to restrict flooding. Another related work is [59] where the authors demonstrate how the community information can improve the forwarding performance. In [67] a community-based mobility model for ad hoc network research is presented.

2.2 Measuring Human Mobility

Haggle project has done a lot of work on the area of measuring and analysing human mobility. They have performed measurements in various environments, like research laboratory, university and conferences, using the Intel Motes (iMotes) [49, 58]. The iMote is a small device comprising of an ARM processor, a Bluetooth radio and a flash RAM. The iMotes collect data about nearby Bluetooth devices by performing a device inquiry every 2 minutes and recording any devices in range. The acquired data set contains a list of devices with contact times (duration) as seen by each of the iMotes. The nearby devices can of course be other iMotes or any other Bluetooth device responding to the inquiry. The basic results are presented in terms of contact and inter-contact times between pairs of nodes. The contact times follow an approximate power law. Similarly the inter-contact times can be approximated by a power law but only on moderate time scales (less than a day). A detailed report of the practical limits and experiences from these experiments is given in [69]

In [50] the authors present a more theoretical analysis of the impact of human mobility on opportunistic forwarding algorithms based on several existing data sets. They consider the inter-contact time to have an especially great impact on the opportunistic forwarding algorithms performance since it basically measures how frequently an opportunity to transport data between a pair of devices occurs. They also show how the power law model (which is defined by a parameter called heavy-tail index) affects the performance of some well-known forwarding algorithms. Actually, many of the existing algorithms seem to perform very badly with this mobility model.

Few other projects have also been collecting and publishing human mobility traces. In [56] the authors report a large-scale campus 802.11 WLAN network usage study where they collected traces and usage statistics from access points and user equipment. The Reality Mining project of MIT [52] used Nokia mobile phones to track nearby Bluetooth devices and cell tower ids during several months in the campus environment (MIT and business school students). CoSpere project at Telematica Instituut [6] has collected a 1-month trace with multi-homed devices storing information such as the cell id, in-range operators, 802.11 access points and Bluetooth devices nearby.

2.3 Wireless Networking Technologies

2.3.1 Bluetooth

Bluetooth is an industrial specification for wireless personal area networks (PANs). Bluetooth provides a short-range wireless communications system indented to replace

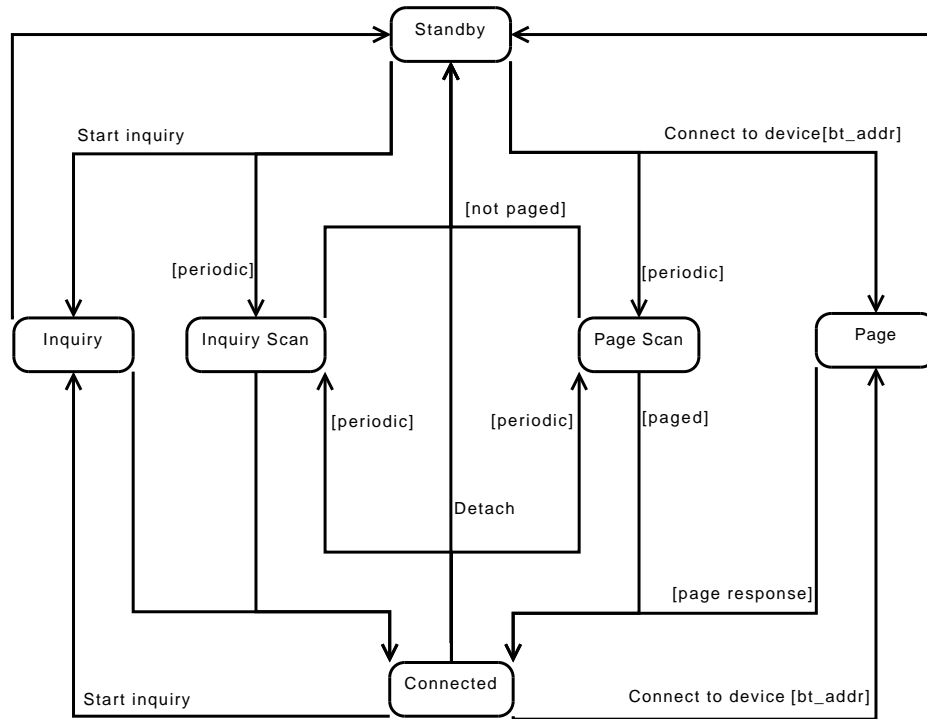


Figure 2.1: Bluetooth device state diagram

the cable(s) connecting various electronic devices such as mobile phones, laptops, PCs, printers, digital cameras, and video game consoles. Bluetooth operates on the licence-free ISM band at 2.4 GHz. The nominal rate is 1 Mbit/s or (Bluetooth v1.1 and v1.2) and with Enhanced Data Rate (EDR), 2 or 3 Mbit/s (Bluetooth v2.0). The operation ranges of Bluetooth devices vary from 1 to 10 to 100 meters (class 3, class 2 and class 1 respectively). The Bluetooth specifications are developed and licensed by the Bluetooth Special Interest Group [5]. The latest defined standard is the Core Specification v2.1 + EDR published in the beginning of August 2007 but in the following we refer to the previous standards, the Core Specification v1.2 and v2.0 [45, 46].

Bluetooth operates on a radio channel that is shared by a group of devices and uniquely defined by a common clock and a frequency hopping pattern. One of the devices is called the master and provides the synchronisation reference, others are known as slaves (up to seven). The group of synchronised devices is said to form a piconet. The ISM band is divided into 79 frequencies that are hopped in a pseudo-random fashion to improve co-existence of Bluetooth with static (non-hopping) ISM systems.

Bluetooth uses separate physical channels for device discovery (inquiry scan channel) and for connecting devices (page scan channel). A discoverable device listens for inquiry requests on its inquiry scan channel, and then sends responses to these requests. In order for a device to discover other devices, it iterates (hops) through

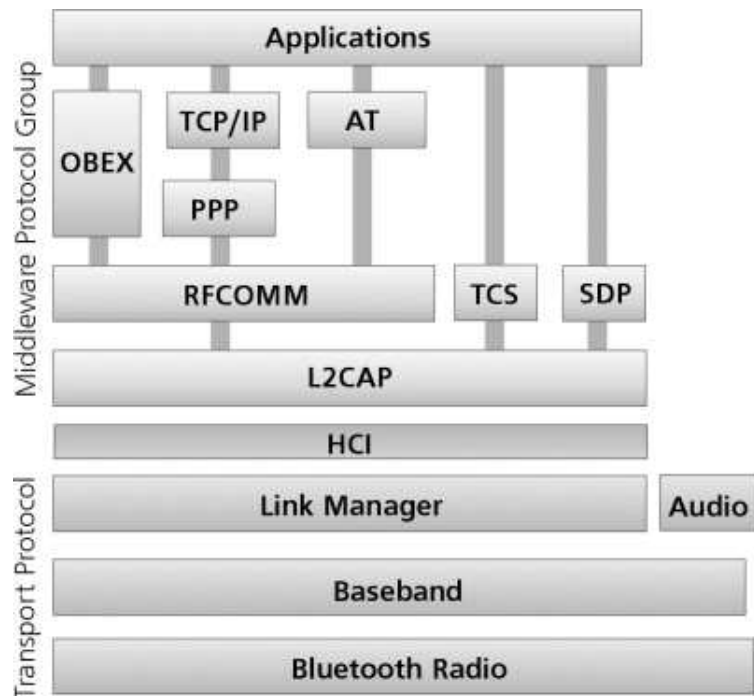


Figure 2.2: Bluetooth protocol layers

all possible inquiry scan channel frequencies in a pseudo-random fashion, sending an inquiry request on each frequency and listening for any response. The inquiry scan channels have a reduced number of hop frequencies and a slower rate of hopping. A recommended inquiry duration is at least 10.24s. The inquiry responses include a 48-bit device MAC address, the device code and clock synchronisation information. The friendly name of the device can be fetched using a separate control request before a physical link establishment (paging). Paging is a similar process to inquiry, a connectable device listens on its page scan channel for connection requests and a connecting device sends the requests on all the available page scan channel frequencies. A device may be connected to other devices while performing both inquiries or paging. The previous description is summarised in Figure 2.1 which presents the complete Bluetooth device state diagram.

The Bluetooth protocol layering is shown in Figure 2.2. Physical links are formed between slaves and the master and are full duplex through the use of a Time-Division Duplexing (TDD). The physical link is used to carry one or more logical links which are controlled by the Link Manager Protocol (LMP). Above the Link Manager resides the Host Controller Interface (HCI). The HCI provides a uniform interface method of accessing the low level Bluetooth controller capabilities. The Logical Link Control and Adaptation Protocol (L2CAP) adds a channel-based data service abstraction to applications and services. The Bluetooth specification includes also one service layer protocol that is common to all Bluetooth applications. That is the Service Discovery Protocol (SDP) which is used to advertise and dynamically discover the available

services and their characteristics. The RFCOMM protocol [44] is a commonly used transport protocol for Bluetooth. It provides emulation of serial ports over the L2CAP. The protocol is based on the ETSI standard TS 07.10.

2.3.2 IEEE 802.11b/g

IEEE 802.11 is the wireless local area network (WLAN) standard developed by the IEEE LAN/MAN Standards Committee WLAN working group [15]. 802.11 family of standards consist of a core standard and several amendments to it. The most common versions implemented on the devices today are 802.11b and 802.11g that are both amendments to the core standard, 802.11-1999 [60].

The original standard defines the basic physical layer (PHY) and the medium access control (MAC) functionality. The original specification defines a PHY that operates at 1 and 2 Mbit/s second on the licence-free ISM frequency band at 2.4 GHz. 802.11b [61] adds an additional physical layer specification for the same band with the maximum rate of 11 Mbit/s, while 802.11g [62] defines high-speed extensions for the same band increasing the maximum rate to 54 Mbit/s. The communication ranges vary from 40 to 80 meters (indoors and outdoors respectively) for 802.11b and from 40 to 90 meters for 802.11g. 802.11b is based on Direct-Sequence Spread Spectrum (DSSS) and 802.11g to Orthogonal Frequency-division multiplexing (OFDM) to obtain the higher rates. Both 802.11b and 802.11g use adaptive rate selection using varying modulations. The supported rates for 802.11b are 1, 2, 5.5 and 11 Mbit/s. In addition to those, 802.11g can function at the rates of 6, 9, 12, 18, 24, 36, 48 and 54 Mbit/s. The 802.11g is thus backward compatible to 802.11b.

The basic building block of 802.11 LAN is a Basic Service Set (BSS). A BSS consists of a number of stations (STA) that can communicate between each other (i.e. are within the radio range). The simplest type of BSS is the Independent Basic Service Set (IBSS) or the ad hoc network. In IBSS the STAs communicate directly between each other and have no access to other BSSs or infrastructure. To provide infrastructure connectivity (e.g. Internet) or communications between BSSs, the architecture defines a component called Distribution System (DS). An Access Point (AP) is a special kind of STA that provides the DS services for STAs associated to that BSS. A DS and BSS(s) can be combined together to create an Extended Service Set (ESS) network. In ESS the APs are responsible of providing association, security, MAC service data unit (MSDU) delivery and infrastructure integration services for the STAs within their range.

STAs can discover available BSSs either passively or actively. Both happen via sending and receiving specific management frames. Figure 2.3 shows the general MAC frame format. The Frame Control field within the header is used to distinguish between different frame types. To do passive scanning, STAs listen for so called Beacon frames that are sent periodically by an AP or in a distributed fashion by all the participating STAs in an IBSS. Active scanning is done by sending a Probe Request frame and collecting the Probe Responses sent by APs, or or in an IBSS, by the last node that send a Beacon. Each MAC frame contains four address fields.

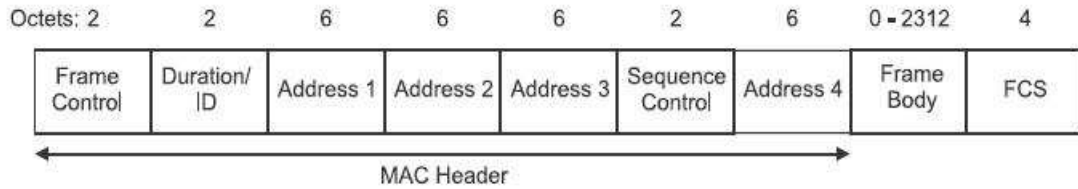


Figure 2.3: 802.11 MAC frame format [60]

Order	Information	Notes
1	Timestamp	A timer value for synchronisation
2	Beacon interval	Number of time units between beacon transmission times
3	Capability information	Includes privacy, BSS type and association control bits
4	SSID	Service Set Identity
5	Supported rates	
6	FH parameter set	Present if the generating STA uses frequency hopping PHYs
7	DS parameter set	Present if the generating STA uses direct sequence PHYs
8	CF parameter set	Present in Beacons generated by APs supporting a PCF
9	IBSS parameter set	Present in Beacons generated by STAs in IBSS
10	TIM	Present in Beacons generated by APs

Table 2.1: 802.11 Beacon frame body format

Depending on the frame type they are used to indicate the Basic Service Set Identifier (BSSID), source and destination MAC addresses and/or transmitting/receiving MAC addresses. The BSSID has the same 48-bit format as an IEEE 802 MAC address and it is used to uniquely identify each BSS. In an infrastructure BSS it is the MAC address of the AP; in an IBSS it is random. The broadcast BSSID is all 1s and is used only when sending Probe Requests. Table 2.1 shows the Beacon frame body and Table 2.2 the Probe Response frame body.

Figure 2.4 summarises the possible states of a STA. The states are defined by two basic state variables, authentication and association.

802.11 MAC uses Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) to share the access to the wireless medium between the STAs of an infrastructure BSS or IBSS (ad hoc network). Prior to transmit, STAs sense the medium to determine if another STA is transmitting. If the medium is busy, the STA waits until

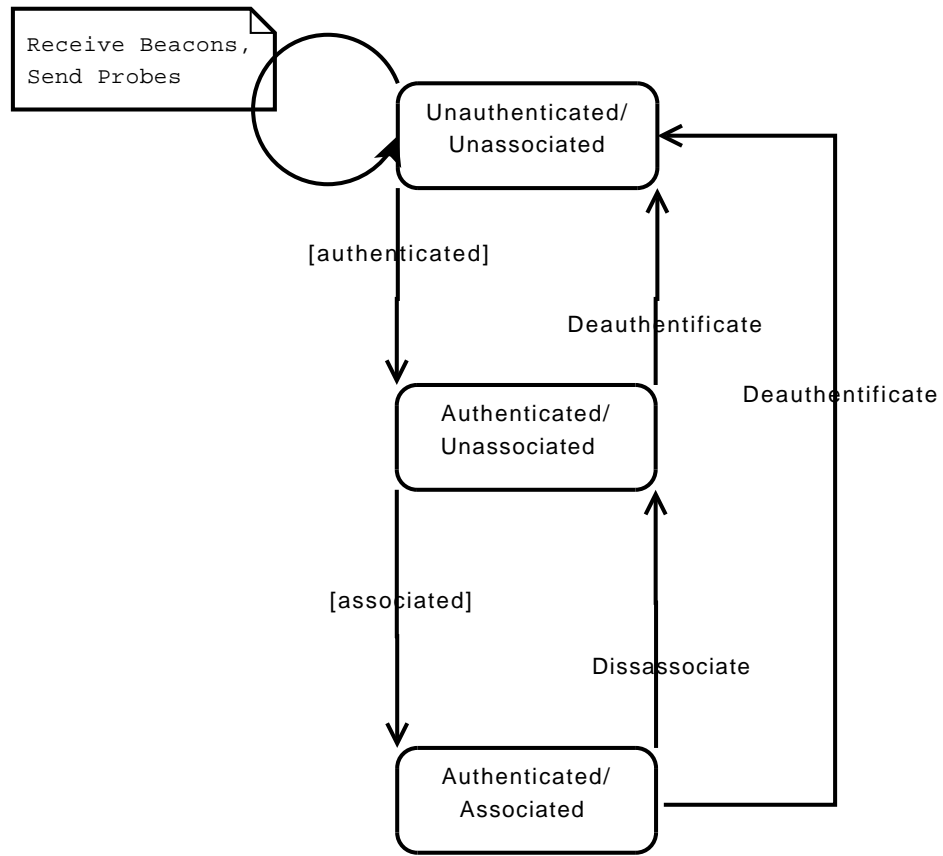


Figure 2.4: 802.11 STA state diagram

Order	Information	Notes
1	Timestamp	A timer value for synchronisation
2	Beacon interval	Number of time units between beacon transmission times
3	Capability information	Includes privacy, BSS type and association control bits
4	SSID	Service Set Identity
5	Supported rates	
6	FH parameter set	Present if the generating STA uses frequency hopping PHYs
7	DS parameter set	Present if the generating STA uses direct sequence PHYs
8	CF parameter set	Present in Beacons generated by APs supporting a PCF
9	IBSS parameter set	Present in Beacons generated by STAs in IBSS

Table 2.2: 802.11 Probe Response frame body format

the medium is free and retries the transmission after a random backoff interval. The basic mechanism can be enhanced by using special control frames, request to send (RTS) and clear to send (CTS), to avoid collisions due to hidden nodes. For details, see [60].

2.3.3 GSM and GPRS/EDGE

GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile) is the most popular standard for mobile communications in the world. GSM is an open standard maintained by the 3rd Generation Partnership Project (3GPP) [1]. 3GPP is a grouping of international standards bodies, operators and vendors originally formed (1998) to define the *3rd generation* (3G) mobile system and related radio access standards, but later amended to take over also the GSM standardisation including its evolved features such as General Packet Radio Service (GPRS) and Enhanced Data rates for GSM Evolution (EDGE). [28]

GSM is a *2nd generation* (2G) mobile radio technology. The *1st generation* (1G) mobile telephone systems (NMT, AMPS, TACS) were analog while the 2G systems are digital (other 2G standards include iDEN, IS-136 and IS-95 and mainly in use in the US). GSM is a cellular network operating on four different radio frequencies: 850 MHz, 900 MHz, 1800 MHz and 1900 MHz. Each frequency band is divided into uplink and downlink bands that are further subdivided into individual carrier frequencies. These frequencies are finally shared using time division multiplexing (TDMA). GSM (and cellular systems in general) are based on frequency reuse i.e. the available channels are grouped in N non-overlapping sets which are then assigned in a repeating pattern

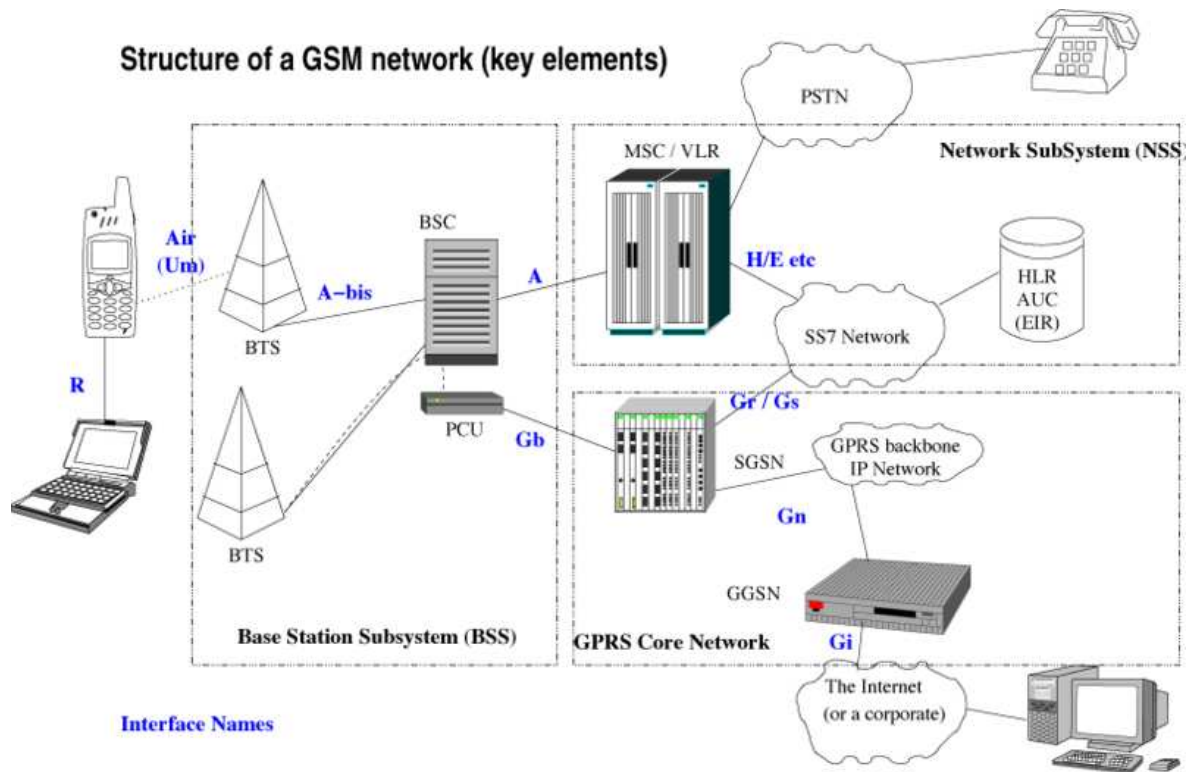


Figure 2.5: GSM network structure [28]

to a hexagonal cell grid. The choice of N and the cell size depend on the traffic distribution and demand, environment and cost. The GSM standard defines four cell types based on the coverage area or size. *Pico cells* are small, mainly indoor cells with a diameter of few dozen meters. *Micro cells* are formed by using antennas below the average roof top level and typically used in urban areas for a range of less than 1-3km. *Macro cells* are formed by installing antennas on the roof top level and usually they reach a minimum distance of 1km, but typically exceed 3km. Finally, *umbrella cells* are used to cover large areas to reach shadow areas and to fill in gaps between the other cell types. The longest supported radius is about 35km. [71, 28]

Figure 2.5 shows the main architectural components of a GSM network or Public Land Mobile Network (PLMN) as referenced in the standardisation. The Base Station Subsystem (BSS) is the basic network building block consisting of Base Transceiver Stations (BTSs) and a Base station Controller (BSC) that manages several BTSs. Each cell has a BTS that takes care of the radio signal transmission and receiving. The BSC is responsible of the actual radio channel allocation and handovers between BTSs within its range. It also handles the traffic and signalling between the mobile node and the Network Switching Subsystem (NSS). The main component within a NSS is the Mobile Switching Center (MSC) which carries out switching functions i.e. manages the calls between mobile nodes and to and from the Public Switched Telephone Network (PSTN). MSC also takes care of inter-BSS handovers. A GSM

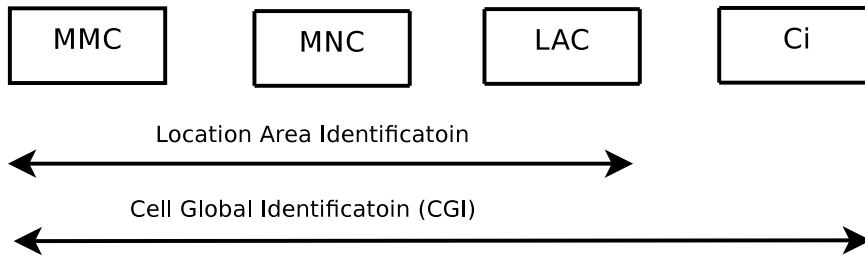


Figure 2.6: GSM identifiers for location areas and base stations [42]

system has several databases to support its functionality. The main ones are the Home Location Register (HLR) for the persistent subscriber information within the home network and the Visitor Location Register (VLR) for the temporary subscriber information when roaming. In addition to the functionality, the GSM standard defines clear interfaces between each component assuring interoperability between various implementations. [71, 41]

In order to route calls in the network, GSM defines a hierarchic numbering scheme for PLMNs. Figure 2.6 shows the basic scheme. Each cell has a Cell Identity (CI) which concatenated with the Location Area Identification (LAI) forms the Cell Global Identification (CGI). Cell Identity must be unique within a location area. The LAI is composed of a Mobile Country Code (MCC) which defines the country where the PLMN is located; a Mobile Network Code (MNC) which identifies a particular PLMN within a country (i.e. an operator); and a Location Area Code (LAC) which identifies a location area within a single PLMN. A location area is defined as an area in which a mobile station may move freely without updating the VLR. A location area may include one or several cells. [41, 42]

While the Subscriber Identity Module (SIM) defines the identity of the subscriber and is used to authenticate the user to a network, it does not contain any further information about the entitled services nor the PLMN service provider i.e. the operator. [53] To provide dynamic way to discover and update PLMN identities and other parameters such as the current time and the time zone, a service called Network Identity and Timezone (NITZ) is defined in [40]. The network identity is composed of a "short" and a "long" name of the operator.

General Packet Radio Service (GPRS) is a packet data service for GSM networks. As the name says, GPRS provides a packet switched service in contrast to the circuit switched operation of the GSM network. For this reason GPRS introduces new entities, GPRS Support Nodes, to the network (GSN). Gateway GSN (GGSN) is the interface between the GPRS backbone and the external packet data network (like Internet). Serving GSNs (SGSN) keep track of the individual mobile stations and are responsible for the delivery of packet to and from the mobile stations within its service area. The GPRS packet service is based on the Internet Protocol (IP) and each device has an IP address when communicating. Enhanced Data rates for GSM Evolution (EDGE) is an improvement to GPRS increasing the general capacity, and

thus the perceived per user data rates. In practise, GPRS offers data rates up to around 60 kbit/s, while EDGE can achieve about 150 kbit/s. [47, 27]

Chapter 3

Java for Mobile Devices

In this chapter we give an overview to the Java Platform Micro Edition, discuss the general requirements for the platform in terms of the testbed functionality, and finally, we present the available JVMs for Windows Mobile devices and our experiences with them. In the last section we conclude the Java platform evaluation and why Java was not chosen for the testbed development.

3.1 Java Platform Micro Edition

The Java Platform Micro Edition (Java ME or J2ME) [16] is the Java application platform for mobile devices. It is designed to work on a wide range of devices from mobile phones to set-top boxes. Java ME is a collection of technologies and specifications that can be combined to create a specific Java runtime for each environment. The three basic elements of the platform are:

- **Configuration** defines the basic set of libraries and JVM capabilities.
- **Profile** provides an additional set of APIs for narrower range of devices.
- **Optional packages** are the technology-specific APIs.

The figure 3.1 where Java ME sits in the overall Java platform architecture.

Currently there are two configurations defined for Java ME: **Connected Limited Device Profile (CLDC)** [76] and **Connected Device Profile (CDC)** [77]. The former is the configuration for small, less powerful mobile devices. The latter is targeted towards more capable mobile devices like PDAs, phones with a touch screen or set-top boxes. The CDC configuration is also a proper subset of the Java Standard Edition, so the same code can be executed both on a desktop J2SE JVM and on a CDC compliant JVM. One notable difference between the CDC and the CLDC configurations is that only CDC supports Java Native Interface (JNI) [75, 77].

On top of the configurations, Java ME specifies a number of profiles that offer higher level APIs and application frameworks. The generally used profile with the CLDC configuration is the **Mobile Information Device Profile (MIDP)**. The

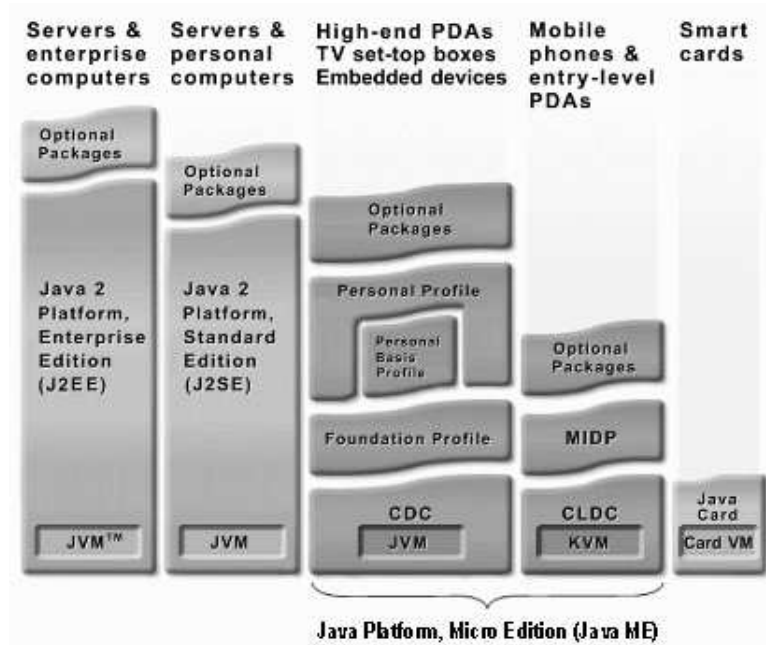


Figure 3.1: Java platform architecture [16]

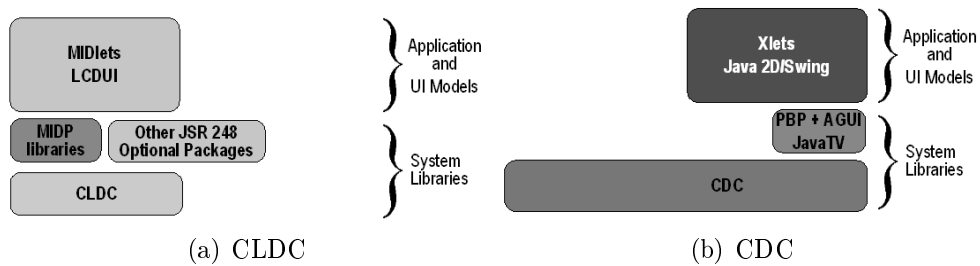


Figure 3.2: Java ME configurations [16]

combination CLDC+MIDP is the most common Java ME environment(3.2(a)) that is found on the mobile phones today. The application framework is based on MIDlets that provides common application life cycle handling routines and specific UI libraries aimed for resources restricted mobile phones.

The CDC(3.2(b)) configuration supports three different profiles: **Foundation Profile**, **Personal Basis Profile** and **Personal Profile**. The foundation profile is the most basic one and does not include any GUI framework. The personal basis profile is a super set of the foundation profile adding the Xlet application interface which is a GUI framework based on lightweight components. The personal profile has all the features of the previous ones plus the standard AWT and applet GUI libraries.

The table 3.1 presents the current Java ME APIs and the specification versions and names in use. It also lists the standardised optional packages.

CLDC		
Connected Limited Device Configuration	1.0	JSR30
Connected Limited Device Configuration	1.1	JSR139
Mobile Information Device Profile	1.0	JSR37
Mobile Information Device Profile	2.0	JSR118
CDC		
Connected Device Configuration	1.1.2	JSR218
Foundation Profile	1.1.2	JSR219
Personal Basis Profile	1.1.2	JSR217
Personal Profile	1.1.2	JSR216
Optional Packages		
Java APIs for Bluetooth (Bluetooth, OBEX)	1.1	JSR82
Content Handler API	1.0	JSR211
Mobile Media API	1.2	JSR135
Java Binding for the OpenGL	1.0	JSR239
J2ME Web Services Specification (JAXP, JAX-RPC)	1.0	JSR172
Security and Trust Services APIs	1.0	JSR177
Security	1.0	JSR219
Advanced Graphics and User Interface	1.0	JSR209
RMI	1.0	JSR66
JDBC	1.0	JSR169
Java TV API	1.1	JSR927

Table 3.1: Java ME APIs, versions and specifications

3.2 General Issues with Java ME

As explained in the introduction, the testbed should be able to detect the wireless neighbourhood of the device by collection information about the nearby Bluetooth devices; Wifi access points and ad hoc nodes; and the cellular environment. We would also like to run the application on the background with as little user intervention as possible.

The Bluetooth API for Java ME, JSR-82, has the required functionality: it provides the device and service discovery functions as well as data communications (type of socket abstraction). The only missing feature is the possibility to explicitly turn on and off the Bluetooth device. The JSR-82 API specification states that if an application tries to invoke Bluetooth methods and the device is off, the implementation should prompt the user for an approval to turn on the device. There is an API method to check the current device power state. [66]

Since the Bluetooth API is an optional package for Java ME, the support depends on the JVM. There are also few open-source and commercial JSR-82 implementations available. We test a commercial implementation from Avetana [3] and an open-source

	Smartphone (Samsung)	Pocket PC (i-mate)
Esmertec Jbed CLDC1.1/MIDP2.0	Not available	OK (no JNI, no JSR-82)
NSIcom CrEme CDC1.0/PP	Failed	Not tried
IBM J9 CLDC1.1/MIDP2.0	OK (no JNI, no JSR-82)	OK (no JNI, no JSR-82)
IBM J9 CDC1.1	OK (console only)	OK
PhoneME CDC and CLDC	not tried	not tried

Table 3.2: Java Virtual Machines for Windows Mobile 5.0

version, bluecove [4]. Both of them work fine on a Java SE JVM on a Windows PC and with the CDC configuration on Windows Mobile devices (bluecove requires slight changes and a recompilation for the target platform). The open-source implementation can be extended to contain the power on/off functions but that kind of an 'hack' breaks the API compatibility and portability across multiple platforms.

Low level Wifi interface access is not supported by the standard Java APIs. There is no API for scanning nearby access points, switching the device mode from infrastructure to ad hoc, turning on/off the device, and so on. The standard way to provide such low level functionality to the JVM, is to program a Java Native Interface (JNI) compatible library using the native platform languages and tools, and to access that library using JNI calls. This is how the current Huggle prototype accesses the Wifi card or how the bluecove JSR-82 API implementation is done too. The same restriction holds for the cell radio programming i.e. getting the list of available operators, the current cell id and so on.

3.3 Java Virtual Machines for Windows Mobile

Windows Mobile does not contain a Java Virtual Machine (JVM) by default. Some phones come with a pre-installed JVM, on others it is possible to install one of the available free or commercial JVMs. We try out several available JVMs on a Windows Mobile 5.0 Smartphone (Samsung i600) and a Windows Mobile 5.0 Pocket PC (i-mate JAQ3). See Appendix A for the device specifications. The table 3.2 summarises our experiences.

3.3.1 Esmertec Jbed

Esmertec Jbed [10] is a commercial JVM that comes both in CLDC/MIDP and CDC configurations. The commercial strategy seems to be to sell the product directly to the phone manufactures. No evaluation downloads nor pricing information of the JVM is available on the web.

However, the i-mate has the Esmertec Jbed Advanced (CLDC 1.1 / MIDP 2.0) pre-installed, so we try it out. The basic MIDlet installation is smooth using the AMS interface provided by the software and the execution of the example MIDlets is fast.

The drawbacks of the Esmertec Jbed are that the i-mate for example should be capable of running the more advanced CDC configuration (since it is a pocket PC with a touch screen) but for some reason that version is not installed. The installed version of the JVM did not seem to have the JSR-82 optional package (no documentation was available). Also native access using JNI is unsupported.

3.3.2 NSIcom CrEme

NSIcom CrEme [18] is another commercial JVM for Windows CE (includes Windows Mobile) devices. NSIcom provides a free evaluation version for 30 days after a simple registration. No pricing information is available.

The latest version of CrEme is compliant with CDC 1.0 specification with the Personal Profile, and the AWT and more advanced Swing GUI libraries. The JVM is JNI compatible and should also include a set of optional interfaces (not specified). CrEme also implements an applet plugin for the Pocket Internet Explorer.

The demo versions are provided for a number of different architectures including Inter ARM, Samsung, TI OMAP and x86. Depending on the CPU chosen, there are versions for several versions of Windows Mobile. Curiously, they offer a CDC version of the JVM for WM5.0 Smartphone (by definition some UI problems were anticipated). That version (CDC1.0 / Intel PXA / Windows Mobile 5.0 Smartphone) was tried on the Samsung since its would offer the support for JNI.

The installation is easy using the application installer that handled the installation automatically over ActiveSync. Running the java application happens by creating a special link file (a file ending in .lnk and containing the full command to launch CrEme). However, both the provided example application and a simple test AWT application do not work correctly. They will launch but the GUI is totally unusable. Also, the basic JNI calls seem not to work.

CrEme is not tried on the Pocket PC (the i-mate) since a working Java environment for it was found with the IBM J9 (see next section).

3.3.3 IBM WebSphere Everyplace Micro Environment J9

IBM provides a practically free JVM (~\$ for a licence) for mobile devices called J9 or WebSphere Everyplace Micro Environment [26]. The website provides trial downloads and the licenced versions can be bought from a webstore called Handango.

J9 comes in various configurations for several platforms. We try the CLDC 1.1 / MIDP 2.0 and CDC 1.1 configurations on x86 Windows XP, the CLDC 1.1 / MIDP 2.0 on an ARM Windows Mobile 5.0 Smartphone (the Samsung) and both the CLDC 1.1 / MIDP 2.0 and CDC 1.1 on an ARM Windows Mobile 5.0 Pocket PC (the i-mate). The JVM comes without any optional packages, and the CDC version implements JNI.

On a Windows PC, both versions run well with the example MIDlets and AWT applications. Also the Bluetooth access was tested and worked using the bluecove JSR-82 implementation.

Similarly, on the Pocket PC, both configurations and the provided examples run well. The Bluetooth API (modified bluecove) could be used with the CDC configuration.

On the Smartphone, the CLDC version works fine but it does not provide the Java Bluetooth API nor JNI as mentioned. The CDC version is tried too (even if it is not really compiled for Smartphones), and actually basic console application does run well and can access the Bluetooth API using JNI. GUI applications on the other hand, will not even launch as expected. It is not completely clear if we could just use the CDC configuration in console mode on the Smartphone for our purposes. One possible problem is that there could be some underlying API calls in the JVM that are supported by the Pocket PC and not by the Smartphone which could create unexpected problems that would be hard or impossible to fix.

Overall J9 seems to have a good performance and is quite easy to use after some try and error with the command line parameters. It also provides an useful console mode for running the applications which is a great help for simple system out style debugging. The CLDC version also provides a MIDlet manager to install and run applications.

3.3.4 PhoneME

While the other JVMs presented are all more or less commercial, there is at least one well known open-source Java ME JVM project going on, namely the phoneME at the java.net [19]. The project is working on both a CLDC compliant JVM, phoneME Feature, and a CDC compliant JVM, that they call phoneME Advanced.

To use phoneME on Windows Mobile 5.0, a custom build is required (no binary downloads available). Build instructions are on the project wiki and according to the wiki the JVM should run on a WM5.0 Pocket PC.

Anyhow, this option is not investigated further. It is still a work-in-progress and we do not know much about the stability and performance of the JVM. Also it is not clear how much of the standard functionality and optional packages are implemented and what is still missing.

3.4 Discussion

The need for JNI for the low level networking functionality effectively rules out the possibility to use a simple CLDC / MIDP JVM. As a work-around, it is proposed that the native methods could be exposed to the JVM via a localhost TCP service [2] but we choose not to try that because of the added complexity and possible overhead.

The CDC configuration does not have this limitation, so it would be the ideal choice for the Java platform. More over, CDC applications would be portable across

mobile devices and standard desktop environments. That is one of the reasons we investigate Java in the first place since the Huggle project early prototype is written in Java and is CDC compliant. However, finding a working CDC JVM for Windows Mobile devices turned out to be quite hard. Sun does not provide one for Java ME. The best and almost free option is the IBM J9. The only drawback with J9 CDC JVM is that there is no real version targeted for the WM5.0 Smartphone edition. Even if we manage to run another JVM version (the Pocket PC one) on the Smartphone, we would be restricted to the console mode only. And limiting ourselves to just to Pocket PCs is also considered not an ideal solution.

The second drawback with Java is the performance. The memory print of the tested JVMs was around 15MB or more (as reported by the OS memory manager application) when running a simple test application. This is quite a lot considering that the devices have about 20 - 40 MB memory free while only the OS is running.

To conclude, based on these experiences and on the fact that we need native software for the low level networking tasks in any case, we decide to implement the testbed using the native Windows Mobile APIs and development language(s).

Chapter 4

Windows Mobile Platform

In this chapter we present the Windows Mobile Operating System and the development environment. Specifically we take a look into few application development related issues like debugging and the platform security model. Then we present the networking related native Windows Mobile APIs and the system and device power management on the Windows Mobile.

4.1 Platform Overview

Windows Mobile 5.0 is a compact operating system targeted to resource constrained hand held devices ranging from embedded systems to PDAs and Smartphones. The vast majority of the devices in use today are Windows Mobile 5.0 powered which has been around since May 2005 [29, 17].

Windows Mobile 5.0 comes in two flavors: Smartphone and Pocket PC. The biggest differences between the two lie in the display hardware support and the system power management. Basically the Pocket PCs have a touch screen while Smartphones do not. Also the system power states are different between the two editions. Apart from these two major differences, the functionality and the SDK APIs for Smartphones and Pocket PCs are almost identical. Both the Smartphone and the Pocket PC editions are specialized versions of the Windows CE 5.0 which is the Microsoft's operating system for resource restricted devices and embedded systems.

Microsoft has published in the beginning of 2007 the version 6.0 of the Windows Mobile OS, but most of the available devices today are still running WM5.0 or slowly upgrading to 6.0. The WM6.0 is still based on the Windows CE 5.0 internally, so applications written for WM5.0 and that use standard and documented APIs should run without changes on WM6.0 according to Microsoft. The edition naming scheme changes with WM6.0. The Smartphone edition is called WM6.0 Standard and the Pocket PC (Phone Edition), is called WM6.0 Professional.

4.2 Development Environment

4.2.1 Languages and Tools

Windows Mobile offers two main technologies for application development: Native code and Managed code. The native code development means coding in c/c++ using the Windows Mobile native APIs. The managed code refers to the .NET Compact Framework (or .NET CF for short) which can be programmed in C#, J#, C++ or VB.NET languages. .NET is sort of Microsoft's version of Java. The runtime, Common Language Runtime (CLR), comes pre-installed on Windows Mobiles (5.0 has .NET CF 1.0). .NET has, however, a restricted API and some native methods should be called anyways to access the Bluetooth, Wifi and cell radio interfaces. For this .NET offers a mechanism that is similar to Java's JNI which they have named p/Invoke (Platform Invoke).

The main development IDE for Windows platforms is the Microsoft Visual Studio. Only the latest version, Visual Studio 2005 (standard edition or better) fully supports smart devices development with the Windows Mobile 5.0 Pocket PC and Smartphone SDKs. While Visual Studio Standard is not free, the SDKs itself are available for free download. They also include a number of emulators for Visual Studio to test the applications without the actual devices.

Testing and debugging with real devices is done over ActiveSync. There are, however, some restrictions in debugging the networking functions. Windows Mobile 5.0 does not support "multihoming", so any active Wifi connection is terminated and the Wifi is disabled (in practise the controls are removed from the device menus) when the device is connected to the host using the USB cable. This is because the device connection is implemented as a network adapter. Thus debugging the IP networking functionality over ActiveSync is not possible.

4.2.2 Platform Security Model

Windows Mobile OS has a security model that poses some restrictions on running new applications on devices and the operations they are allowed to do. Basically, an application can be either privileged, unprivileged or unsigned. The privileged and unprivileged applications are signed with the privileged or unprivileged certificates respectively which must be stored on the device for verification. Privileged applications have essentially no restrictions in what they can do, while unprivileged do. [33]

Each device has a security configuration that defines the access levels for applications. The configurations are OEM or operator defined and vary from completely locked devices to non restricted. On a *Security-Off* configured device, there's no restrictions and even unsigned applications will run and have full access to the APIs and the registry. *One-Tier-Prompt* configuration requires that the SDK (or other valid) certificates must be installed on the device and application are run in a trusted in which they are allowed to make any API call and to access all the registry values. With the *Two-Tier-Prompt* configuration, the SDK (or other) certificates are required

too and applications can run either in trusted or normal mode. In normal mode some API calls are not allowed as well as the access to some registry keys). *Third-Party-Signed* and *Locked* devices require always that an application is signed by a certificate known by the devices certificate store. The certificate store is pre-configured by the device OEM or the operator, and usually they also control the installation of new certificates. Getting a third-party signed certificate costs money. The SDK certificates should be used for the development phase and a real certificate should be acquired to sign the release version of the application. [33]

4.3 Native Networking APIs

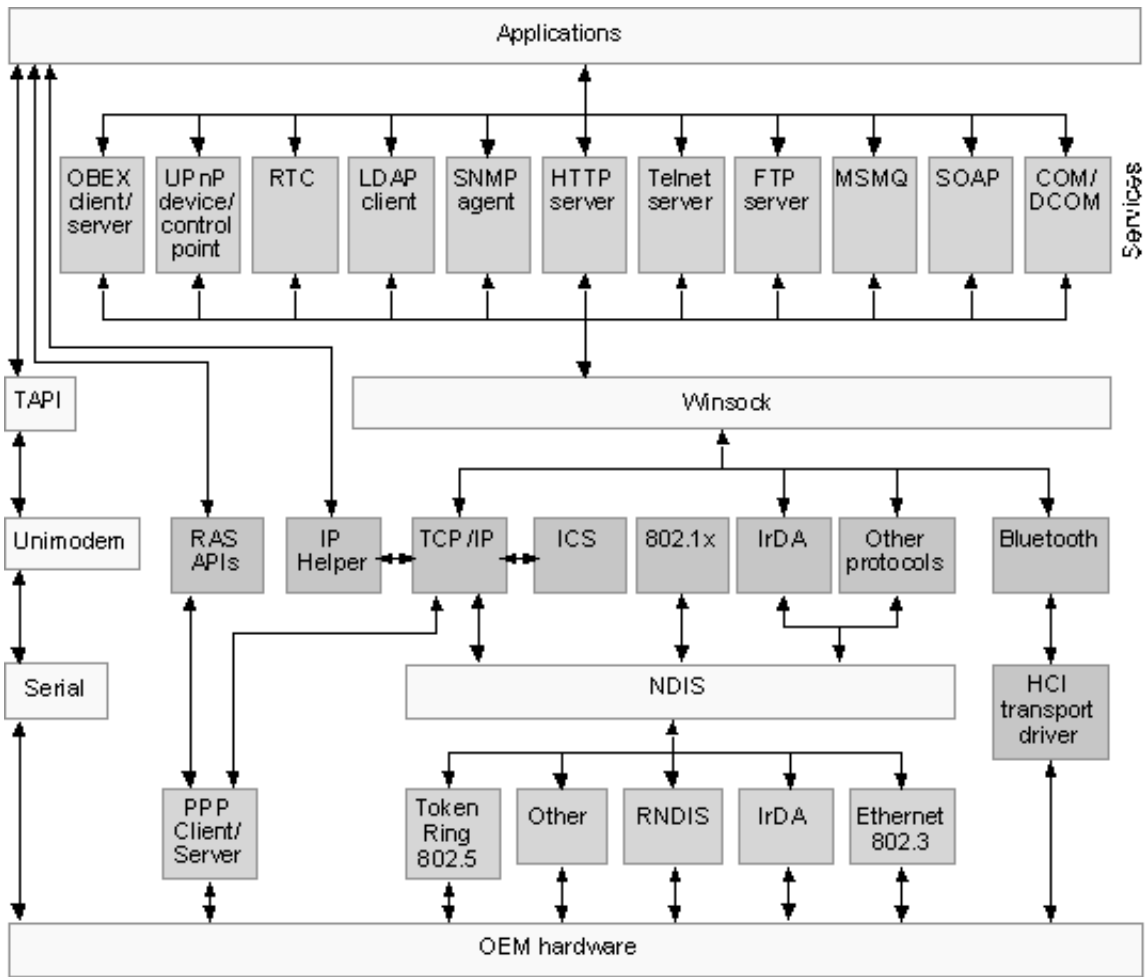


Figure 4.1: Windows CE 5.0 communications architecture [34]

Figure 4.1 shows an overview of the Windows Mobile communications and networking architecture (excluding the Radio Interface Layer for cellular radio). Windows Mobile includes wireless features such as Bluetooth and 802.11 (802.1x, Extensible

Library	Description
Btd.lib	Microsoft core Bluetooth stack. This library exposes functions that are used to extend various layers of the stack.
Bthguid.lib	Contains GUIDs that are used to instantiate various Bluetooth COM components.
Bthutil.lib	Contains functions that are used to turn off or turn on Bluetooth on the device. These functions are declared in the Bthutil.h header file.
Coredll.lib	Contains functions that are used to set up an event notification system from the Microsoft Bluetooth stack. These functions are declared in the Bt_api.h header file.
Ws2.lib	Contains common Winsock2 functions that are used to perform various Bluetooth operations, such as device and service discovery.

Table 4.1: WM5.0 SDK Bluetooth libraries

Authentication Protocol and 802.11 automatic configuration), and services and APIs such as Winsock 2.2 and Object Exchange Protocol (OBEX). In addition, Windows CE includes an updated TCP/IP stack and conforms to the Network Driver Interface Specification (NDIS) 5.1 [34]. In the following subsections we present the relevant components and APIs for our work which are the Bluetooth API; the native Wifi and NDIS APIs; and the Radio Interface Layer API.

4.3.1 Bluetooth

The Windows Mobile 5.0 SDK includes a set of libraries for programming the Bluetooth device (only compatible with the Microsoft Bluetooth stack). The tables 4.1 and 4.2 present the related library and header files. [36]

The Bluetooth API includes functions to query and change the radio mode (on/off). When the device is on, it can be either connectable (device can initiate connections and accept connections) or discoverable (other devices can see it, implies connectable).

The local device mac address can be obtained by creating and binding a socket and performing `getsockname` function on it. The local device name is stored in the registry under the key `HKCU\Software\Microsoft\Bluetooth\Settings\LocalName`.

The device and service discovery, SDP record registration and data communications using the RFCOMM protocol are implemented using specific winsock functions (version 2.2 required).

The device discovery provides the remote device friendly name, the remote mac address and the device class (cod). No signal strength information for example is available. The device discovery time or the maximum number of devices to return can be specified when initiating the query by adding an additional data element,

Header	Description
Bt_api.h	Defines client-level functions for calling into the Bluetooth stack.
Bt_sdp.h	Defines structures and constants for SDP queries.
Bthapi.h	Defines interfaces for managing SDP records.
Bthutil.h	Defines functions for setting and retrieving the mode of the Bluetooth operation.
Winsock2.h	Defines standard Windows socket functions that can be used to perform Bluetooth operations.
Ws2bth.h	Defines Bluetooth specific Winsock constants and structures that can be used in Bluetooth applications over Winsock.

Table 4.2: WM5.0 SDK Bluetooth header files

BTHNS_INQUIRYBLOB, to the query.

The SDP service records must be created by hand (API does not provide any helper functions for that). The service record for the RFCOMM protocol based services contains the channel on which the RFCOMM server is listening for the incoming connections. The service discovery to find out the existence of services, protocols and channels to access them, can be performed either explicitly in a similar fashion as the device inquiry, or implicitly by defining the remote service id (GUID) to connect to in the SOCKADDR_BTH structure. Alternatively to a GUID, a hard-coded channel can be used on the server and client side.

4.3.2 802.11 and NDIS

Windows Mobile network drivers are based on the Network Driver Interface Specification (NDIS). Windows Mobile 5.0 supports NDIS 5.1. NDIS is meant to facilitate the communication between the operating system, the network adapter and the protocol drivers. The NDIS interface (or NDIS Wrapper) is located between the upper-level protocol drivers (such as the TCP/IP protocol driver) and the lower-level communications architecture (intermediate and miniport drivers and a network adapter at the bottom) as shown in the figure 4.2. [30]

To perform nearby AP scanning, switching between ad hoc and infrastructure modes and to associate with BSSs (with or without encryption) we can use the NDIS user-mode I/O (NDISUIO) protocol driver. NDIS protocol driver is an upper level driver on top of the NDIS wrapper (like the TCP/IP driver for example). The table 4.3 lists the related header files.

The basic NDISUIO usage consists of creating a file handle to the NDISUIO driver using the `CreateFile` function, fetching the list of bound adapters (there can be several) and sending control messages to the specified adapter with the `DeviceIoControl` function. The bound network adapters are retrieved using a `IOCTL_NDISUIO_QUERY_`

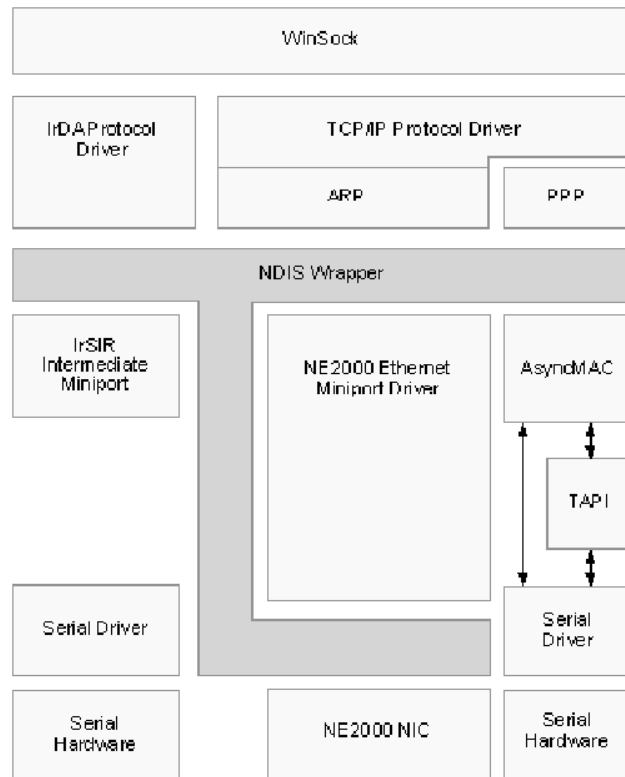


Figure 4.2: Windows CE 5.0 NDIS architecture [30]

Header	Description
ntddndis.h	Defines all constants and types for accessing the network driver interface device.
nuiouser.h	Constants and types to access the NDISUIO protocol driver.

Table 4.3: WM5.0 SDK NDIS header files

`BINDING` control message. The drawback is that it lists only currently bound devices and thus if the Wifi device is disabled by the user from the Wireless Manager, the device is not bound and is not accessible through the `NDISUIO`. Once the Wifi network device is available and its name is known, it can be controlled by sending `IOCTL_NDISUIO_QUERY_OID_VALUE` and `IOCTL_NDISUIO_SET_OID_VALUE` control messages to get and set the NDIS Object Identifiers (OIDs) values that control the device.

Wireless Zero Configuration (WZC) is a service taking care of the network interfaces configurations by default on Windows Mobile. Conceptually WZC resides on top of the `NDISUIO` wrapper and it is using the NDIS API to control the Wifi device. In order to implement the AP/ad hoc association using `NDISUIO`, the Windows Zero Configuration service must be disabled because otherwise it would interfere with the configuration. Windows CE 5.0 has an API, `wzcsapi.h`, that can be used to access the WZC functions [35]. Through the API we can disable and enable WZC for a selected adapter.

The other option to programmatically configure networks is to use the WZC API directly instead of disabling it. The WZC API offers methods to configure networking interfaces via a function call while when using the `NDISUIO`, each step (set mode, encryption, WEP keys, SSID) needs to be performed manually. The second advantage of using the WZC API is that the active configuration is all the time visible and accessible also using the operating system tools which is more convenient from the user point of view. WZC API allows also to reuse any existing preferred network configurations.

The IP address handling can be done using the IP helper, `iphlpapi.h`, API. It supports both static and dynamic address configuration with DHCP [37].

4.3.3 Radio Interface Layer

The Radio Interface Layer (RIL) provides an interface that handles the communication between the CellCore system software and the radio hardware. RIL consists of two modules: the RIL proxy and the RIL driver. An overview of the radio architecture is shown in Figure 4.3. The proxy layer manages callback notifications and inter-process function calls into the driver layer. [32]

The RIL API, `ril.h` and the library, are not included in the WM 5.0 SDKs but are available on a developer site in the Internet [57]. The API is documented by Microsoft starting from the Windows Mobile 6.0 [38]. The custom applications function by registering with the RIL two callback functions. One is used for unsolicited notifications, and the other is used for responses to function calls. For example, when the phone registers with a new cell, the RIL uses the unsolicited notification callback to notify registered applications about the cell id change. Similarly, when an application requests for example the operator list, the call returns immediately and the real operator list is returned through the function response callback. This asynchronous architecture of the RIL requires special actions from the applications that want to function in a synchronous manner i.e. to block waiting for responses. [32]

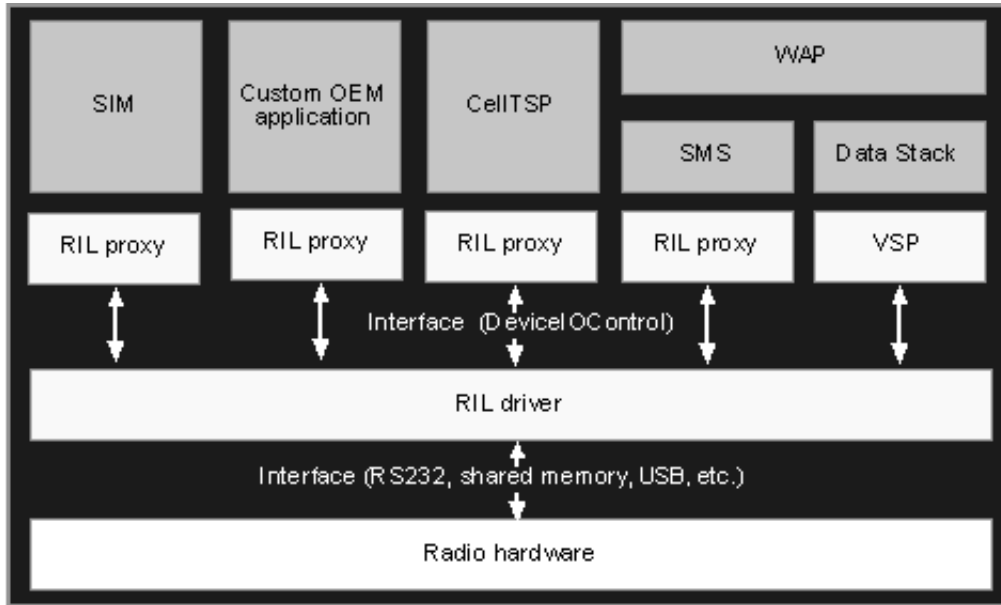


Figure 4.3: Windows CE 5.0 RIL architecture [32]

4.4 System and Device Power Management

A fairly complete discussion about the system power management on Windows Mobile and the differences between Smartphones and Pocket PCs can be found at [20]. To summarise it, Pocket PCs sleep while Smartphones do not. Smartphones do save energy by turning the backlight off and eventually by turning off the screen but all the devices may stay turned on and the applications keep running all the time. On the contrary, the Pocket PC goes to sleep after a defined timeout which means it turns off all the devices (almost, will still receive calls and sms etc.) and stops the running applications. The subsections below give the details of the system power states on Smartphones and Pocket PCs.

The device power management has two main actors on Windows Mobile. The device drivers can be very active and turn on/off the device as required. On the other hand, there is the Power Manager service that manages the general system power states and transitions between them. The Power Manager can control only compatible devices and the support is not required. So unfortunately, the device power management and how it can be controlled is very device and OEM specific issue. [31]

4.4.1 Smartphone

Smartphones take an "Always on" approach to the power management. From the application development point of view this means that the developer has to take care of the CPU utilisation (do no useless work while the phone is idle for example) and

of the device power states when possible. Smartphones have four basic power states listed below: [21]

- **On** The user is interacting with the device. Everything is on.
- **BacklightOff** There has been a brief period of user inactivity. The backlight has been turned off, but everything else is on. When setting the backlight timeout values in the control panel, we define how long the system should wait before going into this state.
- **UserIdle** There has been a longer period of user inactivity. Both the backlight and LCD have been turned off. When setting the screen timeout value on a Smartphone control panel, we define how long the system should wait before going into this state.
- **ScreenOff** The device goes into this state when someone specifically says to turn the screen (and backlight) off. This state is different than UserIdle. In ScreenOff, pressing a button (other than the power button) does not turn the screen back on. In UserIdle, pressing a button does turn the screen back on.

4.4.2 Pocket PC

On the Pocket PC the things are different. The Pocket PC has a suspended (sleep) mode in which the device is completely idle: all the running programs are stopped and devices are powered off. The Pocket PC system power states are listed below: [21]

- **On** The user is interacting with the device. Everything is on.
- **BacklightOff** There has been a brief period of user inactivity. The backlight has been turned off, but everything else is on. When setting the backlight timeout values in the control panel, we define how long the system should wait before going into this state.
- **UserIdle** Generally not used on Pocket PC, since after the user idle timeout the system goes to Suspended state.
- **ScreenOff** The device goes into this state when someone specifically says to turn the screen (and backlight) off. This state is different than UserIdle. In ScreenOff, pressing a button (other than the power button) does not turn the screen back on. In UserIdle, pressing a button does turn the screen back on.
- **Unattended** This a state in which the screen, the backlight, and the audio are off. It is used by applications which need to do things without alerting the user (like ActiveSyncing on the background). While the PocketPC is in this state, the user thinks the device is asleep.

- **Resuming** This is the state the PocketPC goes into when it wakes up from sleep. In this state, the screen is off, and there is a very short (15 second) timer before it goes back to sleep. The only way to keep the device from going back to sleep is to have something put it into one of the other states. This is for dealing with spurious wakeups and for giving the system a way to get into Unattended without letting the user know about it.
- **Suspended** This is the PocketPC "Sleep" state. Everything is off, and the system is not going to wake back up until some piece of hardware wakes it up.

Chapter 5

Testbed

In this chapter we present the core networking library and the human mobility measurement testbed. The first section details the library and testbed requirements and functionality, then we overview the design and finally we describe the implementation and issues we run into during the development. We discuss our experiences and outline some future work in the final section.

5.1 Requirements and Functionality

The core functionality of the networking library and the testbed is the neighbourhood detection using various wireless interfaces in order to enable opportunistic communications and to produce traces about human mobility. The library and the testbed application should not interfere with the normal usage of the device. The testbed should be as transparent as possible to the user i.e. the user interaction with the application should be reduced to very simple actions and the impact on battery life, data connections etc. should be minimal. The library and the testbed are targeted to run on any Windows Mobile 5.0 (and 6.0) Smartphone or Pocket PC device.

By neighbourhood detection we mean the Bluetooth device inquiry, the 802.11 access point and ad hoc node scanning, and the cell id and the available operators listing using the radio interface. The discovery and trace logging are done by a background process at configurable intervals without any user intervention. A simple GUI is provided to select the radio interfaces to be scanned, to set the scan interval length and repetition intervals for each radio, to start and stop the scanning and to indicate that the scanning is in progress and running as expected. A screen capture of the GUI is shown in Figure 5.1.

The mobility traces are recorded in a simple text file on the device storage card or to the ROM if an external memory is not available. For all the contacts we record a timestamp (from the system clock) and a general flag to distinguish between different radios (second and third fields in the example below). For Bluetooth contacts we record the seen device MAC address, the device code and the friendly name of the device if available. For 802.11 APs and ad hoc nodes, we log the BSSID, if it is an infrastructure or an ad hoc node, if the node is using WEP/WPA for privacy, the

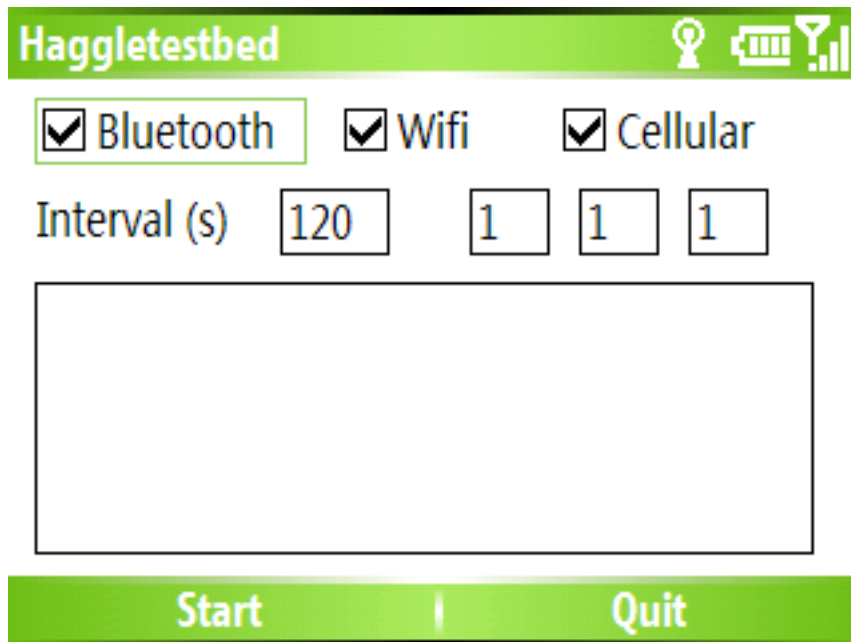


Figure 5.1: Testbed control GUI

received signal strength indicator (RSSI), and the SSID (the “name” of the network). Of the cellular networks we trace the operators’ id, availability and the long name as provided by the network and the current associated cell id when known. In addition, in the beginning of each scan interval, we log some general information about the device state such as the memory consumption, battery level and AC connection status. An example from trace log is shown below (shortened and MACs modified):

```
# 2007/08/03 13:55:20 mem 49 %, avail phy 23494656 b, \
  avail virt 28966912 b, battery 20 %, AC 1
# Wifi scan start: 1186142120
00:00:00:00:00:00 1186142120 0 0 1 166 TMS-WLAN1
00:00:00:00:00:00 1186142120 0 0 1 166 WANAD00-E9F8
# Wifi scan end: 1186142130
# Bluetooth scan start: 1186142130
00:00:00:00:00:00 1186142130 1 256 0 0
00:00:00:00:00:00 1186142130 1 512 0 0 T610
00:00:00:00:00:00 1186142130 1 256 0 0 BOULL060522001
# Bluetooth scan end: 1186142176
# Cell scan start: 1186142176
20820 1186142176 2 32318 0 0 BYTEL
20810 1186142176 2 0 1 0 SFR
20801 1186142176 2 0 1 0 Orange F
# Cell scan end: 1186142206
```

The trace log files are rotated automatically so that the application can run for long periods of time without the trace files growing unmanageably large. Since we assume that each device running the application has a large enough storage card no further trace data compression is done. The traces are collected manually using ActiveSync to copy file from the device to a PC.

In addition to neighborhood discovery, the core network library includes functions to control network interfaces power for autonomous operation. Finally, to get some insight into the feasibility of a Huggle -like PSN application on real devices, and specifically on the chosen platform, we investigate and implement the data transmissions over Bluetooth, Wifi and GPRS. Wifi (and GPRS) data transmission is a basic socket operation but to enable real opportunistic communications we also need to take care of discovering and associating with APs or ad hoc nodes and configuring an IP address (static or DHCP) for the Wifi interface. For Bluetooth communications we experiment with the RFCOMM protocol.

To perform simple throughput measurements we port an open source tool called Test TCP (TTCP) [23] for Windows Mobile and implement an additional option to send and receive data over Bluetooth RFCOMM channels.

5.2 Design

Figure 5.2 shows the general components of the core library and the testbed software. The networking classes present the biggest part of the code and the actual applications are on purpose quite basic. The networking code is packed in a static library that is used by the application projects. Each network interface is presented by an individual class within the library as shown in the component schema. In addition, the Wifi class uses two helpers that wrap the IPHelper API and the WZC API respectively. The main application GUI is a native windows application (not using MFC or ATL) while the actual trace application and the initial data transmission test application (ported TTCP) are basic console applications. The trace and the data transmission processes can be controlled from the testbed GUI but they can also be run as stand-alone applications from the command line.

5.3 Implementation

The testbed is implemented completely using native Windows Mobile APIs and object-oriented C++. Below we describe some of the issues we run into while developing the software and how we solved the problems. The test devices we reference in this section are presented in Appendix A.

5.3.1 General Issues

The Windows Mobile Smartphone and Pocket PC SDKs are almost identical but distributed separately due to the differences outlined in the previous chapter. The

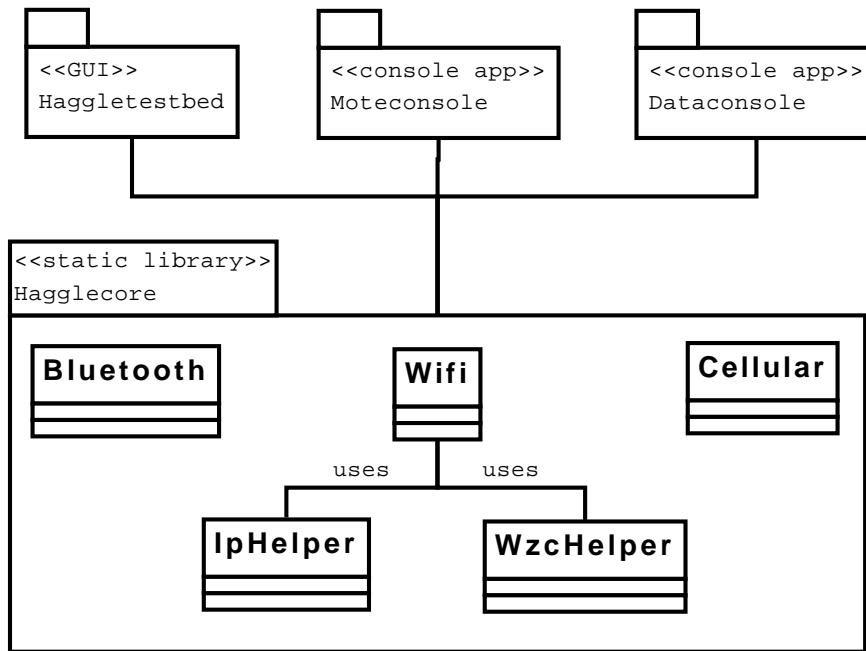


Figure 5.2: Testbed components and architecture

power management and some GUI features in our code differ somewhat and thus we need to have separate builds of the software for Pocket PCs and Smartphones.

Some of the APIs used require privileged access and the software needs to be signed with appropriate certificates. We use the SDK test certificates provided by Microsoft. A certificate installer for the devices comes with the Windows Mobile SDK and must be run on most of the phones before utilising the testbed.

The biggest problem in working with Windows Mobile is the closed nature of the system. In particular while trying to work with very low level APIs, the documentation is often insufficient or unavailable. In addition, the OEMs have the possibility to make custom builds of the OS and to add some OEM specific hooks here and there which makes the development even harder. In some cases we have to rely on information and “hacks” found from the Internet and figure out by trial and error the working ones.

Finally, we encountered one very strange problem with the MicroSD memory cards of some manufactures (Kingston and Dane-Elec at least) and with a certain device, namely the HTC s620. The device freezes completely when we try to run the testbed on that hardware combination. With a SanDisk memory card on the HTC s620, or on any other device – memory card combination, we do not have this problem. The reason for the freezing seems to be that the testbed has two file handles open to the memory card ¹ at the same time. This was verified by disabling the debugging after which the problem disappeared with a Kingston card. But since the problem was

¹One for the trace file and another for a debug log

observed only with this unique combination, we did not spend much time in trying to debug it further or to change the code to overcome the problem. Instead we use SanDisk memory cards with the test devices.

5.3.2 Bluetooth

The Bluetooth API works fine on each test device. One "feature" of the Samsung i600 is that it switches on the screen when returning from the `WSALookupServiceNext` function i.e when it discovers a Bluetooth device. This is not good for the battery life but so far no work-around nor an explanation for this has been found. We did not observe this behaviour with other devices.

General problem with the Bluetooth device discovery is that the duration increases a lot when there are many devices around and we want to fetch the friendly names. This is because the friendly name discovery is an independent link layer request that must be sent to each discovered device separately and it seems to take about 2-3 seconds per device to complete.

The data transmission over RFCOMM was implemented both in client and server mode using either a hard coded channel number or a service record with a known service GUID (globally unique identifier). The Bluetooth APIs do not provide any support for the service record generation so the SDP record (a byte array in practise) has to be constructed by hand as described in the Bluetooth specifications. The service discovery on the other hand, is done implicitly when the service GUID is set to the socket descriptor (instead of a channel number).

5.3.3 802.11

We had few problems to solve with Wifi too. First, the NDISUIO driver can list and access only the currently bound network adapters. If the Wifi device is disabled by the user from the phone's Wireless Manager, the device is not bound and is not accessible through the NDISUIO. We choose to save the Wifi device name to the registry upon the first launch of the software so that it is available to us after that and we can turn on the Wifi device from our code automatically when required.

The second big problem with NDIS is that since there is no common set of compulsory IOCTL OIDs that the device drivers should recognise, the OIDs have to be manually tested on each device. We run into some problems with the Wifi association using NDIS. The code seems to work on some devices while on others it does not. This might be due to the OEM specific customisations. Also any of the power management related OIDs for NDIS do not seem to work (see details below).

As a work around to the association problem we implement the association using the WZC directly. This seems to work nicely both in infrastructure and ad hoc modes. Additionally we can reuse the user network configurations and thus we do not need to store for example the WEP keys separately for our application. WZC also takes care of the IP assignment automatically based on the network mode. In ad hoc mode,

it assigns a Windows default IP within a subnetwork 169.254.0.0, so we need to take care of assigning our own IP (if required) only after the autoconfiguration is ready.

To enable ad hoc communications we need a way to discover neighbour nodes' IP addresses. Since IPv4 [70] does not offer any mechanisms to automatically detect neighbouring nodes in the subnet (vrt. Neighbor Discovery in IPv6 [68]), we implement a simple UDP-based node discovery service, that sends a query message to the broadcast address and a known port. Any listening node in the same subnet will answer to the request and the nodes will discover each others IP addresses and can start communicating directly.

5.3.4 Radio Interface Layer

The asynchronous nature of the RIL API requires some synchronisation code since we want our API to behave in a synchronised (blocking) manner. We use system events and critical section mechanisms to implement the blocking behaviour.

To enable the cell id notifications on the devices, we have to rely on something that seems to be a de-facto hack, namely calling the `RIL_DeviceSpecific` function with an undocumented but working value that is available in the Internet [57].

Some of the devices allow fetching the operator information from the network even without a SIM card (Samsung for example), others require always a valid SIM card for that. No cell id is available unless the SIM is valid and able to connect to an operator.

The 3G devices we have (Samsung and Toshiba) report twice each operator providing both GSM and 3G service. There seems to be an OEM specific bit mask for a parameter in the operator list provided by the RIL API that obviously carries this information, but there is no documentation, and we do not manage to find any example how to read it.

5.3.5 Power Management

The Smartphones we test have some device/OEM specific power controls. The Bluetooth device is still completely controllable and does not have any automated power management due to its low power consumption. The Wifi device on the other hand, has an OEM specific timeout after which it turns itself off. This can be disabled by the user and the configuration is stored in the registry. The registry values are usually found easily but they are device specific. So automatic configuration is not really feasible by changing these values.

As a work-around, we find that the Smartphones are responding to some of the Power Manager API functions. We use the function `SetDevicePower` to turn on and off the Wifi device in order to override the automatic power savings when needed. This is a bit brute force approach but the function `SetPowerRequirement` for example does not work, neither does the NDIS power management OIDs even if no error is usually reported. Calling the function requires the knowledge of the actual adapter name. The name can be fetched by listing the bound adapters through the NDISUIO

but that will only see the Wifi adapter after the user has enabled it in the Wireless Manager like explained above i.e. after it is powered on. So this is a second reason why we need to store the device name to the registry so that we can automatically turn on and off the Wifi device.

However, there is still a problem when using the `SetDevicePower` to turn off the Wifi device. It looks like after that call, Windows is unable to activate the Wifi again (from the Wireless Manager or automatically if configured so). A reboot is required to reset the device state back to normal. We can still turn on the Wifi but somehow the method confuses the normal operation. Currently we chose to only turn on the Wifi as needed, and let the system take care of the power management and turning off the Wifi device. This is quite a performance hit since we cannot turn off the Wifi once we're done with the background scanning for example when the user was not actively using the Wifi interface.

On the Pocket PCs the device power control seems to work almost as with the Smartphones. An additional problem is to turn on the Wifi device when it is disabled using the phone Wireless Manager. The call `SetDevicePower` does not work on this as was explained above for the Smartphones and a workaround is not yet found. Based on the experiences of others (for example the CoSphere project [6]), the Wifi device power management might indeed be quite tricky and very device specific on WM5.0 Pocket PCs. Once the device is bound to the NDISUIO initially, changing the device power states seem to work using `SetDevicePower`.

The second difference in the power management on the Pocket PCs is due to the system sleep state that does not exist on the Smartphones. To keep a background scanning or other application running all the time on a Pocket PC, we need to set the software to go to the unattended mode and to call `SystemIdleTimerReset` function every 30s from keeping it to fall a sleep. These two actions increase the energy consumption since we effectively hinder the system from doing its normal power management.

5.4 Discussion

The software development took some time due to a quite steep learning curve on Windows Mobile development and especially the effort needed to get to know the low level APIs. Currently the core library and the testbed have the initially envisaged neighbourhood detection and mobility trace collection functionality, interfaces management functions, some basic data transmission functions and the test software for Bluetooth and Wifi configuration and data transmissions. The testbed can be used for performance evaluation on such benchmarks as the battery life; Bluetooth, Wifi and cellular neighbourhood detection efficiency (number of nodes seen, time it takes, sampling effects, and so on) and the basic throughput while sending/receiving over Wifi and Bluetooth.

Future work includes among others improving the Wifi power management, trying out the GPRS data transmissions and looking into how the GPRS configuration can

be accessed and modified from a custom application code. Also, an automatic log upload would be nice-to-have if running larger experiments.

We also think that it is better to repack the networking library as a dynamic library (dll) so that the internal state handling becomes easier. This would also ease the refinement of the application interface on top of the networking library. Currently each networking class has its own access functions and the calls are blocking. A better way to do this would be to provide a common way to communicate with the whole networking library using for example an event-based IPC mechanism. Event-based communication would be a good choice because of the dynamic nature of the networking events and changing conditions.

A final note on the development environment chosen. While the native APIs and C++ are almost inevitable to program the low level functionality required for truly autonomous ad hoc communications, they are very slow (in terms of development time) and cumbersome for building a graphical user interface. We suggest that any further UI development should be done either using at least the Microsoft Foundation Classes (MFC) or probably better, the .NET framework.

Chapter 6

Device and Testbed Evaluation

In this chapter we present the experimental results and analysis. We perform two kinds of experiments. The first section describes the experiments that are performed in order to evaluate the test devices performance and to choose one of them for the large-scale testbed deployment and the first human mobility experiment. The second section describes the first large-scale experiment, a performance evaluation of the testbed and the initial statistics and analysis of the human mobility traces we collect during the experiment.

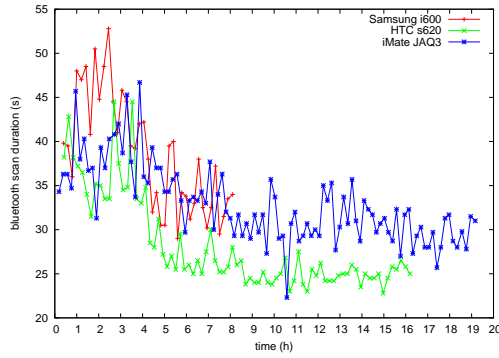
6.1 Devices Performance Evaluation

The devices we compare in this section are presented in the Appendix A. We use the testbed application presented in the previous chapter to evaluate the devices performance in a controlled environment and real life. The experiments were performed using a varying number of devices because they were acquired at different times and we did not repeat each experiment once we had them all.

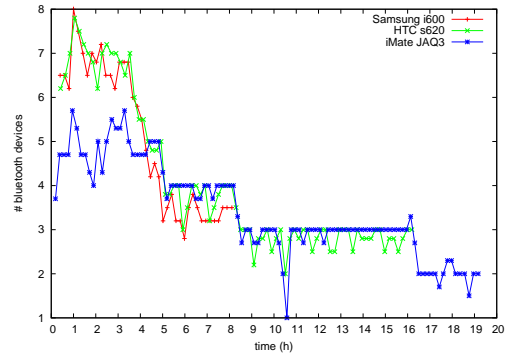
6.1.1 Simple Battery Life Experiment

We perform a simple battery life comparison test with three devices: Samsung, i-mate and HTC s620. The setup is the following: each device scans 802.11 APs and ad hoc nodes, Bluetooth devices and cell towers every 2 min in an uncontrolled but a static environment (all are kept in the same room during the experiment). Results are presented in Table 6.1.

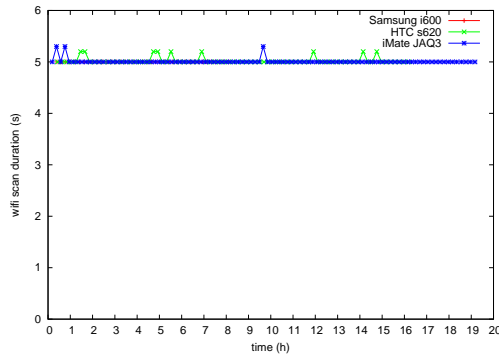
The first observation is that the battery life of the Samsung is very bad compared to the HTC and i-mate that last more than twice as long. One reason can be the fact that Samsung turns the screen on at every Bluetooth scan, which consumes power. The second reason can be that it is the only 3G phone and listens also on the 3G frequencies. The Samsung does not have a SIM card during the experiment but this should not affect the power consumption. Since the Samsung can scan both the GSM and the 3G frequencies, it sees twice the number of operators compared to the other two.



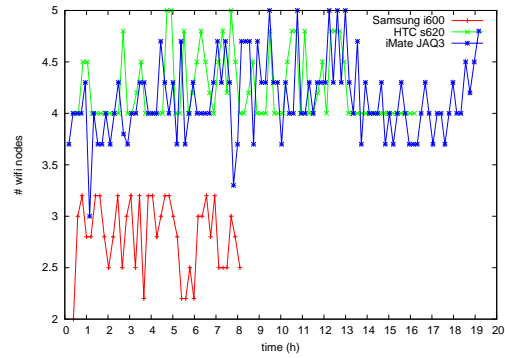
(a) Bluetooth scan duration



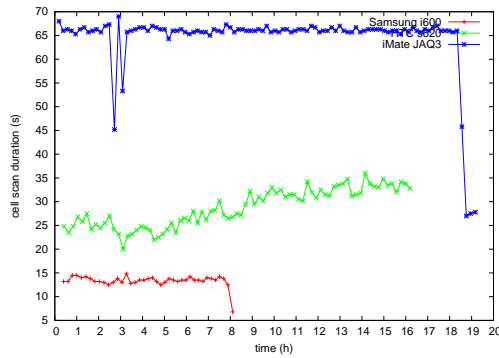
(b) Bluetooth devices



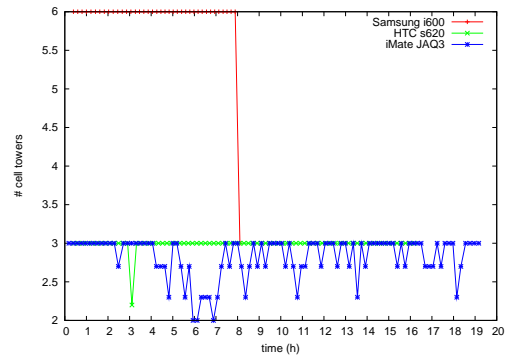
(c) Wifi scan duration



(d) Wifi nodes



(e) Cell scan duration



(f) Operators

Figure 6.1: Simple battery life experiment scan results

	Samsung i600	HTC s620	i-mate JAQ3
Total duration	29401s (8.2h)	58327s (16.2h)	69034 s (19.2 h)
Scans (succeeded/failed)			
Bt	166/0	321/0	312/0
Wifi	166/0	321/0	312/0
Cell	162/4	320/1	312/0
Average scans/hour	61	59	49

Table 6.1: Simple battery life experiment statistics

Figure 6.1 shows the scan durations and discovered devices per radio interface. The Bluetooth and cell scan durations are quite variable while Wifi scan duration is almost constant. The Bluetooth scan duration can be controlled through the API. One can set a maximum duration or a maximum number of devices to be discovered (here we have around 20 seconds or 30 devices) but since we perform also the friendly name query, the durations vary depending on the number of discovered devices. The Wifi scan duration is not configurable, and it is more or less a constant 5 seconds for each device due to a manually programmed sleep before reading the scan results. The cell tower scanning (i.e. the operator information request) length is not configurable either. The i-mate is especially slow in this. The reason for the i-mate performing considerably less scans than the other two is due to this delay too: the initial version of the testbed used in this experiment made a 2-minute sleep after each scan round no matter how long the previous round took. This is fixed in the later versions of the testbed that starts a new round exactly at every configured interval (except if the interval is passed in which case we start a new round immediately).

The number of discovered devices vary also from device to device, especially for Bluetooth and Wifi. This is partly due to the environment which is not controlled: external Bluetooth devices for example can pass by at anytime, and due to interference and possible simultaneous scanning (a device cannot answer to a Bluetooth inquiry while scanning itself) the devices do not discover the same number of devices at the same time. For Wifi, the reason for the variation is partly in the API which does not define when or how exactly the device should collect beacons (or send a probe) and what nodes to return when asked to do a “scan”. The Samsung does not seem to pick up as many APs as the others, while the i-mate does not record as many Bluetooth devices when there are more of them around. The reason why the number of discovered Bluetooth devices decreases towards the end of the experiment is that the experiment starts in the afternoon and runs during the night. At the night time there are not so many external Bluetooth devices (i.e. people with mobile phones) around. The devices see also each other, so the number of potential devices within the range decreases always by one as a device disappears when the battery runs out.

	Samsung i600	HTC s620
Duration	83h	88h
Resets	10	9
Max cont. per scan	27	21
Devices seen	835	692
Contacts (active/total)	All 650/1355 Bt 176/640 Wifi 474/715 Wifi open 92/152	All 1037/1614 Bt 111/392 Wifi 926/1222 Wifi open 615/761
Median cont. time	338s	358s

Table 6.2: 4-day real life usage experiment statistics

6.1.2 First Real Life Usage Experiment

In this experiment two users use the Samsung i600 and HTC s620 as their normal mobile phone while the test software is running and collecting traces from Wifi APs and Bluetooth devices (cell scanning is not yet implemented). The experiment is performed in a very early phase of the testbed development and the main purpose is to test the software and the battery life in real life usage.

Both test users report that the battery lasts at least one day in a normal usage while scanning every 2 minutes. The collected trace files are about 30kb per day, so the 2GB MicroSD is more than enough to collect several days data. Both users report the software crashing even after some iterations that fix few memory handling problems. Even with a quite complete debug log, we were unable to find a single reason for the crashes. For this reason a watcher thread is implemented to the testbed that restarts the scanning process in case it crashes or stops running.

Table 6.2 presents some basic statistics from a four day continuous trace set from both devices. The traces were collected from Wed 16/5/2007 to Sat 19/5/2007. The period includes two normal office days and two days off (17/5/2007 was a bank holiday). The Samsung is out of the office almost the whole period and travelling on the country side. These environmental differences can be noted in the statistics. The Samsung records a larger number of unique devices due to its own mobility. When we look at the contact times, we can see that the Samsung has a lot of Bluetooth contacts and less Wifi contacts since it was out of town. The HTC sees a lot of Wifi contacts which is normal since the city (and office of course) has a lot of APs around. In the table by an active contact we mean that the device was seen more than once. The software version used in this experiment had still some bugs and suffered from several resets during the period. It had to be manually restarted after a crash which means there a varying missing periods of contacts in these data set.

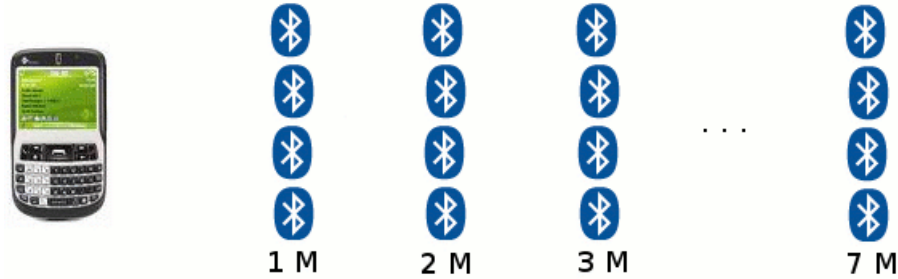


Figure 6.2: Controlled battery life experiment setup

	Samsung i600	HTC s620
Total duration	19705 s (5.5 h)	77336 s (21.5 h)
Scans (succeeded/failed)		
Bt	98/0	404/0
Wifi	1/97	404/0
Average scans/hour	36	38
	HTC Touch	Toshiba
Total duration	66595 s (18.5 h)	14143 s (3.9 h)
Scans (succeeded/failed)		
Bt	373/0	66/0
Wifi	373/0	66/0
Average scans/hour	40	34

Table 6.3: Controlled battery life and sampling experiment statistics

6.1.3 Controlled Battery Life and Bluetooth Sampling Effect Experiment

We do another performance experiment for the devices in a static environment where each device is scanning 802.11 APs and ad hoc nodes and Bluetooth devices every 2 min starting with a full battery until it drains out. The cell tower scanning is left out and the phone radio is disabled on each device for the duration of the experiment. The experiment is performed in a separate room with very little people going in and out. We place 30 iMotes (devices used in the previous experiments at Cambridge and at Thomson) in the room in groups of 4-5 iMotes distributed every one meter (see Figure 6.2). The closest group of iMotes is one meter away from the scanning devices and the furthest away is about 7 meters away. The first two groups contains 5 iMotes and the rest 4 iMotes each. Each iMote is simply answering to the Bluetooth inquiries (not inquiring themselves). During the experiment three of the iMotes fail because of the battery (1 from each group of five iMotes and one from the last group).

Table 6.3 does not include the i-mate, because it fails to initialise the software correctly and this is noticed too late. The Samsung has a problem too with controlling

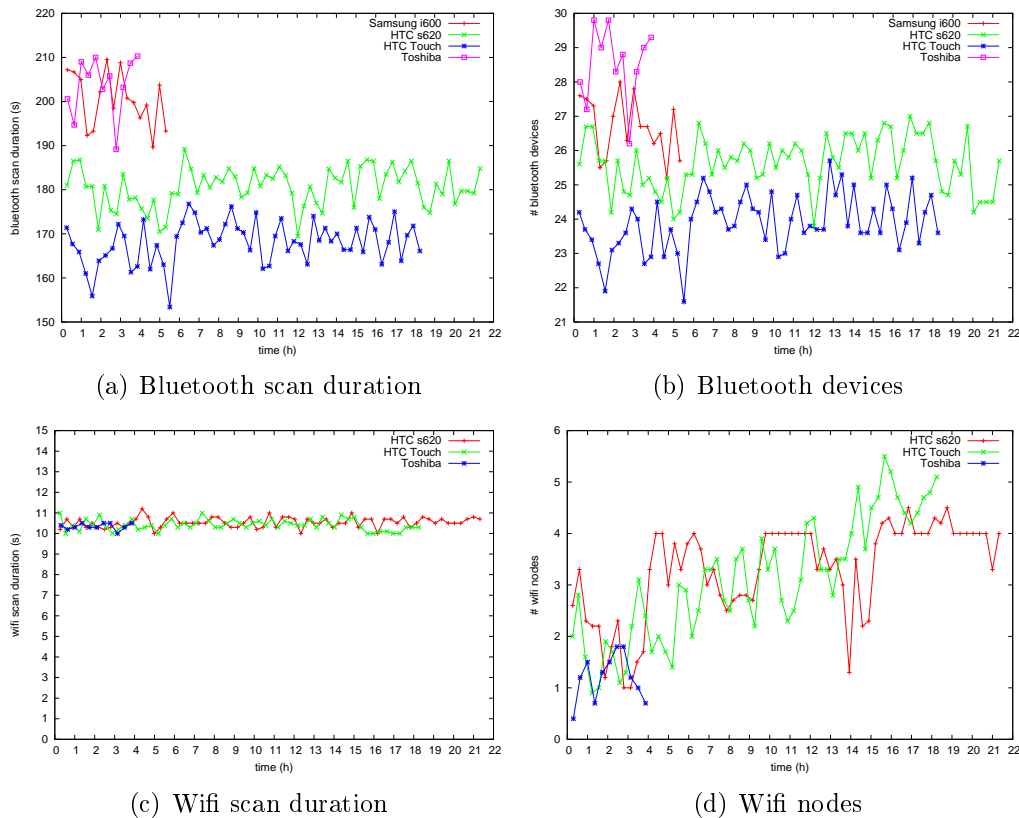


Figure 6.3: Controlled battery life experiment scan results

the Wifi device power, and it fails to perform any Wifi scans after the first one. Despite this, the total durations show again that the Samsung has a poor battery life and the HTC s620 is performing very well. Of the new devices introduced in this experiment, the Toshiba seems to be even worse than the Samsung in terms of the battery life and the HTC Touch is almost as good as the other HTC.

Figure 6.3 shows the overall Bluetooth and Wifi scan results. The Bluetooth scan durations are very variable, and very long, around 3 min in average. The duration can be explained by the fact there are now at least 30 Bluetooth devices active and the scan duration (20 seconds in this experiment) is enough to pick up most of them. The actual scan duration increases above that because we query each device for their friendly name (which would not have been strictly necessary in this experiment). All the test devices seem to find most of the iMotes around, the Toshiba being quite clearly the most efficient. The Wifi scan duration is constant (set in the software), and the number of APs seen varies between the devices. The Toshiba seems to perform less well, discovering occasionally no APs at all (not show in the picture). The Samsung was excluded from the Wifi plots because of the failure.

Table 6.4 shows the average discovery rates of the iMotes by distance from the scanning devices. In the table, 1m means that the motes in that group were roughly

	Samsung i600	HTC s620	HTC Touch	Toshiba
1m	99.0 %	67.8 %	79.8 %	92.4 %
2m	100.0 %	97.5 %	99.1 %	95.8 %
3m	74.2 %	73.3 %	69.3 %	73.9 %
4m	99.5 %	99.3 %	99.2 %	98.1 %
5m	96.9 %	97.5 %	95.6 %	98.5 %
6m	78.1 %	60.6 %	35.8 %	92.8 %
7m	74.1 %	90.0 %	68.9 %	96.5 %

Table 6.4: Imote discovery rate by distance

1m away from the devices and so on. The values do not include the motes that failed during the experiment. As we can see from the table, the distance does not seem to play any role in which devices get discovered and which not. This is expected since the scan duration is long enough to discover all the devices and all of them are within the Bluetooth radio range (10m for Class 2 devices).

6.1.4 Discussion

Based on the experiences we had during the testbed implementation and the performance evaluation presented above, we choose the HTC s620 as the device for the large-scale human mobility experiment. A Smartphone was considered more usable from the user point of view since it resembles more an ordinary cell phone compared to a Pocket PC. Also from the developer point of view, the Smartphone API is somewhat more convenient, at least what comes to the power management. Finally, among the Smartphones we tried, the HTC s620 had a good performance in terms of battery life and radio functionality. It offers also a nice cost (price) vs. functionality compromise by being the the cheapest (as of June 2007) of the devices we test.

6.2 Testbed Deployment

This section presents the first large-scale human mobility measurement experiment conducted with the testbed at the Thomson Paris Research Lab. The experiment starts in the end of July 2007 and ends in the end of August 2007.

6.2.1 Experiment Setup

We acquire 18 new HTC s620 Smartphones (+ the HTC s620 and Touch we have already) for the Thomson Paris Research Lab personnel (researchers, post-docs, PhD students and interns). Each device is equipped with a 2GB MicroSD memory card. The participants are asked to use the device as their personal mobile phone and to run the testbed application that collects traces using all the three wireless interfaces available on the device: Bluetooth, Wifi and cellular radio. The scanning period for

each interface is set to 2 minutes which means that the device battery lasts about 8 hours in normal usage and must be charged daily.

We encounter some problems before the experiment starts to run smoothly. Initially the testbed UI contained still some development and testing screens that caused some confusion for the users. Also some of the device's hot buttons' (like the back button) default functionality was causing problems (jamming the testbed UI). In addition to these UI fixes, we also set the software to start up automatically upon a reboot (requires still a user confirmation to start) so that the user remembers to run it. In the beginning the testbed also seems to suffer from occasional crashes. We fix a small synchronisation bug in the cellular code which helps to most of the crashes and in the final results we observe them less often. For these reasons the first few weeks of the collected data are not so complete. Also, since August is the summer holiday month in France, all the devices are not updated as soon as the fixes are published because the users were out of office.

6.2.2 Initial Results

Collected Logs Duration and Testbed Stability

Of the 20 devices we have, finally 19 are distributed to the lab personnel. During the experiment, 1 device gets stolen, 2 are returned for other reasons and 1 has constantly wrong date and time settings, so the global statistics presented here include a total of 15 devices. However, the excluded devices are counted as internal devices in all the statistics. The experiment runs from 23/7/2007 until 4/9/2007 and during that period we collect 13859 hours (577 days) of traces. The average trace duration (from the first recorded log line to the last one) per device is 923.9 hours (38.5 days). Within that duration each device is 'offline' i.e. not collecting data for a variable amount of time as shown in Figure 6.4. The average offline time is 278.5 hours.

The offline duration of a device relates to the testbed stability but is also an user and usability issue. Some users put a secondary SIM card (or no SIM) to the device which means it is not really in use and the user does not pay so much attention to the battery life etc. A more detailed view to the testbed stability and the user behaviour can be obtained from the number of trace log files and the reasons why the trace log files "end". A new trace log file begins every time the trace application process is started. We collect a total of 493 trace log files of which 23.9% end because the user stops the testbed and 11.4% because the file is automatically rotated. 17.4% of the files seem to end because the device battery dies out (using a threshold of $\leq 3\%$). These are the clear cases. Rest of the time we cannot really tell if the user just kills the process (using the device's task manager) or turns off the device; or if the software actually crashes or gets blocked. We observe that in 6.3% of the cases, a new trace log file (and thus a process) is started quite fast after the previous which is most likely because the watcher thread restarts the application after determining that it is no more running. These cases are most likely crashes. For the remaining 41.0% of the cases, we cannot give a clear reason why the trace file, and thus the whole process,

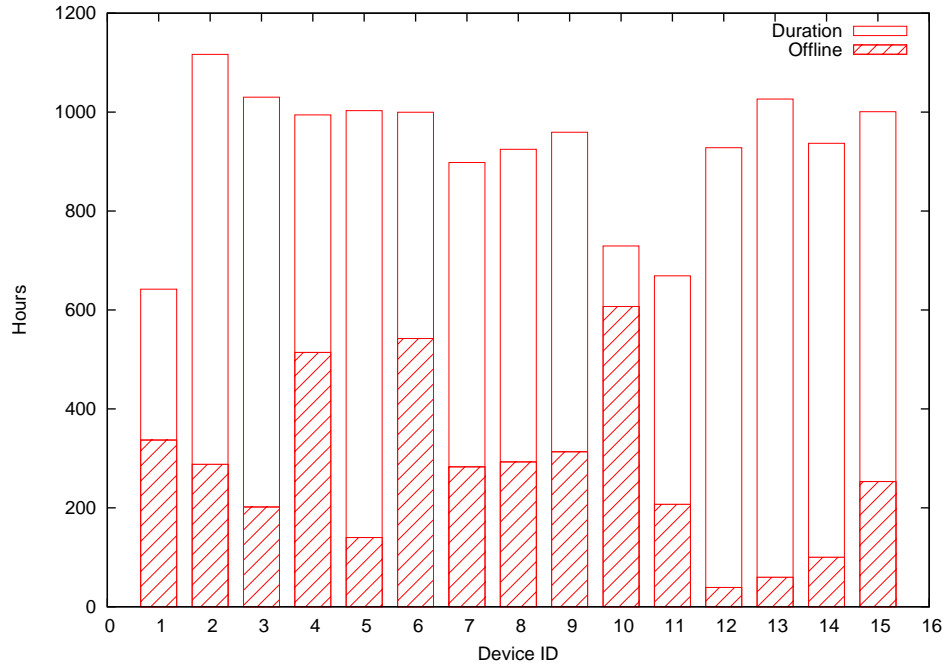


Figure 6.4: Lab experiment trace and offline durations per device

	Total	Internal	External
Contacts	90787	24801 (27.3%)	65986 (72.7%)
Unique devices	34422	20	34402

Table 6.5: Lab experiment Bluetooth contacts and unique devices

ends.

Overall, the number of unclassified cases is large and requires more investigation even though the actual most likely crash cases seem to be reasonably rare. The debug logs hint that the place where the application usually blocks is still within the cellular scanning. Resolving this problem should help to decrease the offline times. Also, based on user experience and received improvement ideas, an even more simpler UI (requiring no user interaction for starting and with a simple icon indicating that the application is running) would decrease the offline times.

Bluetooth Contacts

The total number of Bluetooth contacts and unique devices seen by all the 15 devices are shown in Table 6.5. A contact is defined as a period of time during which a device is seen by the scanning device and they could have changed data if they had wished to. To mitigate the sampling effect (i.e. we miss some devices around due to interference and simultaneous scanning), a contact is considered finished only if the device is not seen during two consecutive scans. The internal contacts present

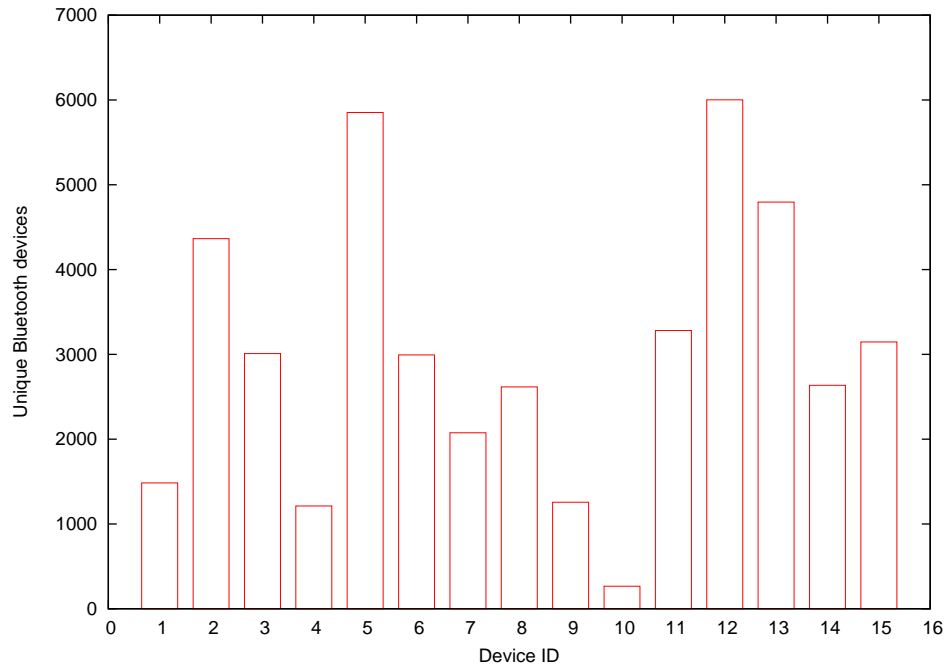
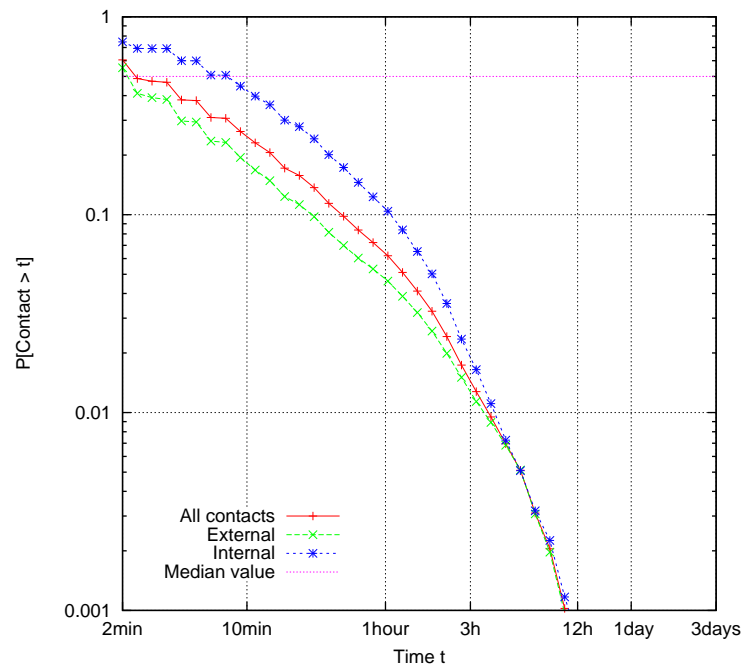


Figure 6.5: Lab experiment unique Bluetooth devices per device

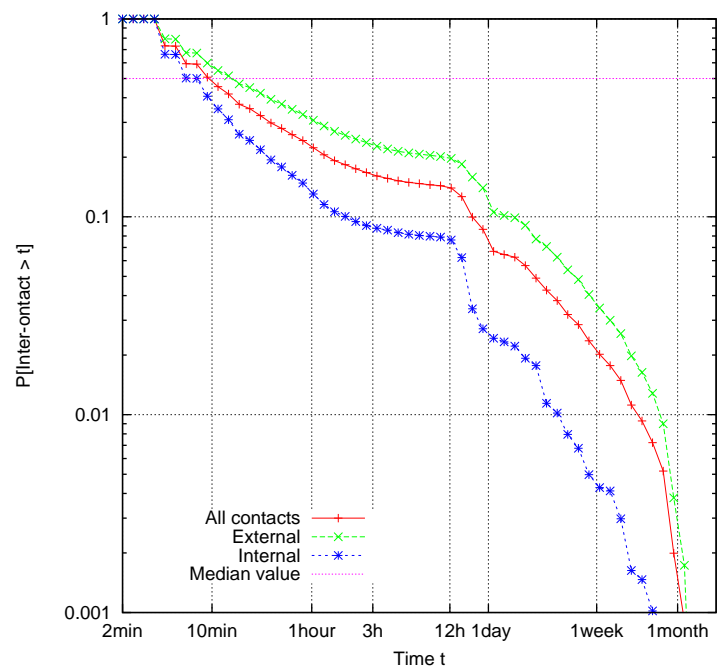
a considerable portion of the total number of contacts considering the number of internal devices compared to the external ones. This is quite obvious result since the internal devices meet at the office more or less every working day while most of the external devices are probably encountered only once since the experiment runs in a large city. Figure 6.5 shows how the the number of unique devices seen is distributed among the participating devices. The differences between participating devices give an idea about the relative node mobility, but are also of course affected by the trace durations and offline time.

Figure 6.6 shows the contact and inter-contact time distributions. Like already explained below, a contact, and thus the contact time measures the possibility to exchange data between devices i.e. the capacity of the network. The median contact time with any node is 2.2 minutes, being considerably larger, 8.0 minutes, between the internal nodes. This is a very promising result considering the community-based ad hoc communications using Bluetooth (we can see our the devices participating in the experiment as a sort of community that meets regularly). An 8-minute contact is already a reasonably long transport opportunity for an ad hoc Bluetooth connection.

Inter-contact time is the interval between two consecutive contacts. It measures the delay to reach a certain device in the network and affects the frequency with which packets can be transferred between two devices. As can be seen from Figure Figure 6.6, the median inter-contact times between the internal devices are around 6.0 minutes which is less than with external nodes. This means that within a community or a close group of nodes the contact opportunities arise more often. We can also see



(a) Contact times



(b) Inter-contact times

Figure 6.6: Lab experiment Bluetooth contact and inter-contact time distributions

	Total	Access Point	Open Access Point	Ad hoc
Contacts	278422	270046 (97.0%)	105941 (38.1%)	8376 (3.0%)
Unique nodes	65191	61679 (94.6%)	26106 (40.0%)	3512 (5.4%)

Table 6.6: Lab experiment Wifi contacts and unique nodes

that the contact and inter-contact time curves have a similar shape as obtained in the previous iMote experiments [49].

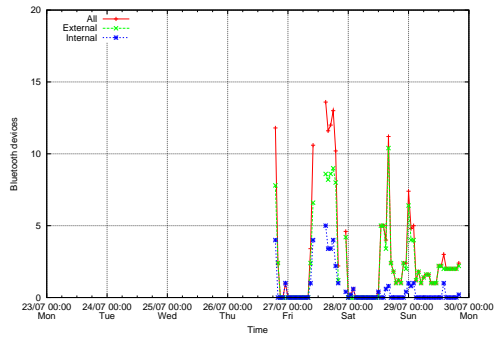
As an example of a single device trace, we show how the number of discovered Bluetooth devices varies over the whole trace period as recorded by the device 12 in Figure 6.7. The graph is drawn by calculating an average number of devices during 10 consecutive scans sampled every hour from the trace logs. On average, the device sees 4.3 devices per inquiry, minimum being 0 and maximum 30 devices. The holes in the contact curves mean that the device was offline. From the timeseries we can observe the normal workday pattern where the number of Bluetooth devices seen peaks usually on the way to the work and back (the person uses the metro in Paris to go to the work) and stays quite high during the day. Also the number of internal devices seen increases once the person is in the office as expected.

Wifi Contacts

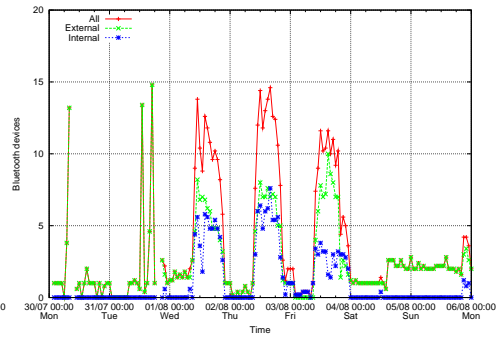
Table 6.6 shows the total number of Wifi contacts and unique nodes. We can see that in general the number of Wifi contacts is larger compared to the Bluetooth contacts. The number of ad hoc contacts is not very important, and most of the ad hoc contacts are actually recorded at the Thomson laboratory where we have few ad hoc test networks and devices running. Figure 6.8 shows how the number of unique Wifi nodes (access points and ad hoc nodes) seen is distributed between the participating devices.

In Figure 6.9 we show the contact and inter-contact time distributions for Wifi nodes. Here we measure the contact times to a Wifi AP or ad hoc node not between the participating devices. We use a skip limit of 2 for calculating the Wifi contacts (i.e. a contact ends when a device was absent during 3 consecutive scans) due to the NDIS API that does not allow us to control the scanning time exactly. This increases the median contact times and decreases the number of contacts but we believe that it is more representative of the actual contact times since the Wifi ranges are typically larger than that of the Bluetooth and the contacts should in general last longer. We might of course lose some fine-grained mobility information by increasing the skip-limit.

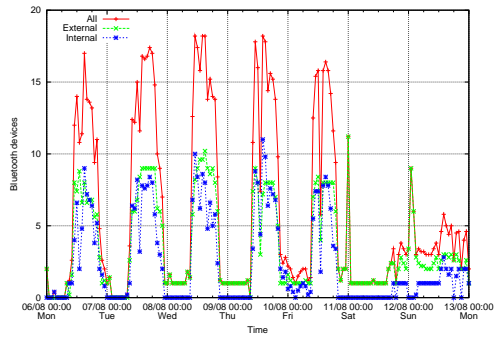
Figure 6.10 shows an example Wifi trace from the device 12. The average number of Wifi nodes found per scan is 7.4, varying from 0 to a maximum of 32 nodes per scan. As with Bluetooth, the timeseries shows a quite clear weekly pattern of being at work or at home. The timeseries reveals also that in an urban setting Wifi connectivity starts to be fairly ubiquitous in terms of visible access points. From the opportunistic communications point of view, it is also encouraging that around 40% of the contacts



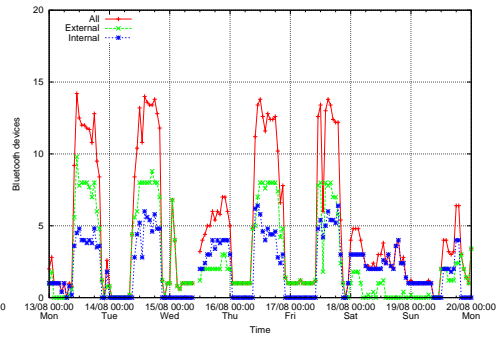
(a) Week 1



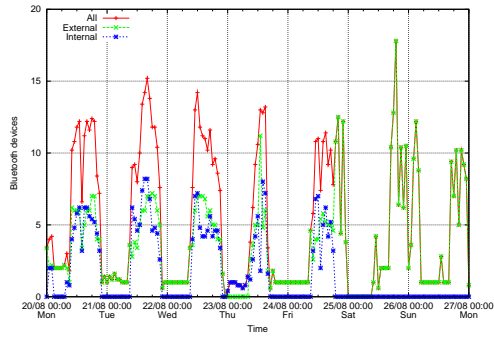
(b) Week 2



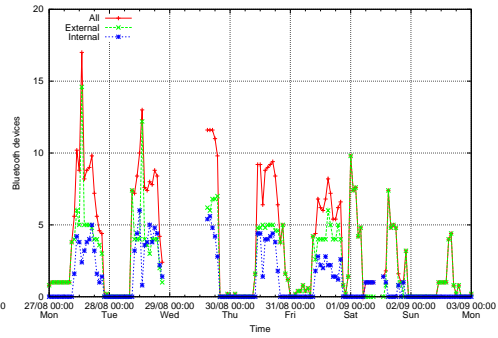
(c) Week 3



(d) Week 4

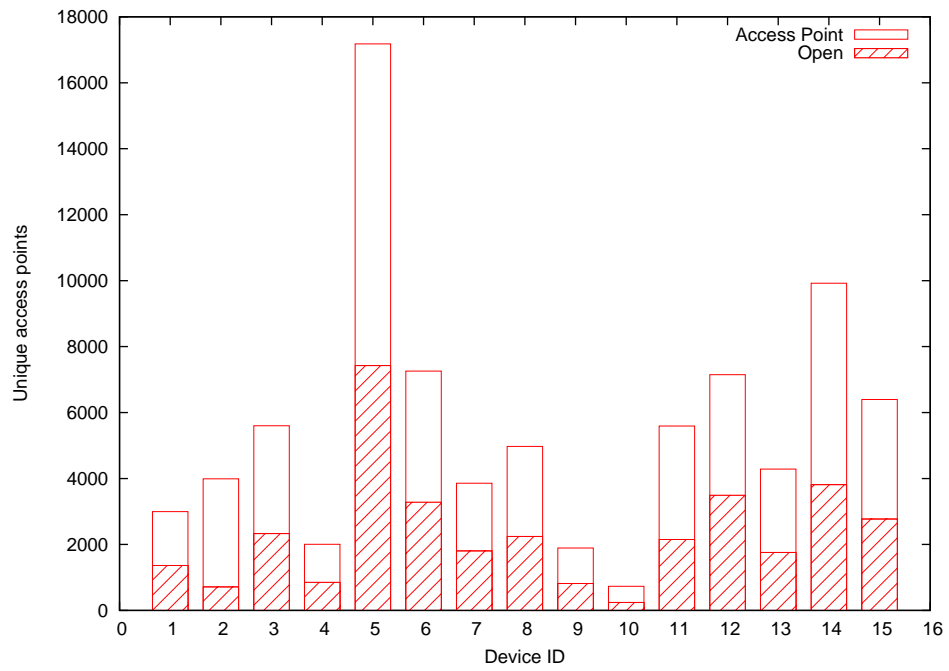


(e) Week 5

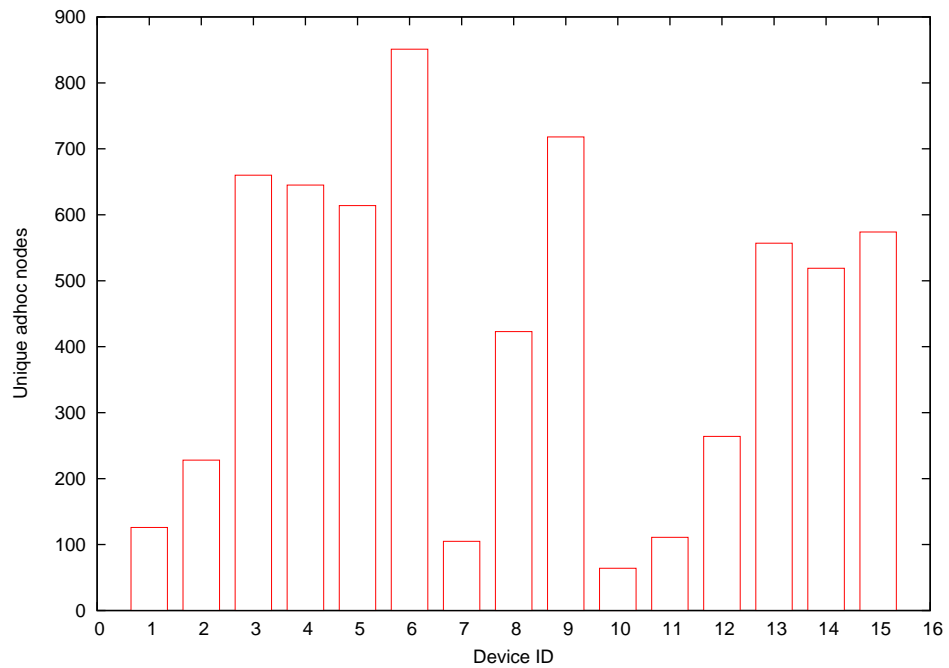


(f) Week 6

Figure 6.7: Lab experiment discovered Bluetooth device trace from device 12

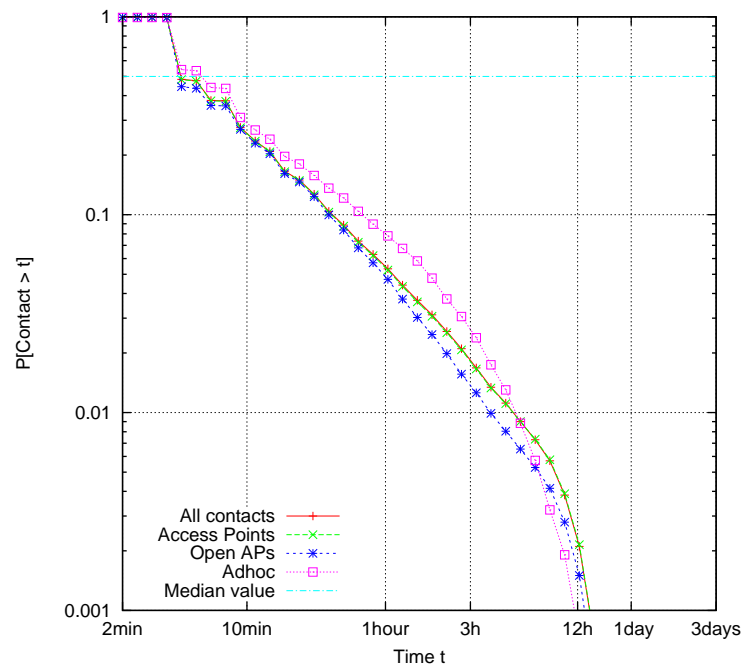


(a) Infrastructure

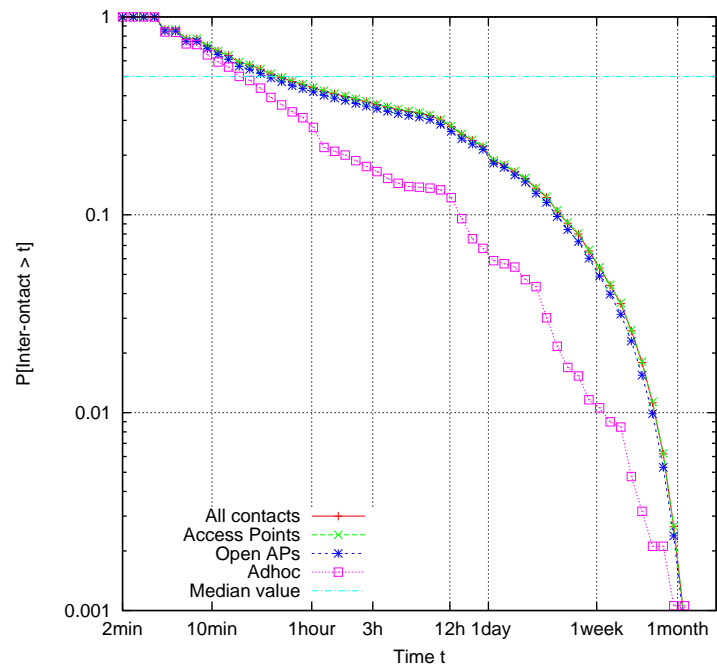


(b) Ad hoc

Figure 6.8: Lab experiment unique Wifi nodes per device

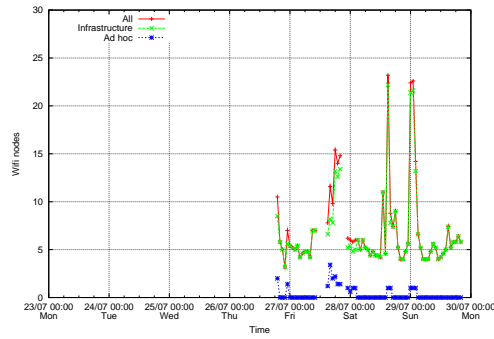


(a) Contact times

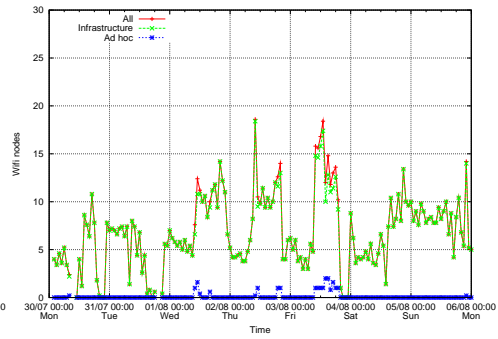


(b) Inter-contact times

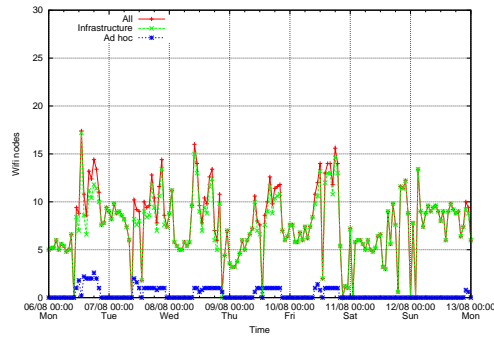
Figure 6.9: Lab experiment Wifi contact and inter-contact time distributions



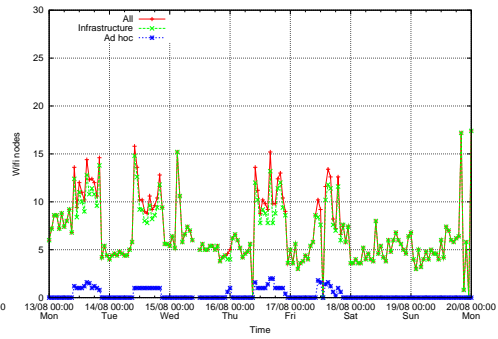
(a) Week 1



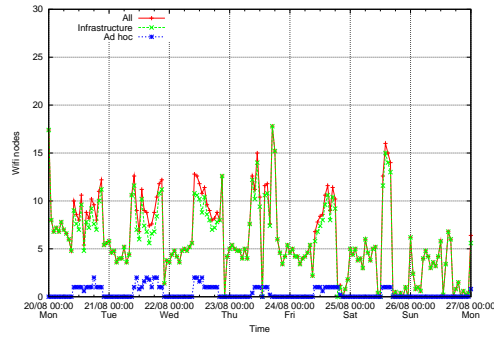
(b) Week 2



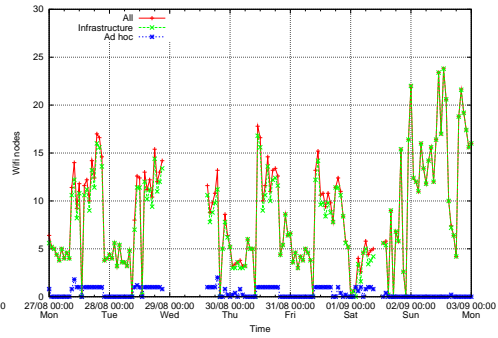
(c) Week 3



(d) Week 4



(e) Week 5



(f) Week 6

Figure 6.10: Lab experiment Wifi contacts trace from device 12

occur with open access points i.e. they could be used for transmitting data. We cannot of course tell from these results if the open access points seen are actually available (they could be using mac filtering for example) or if they are connected to the Internet.

Cellular Contacts

Figure 6.11 shows the number of unique operators and cells encountered per device (device number 11 was not recording cellular data). In France the number of available operators is three and we can see that three of the devices do not go close to any other country, while the rest eleven, and the number 2 in particular, do pass occasionally abroad or at least close to the border where they are able to capture foreign operators' networks.

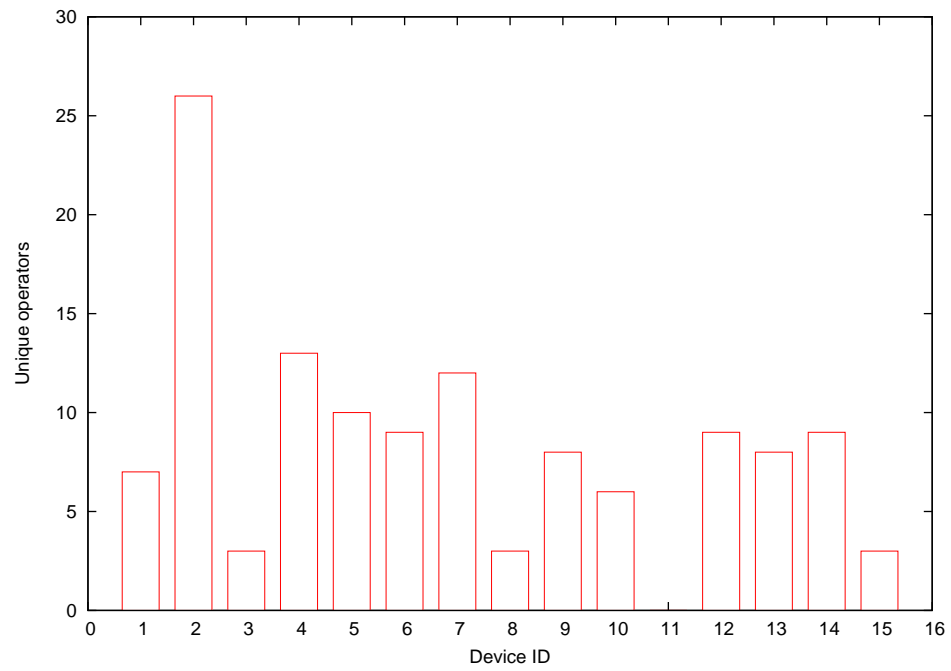
Figure 6.12 shows the contact and inter-contact times with recorded GSM cells. The contact-time median is much over 10 minutes which is due to the large cell size (compared to Bluetooth and Wifi radio ranges). We can remark the same about the inter-contact times.

6.2.3 Discussion

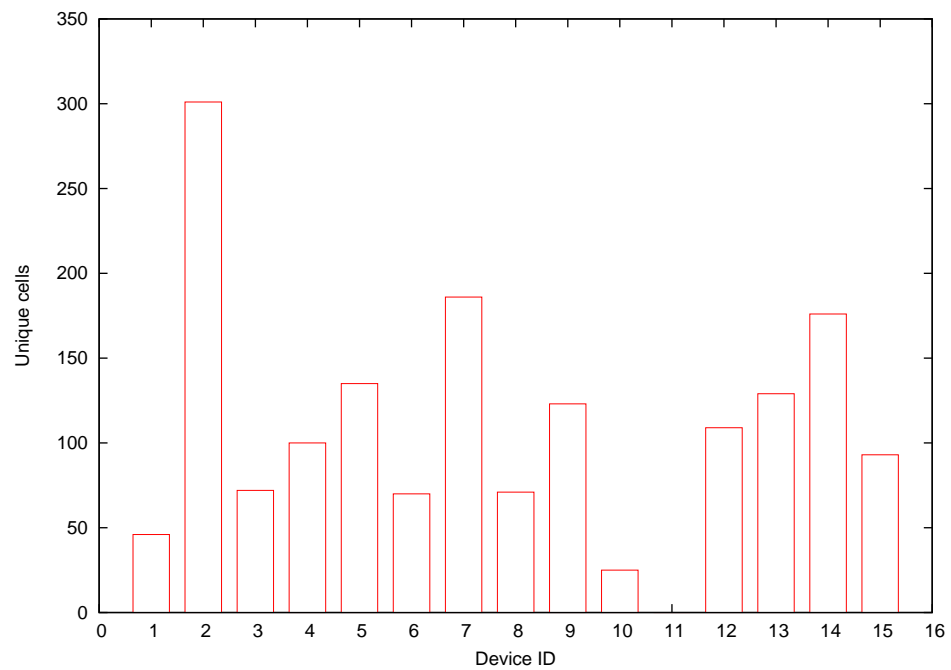
During the first large-scale experiment we collect an unique, over one month long human mobility trace using 15 Smartphones. The data set includes contacts with other Bluetooth devices; Wifi access points and ad hoc nodes; and GSM cells and operators. Below some miscellaneous remarks of the collected trace and future improvements and work suggestions.

The offline times present still a quite considerable proportion of time for some of the participating devices and those traces should most likely be removed from the data set before analysing the data further or using the traces in any simulations. Some reasons for the offline times and improvement ideas to decrease the offline time were discussed already above, but to summarise, looks like the UI should still be simplified and the tracing should happen completely automatically. The one-day battery life seems to be quite sufficient for all the users even though we still observe a fair amount of "dead batteries" during the experiment. The battery life could be further enhanced by tuning the scanning intervals. For example scanning Wifi nodes for every 2 minutes might be a bit too often knowing the Wifi radio range and the human walking speed. Of course increasing the interval will result in missing some contacts but that is the basic trade-off to make.

The closest available public trace from a similar setup is the CoSphere trial [6]. The published statistics from their data set are not very comprehensive and use different representations that we do, but in terms of unique nodes/devices/operators seen our data set seems to contain much more information. This can be partly explained by the longer scanning intervals used in the Cosphere trial compared to the 2-minute interval in our setup. One notable difference is in the number of unique cells encountered. Basically we record them less. This requires closer look but may indicate that either

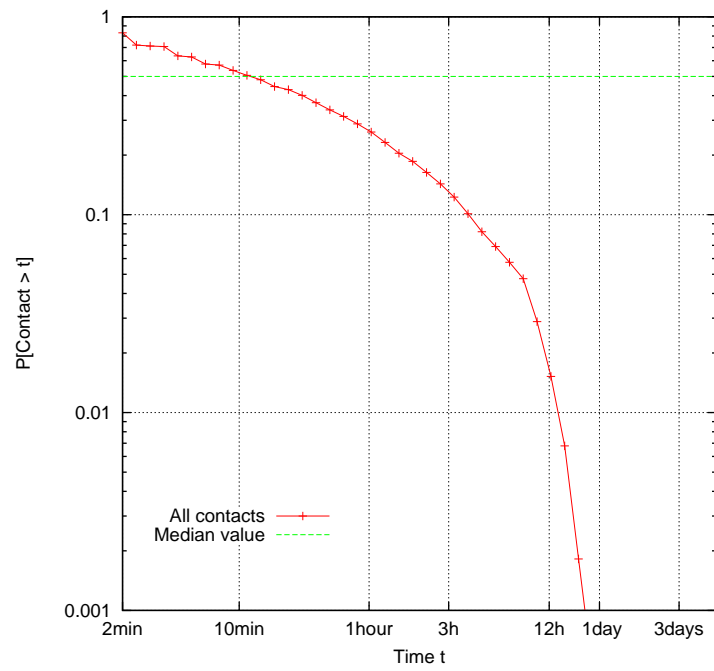


(a) Operators

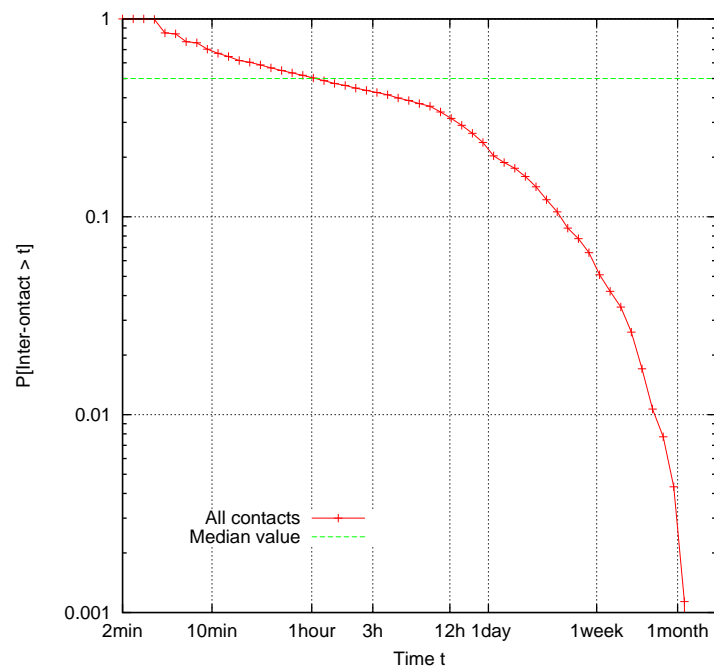


(b) Cells

Figure 6.11: Lab experiment unique operators and cells per device



(a) Contact times



(b) Inter-contact times

Figure 6.12: Lab experiment cell contact and inter-contact time distributions

the RIL API does not work as expected or there is something wrong still in the way we collect the data.

From the collected traces we note that the Bluetooth scan durations are quite variable from about 20 seconds to almost 3 minutes or more (average being around 30 seconds). This is due to the friendly name discovery. The friendly name discovery is a bit redundant information, and for the next experiments it could be turned off. Also based on our experiments with the Bluetooth scan durations and number of discovered devices, the default discovery length (around 20 seconds) seems to be unnecessary long and could be decreased. This would help in saving some battery too. Finally, the maximum number of Bluetooth devices discovered per scan is currently 30 due to a programmed limit. This should be increased in order to not miss devices in crowded environments.

To further improve the collected data set, it would be interesting to add information about the currently connected Wifi network and access point. This information is now excluded but could provide more insight into the mobility and connectivity and would help to better analyse the Wifi scanning results we get now.

This section highlighted only some initial statistics from the data set collected during the first experiment. Future work includes a more detailed analysis of the traces including for example mobility pattern and community detection and analysis. Another interesting research question would be the correlation between the devices/nodes/cells seen and how that information could be used for predicting future contact opportunities for example. It would also be interesting to look into the actual contacts between the devices over Wifi (i.e. when two devices share a common access point that could be considered as a contact opportunity). The collected data set can also be used for trace driven simulations for example to analyse forwarding algorithms.

Chapter 7

Conclusion

In this chapter we provide a brief summary of this report by emphasising our contributions and conclusions drawn during the work. Finally we discuss some possible future work directions.

7.1 Summary

The report started by presenting the motivation and background to this thesis work and by setting the scope for our work. The main goals were the design and implementation of a networking library for PSN and a testbed using that library for measuring human mobility. We also aimed in performing a first human mobility measurement experiment with the testbed to evaluate its performance and to collect a new mobility data set.

In the related work chapter we discussed the existing work on opportunistic mobile communications including the DTN and Huggle architectures and gave an overview to the forwarding algorithms for PSN. We also reviewed previous work on human mobility measurements. The background was concluded by a brief technology review on 802.11, Bluetooth and GSM standards.

A part of the time during this thesis work was devoted on experimenting with development platforms and different devices based on Windows Mobile operating system. This was done in order to select a suitable platform and a device for the actual networking library and the testbed implementations and for performing experiments with them. Thus, we overviewed the Java technologies for mobile devices and the native Windows Mobile development tools and APIs. After having tried several available JVMs on Windows Mobile Smartphone and PocketPC devices we found out that none of them really provides the required functionality for our purposes i.e. an access to the low-level networking functions to enable autonomous communications. We choose to use the native code and Windows Mobile APIs to implement the library and the testbed.

We presented the networking library and the testbed requirements and implemented functionality in detail together with our design. We also provided a comprehensive discussion on our experiences with the Windows Mobile APIs while developing

our code. In practise many of the low-level networking features we use are not very well documented, or simply, there is no documentation nor working examples available at all due to the closed nature of the Windows Mobile OS and the pioneering nature of our work. Also since each OEM has some freedom in implementing some of the interfaces and functionality on WM, we encountered several problems in making our code work seamlessly on all the devices (at least on the ones we had access to). In the end we believe that our implementation works on any Windows Mobile 5.0 or 6.0 device making it thus possible to preform very large-scale experiments with any existing (Windows Mobile) devices.

Finally, we explained how we evaluated and selected one device for the larger-scale human mobility experiment. Then we presented our initial human mobility measurement experiment setup, and analysed the testbed performance and the initial mobility statistics collected during the experiment. We collected a considerable data set from 15 Smartphones used by the Thomson Paris research laboratory personnel during August 2007. The data set contains around 577 days of logs including the discovered Bluetooth devices, surrounding Wifi access points and ad hoc nodes, and the available operators and the current cell id recorded every 2 minutes. This data set is quite unique by covering a such a long period and data from several networking interfaces. Also the use of real devices in a real-life environment distinguishes us from many of the previous human mobility experiments, like the iMote experiments in conference environments.

7.2 Future Work

The PSN networking library and the testbed development is still a work in progress and some practical improvements ideas and related future work was already outlined in the end of Chapter 5.

Similarly, the first human mobility experiment presented in this report will be continued, and in addition new experiments are planned to be conducted in up-coming conferences before the end of the year. Complete long-lasting traces from variable environments are valuable for the whole research community so we plan also to publish our traces in an online community data repository such as CRAWDAD [7].

The device evaluation presented in this report and the initial data transmission experiments we have been doing give already some hints about the feasibility of opportunistic communications with mobile devices, and with Windows Mobile based devices in particular. We are still continuing the work and hope to publish a detailed article on the topic in the near future. Also the PSN architecture development for mobile devices is an on going work. The library code we developed and the experiences learned while doing it are used by the Huggle architecture developers currently working on their mobile prototype.

Appendix A

Test Devices

Throughout the project we used and evaluated the following Windows Mobile based devices: i-mate JAQ3 [14], Samsung i600 [22], HTC s620 [12], Toshiba Portégé G500 [25] and HTC Touch [13]. Table A.1 presents a summary of the key technical characteristics of each device, and all the devices are pictured in Figure A.1. In terms of features they are very similar. The biggest differences come from the Pocket PC and Smartphone editions of the Windows Mobile OS. The i-mate JAQ3 and the HTC Touch are, with a touch screen, more like a PDAs and they are running the WM Pocket PC edition. The Samsung i600, the HTC s620 and the Toshiba G500, are WM Smartphones. The Samsung and the Toshiba are the only 3G phones among the devices we test. The prices listed in the table are the prices advertised in the Internet for France in June 2007.

	i-mate JAQ3	Samsung i600	HTC s620
Processor	TI OMAP 850 200Mhz	Intel PXA272 416Mhz	TI OMAP 850 200Mhz
RAM	64MB	64MB	64MB
ROM	128MB	128MB	128MB
Extension	MicroSD	MicroSD	MicroSD
Battery	1200mAh	1200mAh	960mAh
Platform	WM5.0 Pocket PC	WM5.0 Smart- phone	WM5.0 Smart- phone
Radio	GSM/GPRS/EDGE 850/900/1800/1900	GSM/GPRS/EDGE UMTS/HSDPA 900/1800/1900	GSM/GPRS/EDGE 850/900/1800/1900
Wifi	802.11b/g	802.11b/g	802.11b/g
Bluetooth	V1.2	V2.0	V1.2
Price	490e	420e	370e
	Toshiba G500	HTC Touch	
Processor	Intel Xscale PXA270 312Mhz	TI OMAP 850 200Mhz	
RAM	64MB	64MB	
ROM	64MB	128MB	
Extension	MiniSD	MicroSD	
Battery	1100mAh	1100mAh	
Platform	WM5.0 Smart- phone	WM6.0 Profes- sional	
Radio	GSM/GPRS/EDGE UMTS/HSDPA 900/1800/1900	GSM/GPRS/EDGE 900/1800/1900	
Wifi	802.11b/g	802.11b/g	
Bluetooth	V2.0	V2.0	
Price	400e	450e	

Table A.1: Technical details of the test devices



(a) i-mate JAQ3



(b) Samsung i600



(c) HTC s620



(d) Toshiba G500



(e) HTC Touch

Figure A.1: Test devices

Bibliography

- [1] 3gpp home page. <http://www.3gpp.org/>.
- [2] Accessing native methods from a midlet. <http://forum.java.sun.com/thread.jspa?threadID=617976>.
- [3] Avetana jsr-82 implementation. <http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml>.
- [4] bluecove. <http://code.google.com/p/bluecove/>.
- [5] Bluetooth special interest group (sig). <http://www.bluetooth.org>.
- [6] Communication context for adaptive mobile applications. <http://cosphere.telin.nl/index.html>.
- [7] A community resource for archiving wireless data at dartmouth. <http://crawdad.cs.dartmouth.edu/>.
- [8] Delay-tolerant networking at tkk netlab. <http://www.netlab.hut.fi/~jo/dtn/index.html>.
- [9] Delay tolerant networking research group. <http://www.dtnrg.org>.
- [10] Esmertec mobile multimedia solutions. http://www.esmertec.com/solutions/mobile_multimedia/.
- [11] Huggle. <http://www.huggleproject.net>.
- [12] Htc s620 product page. <http://www.europe.htc.com/products/htcs620.html>.
- [13] Htc touch product page. <http://www.europe.htc.com/products/htctouch.html>.
- [14] i-mate jaq3 product page. http://www.clubimate.com/t-DETAILS_JAQ3.aspx.
- [15] Ieee 802.11 the working group for wlan standards. <http://www.ieee802.org/11/>.
- [16] The java me platform. <http://java.sun.com/javame/index.jsp>.

- [17] Microsoft windows mobile home. <http://www.microsoft.com/windowsmobile/default.aspx>.
- [18] Niscom creme. <http://www.nsicom.com/Default.aspx?tabid=138>.
- [19] phoneme project. <https://phoneme.dev.java.net/>.
- [20] Power to the people. <http://blogs.msdn.com/windowsmobile/archive/2005/08/01/446240.aspx>.
- [21] Power to the system. <http://blogs.msdn.com/windowsmobile/archive/2005/08/10/450186.aspx>.
- [22] Samsung sgh-i600 product page. http://www.samsung.com/sg/products/gsm/gsm/sgh_i600.asp.
- [23] Test tcp (ttcp) benchmarking tool for measuring tcp and udp performance. <http://www.pcausa.com/Utilities/pcattcp.htm>.
- [24] Tetherless computing. <http://blizzard.cs.uwaterloo.ca/tetherless/>.
- [25] Toshiba. <http://www.toshiba-europe.com>.
- [26] Websphere everyplace micro environment. <http://www-306.ibm.com/software/wireless/weme/>.
- [27] Wikipedia: Gprs core network. http://en.wikipedia.org/wiki/GPRS_Core_Network.
- [28] Wikipedia: Gsm. <http://en.wikipedia.org/wiki/GSM>.
- [29] Wikipedia: Windows mobile. http://en.wikipedia.org/wiki/Windows_Mobile.
- [30] Windows ce 5.0 network drivers. <http://msdn2.microsoft.com/en-us/library/ms892542.aspx>.
- [31] Windows ce 5.0 power management. <http://msdn2.microsoft.com/en-us/library/aa447554.aspx>.
- [32] Windows ce 5.0 radio interface layer. <http://msdn2.microsoft.com/en-us/library/ms890075.aspx>.
- [33] Windows mobile 5.0 application security. <http://msdn2.microsoft.com/en-us/library/ms839681.aspx>.
- [34] Windows mobile 5.0 communications services and networking. <http://msdn2.microsoft.com/en-us/library/ms880996.aspx>.

- [35] Windows mobile 5.0 sdk automatic configuration reference. <http://msdn2.microsoft.com/en-us/library/aa448301.aspx>.
- [36] Windows mobile 5.0 sdk bluetooth. <http://msdn2.microsoft.com/en-us/library/ms847080.aspx>.
- [37] Windows mobile 5.0 sdk ip helper api. <http://msdn2.microsoft.com/en-us/library/ms850369.aspx>.
- [38] Windows mobile 6.0 sdk radio interface layer. <http://msdn2.microsoft.com/en-us/library/aa920475.aspx>.
- [39] Zdnet news: Global mobile phone use to pass record 3 billion. http://news.zdnet.com/2100-1035_22-6193559.html.
- [40] 3GPP. *3GPP TS 22.042 Network Identity and Time Zone (NITZ) service description; Stage 1*, October 1999.
- [41] 3GPP. *3GPP TS 23,002 Network Architecture (Release 1999)*, October 2006.
- [42] 3GPP. *3GPP TS 23,003 Numbering, addressing and identification (Release 1999)*, October 2006.
- [43] A. Balasubramanian, B. Levine, and A. Venkataramani. Dtn routing as a resource allocation problem. In *Proceedings of ACM SIGCOMM*, 2007.
- [44] Bluetooth Special Interest Group (SIG). *RFCOMM with TS 07.10 Serial Port Emulation*, June 2003.
- [45] Bluetooth Special Interest Group (SIG). *Specification of the Bluetooth System. Core Package version: 1.2*, November 2003.
- [46] Bluetooth Special Interest Group (SIG). *Specification of the Bluetooth System. Core Package version: 2.0*, November 2004.
- [47] G. Brasche and B. Walke. Concepts, services, and protocols of the new gsm phase 2+ general packet radio service. *IEEE Communications Magazine*, pages 94 – 104, August 1997.
- [48] B. Burns, O. Brock, and B.N. Levine. MV routing and capacity building in disruption tolerant networks. 2005.
- [49] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Pocket switched networks: Real-world mobility and its consequences for opportunistic forwarding. Technical report, University of Cambridge, Computer Laboratory, 2005.
- [50] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on the design of opportunistic forwarding. In *Infocom*, 2006.

- [51] James A. Davis, Andrew H. Fagg, and Brian N. Levine. Wearable computers as packet transport mechanisms in highly- partitioned ad-hoc networks. In *ISWC '01: Proceedings of the 5th IEEE International Symposium on Wearable Computers*, page 141, 2001.
- [52] N. Eagle and A. Pentland. Reality mining: Sensing complex social systems. *Journal of Personal and Ubiquitous Computing*, 2005.
- [53] ETSI. *igital cellular telecommunications system (Phase 2+); Subscriber Identity Modules (SIM); Functional characteristics (GSM 02.17 version 8.0.0 Release 1999)*, November 1999.
- [54] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.
- [55] M. Grossglauser and D. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, 2002.
- [56] Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 187–201, 2004.
- [57] W.J. Hengeveld. Misc notes on the cds and windows ce. <http://www.xs4all.nl/~itsme/projects/xda/>.
- [58] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and the consequences of human mobility in conference environments. In *Proceedings of ACM SIGCOMM first workshop on delay tolerant networking and related topics*, 2005.
- [59] P. Hui and J. Crowcroft. How small labels create big improvements. In *PerComW'07: Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 65–70, 2007.
- [60] IEEE Standards Board. *ANSI/IEEE Std 802.11, 1999 Edition (R2003), Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, June 2003.
- [61] IEEE Standards Board. *IEEE Std 802.11b-1999 (R2003), Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, June 2003.
- [62] IEEE Standards Board. *IEEE Std 802.11g-2003, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band*, June 2003.

- [63] Burgess J., Gallagher B., Jensen D., and Levine B. N. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *INFOCOMM '06: Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–11, 2006.
- [64] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, 2004.
- [65] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3):19–20, 2003.
- [66] Motorola and Sun Microsystems. *Java APIs for Bluetooth Wireless Technology (JSR 82)*, September 2005.
- [67] Mirco Musolesi and Cecilia Mascolo. A community based mobility model for ad hoc network research. In *REALMAN '06: Proceedings of the second international workshop on Multi-hop ad hoc networks: from theory to reality*, pages 31–38, 2006.
- [68] T. Narten, E. Nordmark, and W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461 (Draft Standard), December 1998. Updated by RFC 4311.
- [69] Eric Nordström, Christophe Diot, Richard Grass, and Per Gunningberg. Experiences from measuring human mobility using bluetooth inquiring devices. In *MobiEval '07: System Evaluation for Mobile Platforms*, 2007.
- [70] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.
- [71] M. Rahnema. *IEEE Communications Magazine*, pages 92 – 100, April 1993.
- [72] James Scott, Pan Hui, Jon Crowcroft, and Christophe Diot. Hagggle: a networking architecture designed around mobile users. In *Proceedings of The Third IFIP WONS Conference*, 2006.
- [73] A. Seth, P. Darragh, S. Liang, Y. Lin, and S. Keshav. *An Architecture for Tetherless Communication*, July 2005.
- [74] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay- tolerant networking*, pages 252–259, 2005.
- [75] Sun Microsystems. *JNI 1.1 Specification*, 2003. <http://java.sun.com/j2se/1.4.2/docs/guide/jni/index.html>.

- [76] Sun Microsystems. *JSR 139: Connected Limited Device Configuration 1.1*, March 2003.
- [77] Sun Microsystems. *JSR 218: Connected Device Configuration (CDC) 1.1*, August 2005.
- [78] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, UCSD, 2000.