

# Connectivity Management for Opportunistic Communications in PSN

Anna-Kaisa Pietilainen, Christophe Diot

Thomson, Paris, France.

email: [firstname.lastname@thomson.net](mailto:firstname.lastname@thomson.net)

Thomson Technical Report  
Number: **CR-PRL-2008-03-0001**  
Date: March 14th 2008

**Abstract:** Opportunistic communications through Pocket Switched Networks (PSN) is a communication paradigm which aims to exploit contact opportunities between mobile devices and human mobility to transfer data in a peer-to-peer fashion. Building real PSN systems is challenging: mobile devices have resource limitations, network interfaces and protocols do not readily support ad hoc communications, and finally, the mobile operating systems and development platforms offer insufficient or limited support of the required functionality. In this paper we design and implement connectivity management software to support opportunistic communications in PSN using Windows Mobile devices. We motivate our design choices and evaluate the connectivity manager with laboratory benchmarks and three real-life experiments. We get insights into how to tune the system parameters in order to maximize the use of contact opportunities under various resource constraints. We show that opportunistic networking between mobile devices is feasible and offers interesting possibilities for new communication services. However, the operating system functionality and wireless network technologies are limiting the number of devices we can contact opportunistically, and consequently, the capacity of Pocket Switched Networks.

## 1. INTRODUCTION

In Pocket Switched Networks, data transmission takes advantage of the communication opportunities that occur when mobile devices “meet”, as well as of the device’s user mobility [19]. PSNs are an attractive communication model when infrastructure connectivity is intermittent or when its usage could be undesirable because of price or privacy issues. In addition to partial connectivity, PSNs are characterized by asymmetric data rates and long delays. The internet protocol suite performs poorly in such environment. To overcome these challenges, a new communication architecture is needed.

Prior work on PSNs has focused on understanding the impact of human mobility on the performance of forwarding algorithms [4]. Human mobility has been studied using WiFi enabled laptops and computers [8], cell phones [5] and iMotes [9] (a small Bluetooth sensor running on an ARM processor with limited battery and memory). Most of the data sets are available on *Crawdad*<sup>1</sup>.

While all these data sets have improved our understanding of human mobility, they have also several limitations that make them very specific to the experimentation environment or difficult to reproduce: measuring WiFi contacts using laptops is a coarse measure for human mobility; experimental code for cell phones is usually very device specific and hard to port even on devices that use the same operating system; and finally, the iMotes have very limited resources and are unsuitable for application development.

The first objective of this work is therefore to design communication software for large scale mobility measurement and PSN application prototyping. This software should be easily deployable on a large number of devices and support multiple wireless interfaces. To reach these goals, we have chosen to develop our PSN connectivity manager on Windows Mobile smartphones. Windows Mobile is probably the most popular and the most “open” general operating system for smartphones. We identify smartphones as the most appropriate platform for prototyping PSN applications; smartphones have good storage and CPU capacity and even the battery life is reasonable. It is becoming also more common to have several networking interfaces on a single device including cellular, WiFi (802.11) and Bluetooth. Smartphones will make running large-scale experiments possible as the software can be installed on many compatible devices.

PSNs are a special case of Delay Tolerant Networks [6]. However, while being tolerant of delays in general, we want to avoid introducing any unnecessary additional delays in finding and using contact opportunities. The

performance of PSNs (network capacity and message delays) and even the overall usefulness of opportunistic networking depends on how we manage connectivity, a fact that has been often ignored in prior work. Thus the second goal of our work is to understand and characterize the system parameters and resource constraints affecting contact discovery, link establishment, capacity and delays over various radio technologies.

The contributions of this paper are the following. We overview the available mobile computing platforms in Section 2 motivating our choice to use Windows Mobile smartphones. We discuss the design and implementation of the connectivity management software and propose an API for the application development in Section 3. We provide a performance and feasibility evaluation using laboratory benchmarks (Sections 4) and real-life experiments (Section 5). Related work is discussed in Section 6 followed by our conclusions (Section 7). We plan to make the connectivity software publicly available in order to allow other research groups to collect data, test forwarding algorithms and develop their own applications. We hope that this work will help boost the PSN research and contribute to the development of a community of PSN users.

## 2. MOBILE COMPUTING PLATFORMS

The selection of a suitable experimentation device, platform and development environment must be considered carefully. We want to avoid unnecessary resource limitations due to hardware or software choices. Secondly, we need to be able to access the (low-level) networking functionality on the device for opportunistic communications. Finally, we would like to work on a platform that is popular and allows to build portable software in order to promote the creation of a real PSN user community.

### 2.1 Mobile Devices

Mobile devices vary from cheap low-end cell phones to sophisticated smartphones and PDAs. While the low-end phones are still too restricted in terms of resources and openness for development<sup>2</sup>, smartphones provide an interesting research platform. A typical smartphone today is running an ARM processor based on 32-bit RISC design. The processor speeds vary from 100 - 500 MHz. Typically the amount of RAM is 64-128MB, and the permanent storage (ROM) is an EEPROM flash memory of 128-256MB. In addition, most devices have a slot for an external memory card such as microSD that can offer up to 8GB (SDHC) of storage. The devices are powered by lithium-ion batteries of about 1000mAh or more, with a life-time depending from the manufacturer and the embedded radio interfaces.

<sup>2</sup>Most of them run proprietary operating systems and offer only a limited Java support.

<sup>1</sup><http://crawdad.cs.dartmouth.edu/>

The networking interfaces available for mobile devices include cellular radio (GSM/UMTS), WiFi and Bluetooth. These technologies have a variable set of capabilities making them complementary rather than exclusive of each other. The cellular radio provides a long-range, nearly ubiquitous network service. GPRS (the packet data service for GSM) offers a theoretical user rate of 22.8kbps, and its enhancement, EDGE, increases the rates up to 64kbps [7]. The currently deployed 3G system, UMTS, offers high speed data access with HSDPA. While the maximum supported rates of HSDPA go up to 10Mbps, available devices today advertise generally 384kbps for upload and 3.6Mbps for download.

WiFi is a wireless local area network (WLAN) standard operating on the license-free ISM frequency band at 2.4GHz. The commonly implemented versions on mobile devices are either 802.11b and/or 802.11g. The former has a maximum rate of 11Mbit/s, while the latter increases the maximum rate to 54Mbit/s. The communication range of 802.11 varies from about 35 to 140 meters (indoors and outdoors respectively).

Bluetooth is a low-power short-range wireless communication technology intended to replace the cable(s) connecting electronic devices. Bluetooth operates on the same license-free ISM band at 2.4GHz as WiFi. The nominal rate for Bluetooth v1.1 and v1.2 is 1Mbit/s. Bluetooth v2.0 introduces Enhanced Data Rate (EDR) which increases the rate up to 2 or 3Mbit/s. The operational ranges of Bluetooth devices vary from 1, 10 to 100 meters (class 3, class 2 and class 1 respectively). Mobile devices are generally class 2 devices, and mainly ship with either a Bluetooth 1.2 or 2.0 radio.

The cellular interface is not very interesting from the ad hoc communications point of view because of the unavoidable infrastructure dependency and the usually high price of data transmissions. At the same time, both Bluetooth and WiFi can be used for ad hoc device-to-device communications without any monetary cost. The biggest differentiators between these two technologies are the radio range, energy consumption and data transmission rates. Even if the cellular radio is not designed for ad hoc communications, it could still be considered, e.g., as a backup or control channel for delivery acknowledgments.

## 2.2 Development Environments

The smartphone operating system market is mainly shared between Windows Mobile, Symbian, BlackBerry, Linux, Palm and the mobile version of Mac OS X. The world-wide market leader is considered to be Symbian while the US market is dominated by Windows Mobile and BlackBerry<sup>3</sup>. Both Symbian and Windows Mobile support natively GSM/UMTS, 802.11b/g (Sym-

bian only from version 9.3 onwards) and Bluetooth. Due to the recent addition of WiFi support, most of the available Symbian devices do not have a WiFi interface. Both OSs are open for third-party applications. However, Windows is a more attractive option due to more uniform editions (Symbian distributions differ from manufacturer to manufacturer), simpler application licensing and standard C++ support. Mobile Linux would be an open and license-free option but an industry standard mobile Linux is in early deployment stage<sup>4</sup> and only few OEM specific devices have been released.

Hence, we choose to work with Windows Mobile. We consider it to be a relatively popular platform supporting the features we need on a range of different devices from various manufacturers providing us a large and varying device base for experimentation. Windows Mobile comes in two editions: Smartphone and Pocket PC. The basic difference between the two is that Pocket PCs have a touch screen while Smartphones do not. Apart from that, the functionality and the development APIs for Smartphones and Pocket PCs are almost identical. The applications are developed either using the platform native APIs and tools, or by relying on an external development environment such as Java.

Java Platform Micro Edition (Java ME)<sup>5</sup> is the Java platform for mobile devices. Java ME comes in two main configurations: Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC). CLDC is the most basic Java ME configuration found on most of the cell phones today. The CDC is a subset of the J2SE and designed to support more sophisticated mobile devices such as PDAs, touch screen devices, and set-top boxes. For Bluetooth, the Java platform defines an optional API, JSR-82, which contains the required functionality except the device power management. In contrast, WiFi and cellular radio functions such as managing the device power, configuring the interface and/or detecting the neighborhood are not supported by the standard Java APIs. The way to provide such low-level functionality to the JVM is to program a Java Native Interface (JNI) compatible native library that in turn uses the platform native APIs. In Java ME, JNI is only supported in CDC<sup>6</sup> which essentially rules out the possibility to use CLDC based JVMs.

Windows Mobile does not support Java by default but there exist several CDC compliant JVMs for Windows Mobile which we try out on several Windows Mobile devices. The general conclusion is that there is no single CDC compliant JVM that would run both on Windows Mobile Smartphone and Pocket PC editions.

<sup>3</sup><http://www.informationweek.com/story/showArticle.jhtml?articleID=196902226>

<sup>4</sup><http://www.limofoundation.org/>

<sup>5</sup><http://java.sun.com/javame>

<sup>6</sup><http://java.sun.com/javame/reference/apis/jsr218/>

We find that the IBM J9<sup>7</sup> is a good choice. Its only drawback is that at the time of testing, it is limited to console-mode applications only on Smartphones. This, together with the fact that the memory consumption of the tested JVMs (including J9) is very high, around 15MB while running a simple test application (out of 20-40MB that is available for user applications), discourages us to use a Java-based solution. However, providing an application interface to the native networking library for Java is still an option; and probably an important one in order to attract external application developers not familiar with native Windows APIs. This could be done either by providing a JNI-wrapper for the connectivity management software, or for CLDC environment, by implementing a socket-based interface.

Windows Mobile platform itself provides two main programming models: Managed code and Native code. Managed code refers to the .NET Compact Framework, but as it has the same basic limitations as Java and we choose to work with the native APIs directly. The native networking APIs<sup>8</sup> for Windows Mobile include the Bluetooth API that is mainly based on Windows sockets (2.2), a low-level WiFi interface access using NDIS and NDIS user-mode I/O (NDISUIO) upper level protocol driver. For the cellular radio access there is Radio Interface Layer (RIL) API.

### 3. DESIGN AND IMPLEMENTATION

In this section we present the design and implementation considerations of the connectivity management software to support opportunistic communications with multiple network interfaces on Windows Mobile. We also propose a simple Application Programming Interface for the connectivity management software library.

#### 3.1 Bluetooth

Bluetooth communication setup takes three steps: (1) device discovery, (2) service discovery (optional), and (3) link and connection establishment. The Bluetooth standard [2] defines a set of separate physical channels for device discovery (inquiry 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 its neighbors, it hops through all possible inquiry scan channel frequencies in a pseudo-random fashion, sending an inquiry request on each frequency and listening for responses. The inquiry mechanism is probabilistic and the recommended inquiry duration time is at least 10.24s. The Windows Bluetooth API uses, by default, a duration that is twice the recommendation but it can be modified. The inquiry duration can also be limited by setting the maximum number of devices to look for (by default

16). The inquiry response includes a 48-bit device MAC address and the device class code. The “friendly” name of the device can be fetched using a separate control request before a physical link establishment (paging).

The Service Discovery Protocol (SDP) defines how the services (or “servers” listening for incoming connections) are advertised and queried. A service is described using a service record which is a simple string consisting of (attribute id, value) pairs. Service discovery is implemented as Winsock functions. Unfortunately, the available Bluetooth/Winsock API does not offer any convenience methods for handling these service records which makes the API harder to use.

For the data transmission we use RFCOMM which is a serial line emulation protocol for Bluetooth. In order to connect to another Bluetooth device over RFCOMM, the client must know the server channel which can be resolved using SDP. It is also possible to use hard-coded channels. Service discovery is more dynamic and recommended since the number of available channels is very limited (30). In addition, the service records can contain more information than just the channel number. Such information could help in deciding whether actually to connect and use the contact opportunity or not, and thus could save some critical resources.

#### 3.2 WiFi

WiFi (TCP/UDP) connection (either in ad hoc mode or through an access point) between two devices requires the following steps: (1) network detection, (2) association and interface configuration, (3) peer device detection, and (4) socket creation and connection (TCP only). In addition to device-to-device connections, we can also use broadcasting, in which case the step 3 can be skipped.

The 802.11 standard [10] defines two possible network discovery modes: active or passive. In active mode, the scanning device sends probes to request for available networks (infrastructure or ad hoc); in passive mode, the scanning device listens for periodic beacons sent by neighboring APs and devices in ad hoc mode. One problem with the Windows NDIS API is that it does not offer any control over the scanning process, i.e. there is no way to define the “duration” of the scan, and we can not decide if the scanning should be active or passive. The scanning procedure is asynchronous: we notify the NDIS driver of the beginning of a scan, and after some timeout, we ask the currently available list of networks from the driver. In practice this results in the API occasionally returning a network list that is incomplete irrespective of the time between the operations. We decide to set this delay (“scan duration”) to 5 seconds in our implementation. The network scan results detail for each found network the SSID, the BSSID, the network mode (ad hoc or infrastructure), privacy requirements

<sup>7</sup><http://www-306.ibm.com/software/wireless/weme/>

<sup>8</sup><http://msdn2.microsoft.com/>

and the RSSI (signal strength).

Once the list of networks is available, we must select one: for example a preferred one (well-known or pre-configured SSID), the one with the best signal strength or simply any open one. After an association with the network, the network interface (NIC) is configured by assigning a static IP or fetching one using DHCP (only in infrastructure mode). Windows Mobile has a service, Wireless Zero Configuration (WZC), that takes care of the NIC configuration. Overriding it seamlessly using NDIS and IPHelper APIs does not work very well, at least not on all the devices we try. Instead, we use the WZC API directly to configure the WiFi NIC.

The third step is to discover the peer device(s) IP address(es) unless broadcasting. IPv4 does not offer any built-in mechanism to discover neighboring devices. IPv6 defines a standard neighbor discovery mechanism [16]; however, IPv6 support on Windows Mobile is OEM specific and usually not available. The two possible application layer solutions are either a request-response based scheme or periodic beaconing. The *request-response approach* is conceptually similar to the active mode network search and can be implemented by broadcasting an UDP request to a well-known port and waiting for the answers from the available devices. In the *beaconing approach*, the discoverable devices broadcast beacons periodically and the scanning devices listen to these beacons during the scanning interval as in a passive network scan. Both discovery schemes can be performed either in ad hoc or infrastructure networks. The request-response scheme is not very energy efficient as the discoverable devices have to stay associated and listening for incoming requests all the time. The beacon broadcasting is better since it does not require the WiFi interface to be continuously powered up, but for it to be practical (not to miss too many contacts), the beacons should be sent quite often. The drawback is the loss of time and energy in constant association and interface (re-)configuration.

### 3.3 Cellular

The cellular radio interface layer (RIL) API on Windows Mobile provides a function to query for a list of available operators. This triggers a network operation and takes a variable amount of time (20-40 seconds) to complete. In addition, we can request for asynchronous notifications of the cell id changes and of the signal strength. To enable cell id notifications, we have to rely on something that seems to be a “de-facto” hack found in an internet forum since the official API has very limited documentation. The API does not define either how to differentiate between 3G and GSM networks. The asynchronous nature of all the methods in the RIL API requires careful synchronization in order to provide a synchronous API for applications. We leave the

```

BLUETOOTH
  [Set|Get]DevicePower
  SetVisible
  GetDeviceList
  GetServiceList
  [De]RegisterService
  [Create|Bind|Accept|Listen|Close]Socket
WIFI
  [Set|Get]DevicePower
  GetNetworkList
  GetDeviceList
  SetVisible
  Associate
  Disassociate
  [Create|Bind|Accept|Listen|Close]Socket
CELLULAR
  GetNetworkList
  GetCurrentCell
  [Create|Bind|Accept|Listen|Close]Socket

```

Figure 1: Connectivity management API

cellular packet data communications implementation as future work concentrating on pure device-to-device opportunistic links for now.

### 3.4 Connectivity management API

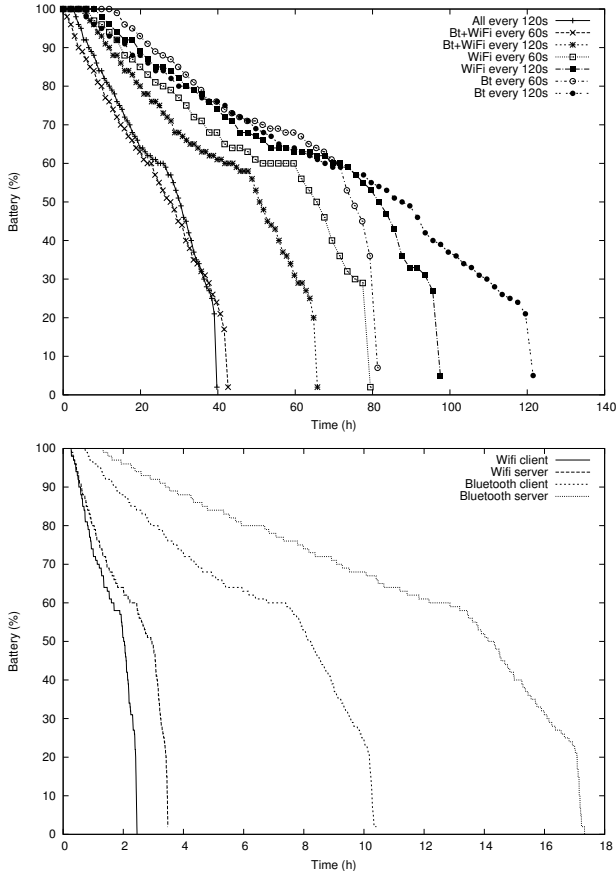
We propose a simple wrapper API to hide the unnecessary details of each available network interface and to unify the handling of them for applications (Figure 1). The API has separate set of functions for each interface. While all the interfaces support the common socket based communication abstraction, the network and device discovery functionality and the configuration of each interface vary. We define explicit power management functions currently only for Bluetooth and WiFi.

We have implemented the connectivity management software as a static library in C++ using the native Windows Mobile APIs. Currently the library supports the WiFi and Bluetooth APIs completely and the discovery part of the cellular API. To assure the portability of our solution, we have tested the implementation on several Windows Mobile 5.0 and 6.0 devices (both Smartphones and Pocket PCs) including various HTC Pocket PCs and Smartphones, i-mate JAQ3, Samsung Blackjack (i600) and Toshiba Portegee.

## 4. LABORATORY BENCHMARKS

This section presents a set of benchmarks on opportunistic communications. We perform the measurements in a static laboratory setting using simple test applications built upon the communication API described in the previous section. The goal of these experiments is to find out about the performance limits and parameters for real-life experiments. The benchmarks are performed using HTC s620 Windows Mobile Smartphones<sup>9</sup>. The device has a 200MHz TI processor, 64MB

<sup>9</sup>[www.htc.com/product/03-product\\_s620.htm](http://www.htc.com/product/03-product_s620.htm)



**Figure 2: Battery-life in discovery (top) and data transmission (bottom)**

of RAM, 128MB of ROM and a MicroSD slot. The radio interfaces include a quad-band GSM/EDGE cellular radio, Bluetooth v1.2 and 802.11b/g. We select the HTC s620 after comparing the performance and programmability of several Windows Mobile based devices from various manufacturers. The battery life of HTC s620 is among the best and it supports all the native APIs described before.

#### 4.1 Battery Life

We first perform simple battery life tests for device discovery and data transmissions. In the discovery test we set the software to scan its neighborhood using various networking interfaces and varying scanning intervals (60 and 120 seconds). The Bluetooth and cellular interfaces are powered on all the time when they are used for scanning. The WiFi interface has the automatic power saving enabled which turns off the interface 10 seconds after scanning activity. Figure 2 (top) shows the battery-life evolution from different setups. The life-time while using all the three interfaces for scanning every 2 minutes is around 40 hours. With just Bluetooth

discovery enabled, the device lasts over 120 hours<sup>10</sup>. When decreasing the scanning interval from 120 seconds to 60, we observe 20 - 40 % shorter life-times. WiFi discovery compared to Bluetooth does not show such a big difference in power consumption, particularly when looking at the shorter, 60-second, scanning interval. This result can be counter-intuitive, but we explain it by the relatively high power consumption of the Bluetooth inquiry [3]. In addition, the Bluetooth inquiry duration is twice as long as the WiFi network discovery (10.24s against 5s). Finally, it must be noted that in this experiment we perform only the network scan for WiFi; actual device discovery would be much more power consuming as it requires association and data transmissions. In the remaining experiments and the life experiments we decide to be conservative in terms of power and we set the scanning interval to two minutes. Based on previously measured contact times [9], two minutes also seems to be a good compromise between the battery-life and scanning frequently enough for not to miss too many contacts.

To measure the worst-case battery life when transmitting data, we perform a test where a device is either continuously sending or receiving data to/from a fixed target PC until the battery drains out. The data traffic consists of 1MB chunks for WiFi and 0.5MB chunks for Bluetooth (the socket connection is reset between each chunk). Both interfaces are powered up during the whole experiment. The recorded life-times are shown in Figure 2 (bottom). Having the WiFi interface constantly on and sending/receiving drains out the battery in less than 4 hours. Bluetooth requires less power, and in the client mode the device runs for 10 hours and in the server mode over 17 hours. In real-life we do not expect PSN applications to create such a continuous data traffic, and thus the impact on battery life would not be as severe as shown here.

From both battery life-time graphs we can also observe that the depletion rate is not linear. There are two visible thresholds: one at about 60% of battery left, and the second when there is about 20% of battery left. Around 60% of battery life-time the curve begins to decrease faster. When it reaches 20%, the depletion rate reaches a critical limit, and the device runs out of battery in few hours when scanning and in only few minutes when sending or receiving data. While this could just be a device model, manufacturer or Windows Mobile specific behavior resulting from the way it reports the remaining battery life-time, it is a useful observation when considering the implementation of power-saving policies on these devices. For example, the reported 20% remaining battery life does not mean that the de-

<sup>10</sup>For comparison, the standby time reported by the manufacturer is up to 220 hours ( 9 days) and talk time up to 5 hours.

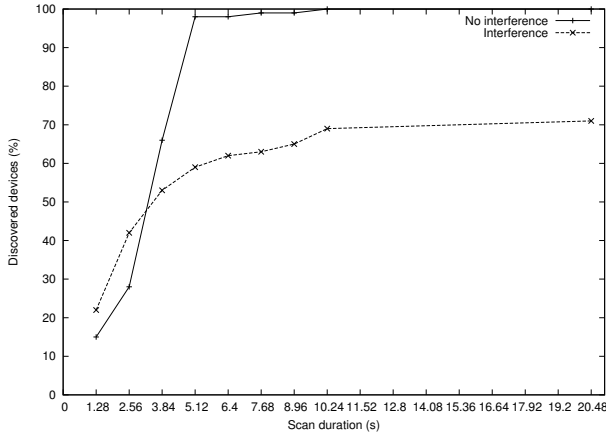


Figure 3: Bluetooth device discovery duration

vice can still stay up about 1/5 of what it has done so far, but that it will run out of battery in a very short time.

## 4.2 Neighborhood Discovery

We evaluate the Bluetooth device inquiry time by performing an experiment in which we set 15 discoverable Bluetooth devices within the range of the inquiring device. We vary the inquiry duration from the shortest possible (1.28s) to the Windows Mobile default (20.48s). Figure 3 shows the average percentage of devices discovered (averaged over 100 successive scans). The first curve shows the discovery rate when the discoverable devices are passive. For comparison, we repeat the test by setting the 15 devices to perform a device discovery every 2 minutes to add some Bluetooth interference. As discussed before, the recommended inquiry scan duration is 10.24 seconds, and we can see that in the interference-free case, we observe a 100% success rate. Based on the experiment, also shorter scan durations are enough to pick up all the devices around. This observation is in agreement with analytical studies such as [14]. However, once we add some interference, even the best-case success rate is only about 70%. 10.24s is again enough to reach this and a longer scan does not improve the rate considerably.

Since the Bluetooth inquiry is a power-consuming task, a shorter scan duration could help in saving the battery on the device, and in addition, it would increase the time that could be dedicated to the data transmissions. However, in real-life, in the presence of multiple inquiring devices and connected piconets, there will be interference affecting the Bluetooth operations like the discovery. In addition, the device mobility will also impact the performance. Consequently, for the remaining benchmarks and the field experiments we choose the duration of 10.24 seconds to be sure to detect a maximum

number of neighbors.

As discussed before, we can not change the parameters of WiFi network discovery and we settle on a scan duration of 5 seconds in our experiments. As to the actual device discovery over WiFi we test the device discovery in a static setting using both the request based method and the periodic beaconing. The discovery delay for both methods is dominated by the time spent on the available network lookup, the association and the network interface configuration of the scanning and the beaconing device. As discussed above the network lookup takes 5 seconds and the configuration steps require around 4 seconds on average. The broadcast request sending and response collection is fast taking usually less than a second. The success rate is normally 100% over a large number of repeated scans and with varying number (2 to 10 in our benchmarks) of discoverable devices. The discovery time and detection rate when broadcasting periodic beacons would depend on how often the beacons are send, on the scanning interval and on how long the scanning device is listening for incoming beacons. We do not perform detailed measurements on beaconing.

## 4.3 Data Transmission

Next we examine the Bluetooth and WiFi connection setup and data transmission characteristics. We measure the data transmission rates and delays using a modified version of the open-source tool Test TCP<sup>11</sup>.

To evaluate the connection setup process, we perform an experiment where we set varying number (10 and 15) detectable and connectable devices in the range of a device that is scanning and trying to connect to the devices it detects on each scan. Upon a successful connection, the test sends a small amount of data (25KB) to the destination device and closes the connection. To see the effect of interference, we repeat the experiment both for the case of passive connectable devices and for the case in which the connectable devices are performing either a Bluetooth inquiry every 2 minutes, or for WiFi, a request-response based discovery over an ad hoc network every 2 minutes. When there is no interference, we manage to connect to 75% of 10 available devices and only 50% of 15 available devices using Bluetooth. With interference, the connection success rates seem to stay about the same. The WiFi connection success rate is 100% without and with interference.

Figure 4 shows the distribution of connection setup delays from the previous experiment. As a CSMA/CA based MAC, WiFi offers constant and good performance; average TCP connection setup time is 114.7ms when there is no interference in the network and 123.8ms with interference. The connection setup times exclude the network association and the interface configuration

<sup>11</sup><http://www.pcausa.com/Utilities/pcattcp.htm>

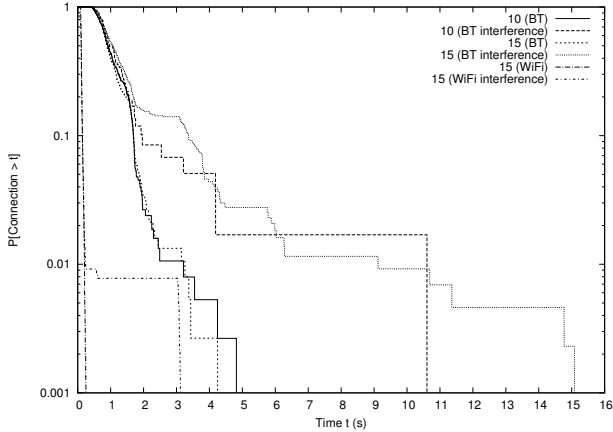


Figure 4: Connection setup delay distribution

which takes 4.5s on average in this experiment. The distribution shows only few cases where the connections setup takes longer when there is interference compared to the interference-free situation. Bluetooth paging (connection) procedure is probabilistic like the inquiry, and we see quite a lot of variation in connection setup times, in particular in the presence of interference. We can also note that to maintain the similar connection success rates between the interference-free and interference setting, we suffer from higher delays. The average connection setup time when we have 15 interfering devices around is 1541.4ms and 1378.7ms for 10 interfering devices. The corresponding values in the non-interference setting are 1051.0ms (15 devices) and 1070.4 (10 devices).

	Bluetooth	WiFi infra	WiFi ad hoc
txt [2KB]	1.0s	1.30s	1.28s
doc [40KB]	1.48s	1.83s	2.89s
jpg [190KB]	4.76s	4.76s	5.34s
mp3 [5MB]	114.65s	15.92s	12.08s

Table 1: Data transmission delays

Table 1 shows the server side times (average over 100 repetitions) to receive files of different sizes (a small text file, a Word document, a JPG image and an mp3 file). We can see that Bluetooth is as fast as WiFi when sending small amount of data. When the file size grows, WiFi outperforms Bluetooth as expected. There is no big difference between infrastructure (both devices connected to the same access point) or ad hoc modes for WiFi. On average, the observed rates both at the sender and the receiver side for Bluetooth are about 50KB/s, and for WiFi varying from 100KB/s to 500KB/s.

Following these lab experiments, we choose to use Bluetooth for data transmissions in our field experiments because of its lower power requirements and built-in device discovery mechanism. We also find that the

transmission rate obtained with Bluetooth is sufficient for the first simple applications that we have in mind (e.g. mobile social networking and newsgroup -style messaging).

## 5. FIELD EXPERIMENTS

This section presents three real-life experiments we perform to evaluate the feasibility, constraints and suitable parameters for real PSN systems. The initial experiment is parametrized based on the laboratory benchmarks. Each experiment is used to gather more information to tune the configuration parameters in order to optimize our communication system for PSN applications.

### 5.1 Experimental Setup

The first experiment (*Summer*) mimics the setup of the previous iMote experiments [9]. We implement an application on top of our connectivity manager library to log contact opportunities using Bluetooth, WiFi and cellular interfaces. We distribute 15 HTC s620 smartphones to a research lab staff working together and living in a major city. The participants are instructed to use the device as their personal mobile phone for about one month and to run the contact log application in the background all the time.

The other two experiments are performed in conference environments. For the first conference experiment (*ConfEU*) we extend the previous scanning application to send data over Bluetooth between participating devices opportunistically when they meet. We still scan all the three interfaces (on most of the devices) every two minutes. We make a small implementation change to the Bluetooth discovery and disable the friendly name query after observing that it increases the scan durations and could impact the number of devices detected. After each scan, the test initiates a Bluetooth data transfer towards detected internal devices. We vary between two modes: sending 50 KB chunk of data to each internal device (fragment test), or sending only 1 MB to a single random internal device (bulk test). The list of participating devices is hard-coded in the software so that we do not waste resources in trying to connect to external devices. We measure the success rate and data throughput on the client and server sides. The application requires no user interaction and the participants are simply asked to keep the devices charged and with them all the time. The second conference experiment (*ConfUS*) has a similar setup as ConfEU. The main differences are that we add a user discovery to find participating devices, and we send only 50KB chunks to a subset of the detected internal devices. The user discovery is performed after each scan by trying to connect to all the discovered devices to perform a short handshake protocol. At ConfUS the application uses only Blue-

	Summer	ConfEU	ConfUS
<b>Duration</b>	6 weeks	3 days	3 days
<b>Participants</b>	15	29	28
<b>Trace length (offline)</b>	38.5d (11.6d)	2.8d (27.1h)	2.2d (32.1h)
<b>Unique Bluetooth MACs</b>	34422 (15 internal)	990 (29 internal)	2024 (28 internal)
<b>Bluetooth contacts</b>	90787 (24801 internal)	21804 (11702 internal)	15109 (10143 internal)
<b>Unique WiFi BSSIDs</b>	65191 (26106 open)	505 (144 open)	n/a
<b>WiFi contacts</b>	278422 (105941 open)	6959 (4873 open)	n/a

**Table 2: The field experiments**

tooth to detect and communicate with the neighboring devices; WiFi and cellular interfaces are disabled unless used by the participant.

Table 2 summarizes the three experiments’ setup and overviews the collected data sets. The summer experiment was 6-weeks long while the conference experiments lasted for three days. The scanning interval for each experiment (and each interface) was set to two minutes which means that the device battery was expected to last about 10 hours in normal usage requiring the participants to charge the devices daily. We define the offline time as the time during which (within the full trace duration) the device is not running our application. The offline time results mainly from depleted batteries and application crashes. The offline times, in particular at the conference experiments, are somewhat high. The big challenge seems to be to motivate the participants to use and charge the device. We envisage that future experiments with real PSN applications will help in increasing the user motivation, and consequently allow us to collect more complete results.

To compare the scope of the collected data, we calculate the number of unique devices seen and the number of contacts for each scanned interface (we omit cellular since it was only fully traced in the first experiment). A *contact* is defined as a period of time during which a device (Bluetooth) or a network (WiFi) is seen by the scanning device. The contact time is a measure of the network capacity. To mitigate the unavoidable “sampling” effect (i.e. we miss some devices due to interference or simultaneous scanning with Bluetooth for example), a contact is considered finished only if the device or network is not detected during two consecutive scans. This definition of contact is consistent with previous studies [9]. An *internal contact* is a contact between two participating devices and is applicable only for Bluetooth in these experiments (we do not perform the WiFi device discovery). The last important metric is the *inter-contact time* which is the time between two contacts.

As the summer experiment lasts much longer than the conference experiments, it logs the highest number of unique devices and contacts. At ConfEU we detect twice fewer unique Bluetooth devices than at ConfUS which is explained by the conference locations:

ConfEU took place in a research center slightly outside of a smaller European city while ConfUS was held at a university campus in a very large city in the US. The participants of ConfUS moved around in the city on a wider area than the participants of ConfEU resulting in a higher number of unique devices at ConfUS. On the contrary, the total number of contacts at ConfEU is about 22000 which is almost 7000 contacts *more* than at ConfUS where we log around 15100 contacts in total. This is partly due to the shorter average trace length of ConfUS (where we also had one device less). This also means that at ConfEU we have contacts with the same devices much more often than at ConfUS which again makes sense because of the different locations. The number of internal contacts in both conference experiments is roughly the same and in the order of 10000.

## 5.2 Battery Life

We can observe the battery life-times in each experiment from the collected traces. In the summer experiment the battery lasts on average 10 - 12 hours. We achieve this life-time when the device is running our application and while users are using the device normally for calling, checking email or Skyping over WiFi. Most of the users also charged the device well before the battery depleted.

The conference experiments are not completely comparable to the summer experiment since most of the participants did not put a SIM card into the device, and the phone usage was typically limited to occasional internet browsing and “playing”. The second difference is that the test application generated constant Bluetooth data traffic as explained before increasing the energy consumption. At ConfEU the usual battery life-times vary from 8 - 14 hours depending on the user activity, whether the device was scanning WiFi, whether the cellular interface was enabled (and scanning), and how much data the device was sending and receiving. To increase the average battery life, we disabled WiFi and cellular interface scanning for ConfUS. Consequently, the observed battery lives improve and the devices run 12 - 14 hours on average if not charged before. We consider a 10-hour life-time to be reasonable for an experiment, but for practical application this is probably still short for most users since the device has to be charged every night.

### 5.3 Neighborhood Discovery

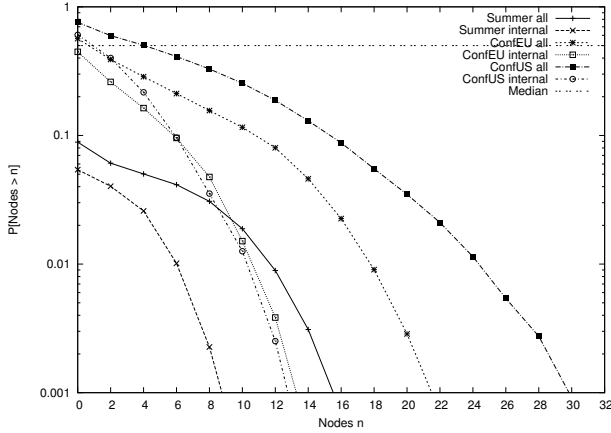


Figure 5: Bluetooth contact detection rates

We first compare the Bluetooth device detection rates from each experiment in Figure 5. The figure shows the distribution of the number of Bluetooth devices detected per scan interval, as well as the number of the internal devices per scan. We exclude the night hours (10pm to 8am) from the distribution as the devices usually stay in a static environment during that period. For the summer trace we consider only one representative week.

In the summer trace we observe very low contact rates. 90% of the time the devices are discovering no other Bluetooth devices. The tail of the distribution shows that only a small fraction of scans detect 8 or more internal devices (or 15 or more in total). The conference traces are much denser: 50% of the scans discover at least one Bluetooth device at ConfEU and four at ConfUS. At ConfEU we detect at least one internal device 30% of the scans while at ConfUS one or more internal devices is seen by 50% of the scans. The tails of the distributions ( $P \leq 10\%$ ) for internal devices in the conference experiments are almost identical, and we rarely detect more than 13 internal devices per scan. The maximum number of all devices seen are different in the two conference experiments: at ConfUS we observe a small percentage of scans that detect almost 30 devices or more while the respective number for ConfEU is less (around 22).

The main reason for the differences in contact detection rates come from the experimental environment (location and participants). In the summer experiment we have twice fewer participants that only meet each other during the office hours, and even then not everybody is present as the participants travel frequently for their work or go on a holiday which lowers the number of internal contacts. The conference experiments are denser because people are typically sitting in a single room lis-

tening for the talks during the day and stay in contact also after the working hours to go sightseeing, on social dinners, or because they are sharing a hotel. The experiment location has an impact on the total number of devices seen and in a larger cities the number of external Bluetooth devices per scan is higher. However, in the summer experiment we never see as many devices per scan than in the conference experiments even if it is performed in a relatively large city. This is partly affected by the fact that the friendly name query was still enabled in this experiment which could have resulted in missed contacts.

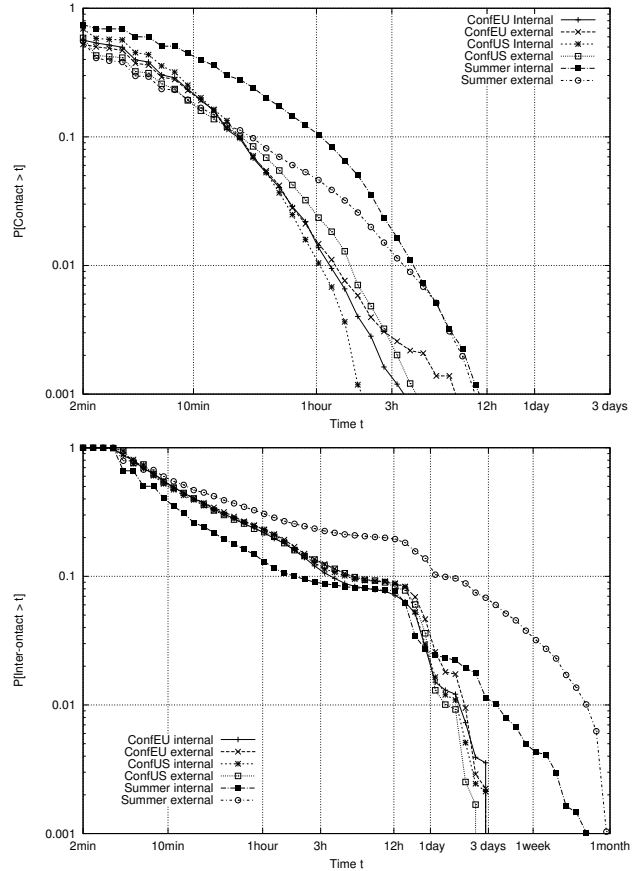


Figure 6: Bluetooth contact times (top) and inter-contact times (bottom)

Figure 6 (top) illustrates the contact time distributions from the three experiments. The contact times follow the same general trend as reported before in [9]. The summer experiment distinguishes itself from the conference experiments in two ways. First, the external contacts are typically very short which makes sense as most of them are likely happening with devices that are met shortly while moving in the city. The second observation is that the contacts with internal devices are long: the median contact time is 8 minutes. This re-

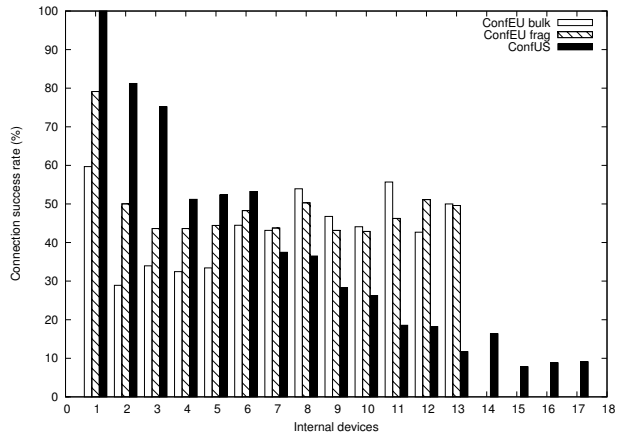
sults from the fact that when the participants meet each other, it usually happens in the office when working together, having a meeting or going for lunch. In contrast, at the conference experiments the contact times are shorter and the median contact time between internal devices is only around 2 to 4 minutes. This seems to be in contradiction with the previous discussion about the higher number of devices discovered per scan. However, we think that the explanation to this behavior is that even if the devices are co-located, the set of detected devices vary on each scan resulting in short contacts. This sampling happens because the Bluetooth device discovery is probabilistic and very sensitive to interference as we could see also from our laboratory benchmarks.

From the inter-contact time distribution (Figure 6 bottom) we can see that the median inter-contact duration is around 10 minutes for the conference experiments. During the summer experiment we observe shorter inter-contacts for internal devices, and on the other hand, longer external inter-contact times. The tail of the distribution goes up to the experiment duration (three days for conferences and about a month for the summer experiment). The inter-contact time patterns are similar in all experiments and follow the power law shape observed and analyzed in previous work [9].

In conclusion, Bluetooth device discovery does have some limitations as the number of co-located devices increases. For this reason we think that opportunistic networking applications will benefit from maintaining a contact history list that spans over multiple scans. This would not add too much overhead since the number of discovered devices is quite low in general. Second, the low-level discovery process could be enhanced by propagating discovery results from device to device as has been proposed in [18]. This would increase the discovery duration since we need to connect to each (or a subset of) neighboring device(s). However, as real applications might do this in any case, that would just add a small extra piece of information to be exchanged upon a connection. In both schemes, the aggregated contact list might contain unreachable devices due to mobility. We leave for future work experimentation with these ideas and understanding of the real benefits and trade-offs they present.

## 5.4 Data Transmission

Figure 7 plots the Bluetooth data connection success rates as a function of number of detected internal devices. The trend observed in the ConfUS experiment follows the common accepted assumption [1] that the success rate should decrease as more and more devices are trying to communicate between each other. When there is only a single internal device in the neighborhood, the connection succeeds always; when the num-



**Figure 7: Bluetooth data connection success rates with respect to detected internal devices**

ber increases to 10 internal devices, we only manage to establish a connection with three of them on average. For ConfEU we separate the bulk and fragment tests. Remember that in the bulk test we try to connect only to one of the detected internal devices, while in the fragment test we try to connect and send data to all the internal devices detected (as in ConfUS). In this experiment we do not see a clear correlation and on average about 40% of the data connections are successful independently of the number of scanned internal devices. This can be explained by the fact that the bulk and the fragment tests were executed at different times on different devices resulting in fewer simultaneous connection attempts; at ConfUS we systemically tried to communicate with all detected internal devices after each scan.

Figure 8 shows the average client side data transmission rates as a function of detected internal devices (error-bars show the minimum and maximum rates). The transmission rates do not seem to be affected by the number of devices in the neighborhood i.e. the bandwidth allocation is fair. At ConfEU the average client side rate is 15KB/s and the average server side rate 28KB/s. The respective values for the ConfUS are 5KB/s at both ends. The real-life performance at ConfEU is thus only around 50% of the lab benchmark rate, and at ConfUS the rate decrease to 10% of the benchmarked rate. We conjecture that the higher contact rate and increased Bluetooth traffic at ConfUS affects also the perceived user bit rates.

There have been various proposals to use Bluetooth (or any low power technology in general) for the neighborhood detection and parameter negotiation, and in case of a bigger data transfer, switch to WiFi to send the data [17, 11]. Based on our experiences presented here, this could be a reasonable solution at least with

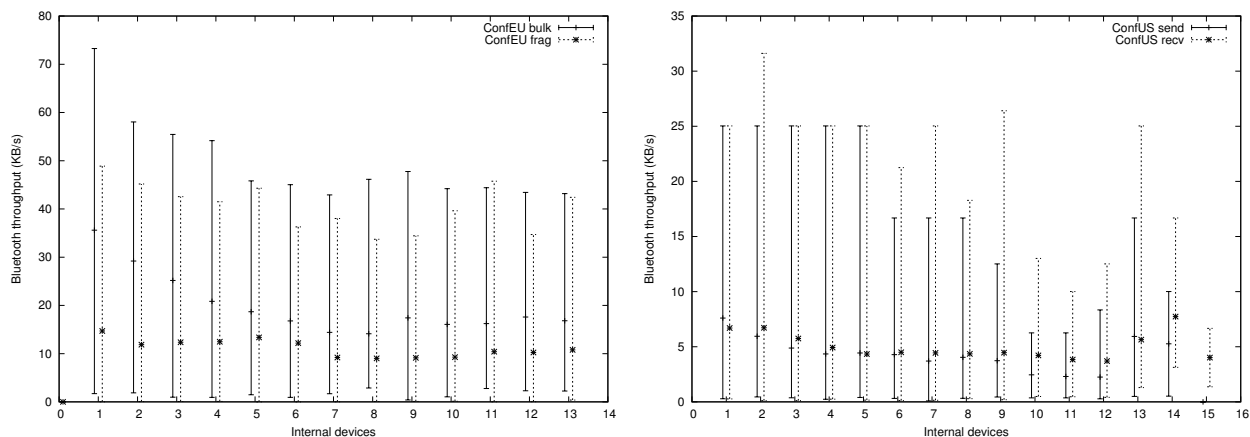


Figure 8: Bluetooth data transmission rates with respect to detected internal devices

the current radio hardware and protocols for building energy-efficient high capacity opportunistic communication systems.

## 6. RELATED WORK

Pocket Switched Networks (PSN) were introduced by the Huggle project<sup>12</sup>. PSN falls under the more general concept of delay tolerant networking [6]. The DTN working group<sup>13</sup> has proposed a set of standard protocols and provides a reference implementation which has also been ported to Symbian [15]. However, this mobile version is a proof-of-concept and no detailed network performance analysis with multiple mobile devices is yet done. Huggle is also a complete cross-layer architecture for opportunistic communications [19]. A Java based (CDC compliant) prototype exists, but similarly, the reported real-life experiments are limited to a simple demonstration scenario with PCs. In this work we take a bottom-up approach to understand the feasibility and performance of opportunistic (and delay tolerant) communications using mobile devices: we study the different mobile computing platforms and communication technologies; and design portable connectivity management software for Windows Mobile. In addition to laboratory benchmarks, we perform three large-scale life experiments.

There exists already some work on applications that use opportunistic connections. The authors of [12] study content sharing over Bluetooth and propose a bittorrent-like system that they evaluate through simulations and emulations. They also provide a detailed analysis of the optimal low level device discovery parameters for Bluetooth. Another recent work [13] presents a receiver-driven delay tolerant broadcasting system and a proof-of-concept prototype for PDAs using Java and Blue-

tooth. While in this work we do not yet consider any specific application, the connectivity manager we propose could easily support both p2p file sharing or delay-tolerant broadcasting. In addition, we provide more insight into the real-life feasibility and performance issues in such systems in presence of interference and user mobility.

## 7. CONCLUSION

Opportunistic device-to-device communications with currently available mobile devices, development platforms and wireless technologies is feasible but has some challenges. In this work we design and implement connectivity management software to help experiment with opportunistic communications. The connectivity manager is currently still a simple wrapper for the native Windows Mobile APIs, but we are extending it towards a complete open framework for PSN application development.

While experimenting with our connectivity manager, we observe that the main limitations for PSNs seem to come from the wireless technologies which have not been designed to support such utilization. The connectivity management has a great impact on the capacity and overall delays in PSNs. As a next step we try to better understand how to maximize PSN capacity using a combination of Bluetooth, WiFi, or even the cellular interface. In addition, we are also looking into proposed enhancements of the existing technologies such as the latest versions of the Bluetooth standard and low-power WiFi extensions. New ultra-wideband (UWB) radio technologies are also currently under standardization. One (or a combination) of these might turn out to be the ideal next generation wireless technology for efficient opportunistic networking.

<sup>12</sup><http://www.huggleproject.org>

<sup>13</sup><http://www.dtnrg.org>

## 8. REFERENCES

- [1] S. Asthana and D. Kalofonos. The problem of bluetooth pollution and accelerating connectivity in bluetooth ad-hoc networks. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, 2005.
- [2] Bluetooth Special Interest Group (SIG). *Specification of the Bluetooth System. Core Package version: 1.2*, November 2003.
- [3] J.-C. Cano, J.-M. Cano, E. González, C. Calafate, and P. Manzoni. Evaluation of the energetic impact of bluetooth low-power modes for ubiquitous computing applications. In *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, 2006.
- [4] A. Chaintreau, P. Hui, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 6(6), June 2007.
- [5] N. Eagle and A. Pentland. Reality mining: Sensing complex social systems. *Journal of Personal and Ubiquitous Computing*, 2005.
- [6] K. 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*, 2003.
- [7] A. Furuskar, S. Mazur, F. Muller, and H. Olofsson. Edge: enhanced data rates for gsm and tdma/136 evolution. *IEEE Personal Communications*, 6, June 1999.
- [8] T. Henderson, D. Kotz, and I. 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*, 2004.
- [9] 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.
- [10] 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.
- [11] H. Jun, M. H. Ammar, M. D. Corner, and E. W. Zegura. Hierarchical power management in disruption tolerant networks with traffic-aware optimization. In *CHANTS '06: Proceedings of the 2006 SIGCOMM workshop on Challenged networks*, 2006.
- [12] S. Jung, U. Lee, A. Chang, D.-K. Cho, and M. Gerla. Bluetorrent: Cooperative content sharing for bluetooth users. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, 2007.
- [13] G. Karlsson, V. Lenders, and M. May. Delay-tolerant broadcasting. In *CHANTS '06: Proceedings of the 2006 SIGCOMM workshop on Challenged networks*, 2006.
- [14] J. Kharoufeh, B. Peterson, and R. Baldwin. Bluetooth inquiry time characterization and selection. *IEEE Transactions on Mobile Computing*, 5(9), 2006.
- [15] O. Mukhtar and J. Ott. Backup and bypass: Introducing dtn-based ad-hoc networking to mobile phones. In *Proceedings of the RealMAN Workshop*, 2006.
- [16] T. Narten, E. Nordmark, and W. Simpson. Neighbor discovery for ip version 6 (ipv6), 1998.
- [17] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, 2006.
- [18] F. Siegemund and M. Rohs. Rendezvous layer protocols for bluetooth-enabled smart devices. *Personal Ubiquitous Computing*, 7(2), 2003.
- [19] J. Su, J. Scott, P. Hui, J. Crowcroft, C. Diot, A. Goel, E. de Lara, M. H. Lim, and E. Upton. Haggie: Seamless networking for mobile applications. In *Proceedings of UbiComp*, 2007.