A Survey of Platforms for Mobile Networks Research

Earl Oliver
David R. Cheriton School of Computer Science
University of Waterloo
eaoliver@uwaterloo.ca

The use of smartphones is growing at an unprecedented rate and is projected to soon pass laptops as consumers' mobile platform of choice. The proliferation of these devices has created new opportunities for mobile researchers; however, when faced with hundreds of devices across nearly a dozen development platforms, selecting the ideal platform is often met with unanswered questions. In this paper I consider desirable characteristics of mobile platforms necessary for mobile networks research. Based on these characteristics, I assess five smartphone platforms: Android (Linux), BlackBerry, iPhone (Mac OS X), Symbian, and Windows Mobile. This survey is current as of December 2008. A living version of this survey is available at: http://blizzard.cs.uwaterloo.ca/eaoliver/platforms/.

I. Introduction

The recent year has seen an enormous growth in the popularity and visibility of smartphones. In 2007, smartphone sales rose 60% to nearly 115 million devices [24]. By 2010, smartphone sales are expected to surpass those of laptops [27]. The proliferation of these devices has created new opportunities and challenges for mobile networks researchers: smartphones have significant power constraints, low-power CPUs, limited RAM and persistent storage, and slow I/O. These devices are inherently mobile, operate primarily over wireless channels, and are nearly always connected to the Internet and/or a cellular network. When faced with hundreds of devices across nearly a dozen development platforms, the choice of mobile device and platform is often met with unanswered questions. The choice of development platform can have a direct effect on the quality and completeness of a researcher's work.

In this paper I analyze the five most popular smartphone platforms: Android (Linux), BlackBerry, iPhone, Symbian, and Windows Mobile. Each has its own set of strengths and weaknesses; some platforms trade off security for openness, code portability for stability, and limit APIs for robustness. This analysis focuses on the APIs that platforms expose to applications; however in practice, smartphones are manufactured with different physical functionality. Therefore certain platform APIs may not be available on all smartphones.

To aid in the decision of platform selection, I have distilled a set of requirements from an existing body of mobile networks research [2, 18, 20, 22, 21, 4, 5, 9]. In the next section I explain each research requirement. In Section III I compare each mobile platform, and in Section IV I summarize and conclude this survey.

II. Requirements

An ideal mobile research platform should expose the following functionality to third party applications:

Network scanning: To access a wireless network, each mobile device is required to scan for APs or cell towers. The data returned from a scan contain information needed to associate with the network. The results of a scan can also be used to approximate the location or assess the mobility of a user. Cell tower scans are typically automated, whereas WiFi and Bluetooth scans are almost always user initiated. However, scanning can rarely be initiated by a third party application. The results of the scan are also typically inaccessible. The ability to programmatically scan and connect to a wireless network can significantly impact the ability of a mobile device to serve as research platforms.

Interface selection: The presence of multiple network interfaces is common on many current smartphones. The most common interfaces are cellular, WiFi, and Bluetooth. Each wireless technology has a unique set of benefits and costs. For example, WiFi has low monetary cost, but consumes more power than a cellular data service such as EDGE. When multiple interfaces are within coverage, most mobile platforms naïvely direct all data traffic over the WiFi interface to minimize monetary cost. Immediately routing data over the least expensive interface is not necessarily optimal [29]. Moreover, this strategy restricts experimentation and measurement across multiple interfaces. Researchers require the means to route data over a specific network interface.

Bluetooth I/O: Bluetooth's short range and low power consumption makes it an ideal method for inter-device wireless communication. As researchers we require the ability to create local Bluetooth services for other devices to connect to.

Interface control: The ability to independently enable and disable network interfaces can have a significant effect on the battery life of a mobile device. The wireless chipsets on most recent devices implement an idle state that reduces the energy required to keep an interface enabled [4]; however, when scanning or transmission on a specific interface is not required, it is more efficient to simply disable the interface.

Background processing: Experimentation often requires monitoring and recording information in parallel with user activities and mobility. Therefore the ability to run background applications is essential for researchers.

Energy monitoring: For the foreseeable future, maximizing battery life will continue to be a key concern for mobile networks research. Scanning for APs, transmitting data, and location sensing all have an energy cost that must be measured. As researchers, we require a means to programmatically measure, (or at least approximate), the energy consumed by a mobile device.

Power saving control: A natural extension to measuring power consumption is the ability to control it. Because maximizing battery life is critical to all devices, these features are typically built into the OS. Devices commonly disable the screen, switch to a lower performance CPU mode, or disable the WiFi interface. On a mobile device, switching to a power saving mode is typically a function of parameters such as: remaining battery level, idle time, input from sensors, physical orientation (holstered), etc. However, there are many situations where an aggressive approach to power savings can have an adverse effect on research. For example, in [18] we found that the Pocket PC version of Windows Mobile frequently disabled the WiFi interface when the screen turned off. To periodically scan for WiFi APs, it was necessary to disable power saving features. As device manufactures continue to extend battery life and implement increasingly aggressive power saving functionality, the ability for researchers to disable it will become increasingly important.

Low-level memory management: Despite low-cost and ample supply, RAM in mobile devices continues to be a scarce resource. Even in high-end $500+ devices, 64 MB of on board RAM is common, and virtual memory paging is uncommon due to limited storage. Running out of RAM often has an adverse effect on an experiment that must be detected and accounted for. In a recent experiment, we distributed 24 smartphones to participants at a major conference [18]. As in other mobile studies, the participants were instructed to use their device as they would normally. We observed frequent peaks in memory allocation due to the user using the camera, surfing the web, and checking emails. Insufficient memory during these allocation spikes translated into failed Bluetooth and WiFi scans, failed local database queries, and crashes in the experiment driver. In the short term, I expect that memory related problems will become a norm in mobile research. Actively monitoring memory levels during an experiment will become an invaluable tool to mobile researchers.

Persistent storage: The ability to write to a persistent file or database is a basic requirement of any research platform; a feature that may not exist on some devices. Many mobiles platforms prevent third party applications from accessing the file system. This has the obvious positive side effect of protecting the device from running out of space or overwriting critical files, but prevents basic logging of data. Mobile devices that cannot log data to persistent storage are practically useless as research platforms.

Location sensing: Low-power GPS receivers and cell tower/AP triangulation techniques [12] are now found in many mobile devices. Although location sensing comes with an energy cost, (and occasionally a monetary cost), location information is commonly used to provide context to a number of applications including: navigation, search, and social interaction. As these technologies continue to improve and scale to large numbers of devices, location sensing in mobile devices will undoubtedly become ubiquitous. As researchers, we require the means to enable and disable location sensing sub-systems and query the current location from within applications.

II.A. Other Platform Requirements

While this list covers most research application requirements, it clearly does not cover all of them. For example, in [23] the authors exploit audio APIs in Windows Mobile for proximity detection. In [13], combinations of vibration and audio are used to signal the presence of buddies. The accelerometer, a feature now found in every mobile platform, is used

	Android (Linux)	BlackBerry	iPhone	Symbian	Windows Mobile
Network scanning	●	○	●	∼	●
Interface selection	○	●	●	●	●
Bluetooth I/O	○	●	○	●	●
Interface control	●	●	○	○	●
Background processing	●	●	●	●	●
Energy monitoring	●	●	●	●	●
Power saving control	●	●	∼	●	●
Low-memory management	●	●	●	●	●
Persistent storage	●	●	●	●	●
Location sensing	●	∼	●	●	●

Table 1: Summary of mobile platforms requirements. Satisfied: ● Partially satisfied: ∼ Not satisfied: ○

in [8] to detect potholes while driving. Camera APIs are used [26] to provide local service discovery. Mobile social networking applications may also use the address book and other personal information APIs to infer trust and aid in content dissemination [1]. This list is only limited by the imagination of researchers. I encourage the community to submit additional requirements for inclusion in the online version of this survey.

III. Mobile Platforms

In this section I analyze the five most popular mobile platforms and their success in meeting our research needs. The results of this analysis are summarized in Table 1. This platform analysis focuses purely on the functionality provided by the OS. With the exception of the iPhone, each platform depends on the features of the physical device to implement its APIs. For example, devices without WiFi antenna cannot scan for WiFi APs, even though the platform may support it.

III.A. Android (Linux) [3]

Several Linux based platforms have recently emerged for mobile devices: Maemo [15], Openmoko [17], Qtopia [19], LiMo [14] and have been adopted by Nokia, Samsung, Motorola, and other manufacturers. However, none of them have garnered the attention and wide spread support that Google's Android has. Since Android's release it has become a popular mobile platform. Unfortunately, Android's current Java API does not live up to our needs. Android supports programmatic scanning for WiFi APs, and allows applications to keep the WiFi radio awake. However, Android does not allow applications to control or establish connections over a specific network interface. Bluetooth is also not supported in the initial (1.0) release of the Android SDK due to certification issues of its profile implementations. Android does provide memory and thread statistics and run-time memory tracing through its debugging sub-system. Location sensing using both GPS (if available) and cell tower triangulation are also present in Android.

Power considerations are built directly into the Android platform. While third party applications are able to query the battery level and AC charging state, they are also able to schedule and manipulate energy saving features. Applications may force the device to go to sleep or maintain a specific power level. In the context of performing measurements, fine grained control over power consumption could be useful.

Android provides background non-user interactive processing using a 'Service' model. Services are invoked by the Android OS, or through an alternate application. These services may run for "an indefinite period of time" with the caveat that they may be killed if the system is under "memory pressure." Although Android employs a LRU termination strategy and attempts to restart terminated services, it is important to keep this caveat in mind when designing experiments. To accommodate low-memory situations, Android provides two forms of persistent storage: an embedded SQLite database and conventional file I/O. These mechanisms can be used to store a Service's state and provide storage and logging to third party applications.

Android runs on top of the Linux 2.6 and is licensed under the GPL. Although Android currently lacks the complete set of ideal requirements, it is reasonable to assume that they could be written once the platform is fully released and open to the community.

III.B. BlackBerry [6]

Contrary to popular belief, BlackBerry is an open platform. Like Android, all BlackBerry applications are written in Java and run in a protected run-time environment. As a development platform, BlackBerry lives up to its reputation for robustness and reliability; it has a rich set of libraries, and allows applications to interoperate seamlessly. Unfortunately BlackBerry falls short of the ideal research platform. The platform does not allow programmatic scanning for cell towers or WiFi APs. A third party application may only retrieve data about the cell tower or AP that it is *currently* connected to. However, BlackBerry provides full support for the Java Community Process JSR 82 Bluetooth specification. BlackBerry applications can therefore scan for neighbouring devices, initiate service discovery, and connect to or create custom Bluetooth services. Unfortunately, before a BlackBerry can connect to another device over Bluetooth it must be securely paired. Bluetooth pairing on a BlackBerry requires the user to enter a key. For two BlackBerrys to communicate over Bluetooth they must enter the same shared key.

With the recent addition of WiFi, BlackBerry has added support for network interface specification when establishing socket connections. Moreover, applications are able to independently enable and disable the wireless radios, or *Wireless Access Families*, found on a device. The BlackBerry platform provides support for background processing, which can startup automatically when the device starts up. Energy monitoring and memory facilities on the BlackBerry are more refined than those on Android. The platform provides applications many APIs for querying the state of the battery, charging state, temperature, etc. Applications may also enable and disable the screen/backlight (another significant energy consumer), shutdown the device, and schedule it to wake up. Unfortunately, BlackBerry does not allow applications to programmatically control the GPS; this is probably due to privacy reasons. However, once the user has manually enabled the GPS, applications may retrieve the current location and receive notifications when the device is near a specified coordinate or predefined location.

As in Android, applications may directly query the remaining level of Flash memory and RAM. BlackBerry differentiates itself by allowing applications to interact and cooperate with the memory management facilities of the underlying virtual machine (VM). Memory intensive applications may register with the VM to receive low-memory warnings and requests to free memory; a form of participatory garbage collection. Unlike Android, the BlackBerry OS does not kill processes when the device runs low on memory; applications themselves terminate when they fail to allocate memory and an ‘OutOfMemoryError’ is thrown.

Persistent storage on the BlackBerry is sufficient for most research needs. The platform provides standard mechanisms for accessing files and manipulating the file system. When the device storage approaches full capacity, the platform throws an exception and simply prevents the application from writing further. The use of removable storage cards, such as MicroSD, has become common on most mobile devices including the BlackBerry. The card is typically accessed through the device as a USB Mass Storage device. This is the recommended approach to storing large amounts of data; however, using removable storage on the BlackBerry has one catch: enabling USB mass storage to access the card from a PC unmounts the card from the BlackBerry file system. This causes application writing to the removable storage card to immediately fail.

Developing on BlackBerry comes with the caveat that code making low-level API calls must be signed by Research In Motion. This requires a one-time purchase of a $20 ‘code signing PIN’. However, having the author’s identity strongly bound to the application binary provides accountability and security to the user.

III.C. iPhone (Mac OS X) [10]

Like the BlackBerry, Apple’s Mac OS X is tightly coupled with its device: the iPhone. The iPhone is an exceptional case in our survey. Out of the box, the iPhone and SDK provided by Apple are severely limited. The current SDK does not allow applications to initiate a WiFi network scan or retrieve information about neighbouring cell towers. The SDK allows applications to detect if the iPhone has WiFi connectivity; however, applications cannot transmit over a specific network. Moreover, Apple prohibits bandwidth intensive applications to be installed, nor can third party applications run in the background. The SDK does not allow applications to query the state of the battery or the level of available RAM. On the plus side, the iPhone SDK does provide persistent storage through conventional files and an integrated SQLite database. The platform also provides location sensing through the ‘Core Location Framework’, which determines its position using an integral GPS or cell tower/WiFi triangulation.

As a research platform, the functionality provided by the Apple SDK is insufficient. Fortunately, the iPhone has been liberated by members of the black hat community. Unlocking, or 'Jail Breaking', the iPhone takes a matter of minutes using tools such as 'iPlus'. After installing the GNU or BSD sub-systems [7], the iPhone has all the capabilities of a standard Unix system. The iPhone OS provides APIs for both cell tower and WiFi scanning, and network connections are made using conventional BSD sockets. However, I found no evidence in Apple documentation or in a survey of third party application that indicates that applications may enable and disable network interfaces. The iPhone also provides no support for Bluetooth I/O. As in previous systems, the iPhone OS provides APIs to query the battery capacity, voltage, and charging status.

Unlike previous systems, the iPhone has a virtual memory system with paging. Low-memory conditions are therefore less critical to the iPhone; however, one should be aware that frequent paging in conjunction with persistent flash memory I/O will have an adverse effect on battery life. iPhone is also far more aggressive in power management than other devices. By default, when a user is not interacting with the device, the screen is disabled and background processes are suspended. To run applications in the background of an unlocked iPhone, it is necessary to override the power saving setting and force the device to stay on.

Developing on an unlocked iPhone has initial challenges; documentation is generally poor and it will probably take several attempts to successfully unlock the device and install the desired sub-systems. The iPhone has a final caveat, future versions of the iPhone may restrict this openness, but given Apple's recent endorsement of iPhone unlockers [11], it is unlikely.

III.D. Symbian (S60) [25]

Holding (currently) 46.6% of the global smartphone market, Symbian OS is a sophisticated development platform supporting a wide range of languages, features, and hardware configurations. There are currently two major platforms built on top of Symbian: the S60 from Nokia and UIQ from Sony. Nokia's S60 is the dominant Symbian platform, with more features, and supports native Symbian C++, Java, PERL, and Python. For the purposes of this survey I ignore UIQ and focus on the features of Symbian with S60.

Symbian's support for network scanning is mediocre. Symbian applications can programmatically scan and retrieve information about neighbouring WiFi APs, but they cannot extract the same information from the cellular network. Like the BlackBerry, Symbian only allows applications to retrieve information from the single cell tower that it is connected to. Symbian OS sockets are similar to BSD sockets and can be configured to bind a specific network interface. If a connection is created without specifying the interface, Symbian will automatically select an available one. Symbian does not support programmatic control of interfaces, and enables the WiFi interface only when it is being used. Symbian also supports the JSR 82 Bluetooth specification and suffers from the same security feature as BlackBerry: Symbian does not allow Bluetooth I/O unless the devices are securely paired.

Symbian's support for GPS location sensing and persistent storage is consistent with previous platforms. Symbian provides conventional C style mechanisms for file I/O and includes an embedded relational database. Like iPhone, Symbian implements a virtual memory system and allows applications to query the physical memory levels.

Nokia's efforts in Power Management are notable. Of the five platforms surveyed, Nokia is only one with a developer tool, the 'Nokia Energy Profiler', that actively monitors the power consumption on the device. The tool is free to download, easy to use, but requires a Nokia device running the latest version of S60. On the device side, Symbian contains a framework for power management that allows applications to lock the power state, receive notifications of changes to power modes, adjust power requirements, or wakeup/shutdown the Symbian kernel.

Symbian also offers a Java runtime environment; however, unlike Android and BlackBerry, Symbian's Java environment is constrained to the Java 'Mobile Information Device Profile' (MIDP). Constraining Symbian Java applications to MIDP makes them highly portable, but the set of available APIs are very limited and do not satisfy the needs of mobile researchers.

III.E. Windows Mobile [28]

To my great surprise, Microsoft's Windows Mobile satisfies every one of our research requirements. The OS provides facilities to scan for cellular, Bluetooth, and WiFi networks, establish connections on a specific network interface, enable and disable interfaces, determine the current location using GPS, and to run applications in the background. Like the iPhone, Windows employs a virtual memory system and memory is therefore not a concern for applications. However, since Windows Mobile is commonly deployed on devices with less than 64 MB of total Flash memory, applications should be careful not to over indulge in memory.

Bluetooth on Windows Mobile has distinct advantages over BlackBerry and Symbian. The Windows Bluetooth API does not require devices to be paired! Although this ‘feature’ exposes Windows Mobile users to a range of interesting attacks, it has huge advantages when prototyping mobile systems: devices can connect to each other without user intervention.

Windows provides applications with standard file I/O facilities and an integral database engine. Applications may also maintain state in a persistent global system registry consisting of key pairs (a mechanism similar to the Windows registry). As a researcher writing experimental code, I have found that the registry is an invaluable tool for storing state across countless crashes.

As a change to the conventional approach, Windows allows applications to register for changes to the battery status. Applications are notified when, for example, the battery level changes or the device is plugged in for charging. Third party applications may also specify their own power requirements to prevent the device from entering a power saving state.

The downside of Windows Mobile is that Microsoft has no control over which components of its OS are supported by device manufactures (OEMs). Microsoft distributes a tool to OEMs called the “Adaptation Kit for Windows Mobile” that contains approximately 90% of the source code of the Operating System and drivers [16]. OEMs use the kit to customize the OS for their hardware. OEMs are also responsible for implementing their own network stacks and selecting the set of APIs that third party applications may call to access the network. This has translated into a cornucopia of devices each with a unique set of missing features and general instability.

IV. Summary and Conclusion

To the average consumer, each platform is functionally equivalent. They all support making phone calls, saving files, taking pictures, and other common tasks; however, there is more to a mobile platform than meets the eye. Each platform has its own development environment that supports different sets of programming languages and APIs.

Android has a rich set of APIs, but without Bluetooth it currently lacks the necessary requirements for multi-NIC related research. BlackBerry is the most robust mobile platform in our survey. It lacks some of the features of other platforms, but makes up for it with ease of development and stability. In five years developing applications for BlackBerry, I have never experienced a crash. Out of the box, iPhone is a substandard research platform; however, unlocking it exposes a rich set of APIs from its Mac OS X foundation. The unlocked iPhone currently lacks the developer tools found on other platforms, which makes it more difficult to develop and debug applications. Conversely, Symbian includes an excellent set of developer tools and the platform supports many of our requirements. Unfortunately, Symbian is widely regarded as the most difficult platform to develop on. This fact alone negates many of its qualities and hurts its viability as a research platform. Finally, Windows Mobile, running on a wide range of devices and hardware configurations, has a rich set of APIs; unfortunately, support for those APIs is OEM dependent. Moreover, there is little incentive for OEMs to support niche APIs such as programmatic power management and network scanning.

The awkward conclusion of this survey is that: *none of today’s mobile platforms fully meet the needs of researchers*. There is no single platform or device that can do everything reliably. The choice of platform boils down to the needs of the individual researcher; this survey should make that decision easier. Of course, this survey is time-sensitive. Its validity will decay as platforms evolve to meet the demands of consumers and new technologies emerge. I encourage others to contribute new ideas to the online evolution of this survey. Hopefully, as smartphones continue to proliferate and become an invisible tool in our lives, their software APIs will expand and their usefulness as mobile research platforms will increase.

V. Acknowledgement

My thanks Hossein Falaki, Usman Ismail, Stratis Ioannidis, Anna-Kaisa Pietiläinen, Nilam Kaushik, Aaditeshwar Seth, and my supervisor Prof. S. Keshav for their comments on this survey.

References

[1] G. Ananthanarayanan, R. Venkatesan, P. Naldurg, S. Blagsvedt, and A. Hemakumar. SPACE: Secure Protocol for Address-Book based Connection Establishment. 2006.

[2] Ganesh Ananthanarayanan, Venkata N. Padmanabhan, Lenin Ravindranath, and Chandramohan A. Thekkath. Combine: leveraging the

power of wireless peers through collaborative downloading. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 286–298, New York, NY, USA, 2007. ACM.

[3] Android. Last visited 12/11/2008. *http://code.google.com/android/*.

[4] Trevor Armstrong, Olivier Trescases, Cristiana Amza, and Eyal de Lara. Efficient and transparent dynamic content updates for mobile clients. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 56–68, New York, NY, USA, 2006. ACM.

[5] N. Banerjee, A. Rahmati, M.D. Corner, S. Rollins, and L. Zhong. Users and Batteries: Interactions and Adaptive Energy Management in Mobile Systems.

[6] BlackBerry. Last visited 12/11/2008. *http://blackberry.com/developers/*.

[7] Bringing Debian APT to the iPhone. Last visited 12/11/2008. *http://www.saurik.com/id/1*.

[8] Jakob Eriksson, Lewis Girod, Bret Hull, Ryan Newton, Samuel Madden, and Hari Balakrishnan. The pothole patrol: using a mobile sensor network for road surface monitoring. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 29–39, New York, NY, USA, 2008. ACM.

[9] Ken Hinckley, Jeff Pierce, Mike Sinclair, and Eric Horvitz. Sensing techniques for mobile interaction. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 91–100, New York, NY, USA, 2000. ACM.

[10] iPhone. Last visited 12/11/2008. *http://developer.apple.com/iphone/*.

[11] Jail-Breaking iPhones and Other Tales from the Apple Store. Last visited 12/11/2008. *http://www.xconomy.com/2008/03/25/jail-breaking-iphones-and-other-tales-from-the-apple-store/*.

[12] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter, et al. Place Lab: Device Positioning Using Radio Beacons in the Wild. *Proceedings of Pervasive*, 3468:116–133, 2005.

[13] Kevin A. Li, Timothy Y. Sohn, Steven Huang, and William G. Griswold. Peopletones: a system for the detection and notification of buddy proximity on mobile phones. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 160–173, New York, NY, USA, 2008. ACM.

[14] Limo Foundation. Last visited 12/11/2008. *http://limofoundation.org/*.

[15] Maemo. Last visited: 12/11/2008. *http://maemo.org*.

[16] MSDN Blog. Last visited 12/11/2008. *http://blogs.msdn.com/raffael/archive/2008/02/27/support-boundaries-for-windows-mobile-programming-developing-drivers-for-example-or-even-wifi-programming.aspx*.

[17] Open Moko. Last visited 12/11/2008. *http://openmoko.org*.

[18] Anna-Kaisa Pietiläinen, Earl Oliver, Jason LeBrun, George Varghese, Jon Crowcroft, and Christophe Diot. Experiments in mobile social networking. Technical Report CR-PRL-2008-02-0003, Thomson, February 2008.

[19] QTopia. Last visited 12/11/2008. *http://qtopia.net*.

[20] Ahmad Rahmati and Lin Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 165–178, New York, NY, USA, 2007. ACM.

[21] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A networking architecture designed around mobile users. *Proceedings of the Third Annual Conference on Wireless On demand Net work Systems and Services (WONS 2006)*, 2006.

[22] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya. A policy-oriented architecture for opportunistic communication on multiple wireless networks. 2006.

[23] Guobin Shen, Yanlin Li, and Yongguang Zhang. Mobius: enable together-viewing video experience across two mobile devices. In *MobiSys '07:*

Proceedings of the 5th international conference on Mobile systems, applications and services, pages 30–42, New York, NY, USA, 2007. ACM.

[24] Smartphones Showed Strong Growth in 2007. Last visited 12/11/2008. *http://www.palminfocenter.com/news/9617/smartphones-showed-strong-growth-in-2007/*.

[25] Symbian. Last visited 12/11/2008. *http://symbian.com/*.

[26] Eleanor Toye, Richard Sharp, Anil Madhavapeddy, and David Scott. Using smart phones to access site-specific services. *IEEE Pervasive Computing*, 4(2):60–66, 2005.

[27] Why the future is in your hands. Last visited 12/11/2008. *http://news.bbc.co.uk/2/hi/technology/7250465.stm*.

[28] Windows Mobile. Last visited: 12/11/2008. *http://msdn2.microsoft.com/windowsmobile/*.

[29] M. Zaharia and S. Keshav. Fast and Optimal Scheduling Over Multiple Network Interfaces. *University of Waterloo Technical Report, CS-2007-36*, 2007.