UNIVERSITY OF CALIFORNIA, SAN DIEGO

Building Aggressively Duty-Cycled Platforms to Achieve Energy Efficiency

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Yuvraj Agarwal

2009

The dissertation of Yuvraj Agarwal is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

DEDICATION

To Dadi, Shyam Babaji, Papa and Ma.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor Rajesh Gupta, who has been an amazing mentor to me. He has always believed in me and my work, and has constantly strived to let me find my own path and be an independent thinker. He has been exceedingly patient, repeatedly stressing on quality of work much more than quantity of publications, and that is something that will stay with me throughout my research career. I especially treasure his work ethic, his insightful comments, his insistence of building working prototypes, his encouragement in the face of hard decisions, and his willingness to foster collaboration with others without any hindrances or reservations.

My PhD research has benefited tremendously from the advice given by various key people. These include various faculty in the SysNet group at UCSD; specifically Alex Snoeren, George Varghese, Stefan Savage, Amin Vahdat and Geoff Voelker. Thank you for accepting me as a bonafide sysnet member, making me a part of the group and presiding over my thesis committee. Geoff and Stefan's introductory talk when I entered graduate school is something that I still remember to this day, especially the invaluable parts about – "It does not matter who you know, rather who knows you" and "You are responsible for publicizing your own work - go talk to people about it". I am also indebted to the various administrative staff in the CSE department and the folk at CSEHelp who handled basically everything behind the scenes. Thank you for making all the painful processes easy, always listening to my questions, working with me on my hair brained schemes and helping get things done. Additionally, I would not be where I am without the advice and guidance of my mentors on my short stints at various research labs. Roy Want and Trevor Pering at Intel Research taught me the importance of building testbeds, extensive evaluation and taking ideas into real prototypes. Ranveer Chandra and Victor Bahl at Microsoft Research, Remond always had the utmost faith in my abilities, and sometimes pushed me to do things that I did not think I was even capable of doing. Finally, I would like to thank Steve Hodges for making it one of my most memorable summers in Cambridge, for his infectious enthusiasm in everything he did, and always wanting and doing what is correct and fair.

I would like to thank the various members of my lab, starting from my early academic days at UC Irvine to the last five amazing years at UCSD. My entire graduate career has been an unforgettable journey because of these friends and colleagues. Cristiano Pereira for keeping it fun in the lab, Ravindra Jejurikar for his jovial and easygoing nature; Arijit Ghosh, Jean-Michel Rucheton and Oliver Schweizer for always being there, their camaraderie and making life in Irvine even possible. My transition to UCSD and San Diego was made much easier by the amazing colleagues and friends that I have made during the last five years, each of whom I will remember for special reasons. Dennis Koehler for always being there as a good friend. Patrick Verkaik for his wisdom and his willingness to help anytime, anywhere whether it meant listening to my endless ideas, doing proof reads of my papers or just sipping a latte; Sudipta Kundu for always keeping it fun in the lab and always being ready for a discussion; Gjergji Zyba for just being Gjergji. I could probably spend several pages with all the friends that I made since I came to the US that have made my time here unforgettable; Joel Coburn, Ryo Sugihara, Thomas Weng, Didem Unat, Marisol Chiang, Tobias Bang, Kristine Berg, Ola Millnert, Diwaker Gupta, Kashi Vishwanath, Harsha Madhyastha, Yu-Chung Cheng, John Mc-Cullough and Radhika Niranjan. The list of the timeless friends I made during my internships is also long; particularly Raj Bhan, Jurgen Van-Gael, Mallesh Pai and Krishna Ramachandran. The last seven years would not have been the same without the company of all of these friends.

Most importantly, I would like to thank my entire family for making me who I am. My parents, my brother Gautam, Shyam Babaji and Dadi have all been pillars of fortitude in my life. This PhD was only possible because of their constant support, guidance, understanding, love and best wishes. Thank you for pushing me this far and letting me fulfill my dreams by always believing in me. Finally, I would like to thank my lovely wife Gunjan that I had the fortune of meeting only because of my PhD at UCSD. Thank you so much for all your support and love, for your patience and kindness during all the stressful times, for always being ready to listen to my hair brained ideas, for smiling and making my day. This dissertation was not possible without you. I would also like to thank my wife Gunjan's parents and younger brothers who have accepted me whole heartedly into their family and have constantly encouraged me throughout the writing of this dissertation.

Chapter 3, in part, is a reprint of the material as it appears in Proceedings of ACM Mobile Systems, Applications and Services (MobiSys '07), June 2007. Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl and Rajesh Gupta. The dissertation author is the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in Proceedings of ACM Mobile Systems, Applications and Services (MobiSys '06), June 2006. Trevor Pering, Yuvraj Agarwal, Rajesh Gupta and Roy Want. The dissertation author is the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in Proceedings of IEEE International Symposium of Wearable Computing (ISWC '08), July 2008. Yuvraj Agarwal, Trevor Pering, Roy Want and Rajesh Gupta. The dissertation author is the primary investigator and author of this paper.

Chapter 6, in part, is a reprint of the material as it appears in Proceedings of USENIX Symposium on Networked System Design and Implementation (NSDI) 2009. Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Paramvir Bahl and Rajesh Gupta. The dissertation author is the primary investigator and author of this paper.

VITA

1997 - 2001	B. E. in Electrical Engineering, University of Pune, India
2001 - 2003	M. S. in Information and Computer Science, University of California, Irvine
2004 - 2009	Ph. D. in Computer Engineering, University of California, San Diego

PUBLICATIONS

Yuvraj Agarwal, Steve Hodges, James Scott, Ranveer Chandra, Paramvir Bahl and Rajesh Gupta,"Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage." In Proceedings of USENIX Symposium on Networked Systems Design and Implementation **(NSDI '09)**, April 2009.

Patrick Verkaik, Yuvraj Agarwal, Rajesh Gupta and Alex C. Snoeren, "SoftSpeak: Making VoIP Play Fair in Existing 802.11 Deployments." In Proceedings of USENIX Symposium on Networked Systems Design and Implementation **(NSDI '09)**, April 2009.

Yuvraj Agarwal, Trevor Pering, Roy Want and Rajesh Gupta, "SwitchR : Reducing System Power Consumption in a Multi-Clients, Multi-Radio Environment. In Proceedings of IEEE International Symposium of Wearable Computing **(ISWC '08)**, July 2008.

Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl and Rajesh Gupta, "Wireless Wakeups Revisited: Energy Management for VoIP over WiFi Smartphone." In Proceedings of ACM Mobile Systems, Applications and Services **(ACM MobiSys '07)**, June 2007.

Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, Roy Want, "*CoolSpots:* Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces." In Proceedings of ACM Mobile Systems, Applications and Services **(ACM MobiSys '06)**, June 2006.

Yuvraj Agarwal, Curt Schurgers and Rajesh Gupta, "Dynamic Power Management using On Demand Paging for Networked Embedded Systems." In Proceedings of Asia-South Pacific Design Automation Conference **(ASPDAC '05)**, Jan 2005.

ABSTRACT OF THE DISSERTATION

Building Aggressively Duty-Cycled Platforms to Achieve Energy Efficiency

by

Yuvraj Agarwal

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California, San Diego, 2009

Rajesh Gupta, Chair

Managing power consumption and improving energy efficiency is a key driver in the design of computing devices today. This is true for both battery-powered mobile devices as well as mains-powered desktop PCs and servers. In case of mobile devices, the focus of optimization is on energy efficiency to maximize battery lifetime. In case of mains-powered devices, we seek to optimize power consumption to reduce energy costs, thermal and environmental concerns. Traditionally, there are two main mechanisms to improve energy efficiency in systems: slowdown techniques that seek to reduce processor speed or radio power against the rate of work done, and shutdown techniques that seek to shut down specific components or subsystems – such as processor, radio, memory – to reduce power used by these components when not in use. The adverse effect of using these techniques is either reduced performance (e.g., increase in latency) and/or usability or loss of functionality.

The thesis behind this dissertation is that improved energy efficiency can be achieved through system architectures that seek to design and exploit "collaboration" among heterogeneous but functionally similar subsystems. For instance, multiple radio interfaces with different power/performance characteristics can collaborate to provide an energy-efficient wireless communication subsystem. Furthermore, we show that in systems where such heterogeneity is not naturally present, we can introduce heterogeneous components to improve overall energy efficiency. We show that using collaboration, individual subsystems and even entire platforms can be shut down more aggressively to reduce energy consumption, while reducing adverse impacts on performance or usability.

We have used collaboration to do energy efficient operation in several contexts. For battery powered mobile devices we show that wireless radios are the dominant power consumers, and then describe several techniques that use various heterogeneous radios present on these devices in a collaborative manner to improve their battery lifetime substantially, on average by two to three times and in some cases up to 8 times. First we present "Cell2Notify", a technique in which a lower power radio is used purely to wakeup a higher power radio. Next, we present "CoolSpots" and "SwitchR", systems that build a hierarchy of collaborative radios to choose the most appropriate radio interface, taking into account application characteristics as well as various energy and performance metrics.

In the case of wall-powered desktop and laptop Personal Computers (PCs) we show that the dominant power consumers are the processors themselves. We then describe "Somniloquy", an architecture that augments a PC with a separate low power secondary processor that can perform some of the functions of the host PC on its behalf. We show that by using the primary processor (i.e. the PC) collaboratively with the secondary processor we can shut down PCs opportunistically, and as a result reduce the overall energy consumption by up to 80% in most cases.

Chapter 1

Introduction

Energy efficiency has emerged as a key driver for the design of computing devices, which exist in a variety of forms ranging from mobile handhelds to desktop PCs and servers. Mobile devices by definition are untethered to energy sources and thus rely on batteries for operation. The usage models of these emerging mobile devices are increasingly diverse, ranging from watching videos, downloading content, browsing the web and checking email. In order to support these advanced use scenarios, manufacturers are constantly adding more functionality and integrating higher performance parts which until a few years ago were only available on laptop or desktop class computers. Unfortunately, battery storage technology has not advanced at such a rapid pace [62] and as a result the usage lifetime of these platforms is severely constrained by the capacity of their battery.

Recently, improving energy efficiency has become a key design consideration even for "mains-powered" platforms such as desktop PCs and servers as motivated by rising energy costs and environmental concerns. According to some estimates [67] the average annual cost to power IT equipment across the US alone is a staggering US$ 15 Billion per year, in addition to accounting for about 4.5% of the total energy requirement of the US. Incase of data centers, the operation cost of supplying power to and cooling servers has already surpassed the original capital cost of buying these computing machines [47, 50]. Furthermore, the increasing role of shifting computing resources to remote servers available on the Internet even for mobile applications, so called "cloud-computing", has accelerated the trend towards increased consolidation of computing in large data centers around the world. As a result the proportion of IT energy consumption to the total energy demand is slated to continue to grow rapidly.

While the motivations to improve energy efficiency for these different classes of computing devices – wall powered PCs and servers to battery-powered mobile devices – are different, the underlying goal of getting the most useful work done for the least amount of energy used remains the same. The challenging aspect of achieving energy efficiency is that despite various attempts at low power design techniques – at all levels, ranging from circuit optimizations to energy-aware applications — the need for power and energy keeps growing. Once all the low power design techniques have been exhausted, "duty-cycling" – that is shutting down components that do not need to be powered on – remains the only available mechanism of reducing power or energy consumption. There are however two challenges associated with duty-cycling. First we need to identify opportunities to shutdown both at an individual subsystem level and at the level of entire systems. Second, is to architect systems in a way such that they can be duty-cycled while satisfying important user and application needs for reliability, availability, responsiveness and performance.

This dissertation puts forth the thesis that duty-cycling or shutdown can be made much more effective by using "collaboration" between heterogeneous but functionally similar subsystems (e.g. multiple wireless radios on the same device), each with different power and performance characteristics. In this dissertation we will show that this heterogeneity is either already present in existing systems, or in cases where it is not can be easily added to aggressively duty-cycle individual subsystems or entire platforms.

This chapter begins with an overview of previous approaches explored for power and energy management. Next, it provides a detailed vision of our "collaboration" approach to improve duty-cycling and then presents our specific contributions. Finally it concludes by highlighting the organization of the rest of this dissertation.

1.1 Reducing the Energy Consumption of Computing Devices

Energy management at a high level can be modeled as a resource allocation problem. The goal is to allocate energy resources that are needed against work to be done, in a given amount of time. Allocating more resources than the demand incurs additional power costs and causes wasted energy. On the other hand if the work needed to be done exceeds the available resources, quality suffers. There are thus two primary ways to improve energy efficiency of a system: reduce demand by using low power components; and reduce wasted energy. The focus of this dissertation is on the latter.

For the purposes of energy management, we can think of a system as one that provides a "service" in response to the input "work requests". The incoming requests are placed in a queue, the occupancy of which can give an indication of the amount of work pending. To reduce the energy consumption of individual components or devices hardware manufacturers resort to utilizing advances in low-power circuit design [26, 30, 49, 64, 96]. As a result these devices can be in one of the many low **power states** that they support. A power state represents configurational choices and system operation or inoperation that corresponds to a specific level of power consumption. These low power states are either **active** power states, in which the device is powered on and functional, albeit at different power and performance levels; or **sleep** power states in which the device is essential off and cannot provide any functionality with the trade off being the latency and energy to transition back to an active power state.

Based on these power states the two basic mechanisms employed in systems to reduce energy consumption are defined as **"slowdown"** and **"shutdown"**. When utilizing slowdown, individual devices switch to their lower power consumption *active* state, in which a given level of performance can be maintained. For instance, a processor can operate at various clock frequencies, a disk can spin at different rotational speeds. In contrast, when using shutdown, individual devices switch to one of their *sleep* states, taking into account the power consumed in this sleep state and the latency to transition back to an active power state. The basic idea of switching to one of these lower power *active* states (slowdown) under low system utilization or alternatively to one of the *sleep* states (shutdown) have been applied to the processor subsystem [26, 49, 64, 71] and the storage subsystem [36, 54, 59] amongst others.

Consider the processor subsystem of a computing platform. Most modern processors [49, 64] exhibit slowdown in the form of dynamic voltage and frequency scaling (DVS/DFS) [26, 71] that allow the processor to be run at lower power active states by reducing the clock frequency and supply voltage (often called processor *P-states*) to save power. The processor subsystem also exhibits shutdown since processors can be switched to one of the sleep states (often called processor *C-states*) in which most of the parts of the processor are powered off [49, 64].

Individually, both slowdown and shutdown have their advantages. In the case of slowdown since only the active states of the system or the individual subsystems are used, the availability of the system remains largely unaffected, although performance levels may have changed. From the above example, the processor is still active in any of the DVS/DFS states. Energy savings when using slowdown are a function of the **slack** available in the system, i.e. the length of time a given amount of work can be delayed for, while still meeting deadlines. For CMOS devices, due to the quadratic dependence of delay on voltage, given extra time (slack) to execute a task, slowdown usually saves more energy than shutdown.

There are, however, limits to how much savings can be achieved from slowdown. First, these savings apply only to the speed proportional part of the energy consumption. As leakage – from devices or constant drain components that are independent of speed – increases, it reduces the effectiveness of slowdown. More importantly, it most real-life systems consisting of deep hardware and software layers, there is a significant amount of functionality needed to keep a device or machine active even if it is not doing any work. Therefore, often the lowest power *active* state consumes more power than the highest power consuming *sleep* state[64]. For this reason, and the fact that lowered duty-cycles of usage (or higher slack) can continue to reduce energy consumption, shutdown is particularly useful at the system

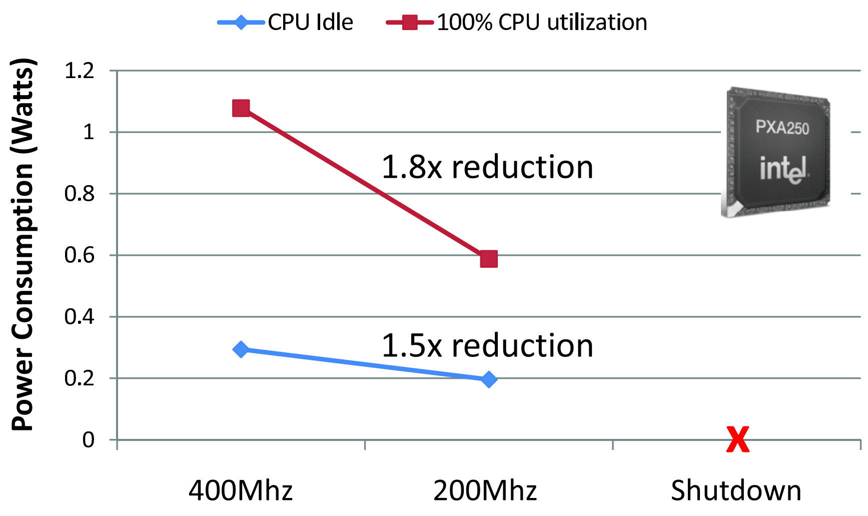


Figure 1.1: Power consumption of a Intel-PXA255 processor using slowdown and shutdown.

level. The disadvantage of shutdown, however, is that the additional savings over slowdown come at the cost of availability and loss in functionality. For example, when using shutdown to save power on the processor subsystem no computation can take place.

Figure 1.1 illustrates the power consumption of a processor used on mobile devices when exhibiting both slowdown and shutdown. As can be seen from the figure there is a 1.5X to 1.8X reduction in power consumption when the processor is run at 200Mhz versus 400Mhz (slowdown). In contrast, when using shutdown the processor power consumption is negligible ($<$ 1mW).

1.2 Using Collaboration to Aggressively Duty-Cycle Platforms

Ideally to get the most energy savings a combination of slowdown and shutdown techniques must be used. For example on a mobile device the wireless radio can be turned off when not in use to save power and a PC can be put into a sleep mode when not in use. Operating systems and device drivers already employ slowdown to an extent, shutdown however is employed to a lesser degree because it often runs into availability issues and usability problems. For example, when the wireless radio of a mobile device is turned off it does not receive any incoming email notifications. Similarly a PC in sleep mode cannot be accessed remotely, over SSH for example. The main challenge with using shutdown to duty-cycle platforms arises from the "Receiver-Side Problem (RSP)", defined as follows. A device (receiver) that uses shutdown to go into a low power *sleep* state, is essentially unavailable. Therefore, the receiver does not know when it needs to transition back to an *active* state to receive an incoming request, which can be asynchronous and hence can arrive any time.

In this dissertation we address the challenge of making duty-cycling of individual subsystems, or even entire platforms, more effective without adversely affecting functionality or requiring changes to user behavior. Our ultimate goal is to achieve the energy savings of shutdown, while achieving the availability of slowdown. To address the RSP described above, the idea explored in this dissertation is to duty-cycle higher-power consumption high performance subsystems and use lower-power albeit low performance alternatives in their place whenever possible. This approach of **collaboration**, can be applied at the level of subsystems or at the level of entire platforms. There are two requirements that must be met for collaboration to work. First, there must exist multiple instances of a particular resource or a subsystem that are **functionally similar**, ideally on the same platform. For example multiple wireless radios are often integrated on a single mobile device, all of which can essentially send bits across the air. Second, each of these resources must be **heterogeneous** to be useful. Specifically they must have different power versus performance characteristics to allow them to be used in a collaborative manner to save energy. For example having two 802.11 radios on the same device is not particularly beneficial from a collaboration standpoint since the radios will most likely have similar power and performance characteristics.

As it turns out, this requirement of functionally similar but heterogeneous subsystems is already met by various existing platforms. Examples include multiple heterogeneous radios present on emerging mobile devices, multiple display adapters on the same PC. Furthermore, we show that in cases where this heterogeneity does not exist, it can be easily added to existing platforms. For example, the addition of a separate secondary processor to existing PCs to allow them to be duty cycled.

1.3 Contributions

We have explored this notion of collaboration within several different contexts. For battery powered mobile devices we show that wireless radios — both the RF as well as the radio electronics — are the dominant power consumers and constitute a major portion of the total energy budget of the device. We describe several techniques that use the heterogeneous radios present on these devices in a collaborative manner to improve their battery lifetime substantially, in some cases up to eight times. The techniques proposed in this dissertation do not require any changes to applications or modification to the operating systems of the mobile devices, thus making them immediately deployable on existing platforms.

In the case of wall-powered desktop and laptop PCs we show that the dominant power consumers are the processors themselves. We describe an architecture that augments PCs with a separate low power secondary processor, which can then be used collaboratively with the the primary processor (i.e., the PC) to reduce the PC energy consumption significantly. Once again, the proposed architecture does not require any changes to the underlying network infrastructure and therefore can be deployed incrementally on existing PCs.

Overall, the collection of techniques proposed in this dissertation supports two major conclusions that are fundamental to achieving aggressively duty-cycled systems: (a) heterogeneity is essential to achieve energy efficient operation. In some systems, heterogeneity is part of the system architecture due to functional needs, and in others, heterogeneous components must be introduced to achieve the efficient operation; (b) the heterogeneous subsystems must be designed so as to be functionally similar but have very dissimilar energy/power usage curves as a trade off against a performance metric.

1.4 Organization

The rest of this dissertation is organized as follows.

Chapter 2 provides background information and introduces key concepts that are useful for this dissertation. It also discusses related work.

Chapter 3 introduces the idea of radio collaboration, especially in the context of voice as an application over emerging smartphone platforms. The key idea explored in this chapter is to reduce energy consumption of smartphones, by using a lower power radio to wake up a higher power radio, despite the two radios having drastically different ranges and characteristics. Since the higher power radio can effectively be duty cycled when not in use, substantial energy savings are enabled.

Chapter 4 builds a complete hierarchy of collaborating radios such that all radio interfaces can be used to send data traffic, rather than just provide wakeup. It presents a radio switching architecture that allows mobile devices to save energy by dynamically choosing the appropriate radio interface, taking into account application requirements and differences in the ranges of collaborating radios. Chapter 5 addresses the challenges associated with deploying a collaborative radio switching architecture within existing wireless infrastructures. Furthermore, we identify and address scalability challenges in the presence of multiple devices, all using radio collaboration simultaneously to save energy.

Chapter 6 applies the collaboration approach to improve duty-cycling of desktop and laptop PCs. It proposes an energy saving architecture that augments the network interface of a PC with a secondary processor; this augmented network interface can then act collaboratively with the host PC, and as a result enable the PC to be duty-cycled much more aggressively.

Finally, Chapter 7 presents future research directions and concludes with a summary of the dissertation.

Chapter 2

Background and Related Work

In this chapter we introduce some of the background material needed to navigate through this dissertation. In the first section we present characteristics of modern mobile platforms and highlight the dominant power consumers in these platforms with measured data. In the next section, we go over basic concepts of power and energy management and the various metrics used to compare energy efficiency. We also present the mechanisms that we use to measure power and energy consumption, in order to evaluate the various collaborative architectures that we have built. Finally in the last section we will go over the related work in the space of energy management, and specifically discuss techniques to reduce power consumption of mobile devices and desktop and laptop PCs. The related work specific to radio collaboration is discussed in Chapter 3 and 5, while the related work pertaining to processor collaboration is presented in Chapter 6.

2.1 Mobile Platforms

Modern mobile computing platforms, such as smartphones and PDAs, are evolving at a rapid pace into highly functional devices that routine support advanced usage models that sometimes rival those of laptop PCs. The variety of applications that are supported on these devices has increased at a rapid rate, especially since the introduction of the Apple iPhone and the phenomenal success of its corresponding “Application Store”. For consumers, these devices are now

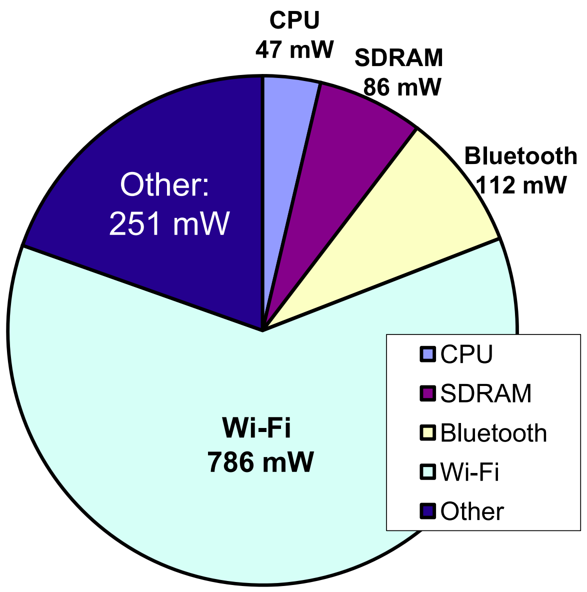


Figure 2.1: Power consumption of the Stargate[21] mobile platform

able to download games, browse the Internet, stream music and video, and play multimedia content either locally, or remotely using wireless connections to proximate environments such as the Digital Home [99]. In the case of enterprise users, these emerging platforms are rapidly becoming a key part in serving all communication needs, ranging from access to email and the Web to Voice-over-IP (VoIP) communication.

Since mobile devices are usually untethered, they rely on wireless radios for all their communication requirements. As a result these devices usually have *multiple* wireless radios integrated on the same device. These *heterogeneous* radios are present on these devices for very specific functions, for example communication with short range Personal Area Networks (PANs), to radios that provide access to Wi-Fi Local Area Networks (LANs) and cellular Wide Area Networks (WANs).

Currently, there are two dominant short range wireless standards that are frequently incorporated into mobile platforms: Wi-Fi and Bluetooth. Wi-Fi, or IEEE 802.11 offers high-bandwidth local-area coverage up to 100 meters. Bluetooth, intended for cell-phone class devices, is primarily a cable-replacement technology up to 10m and focuses on low-power consumption for handheld battery

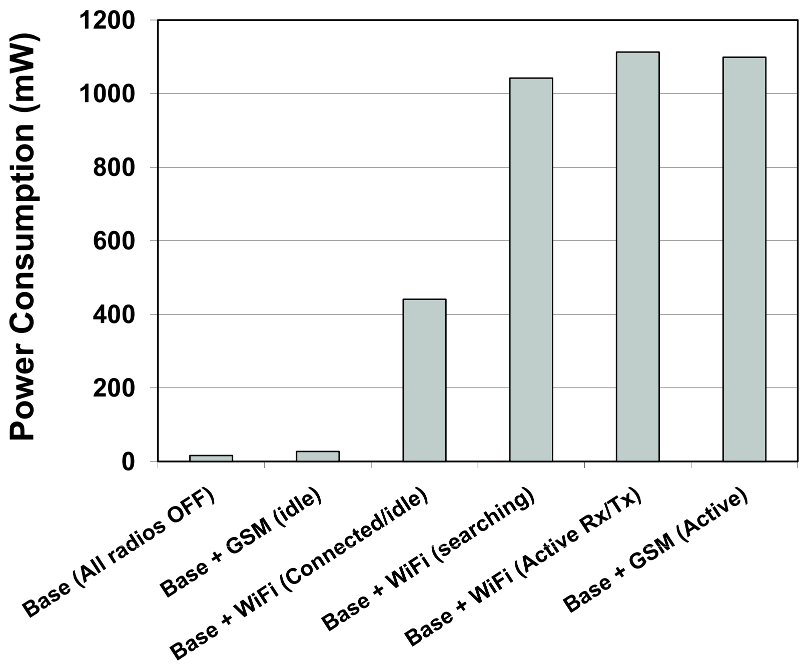


Figure 2.2: Power consumption of the HTC Tornado smartphone platform

constrained devices. The long range WAN interfaces found on these devices are usually based on one of the two prominent cellular radio technologies - either GSM or CDMA. These radios can be used to carry both voice traffic and data traffic. Emerging devices, such as the Apple iPhone, Palm Pre and the Nokia N series smartphones, already possess all of these technologies, allowing them to interface with a wide variety of wireless networks and peer devices.

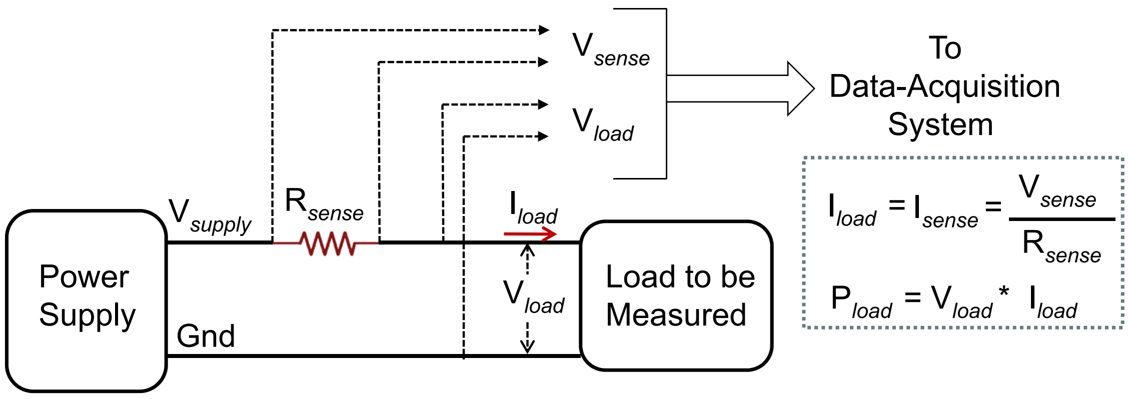
The downside of having these radios integrated on the same mobile device is that they become the dominant power consumers in these platforms and thus limit useful operating lifetime [92, 70, 72]. Figure 2.1, for example, shows the breakdown of power consumption of a mobile research platform [21], where radio power accounts for more than 50% of the total power consumed by the device. Similarly, Figure 2.2 reports our power measurements of a smartphone platform (HTC Tornado - also called Cingular 2125). The power consumed by various radios in different states of operation are shown in the figure confirming that radio power consumption is significant. This fact is not that surprising given the communication centric use patterns of these devices as was mentioned earlier. In the case of handheld mobile devices, such as smartphones, duty-cycling radios is therefore key to achieving overall energy efficiency.

2.2 Power and Energy

There are several metrics that are often used to compare and quantify the energy efficiency of systems. The first is to measure the power consumption, which is defined as the rate of energy usage and is measured in *Watts*. Typically power consumption is an instantaneous quantity, whereas the amount of power consumed over a particular length of time is the energy consumed measured in *Joules*. The storage capacity of batteries is often rated in terms of the amount of energy they can store, with values specified in terms of *Watt-Hour* or *mW-Hour*. For example a 500 mW-hour battery can supply energy for an hour to a device consuming 500mW of power, or for two hours if the device consumes only 250mW.

Each of these quantities are useful in specific contexts, to provide a fair comparison and be useful as evaluation tools. For example, the instantaneous power consumption of a device is important to consider when designing the batteries or the power supplies to make sure they can supply the peak power. In case of battery powered devices energy consumption or the average-power is usually the metric of choice as opposed to instantaneous power consumption values. For example, even though the *power* consumption of a processor on a particular CPU speed may be half than that at the maximum speed, it will still take more *energy* if it takes longer than twice the amount of time to do the same computation. For mobile devices, often the battery lifetime is the most useful measure since it is directly related to the average power consumption and the capacity of the battery.

Finally, in some cases evaluating the efficiency of a system by measuring just the power or energy consumption may give an incomplete picture, since any improvements in energy consumption may come at an associated cost in performance. As an example consider a system that uses a lower throughput radio interface over a higher throughput alternative, which may end up saving energy but also cause the application latency to increase. Some of these performance factors



V_{sense}
V_{load}
To Data-Acquisition System
$I_{load} = I_{sense} = \frac{V_{sense}}{R_{sense}}$
$P_{load} = V_{load} * I_{load}$
V_{supply}
R_{sense}
I_{load}
Power Supply
Gnd
V_{load}
Load to be Measured

Figure 2.3: Power measurement methodology.

although measured easily, for example latency, can also be subjective (i.e. based on user perception). This dissertation focuses on the quantitative and measurable performance metrics.

2.2.1 Measuring Power and Energy Consumption

Most modern handheld platforms and laptops feature smart batteries, which in addition to supplying energy also provide interfaces to query status information such as residual capacity, voltage across the terminals. Using these measurements the average power consumed by a battery device can be calculated, by checking the residual capacity of the battery and then dividing the energy consumed in a particular interval by the length of that interval. There are two issues with this approach. First, batteries are known to be non-linear and this method of calculating power consumption is not very accurate. Second, since we can only measure the total power from the battery, there is no easy way to differentiate power consumed by *individual subsystems*, such as the wireless radio or the processor.

We take an alternate approach to measuring power consumption. Figure 2.3 illustrates our power measurement setup. We first instrument the platforms such that we can get access to the power supplies to individual components or subsystems. We then instrument those power supply lines with sense resistors of appropriate known values (R_{sense}). Using data acquisition systems, which are essentially analog-to-digital converters, we measure and log the voltage drop across this sense resistor (V_{sense}) and the voltage supply to the component (V_{load}) at a high sampling rate (usually 1KHz or more). The current through the sense resistor, and also the actual component we are measuring, can be calculated as $I_{load} = I_{sense} = V_{sense}/R_{sense}$. Given that we know the current and the supply voltage to the component under measurement, the instantaneous power consumption of the component is given by: $P_{load} = V_{load} * I_{load}$. Using these fine grained measurements of power consumption, we can then calculate the energy consumed over any particular interval of time by integrating the instantaneous power consumption values.

2.3 Related Work

Power and energy management have been a topic of research for quite a few years, and as a result there is a large body of previous work in this space. Techniques have been proposed at all layers, ranging from building more energy efficient circuits and hardware, changing protocol behaviors, all the way to modifying applications and adapting them to make them energy-aware. In this section we first go over previous work for power management in battery powered handheld devices. We then discuss several techniques that have been proposed for energy management in desktop and laptop PCs.

2.3.1 Power Management in Mobile Devices

A majority of the techniques proposed within the context of mobile devices focus on optimizing the energy consumption of particular components or subsystems, such as the processor, display, memory, or wireless network interfaces.

For the processor subsystem, most techniques utilize Dynamic Voltage and Frequency Scaling (DVFS) where the processor frequency, and the corresponding operating voltage, is modulated to save power. Modern processors [49, 64] support multiple discrete voltage and frequency levels that can be set by the operating system. Furthermore, several research efforts have explored the algorithms to actually determine the appropriate operating points ranging from prediction based schemes [71] to complex machine learning based approaches [24] and finally techniques that use OS level measurements to infer application requirements [26].

Given the increasingly communication centric usage models of emerging mobile devices, the power consumed by wireless interfaces has started to dominate other subsystems. Previous research shows that in some cases the wireless power consumption can be more than 50% of the total energy budget of a mobile device [72, 81]. Since Wi-Fi networks have become ubiquitous, mobile devices routinely feature Wi-Fi interfaces. Unfortunately, Wi-Fi interfaces consume a significant amount of power and previous research has shown that most of this power is expended even when the interface is idle [90, 81].

Wi-Fi Power Save Mode: To remedy this high idle power consumption, the Wi-Fi standard specifies a Medium Access Control (MAC) layer power management scheme [37, 38] allowing mobile stations to conserve power by switching to low power modes. In the case of an infrastructure Wi-Fi networks the power management is centralized in the AP. A client's radio can be in an *active*, *idle* or *sleep* state. In the *active* state, the radio consumes the maximum power and is used for data transmission and reception. The *idle* state is when the radio is on but is not transmitting or receiving any data. The *sleep* state does not allow data reception or transmission and is a low power state consuming an order of magnitude less power than in the active state.

Using these states, the 802.11 standard specifies two modes of operation: Awake Mode (AM) or Power Saving Mode(PSM). In the case of AM the radio is continuously on and is in either the *active* or the *idle* state. This is a high power mode as the radio consumes power equivalent to transmission, reception or idle power. In PSM the radio switches between *active* state and *sleep* state to save power. The power figures for typical 802.11b cards are shown in Table 2.1. As can be seen from the table, although Wi-Fi PSM consumes less power than Wi-FI AM, the power consumption is still quite substantial.

Table 2.1: Measured power consumption for typical Wi-Fi cards

Vendor	*Average Power*		
	Idle(AM)	Idle (PSM)	Active
Cisco PCM 350	1300mW	390mW	1600mW
Linksys WCF12	690mW	256mW	890mW
Netgear MA701	780mW	264mW	990mW

At a high level the 802.11 Power Save Mode works as follows. Consider an infrastructure based wireless network comprising of various Access Points (APs) and 802.11 clients. When a client goes into PSM it informs its AP about it. The AP buffers messages for this client and at the beginning of a pre-determined interval called the *Beacon Period* (BP), the AP sends out a *Traffic Indication Map* (TIM) to let the various clients know whether the AP has data buffered for them. A client using PSM wakes up at the beginning of every *Listen Interval*, a multiple of the *Beacon Period*, and receives the TIM transmitted by the AP. If the client determines that the AP has data ready for it, it sends a PS-POLL packet to the AP. The AP then sends each buffered packet. If there is no buffered data the client just goes back to sleep.

Other power optimizations schemes at the MAC layer usually adjust one of the parameters of the Wi-Fi PSM. Krashinksy et al. [51] for example propose a bounded-delay addition to PSM that drastically reduces the incurred delay while maintaining power savings of Wi-Fi PSM. Bertozzi et al. [11] give a summary of the various transport protocol optimizations for energy savings in wireless embedded systems, in addition to proposing *lp-tcp* - a low power protocol that utilizes an optimization to the TCP transport protocol to save power. Techniques have also been proposed that use application level knowledge and adaptation [52] to reduce power consumption of the wireless interface. As expected all of the above optimizations and power saving schemes tend to do significantly better than 802.11 AM. These schemes are however unable to do substantially better than 802.11 PSM, as they all employ 802.11 PSM at the MAC layer to turn off the radio. As shown in the preceding section power consumption in PSM remains substantial and is mainly attributed to the complex MAC and supporting circuitry of the 802.11

radio.

2.3.2 Power Management in Laptops and Desktop PCs

Similar to handheld mobile devices, various techniques have been proposed to reduce the power consumption of laptop PCs focusing on individual subsystems such as the processor, the hard drive, and the display subsystem (LCD, backlight, graphics card). In the case of the processor, all the schemes utilize DVS/DFS [49, 64] to change the processor voltage and/or the operating frequency to save power [24, 26, 71, 100].

Strategies to reduce the disk power management aim to spin down the disk to lower rotational speeds [25, 54, 59] adaptively to save power, while attempting to reduce overhead due to spin up time. Techniques have also been proposed to augment disks with flash memory that acts as a buffer to reduce performance issues and keep disks spun down longer [61]. More recently manufacturers have started to use Solid State Disks (SSDs) which use flash memory chips for storage and as a result have lower power consumption due to the lack of any moving parts as compared to conventional disks.

Additionally, laptops and desktop PCs have standardized interfaces and specifications for system power management, which are widely supported by commodity operating systems. The Advanced Power Management (APM) specification [42] defines a standardized interface that is implemented by device drivers and the BIOS. Using this interface devices can be put into low power states based on timeouts, without any interaction with the OS. The more recent specification for power management in PCs is the Advanced Configuration and Power Management (ACPI) specification [1], which has in most cases superseded APM. ACPI specifies a more detailed power management interface to hardware devices, is platform independent, and is geared towards allowing OS-directed Power Management (OSPM). It is important to note that ACPI only specifies the *mechanisms* to control the power state of devices, while the actual *policies* are implemented and handled by the host operating system.

A wide body of related work has explored Dynamic Power Management

(DPM), which aims to design the policies that set the device states appropriately depending on various criteria [10, 9, 82, 25]. Lu et al. [58] introduce the idea of power managers that set the power states of devices appropriately based on their utilization. These DPM algorithms are based on either using simple heuristics such as time-out policies to set power states [25] or more complex prediction based strategies based on stochastic policies [82]. Recent work in this space has even proposed using machine learning techniques to guide power management decisions, by choosing among the most appropriate DPM policies [23, 24] that perform well under different workload conditions. The authors show that for a hard disk, wireless LAN interfaces [23] and for the processor [24] their online learning algorithm is able to converge to the best DPM policy under changing workloads.

In prior work several studies have provided a detailed breakdown of the power consumed by the various components of a laptop PC [17, 57, 60]. Mahesri et. al. [60] for example show that depending on the application workload the total power consumption of a laptop can vary significantly – from 8 Watts to 30 Watts – and more importantly individual components also exhibit large variations in power draw. Several observations can be drawn from their measurements: (a) the CPU is the dominant power consumer for CPU intensive benchmarks and, while DVS/DFS helps, CPU power remains significant; (b) the display subsystem (graphics power, LCD and the backlight) consume a significant amount of power and dominates when the CPU is idle; (c) network interfaces, graphics card, disk drives all consume power under load but are otherwise not dominant power consumers (d); there is a sizable component of unaccounted for power (14%-38%), that is consumed by the "rest of the system", i.e., components other than the processor, power supply, display, hard drives, LCD screen, optical drives and the network interfaces.

Most importantly their data [60] shows that a significant amount of **"base power"** is consumed even in the lowest power idle state of the laptop, with various power management options enabled such as DVS/DFS for the processor, backlight set to a minimum, hard drives spun down, etc. This base power, measured in an idle system, is still 9 Watts as compared to the 13 Watts consumed by the laptop without the DPM options enabled. This "base power" puts an upper bound on the amount of power savings that can be achieved by using DPM techniques since they can only leverage the low power states supported by hardware devices.

All of the power management techniques mentioned in this section are applicable to Desktop PCs as well. While the power consumption data in previous studies [17, 57, 60] is based on laptop PCs, the approximate percentages are similar in the case of desktops. The only exception is that although desktops PCs comprise of similar components as laptop PCs, they usually use higher performance parts (faster processor, hard disk, more memory) which in turn use more power. In Chapter 6 we provide detailed measurements for various laptop and desktop PCs.

Chapter 3

Radio Collaboration - Cellular and LAN Data Radios

In the previous chapter we presented power measurements for various mobile devices, showing that radios are the dominant power consumers in these platforms and thus are key components that need to be duty-cycled. In this chapter we will show how the approach of radio collaboration can be applied to a very diverse set of radios, such as a long range cellular voice radio and a shorter range Wi-Fi radio, to improve the battery lifetime of mobile device substantially.

Let us consider voice communication as an application. Voice traffic is traditionally carried over cellular networks for mobile devices, or the regular public switched telephone network (PSTN) in the case of landlines. However, Voice-over-IP (VoIP) services are rapidly gaining acceptance over these traditional circuit-switched voice communication networks. Although there are many reasons behind this transformation, the two most compelling reasons are lower costs, and new functionality that is difficult to achieve with traditional voice networks. In homes, providers such as Vonage and SunRocket provide very low cost long-distance and international calling services. Skype provides free calling to other Skype users and only charges for calls to users outside the Skype network. In enterprises, VoIP enables new functionality, especially when integrated with Wi-Fi networks: *VoIP over Wi-Fi* adds support for mobility and therefore allows incoming phone calls to be automatically routed to a user's VoIP phone, regardless of where that user connects to the network. Other functionality benefits include integration with network services such as address books, file exchange in parallel with voice conversations, presence notification, video conversations, and call logging.

Simultaneously, emerging devices such as *Smartphones* are rapidly gaining popularity. Smartphones integrate the functionality of PDAs and mobile phones into a single device. They typically run a full-featured operating system, such as Windows Mobile, MacOS or Linux, and most recent smartphones are equipped with multiple wireless network interfaces, such as Wi-Fi and cellular interfaces (GSM or CDMA). As smartphones become ubiquitous, users will demand the ability to use a single device for all their telephony needs. They will use their smartphone as a cellular phone primarily when on the road, and they will use it primarily as a VoIP phone when at work or at home. Therefore, VoIP over Wi-Fi has emerged as a critical application for smartphones. Vendors such as T-Mobile have recognized this trend and are rolling out new functionality that enables the handoff of calls between their GSM networks and their Wi-Fi networks [94].

One critical issue that presents a barrier to the widespread adoption of VoIP on smartphones is that of high energy consumption. In order for smartphones to receive VoIP calls over the Wi-Fi network interface, that interface needs to be on continuously. Unfortunately, as mentioned earlier (Chapter 2), the energy consumption of Wi-Fi interfaces when there is no data transfer taking place is comparable to that of when the interface is active. Furthermore, as we will demonstrate in Section 3.2, the energy consumption of the idle Wi-Fi network interface, even with 802.11 power save mode enabled, vastly exceeds the energy consumption of the smartphone's GSM radio in its idle state. The better energy consumption of the GSM interface is achieved by rapidly duty cycling the GSM radio with predictable timing (based on a TDMA MAC protocol), in addition to tight integration with cellular base stations. In contrast, Wi-Fi uses a distributed MAC (CSMA/CA) protocol where devices must contend for access to the wireless medium thus leading to increased energy consumption due to excessive listening for traffic from other nodes.

In this chapter, we present *Cell2Notify*, an energy management architecture that leverages multiple radios on a smartphone platform and use them **collaboratively** to reduce the idle energy consumption of the Wi-Fi radio. Cell2Notify attempts to minimize energy consumption by powering off the Wi-Fi interface when no VoIP call is in progress, and powering it on only on the reception of an incoming VoIP call. To provide the wakeup mechanism for the Wi-Fi interface, we utilize the voice services of the GSM radio. An incoming ring over the GSM channel, combined with a unique caller-ID of that incoming call, serves as a unique identifier such that the smartphone can distinguish between a *wakeup* ring and a regular incoming phone call over the GSM interface. Upon reception of a wakeup ring, the smartphone powers on the Wi-Fi interface and then receives the actual incoming VoIP call.

Previous research efforts [81, 5] on energy management have proposed radio collaboration to provide wireless wakeups: powering off a higher power radio and using a lower power radio to wake it up whenever needed. Cell2Notify is a continuation along this line of research, with two important distinctions. Previous approaches [81, 5] have faced significant barriers to deployment due to the substantial infrastructure modifications needed, whereas Cell2Notify simply requires software changes on the smartphone devices and on the VoIP proxy. No changes are needed to the VoIP protocol (in our case SIP [76]), and no additional hardware infrastructure needs to be deployed. Moreover, Cell2Notify is targeted at a specific compelling application of VoIP over Wi-Fi.

In this chapter we present the design and implementation of Cell2Notify. We have implemented Cell2Notify on Asterisk, a commonly available open-source SIP proxy, and on a Windows Mobile smartphone platform. Our measurements show that the additional latency imposed by our wakeup mechanism is less than two rings. Based on call logs from cell phones and office phones, we estimate that Cell2Notify can extend battery lifetime of a typical smartphone device by a factor of 1.7 to 6.4. We show the ease of Cell2Notify deployment by demonstrating a working prototype using existing cellular networks and an enterprise Wi-Fi network – we did not require any infrastructure changes to these networks, nor any cooperation from network administrators.

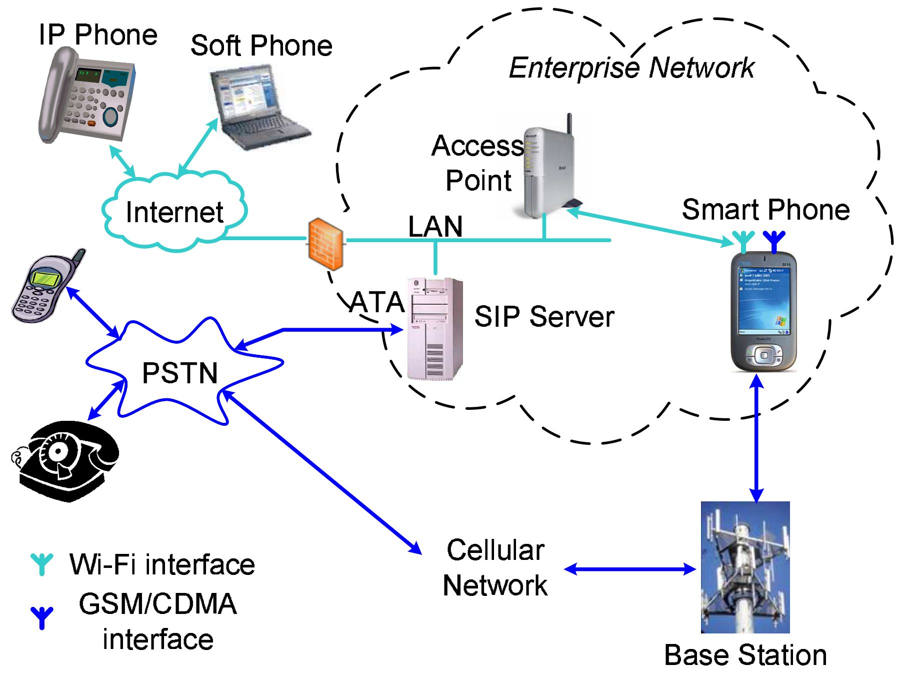


Figure 3.1: A typical enterprise VoIP deployment. Incoming calls to the SIP server can be received over the IP network or over the PSTN line. An Analog Telephony Adapter (ATA) acts as a bridge between PTSN and IP networks.

3.1 Overview of a VoIP Deployment

VoIP enables voice communication over IP-based networks, such as enterprise LANs or WANs as well as the Internet. VoIP protocols digitize voice into packets, and then send them using standard IP routing. Since VoIP does not require a dedicated and complex switching infrastructure as the PSTN does, it is much cheaper to deploy. It can also provide enhanced data services, such as video conferencing at a much lower cost. In the rest of this chapter, we mainly consider VoIP in enterprise LANs, although our protocols can be easily extended to work over the Internet.

A typical enterprise VoIP deployment is illustrated in Figure 3.1. The primary components of any VoIP deployment are a VoIP proxy server, VoIP enabled soft phones, and a VoIP gateway. The soft phones are PCs, PDAs or smartphones that are running software codecs and digitize voice packets. The VoIP proxy server acts as a rendezvous point for VoIP connections. It uses standardized signaling protocols, such as SIP [76] or H.323 [79], to establish a VoIP call between the calling parties. Once the call is connected, it is completed in a peer-to-peer fashion between the calling parties, without routing via the VoIP proxy. A typical VoIP deployment also integrates with the PSTN using a VoIP gateway. The gateway usually has an Analog Telephony Adapter (ATA) that bridges the calls between the IP-based LAN and the PSTN. In scenarios where one calling party is on the PSTN, the VoIP gateway server also plays the role of a VoIP endpoint. The VoIP proxy and the VoIP gateway services are often implemented by the same server.

One of the most popular standards used in VoIP deployments is the Session Initiation Protocol (SIP). SIP [76] is a transport independent application-layer protocol that provides a framework for inviting end-hosts into a conversation. Similar to HTTP, SIP is a text-based protocol which makes it extremely simple, efficient and extensible.

The widespread deployment of enterprise Wi-Fi networks adds an interesting dimension to VoIP in terms of support for mobility. An employee with a Wi-Fi VoIP phone can receive calls when working in a conference room or a colleague's office without relying on explicit call forwarding.

3.2 Alternatives to VoIP over Wi-Fi Radios

In this section, we look at energy consumption and data transfer characteristics of different wireless interfaces. We investigate how these characteristics impact the selection of the best wireless interface to use for VoIP. In particular, we study the characteristics of two cellular data networks (GPRS/EDGE and 1xEVDO), as well as Wi-Fi radios. We then profile the energy consumption of the entire smartphone device while performing various tasks to motivate the need for our Cell2Notify system.

3.2.1 Cellular Data vs. Wi-Fi

Since cellular radios are typically highly optimized to save energy, one possibility for making VoIP calls is to use a smartphone's cellular data connection. We performed a set of measurements to investigate the feasibility of this alternative, and found that the cellular radio consumes significantly more power when used for data transmissions, even more so than the Wi-Fi interface. In this section, we present experimental results to show the energy consumption of two popular cellular data connections: GPRS/EDGE and 1xEVDO, and compare these numbers with the energy consumed over Wi-Fi. To the best of our knowledge, this work is the first to compare energy consumption of these wireless interfaces when used for VoIP communication.

We measured the energy consumption when accessing two different cellular data network technologies prevalent in the US, namely GSM and CDMA. The GPRS/EDGE data service is based on the GSM technology, and is offered by providers such as Cingular and T-Mobile. The 1xEVDO data service is based on CDMA and is offered by Verizon and Sprint. Since it is difficult to obtain accurate power measurements from a smartphone as we demonstrate in the next section, we used PCMCIA cards from Verizon and Cingular inserted in a laptop to obtain power measurements. For Cingular, we used the Sony Ericsson GC83 card to access their GPRS/EDGE network, and for Verizon we used the Verizon V620 card to access the 1xEVDO network. For both networks, we obtained good signal strength in the lab where we performed the experiments. For our Wi-Fi measurements, we used the commonly available Netgear WAG511 802.11a/b/g cardbus adapter.

To measure the power consumption of the various wireless radios, we plugged them into our laptop using a PC card extender device. The extender exposes various pins that help us in measuring the power used by the the PC card. The power measurement setup that is described in detail in Chapter 2. We measure the power consumed by each wireless card in three different states of operation. The first is the "not connected" state, in which the cards were not connected to the data network. This corresponds to the "not associated" state for a Wi-Fi card. The second is the "connected and idle" state, in which the cards are connected to

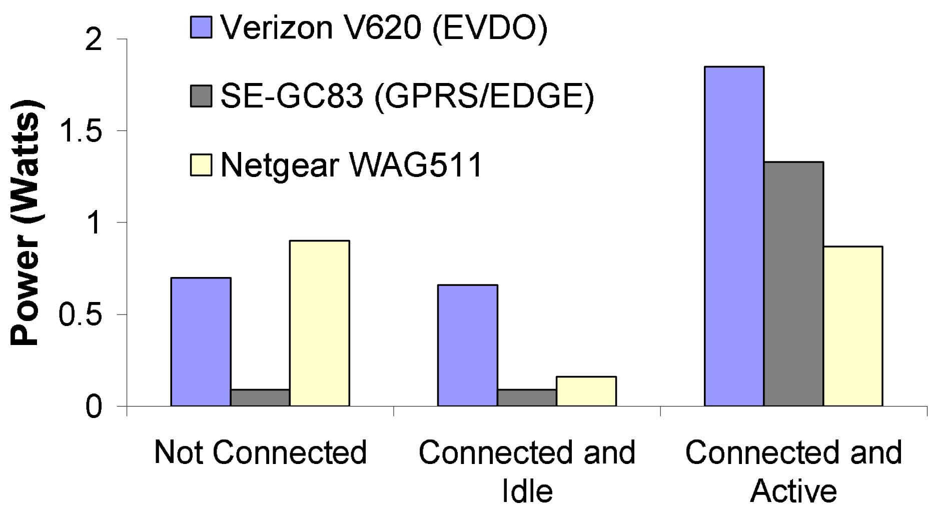


Figure 3.2: Power measurements of 1xEVDO, GPRS/EDGE and Wi-Fi interfaces for different scenarios. The “Connected and Active” measurements show the power when transmitting 32 Kbps of VoIP traffic over UDP. Note that when active, VoIP over Wi-Fi consumes the least amount of battery power.

the data network but not sending any traffic. The third state is the “connected and active” state, where the card is connected to the network and is sending and receiving VoIP traffic over UDP. In our experiments, we used the popular g729 VoIP codec[97], which generates 50 byte VoIP packets at a data rate of 31.2 Kbps. The power measurements for various states of the cards are reported in Figure 3.2.

As shown in the Figure, the power consumption of the V620 (1xEVDO) card is quite substantial in both the “not connected” and the “connected and idle” states. The SE-GC83 (GPRS/EDGE) interface consumes much less power in those states. The V620 utility actively tries to search for the data network, and shows the signal strength of the network even in the “not connected” state. Furthermore, it sends periodic keep-alive messages in the “connected and idle” state, and consumes significant power. On the other hand, the SE-GC83 utility does not connect unless asked to do so, and stays in a low power state when “connected and idle”. Another interesting fact that is unique to the 1xEVDO radio is that the energy consumption is not as much dependent on the number of packets sent on the network as it is on the fact that the interface is switched

Table 3.1: VoIP quality over different network interfaces.

Interfaces	Jitter (ms)	Packet Loss (%)
Verizon V620	25.25	7.6
Cingular SE-GC83	17.24	18.935
Netgear WAG511	0.9745	0

on. This can be seen from the power consumed in the "not connected" and the "connected and idle" states for the 1xEVDO interface, which are similar. Further, the 1xEVDO interface incurs a significant overhead in power, latency and network resources when the radio is woken up from sleep mode. Consequently, the 1xEVDO interface uses conservative policy to decide when to enter a deep sleep mode. Note that the Wi-Fi card consumes the most power when it is not connected, as it keeps scanning for available wireless networks. The power consumption reduces significantly when the card is connected (associated) as it enters IEEE 802.11 Power Save Mode (PSM) [37, 38].

Amongst the three interfaces, Wi-Fi is the *most power efficient radio during an active VoIP call.* It consumes less than half the power of the V620, and less than 75% of the power consumed by the GPRS/EDGE radio. This can be explained by the high transmit power used by the cellular radios to send data over much longer distances (sometimes even miles) compared to Wi-Fi, whose nominal range is 100 meters. This is exacerbated by the strict real time requirements for VoIP and a short inter packet generation time, as a result of which the cellular radios have no opportunities to sleep and save power.

In addition to high power consumption, the performance of current cellular data interfaces is also not well suited for real-time applications, such as VoIP. We measured two metrics, jitter and loss rate, which are usually associated with the quality of a VoIP connection, and we present those results in Table 3.1. All three interfaces had reasonably good connections to their respective networks for these jitter and packet loss measurements. The results show that the quality of VoIP calls is much better over the Wi-Fi connection than over the cellular data networks. In fact, the high packet loss over the cellular data interface makes voice traffic intolerable.

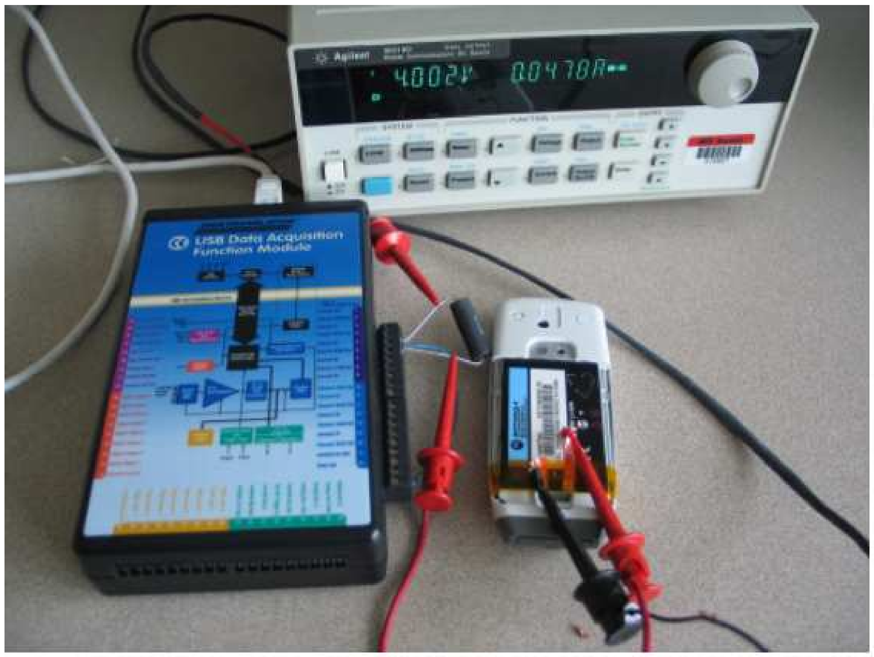


Figure 3.3: Smartphone power measurement setup

There are several other reasons why the cellular data network is not ideal for VoIP traffic in an enterprise. The costs are higher, because all employees (or the enterprise) need to purchase a cellular data plan, and these tend to be expensive. In most cases, this needs to be an unlimited data connection since VoIP calling generates a significant amount of traffic. The enterprise also has no control over calls using this approach, since the first hop from the smartphone is the cellphone carrier. Consequently, it is extremely difficult to implement and manage any call handling system. Given the above factors, we conclude that it is preferable to use Wi-Fi for VoIP instead than a cellular data network.

3.2.2 Smartphone Power Measurements

We now measure the power consumption of a popular smartphone, the HTC Tornado (Cingular 2125). This device has an ARM TI 195 MHz processor, runs Windows Mobile 5.0 and has a TI-1100 802.11g Wi-Fi chipset. We subscribed to the Cingular voice plan for our experiments. We measured the power consumption of the smartphone for various states of its network interfaces, i.e. GSM and Wi-

Table 3.2: Power consumption of the Cingular 2125 smartphone for different states of its network interfaces.

Scenario	Power
All Radios off (Flight Mode)	15.688 mW
GSM Idle	27.38 mW
Wi-Fi (searching)	1042.44 mW
Wi-Fi (connected)	441.82 mW
Wi-Fi (send/recv)	1113.811 mW

Fi, and we show that Wi-Fi is a major power drain if it is in the ON state at all times. These power consumption numbers are also used to evaluate our Cell2Notify protocol.

Since we were unable to get access to the individual power supply lines of each radio, we used a technique described in [31] to measure the power consumption of a smartphone. First, we fully charged the battery of the smartphone and then removed the battery from the device for an hour. We then connected a 0.5 ohm sense resistor in series with the battery of the device, and measured the instantaneous current across the resistor at 50Khz using a data acquisition system. Our measurement setup is illustrated in Figure 3.3. We repeated this procedure for each of our experiments. All our experiments lasted five minutes each. The power consumption of the smartphone can be calculated by multiplying the current with the average supply voltage of 3.7 Volts from the battery. The talk time for the Cingular 2125 is rated at 4 hours. With its 1150 mAH battery, this corresponds to a power consumption in an active cellular voice call of approximately 1150*3.7/4 = 1063.75 mW.

We present the measured results in Table 3.2. In each of our experiments, we measure the total power consumption of the smartphone, not just the power consumption of the interface. We set beaming to off, the backlight timeout to five seconds which is the minimum possible, the display timeout to 1 minute (also the minimum possible), the light sensor to off, and the earpiece volume to the minimum value.

As seen from this table, the smartphone expends very little battery power to keep its GSM interface on when it is connected. However, it consumes much more battery power when its Wi-Fi interface is on. Note that the Wi-Fi card was using the 802.11 PSM. Even when the Wi-Fi radio is idle, the device power consumption is more than 15 times than that in GSM idle mode. These numbers indicate that the total lifetime of a smartphone can be significantly increased if the Wi-Fi radio is turned off most of the time.

3.3 Cell2Notify Architecture

Cell2Notify increases the battery lifetime of smartphones by disabling the Wi-Fi radio when the user is not making a VoIP call. It enables the Wi-Fi interface only when either the user wants to initiate an outbound VoIP call, or when the user is receiving an incoming VoIP call. In the latter case, Cell2Notify sends a wake up signal to the smartphone as a *ring* on the cellular interface (either GSM or CDMA). As noted in Section 3.2.2, the cellular interface consumes significantly less energy than the Wi-Fi interface when not in use, and users rarely disable it. Consequently, Cell2Notify results in significant energy savings when using smartphones for VoIP over Wi-Fi.

The design of Cell2Notify poses two primary challenges. First, the system needs to be easily deployable. Therefore, it should not require changes to the standardized protocols used by VoIP phones. Furthermore, Cell2Notify cannot require extensive changes to network infrastructures it relies upon – neither the Wi-Fi infrastructure nor the cellular infrastructure. Second, disabling the Wi-Fi interface should not result in dropped calls nor significant delays. Cell2Notify must enable the Wi-Fi interface and complete the VoIP call within a reasonable amount of time. Finally it must handle scenarios where the user is an area that lacks either Wi-Fi or GSM coverage.

The Cell2Notify architecture addresses these challenges by requiring minimal modifications to the VoIP architecture illustrated in Figure 3.1. Cell2Notify only requires software changes at the VoIP proxy server and on the smartphone devices. Furthermore, all the software changes are implemented at the user-level, and hence are easily deployable without any modifications to the OS. Our pro-

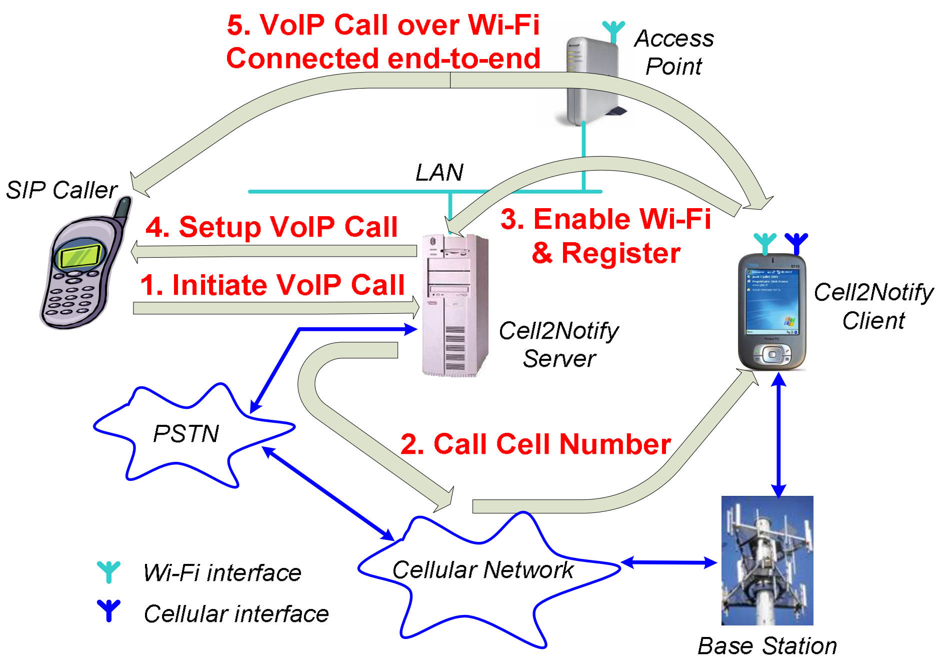


Figure 3.4: Steps of the Cell2Notify protocol.

totype system works with the Session Initiation Protocol (SIP) [76], which is the most commonly used protocol to set up VoIP sessions. All our changes at the proxy server are to the SIP proxy's configuration files, which allows Cell2Notify to be deployed incrementally within existing VoIP proxies. Our system incurs acceptable call setup latencies, and we devise simple protocols to handle scenarios where the user is out of range of either the cellular or Wi-Fi network.

The two new components introduced by Cell2Notify to an existing VoIP system are shown in Figure 3.4. We enhance the VoIP proxy server of a traditional deployment with additional call handling rules, and call it the *Cell2Notify Server*. The Cell2Notify Server also maintains a table that contains the mapping of users (VoIP extensions) to their corresponding cell phone numbers. The other new component in the Cell2Notify system is the *Cell2Notify Client*, which is an existing smartphone running our user-level service. Our service handles notifications sent by the Cell2Notify server. Our architecture is described in detail in the rest of this section.

3.3.1 Cell2Notify Protocol

The main steps of the Cell2Notify protocol are illustrated in Figure 3.4. *Registration*, which is not shown in the figure, is required before a device can utilize this architecture. In the Registration step the network administrator adds a new smartphone to use the VoIP system. During registration, the Cell2Notify server adds a mapping of the smartphone's VoIP extension to its cell phone number. The server also generates a unique Caller-ID (UID) that it will use as the Caller-ID when calling the smartphone to initiate a wakeup. The UID is 10 digits long, and its first digit is set to 0 to prevent collisions with existing phone numbers. This scheme provides basic security against Caller-ID spoofing. Since this UID is randomly generated and is different for different extensions, it is not trivial for attackers to send spurious wakeup calls. We also present a security enhancement to this basic scheme in Section 3.6.2. Finally, the smartphone is updated to set the VoIP extension and to store the UID that will be used by the server to contact it.

The Cell2Notify client disables its Wi-Fi interface whenever it receives a good signal from a cellular base station. When an incoming VoIP call arrives at the Cell2Notify server (Step 1 of Figure 3.4), the server looks up the client's extension in its table and retrieves the corresponding cell phone entry. The server then initiates a call to the client's cell phone number over the PSTN using an ATA (Step 2). Recall, the ATA is used to bridge between the IP and PSTN/Cellular networks. When the Cell2Notify client receives this call, our user-level service traps the Caller-ID, and checks to see if the Caller-ID matches the Cell2Notify server's UID. If the Caller-ID does not match the service allows the call to proceed on the device as a regular call. However, if the Caller-ID does match the server's UID then the service enables the Wi-Fi interface (Step 3). The client associates with a Wi-Fi Access Point (AP) and registers its IP address with the Cell2Notify server. The server can subsequently set up the VoIP call (Step 4), by sending the Cell2Notify client's credentials to the caller. The call is finally carried out end-to-end between the two devices without going through the server (Step 5). After the VoIP call ends, the Cell2Notify client disables the Wi-Fi interface.

We note that after Step 1, if the Cell2Notify server does not find a cell phone number corresponding to the client's extension, it simply proceeds to handle it as a regular SIP server. In other words, it attempts to set up the call if the client has previously registered, and otherwise it will send back a busy tone. Similarly, if after Step 1 the Cell2Notify server finds that the client has already registered, it attempts to setup the call as a regular SIP server, i.e. it directly calls the client's VoIP number.

3.3.2 Connectivity Scenarios

Cell2Notify needs to robustly handle situations where either the cellular network or the Wi-Fi network becomes unavailable. In these situations, our goal is to perform at least as well as a legacy VoIP deployment that does not use Cell2Notify. In this section, we enumerate the connectivity possibilities and describe the system behavior in each of those situations.

Registered Client, in Wi-Fi, Cellular Range

This is the ideal case for our protocol. The smartphone is in range of a known Wi-Fi network and has good cellular coverage. It has also previously registered with the Cell2Notify server, and its DHCP lease has not expired. Moreover, it has not moved recently, so it has cached state of the nearby APs. When someone calls the client, the Cell2Notify server sends a wake-up call on the cellular interface. The smartphone then enables its Wi-Fi interface, connects to the AP whose information it has cached, and sends a SIP register message to the Cell2Notify server. The server then connects the VoIP call over the smartphone's Wi-Fi interface.

Unregistered Client, in Wi-Fi, Cellular Range

In this scenario the client is in a Wi-Fi zone but has not yet connected and registered. In comparison to the previously described case, there is an extra step involved. Upon receiving the wakeup call over the cellular interface from the Cell2Notify server, the device enables its Wi-Fi interface and performs a scan to look for available APs. The rest of the steps are similar to the previous scenario. To address this case, the Cell2Notify server attempts calls to the client's SIP extension multiple times to allow enough time for the mobile device to look for available Wi-Fi APs.

Client in Cellular Range, out of Wi-Fi Range

We now consider the case where a client is not in a Wi-Fi zone. When the Cell2Notify server sends a wake-up call over the cellular interface, the device enables the Wi-Fi interface and scans for wireless networks. Since there is no wireless network available in this case, the Cell2Notify client never sends a SIP register back to the Cell2Notify server and eventually turns its Wi-Fi interface off to save power. To handle this scenario, we use a relatively long timeout value at the proxy. If the proxy cannot connect the call to the mobile device it has several options. Based on user preference, it can either forward the call on the regular cellular line after resetting the Caller-ID to the correct Caller-ID (not the UID), or it can request that the caller leave a voice mail. The first option will complete the call, although the call setup will incur extra latency equal to the timeout value of the SIP server. These options can be configured as part of the call handling rules (described in Section 3.3.3) for the VoIP extension of the smartphone, and can be customized based on user preference.

Client out of Cellular Range

Cell2Notify is based on two key properties of the cellular networks: low power consumption of the cellular radio and near ubiquitous connectivity. However in the rare case that there is no cellular coverage, our user-level service on the smartphone automatically enables the Wi-Fi interface and registers with SIP on the Cell2Notify server. At this point, the Wi-Fi interface only uses IEEE 802.11 PSM [37, 38] to save energy. As soon as the Cell2Notify client detects cellular coverage, it sends a SIP de-register message and turns off its Wi-Fi interface. At this point it reverts to using Cell2Notify wakeups on its cellular interface to enable its Wi-Fi interface.

Client Mobility

Mobility can cause a client to move in or out of cellular or Wi-Fi coverage. This can lead to a window of vulnerability where the state of the client may be different from what is known at the SIP server. For example, when a client moves into cellular coverage, it disables its Wi-Fi interface, although the SIP server might have initiated the signaling of an incoming call on the client's Wi-Fi interface. To handle these mobile scenarios, Cell2Notify requires the SIP server to simultaneously ring the cellular interface of the device while sending a SIP invitation on the client's Wi-Fi interface. So, even in the above scenario, when a client moves into cellular coverage, and disables its Wi-Fi interface, the call setup is successful. In the other scenario where a client moves out of cellular coverage, it immediately enables its Wi-Fi interface, and sends a SIP register message to the SIP server. Therefore, in this case, the latency is better than if the device was in cellular coverage. Finally, we note that the problem of hand off across Wi-Fi APs when a VoIP call is in progress, is out of scope for Cell2Notify, which is a signaling protocol for VoIP call setup.

3.3.3 Modifications to the VoIP Server

The above steps can be implemented over SIP, without any modifications to a standard VoIP proxy server. To implement Cell2Notify, we only need to add *call handling rules* for each VoIP extension or user name that is registered with the Cell2Notify server, and no source code modifications to the VoIP proxy are needed. This rule-based call handling is implemented by many commercial SIP/VoIP proxies [97]. The set of SIP rules at the Cell2Notify server are as follows:

1. Send ring tone to caller.
2. Make call to callee's registered cell phone.
3. Dial the VoIP extension of callee. Retry after timeout.
4. Wait a few seconds for callee's response.

5. Send invalid tone to the caller if no response from callee.

6. Hang up if no response from callee is forthcoming.

In Section 3.4, we present the specific call rules we used in our prototype for the Asterisk SIP server. Step 1 informs the caller that the call is being handled. Step 2 tells the callee to enable its Wi-Fi interface and complete the call. Step 3 attempts to connect to the caller. The server retries this step a few times to account for variation in the time taken by the callee to associate and authenticate with the AP, and obtain an IP address using DHCP. Step 4 waits a little longer for a response. If there is no response from the callee, the server sends back an invalid tone to the caller (or voice mailbox of the callee) in Step 5 and hangs up the call in Step 6.

Since these changes are just rules added to the configuration file of the SIP server, Cell2Notify can be easily added to an existing VoIP deployment without adding any new servers or changing the infrastructure. Furthermore, Cell2Notify works within deployments that have VoIP phones without a cellular interface, or where some users prefer not to use Cell2Notify. Therefore, our system is incrementally deployable as well as backwards compatible.

3.3.4 Modifications to the Smartphone

We require a few changes to the smartphone devices, yet all these changes can be implemented relatively easily. We need the following additional features: (i) The ability to distinguish a *wake-up* call from a regular call over the cellular interface. (ii) The ability to power on the Wi-Fi interface. (iii) The ability to control association and authentication with a Wi-Fi network. (iv) The ability to monitor traffic over the Wi-Fi interface to power it off automatically at the end of a VoIP call.

As described in Section 3.3.1, the Cell2Notify server sends a unique ID (UID) to the mobile device as part of the registration process. The component of the smartphone that handles incoming calls needs to be modified to check the Caller-ID of all incoming calls against this UID. In case of a Windows Mobile based smartphone this can be done by modifying the connection manager. When the incoming Caller-ID does not match the UID, the incoming call is treated as a regular call. When the incoming Caller-ID does match the UID, the connection manager takes the following steps:

1. Do not send the call notification to the user.

2. Power on the Wi-Fi interface.

3. Authenticate and associate to the Wi-Fi network and request an IP address from the DHCP server.

4. Start up the SIP softphone user interface.

5. Send a SIP register message to SIP proxy with the destination address as the IP address acquired from the Wi-Fi network.

When the Cell2Notify server receives the SIP register message from the smartphone device, it can complete the SIP call. An important point to note is that the Cell2Notify server is not required to keep any state, since the SIP call is completed over the Wi-Fi interface of the mobile device and the voice session (using RTP) is established end-to-end. This makes our system highly scalable.

Once the VoIP call ends, the smartphone must detect this event and turn off the Wi-Fi interface to save energy. This may be complicated given the presence of other traffic on the Wi-Fi interface, in which case it may not be clear that the call has ended. To detect the end of a VoIP call, we have implemented an activity detector that monitors the wireless interface for data sent and received. Although VoIP sessions generate an almost constant quantity of data traffic during the lifetime of a session, the actual quantity of traffic is dependent on the codec used. Therefore, automatically distinguishing VoIP from other traffic is very difficult. Instead, our detector simply uses a conservative approach, powering off the Wi-Fi interface after a full ten seconds of network inactivity (although the interval length is configurable).

3.3.5 Other Applications

Until now we have focused on using Cell2Notify solely for VoIP calls. However, this architecture can be used to enable a number of other services for smartphones. For example, the Cell2Notify server can be configured to send e-mail notifications by using a different Caller-ID. The Cell2Notify client can use the Caller-ID to differentiate between VoIP and e-mail notifications. The smartphone can then connect to the mail server over Wi-Fi to download the e-mail message contents. Because many people receive a much larger number of incoming e-mails than phone calls, our notification system may impose a much larger load on the cellular network. To avoid this overload, we can tune the Cell2Notify server to only send these notifications for high priority e-mails, or for e-mails from a pre-specified group of people.

A similar application that can benefit from Cell2Notify is *Fax over Wi-Fi*. Any existing scheme for sending Fax over IP, such as T.38 [69], requires the Wi-Fi client to be enabled and hence drains battery power. With Cell2Notify, the Wi-Fi client can be disabled most of the time, and enabled only to receive the fax transmission. Cell2Notify also has applications outside the enterprise setting. For example, any VoIP provider, such as VoIP-User or Skype, can use Cell2Notify to notify their users of incoming calls at home. They would only additionally need the cell phone numbers of smartphones that would be used as receivers of the VoIP calls. In a similar vein, cell phone providers such as T-Mobile, who are moving towards UMA [94] could benefit from Cell2Notify. UMA allows a cell phone to use a Wi-Fi connection if available. However, the Wi-Fi device always needs to be enabled to receive incoming calls. Using Cell2Notify, they can disable the client's Wi-Fi device unless the client is either receiving a call or making one.

3.3.6 Alternatives to Cell2Notify

There are several alternatives to Cell2Notify. In this subsection we use three metrics to argue that notifications using a call over the cellular network is a better approach. The three metrics are: cost, deployability, and performance.

One alternative to Cell2Notify is Wake-On-Wireless [81]. This scheme requires a custom low power radio to be added to each smartphone, as well as to the enterprise wireless infrastructure. When a user receives a call, Wake-On-Wireless(WoW) sends a signal to the smartphone using the low power radio to enable the Wi-Fi interface. This scheme is more costly as this requires the deployment of other low power radios, and is also less deployable since it requires hardware changes on all the smartphone devices. An alternative approach is to leverage another commodity radio technology, Bluetooth to provide the wakeup functionality. In prior work [5] we implemented this functionality in an architecture called On-Demand Paging, requiring Bluetooth hardware to be added to each Wi-Fi AP. On receiving an incoming connection request, this dual function AP sends a signal via Bluetooth to the smartphone such that it can enable its Wi-Fi interface. Since smartphones mostly have a Bluetooth interface, this scheme is more deployable than Wake-on-Wireless. However, it too requires changes to the infrastructure and is therefore costly. Furthermore, both Wake-On-Wireless and On-Demand Paging suffer from the range mismatch problem: the different wireless interfaces have different coverage ranges, and the low-power wireless interface typically covers a smaller region than the Wi-Fi interface. Therefore, the additional wireless infrastructure must be deployed at a higher density than the existing Wi-Fi deployment of access points.

Another approach to Cell2Notify would be to use an SMS (Short Messaging System) based notification system. This scheme is similar to ours except that it would send an SMS message to the smartphone over the cellular network. Although this scheme is as cheap and deployable as Cell2Notify, it suffers from poor performance. SMS usually incurs higher latency and is more unreliable than phone calls. This reduces the usability of this system.

3.4 Implementation

We are currently implementing the Cell2Notify system on Windows CE, a commonly used operating system on smartphones. In the meantime, for evaluation purposes, we have built a prototype of Cell2Notify using commonly available off-the-shelf components. The components of our prototype are illustrated in Figure 3.5. We implement the Cell2Notify server using a combination of the open-source Asterisk SIP Server [8] and the VoIP gateway provided by Junction Networks [46]. We emulate a smartphone using a combination of a cell phone and a laptop running Windows XP. We use a Sony Ericsson W810i cell phone with a built-in Bluetooth interface. The laptop also has built-in Bluetooth, and we use a Netgear WAG511 Cardbus card as the Wi-Fi interface. Finally, we use a popular SIP client for Windows XP called X-Lite [20] as the VoIP softphone.

Our prototype requires minimal modifications to the above components. We made changes to the call handling configuration files of the SIP server, and we built a user-level *call-manager* service that runs on the Windows XP laptop. Our prototype demonstrates the ease with which Cell2Notify can be incrementally deployed in an existing VoIP system.

The steps of the Cell2Notify protocol for our prototype are shown in Figure 3.5. When someone makes a incoming call to a Cell2Notify client, the Asterisk SIP Proxy looks up the corresponding cellular number for the client, and makes a call to the client over PSTN using the Junction Networks gateway. When our cellphone receives the call, it notifies the laptop of the incoming call via Bluetooth. The call-manager service on the laptop then turns on the Wi-Fi interface and uses it to connect the call. When the call is complete, the call-manager turns off the Wi-Fi interface. In the rest of this section, we describe the implementation details of the Cell2Notify server and client components.

3.4.1 Prototype Cell2Notify Server

The Cell2Notify server only requires minimal modifications to the Asterisk SIP Server. We have added a mapping from SIP extensions to the corresponding cell phone number, and a set of call handling rules for each registered Cell2Notify client. Asterisk supports integration with a back-end database, thus allowing the cell phone mapping table and call handling rules to be implemented as separate tables in the database, and be linked to the Asterisk server. Presently we have manually added these mappings for each Cell2Notify client to the Asterisk con-

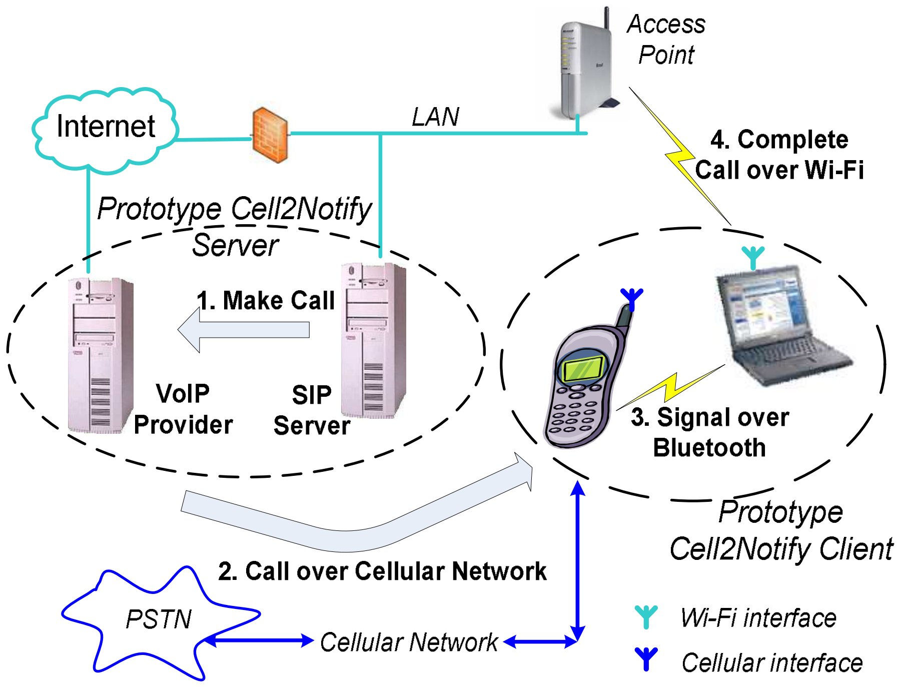


Figure 3.5: Our prototype implementation of Cell2Notify. We implement the Cell2Notify server as a combination of a commonly available SIP proxy and an Internet- based VoIP gateway. We emulate a smartphone using a combination of a cellphone that communicates with a Wi-Fi equipped laptop using Bluetooth.

figuration files. However, this task can be easily automated using the supported database functionality.

We implement the steps described in Section 3.3.3 as call handling rules in the Asterisk server. We define these rules for every registered extension or user name. To define the call handling rules, we use generic functions that are supported by most SIP proxies, such as Ringing, Playback, Dial, Wait and *Set(CALLERID)*. The *Ringing* function sends back a ring notification to the caller. *Playback* plays a default welcome message and *Dial* dials a SIP extension. The *Wait* function waits for a specified duration before executing the next rule. *Set(CALLERID)* is interesting as it allows the Caller-ID of the outbound call to be set to an *arbitrary* number. In the following example, we present the call handling rules for a particular extension, say extension `7676`:

```
1. exten => 7676,1,Ringing

2. exten => 7676,2,Set(CALLERID(number)= UID)

3. exten => 7676,3,Dial(SIP/Cell-Number@jnctn,5)

4. exten => 7575,4,Wait(2)

5. exten => 7676,5,RetryDial(waiting|1|8|SIP/7676|30|Ttm)

6. exten => 7676,6,Playback(Invalid)

7. exten => 7676,7,Hangup
```

These rules define the steps executed by the Cell2Notify server when there is an incoming call for extension 7676. The first argument denotes the destination extension (7676) for the incoming call, the second argument is the rule order(1,2,..,7), and the third denotes the function (Dial, Ringing, etc.). Rule 1 executes the *Ringing* function and sends back a ring tone to the caller. The server then looks at Rule 2 and executes the *Set(CALLERID)* function with the UID as a parameter, essentially setting the Caller-ID to the UID for the next outbound call. As explained earlier the UID is different for each smartphone client using Cell2Notify and is negotiated during registration. Rule 3 places a call to the particular cellular number associated with extension 7676 using the Junction Networks gateway. Rule 3 is essentially needed to send a signal to the Cell2Notify client to turn on its Wi-Fi interface. In rule 4 the server executes *Wait* for 2 seconds to insert some delay before trying to contact the extension. On encountering rule 5 the server executes *Dial* to contact the SIP extension 7676 repeatedly 8 times with a 1 second interval between subsequent retries. These retries are needed because of the latency to turn on the Wi-Fi interface on the laptop device, and the latency to associate and authenticate over the Wi-Fi network. In the case where call is not connected or remains unanswered the server executes the *Playback* function as specified in rule 6, to send the caller an invalid extension or unreachable message. According to Rule 7, the server executes a *Hangup* to end the call. Rule 6 could be modified to playback another message, record a voice mail, forward the call to another extension, or even forward the call to the cellular number of the user.

When a call handling rule (such as Rule 3) requires the server to place a call on the regular telephone network, it uses either an Analog Telephony Adapter (ATA) or an external third party VoIP provider to bridge the IP based network with the PSTN. We have implemented both these options. In the first option, we used the Sipura ATA [84] and a privately leased PSTN line. For the second option, we used Junction Networks, one of the several available VoIP gateways. Using an ATA may be preferable for an enterprise, because the call leaves the IP network within the enterprise itself. However, using a third party VoIP provider may be cheaper.

An architectural requirement for the Cell2Notify server is the ability to place a call over the PSTN using an arbitrary Caller-ID. We implement this using the *Set(CALLERID)* function of Asterisk in conjunction with the VoIP gateway of Junction Networks. The SIP server sets the desired Caller-ID as a parameter to the *Set(CALLERID)* function. Junction Networks allows users to provide their own Caller-IDs for outgoing calls, as long as it is *any* 10 digit number, and then places a call to the destination PSTN number with this Caller-ID using an ATA located in the Junction Networks data center. We are currently working on implementing this functionality on the Sipura ATAs. We discuss the implications of Caller-ID spoofing in Section 3.6.

3.4.2 Prototype Cell2Notify Client

We now describe the implementation of the Cell2Notify client, focusing on three main challenges. First, we need a way to signal an incoming call on the Sony Ericsson cell phone to the call manager service on the Windows XP laptop, and we need to send the Caller-ID of the incoming call to the call manager. Second, we need minimize the delay in completing the call by reducing the delay imposed by the Wi-Fi authentication and association process. Finally, the call manager service needs to determine when the call ends and disable the Wi-Fi interface.

We address the first challenge without requiring modifications to the Sony

Ericsson handset by configuring the Bluetooth interface on the laptop to appear as a Bluetooth headset to the cell phone. Consequently, an incoming call on the cellphone notifies the Bluetooth headset, which is in fact our laptop. We use Float Mobile Agent (FMA) [28] to configure the laptop Bluetooth interface to appear as a headset device. FMA is powerful phone editing software which has extensive support for Sony Ericsson handsets, including a rich set of APIs to control the handset. One feature of these APIs handles a *Call-Notify* event which our call manager service uses to trap an incoming call. We built a separate call handler on the FMA framework that checks the Caller-ID of each incoming call to see if it is from the Cell2Notify server, based on the unique ID that was exchanged as part of the registration process. FMA also provides a way to disconnect a call. If the Caller-ID matches that of the Cell2Notify server, the call handler disconnects the call and wakes up the Wi-Fi interface. If the Caller-ID does not match, the call handler lets the call through and ring on the handset.

To address the second challenge, our service uses caching to quickly associate with an Access Point and complete the call over Wi-Fi. When a wireless card is enabled, it usually goes through a series of steps before it obtains a valid IP address. For example, it scans the network looking for the best available AP, after which it performs the entire association procedure. Associating with an AP using the standard Windows XP Zero Configuration Service takes multiple seconds [15]. We optimize this step by caching the frequency channels of the most commonly used APs. We also turn off the Zero Configuration Service and implement tools to control the wireless interface from our own Cell2Notify service. When the Wi-Fi interface is turned on, we instruct the card to go to specific channels and attempt association to the wireless network. We have measured the total time to associate on a given channel to be less than 20 ms for the Netgear WAG511. Using this optimization, we are able to complete the association within a few hundred milliseconds, as shown in Section 3.5. Once the Wi-Fi card is enabled and has an IP address, we start the X-Lite SIP client. The SIP client sends a register message to the Cell2Notify server with its acquired IP address and completes the call over Wi-Fi.

Finally, we need a way to automatically detect the end of a VoIP call and turn off the Wi-Fi interface. After the Wi-Fi interface is enabled, our call manager service enters an activity monitoring mode. In this mode, it checks the number of packets sent and received on the Wi-Fi interface. It does not immediately disable the Wi-Fi interface when the number of packets is zero, as this might disconnect the call during a period of silence. Instead, the service uses some hysteresis and only disables the Wi-Fi interface if there are no packets sent over it for a certain number of seconds. We experimented with various values and found that a delay of ten seconds was adequate. To avoid modifications to the SIP client application code, we terminate the SIP client process at the end of a call and restart it when a new call is initiated or received.

3.5 Evaluation

The utility of a mobile device is directly related to the useful operating lifetime before its battery needs to be recharged. Thus, the primary metric we use to evaluate our Cell2Notify system is the reduction in energy consumption, which directly translates to increased battery lifetime. We also evaluate the increase in end-to-end latency that a caller experiences when making a call to a Cell2Notify client. Our results show that using Cell2Notify, users can greatly increase the total usage lifetime of their Wi-Fi enabled smartphones when using VoIP, while experiencing only a nominal increase in initial call-setup latency.

3.5.1 Reduction in Energy Consumption

To quantify the energy savings enabled by Cell2Notify, we first measured the power consumption of various commonly used wireless cards. The 802.11 standard [38] specifies various modes of operation for the interface but not the specific implementation details. Table 3.3 below illustrates the power consumption of several Wi-Fi interfaces in the normal mode of operation, Awake Mode(AM), and the Power Save Mode (PSM), achieved by duty cycling the wireless interface. The Cisco PCM-350 is sometimes referred to in research literature for the sake of com-

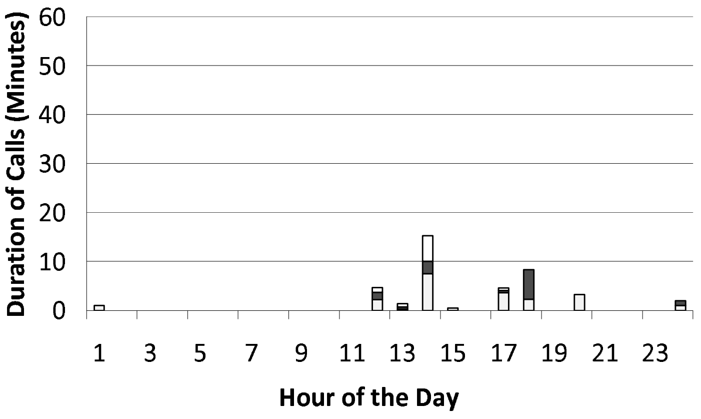
Table 3.3: Measured power consumption for 802.11b cards.

Vendor	*Average Power*		
	Idle(AM)	Idle (PSM)	Active
Cisco PCM 350	1300mW	390mW	1600mW
Linksys WCF12	690mW	256mW	890mW
Netgear MA701	780mW	264mW	990mW

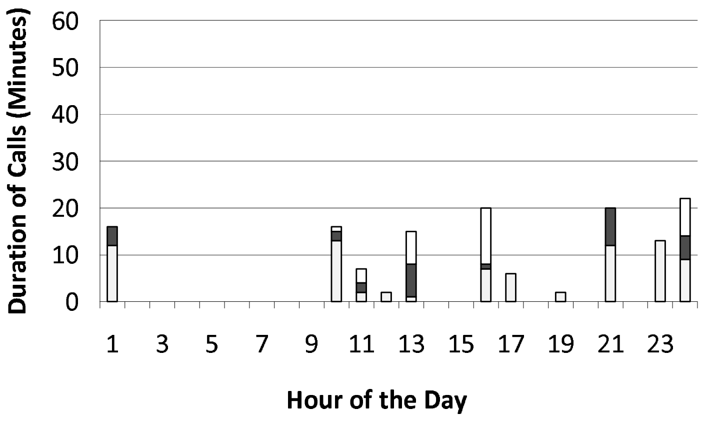
parison, although it is known to be quite power inefficient. The Netgear MA701 and Linksys WCF12 cards are the most power efficient among the cards that we have measured and thus we use the Linksys WCF12 as a baseline for comparison. Once enabled, a Wi-Fi interface usually takes some time to stabilize, before reaching a state where it can perform active data transfer. Similarly, when disabling a Wi-Fi interface it takes some time before the power drawn by it becomes negligible. In addition to measuring the power consumption of these wireless cards, we have also measured the power consumption of a Windows Mobile based smartphone. The power consumption for the Cingular 2125 was reported in Section 3.2.2 earlier.

The effective energy savings for a particular user are somewhat dependent on their usage patterns. As stated earlier, our Cell2Notify scheme keeps the Wi-Fi interface of a smartphone switched off at all times, except during an active VoIP call. Thus, a user who uses their phone for sporadic conversations will end up saving more energy, in contrast to a heavy user who communicates more frequently. Energy saved by our low power architecture is thus directly dependent on the amount of idle time experienced by a mobile device.

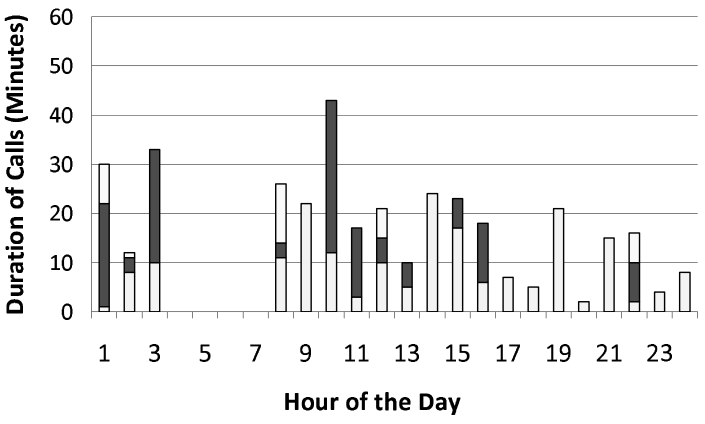
In order to study typical usage patterns, we gathered detailed cellular phone call-logs of different users. Using these call logs we construct a similar trace of periods of communication activity and inactivity, that would be experienced if the users were using VoIP over Wi-Fi instead. Using these call traces we accurately estimate the level of energy savings enabled by the Cell2Notify architecture. We then compare this to the energy consumption of these devices, if they were using the standard 802.11 operating modes, AM and PSM respectively. This technique of using call-logs is similar to the one used in Wake-on-Wireless [81].



(a) James's Office Phone



(b) John's Cell Phone



(c) Beth's Cell Phone

Figure 3.6: Call logs of three different users.

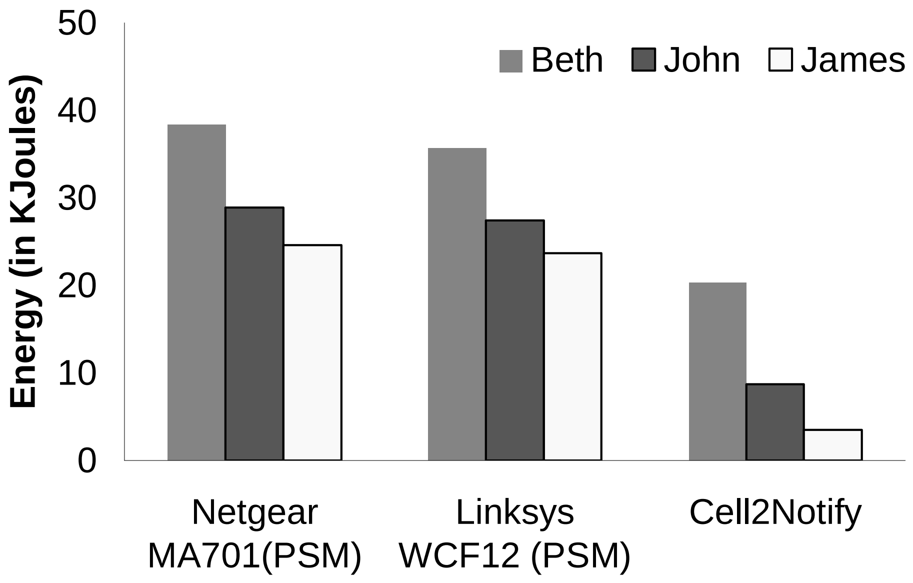


Figure 3.7: Energy consumption using two cards, with and without Cell2Notify for three different users. As expected Cell2Notify saves more energy for lighter usage patterns.

Figures 3.6(a), 3.6(b) and 3.6(c) show the calling patterns of users James, John and Beth respectively. James is a real employee in an enterprise and is the heaviest user among five of his colleagues in our study group. John is a light user with an average talk time of about 5 minutes per hour. Beth on the other end is a hypothetical person with a relatively heavy usage pattern, with an average talk time of 15 minutes per hour. On the horizontal axis, the hour of the day is shown ranging from 0 hours to 23 hours. The total number of minutes that a user was actively communicating over the phone are marked on the vertical axis. The different shaded subsections for each vertical column depict the number of calls made in that hour and the duration of each call. These call logs are illustrative traces that help evaluate the *estimated* energy savings for these three usage patterns.

Figure 3.7 plots the total *communication* energy consumption for the various users calling patterns in a 24 hour period. The graphs shows the energy consumed in the wireless interface when two low power Wi-Fi cards (MA701, WCF12)

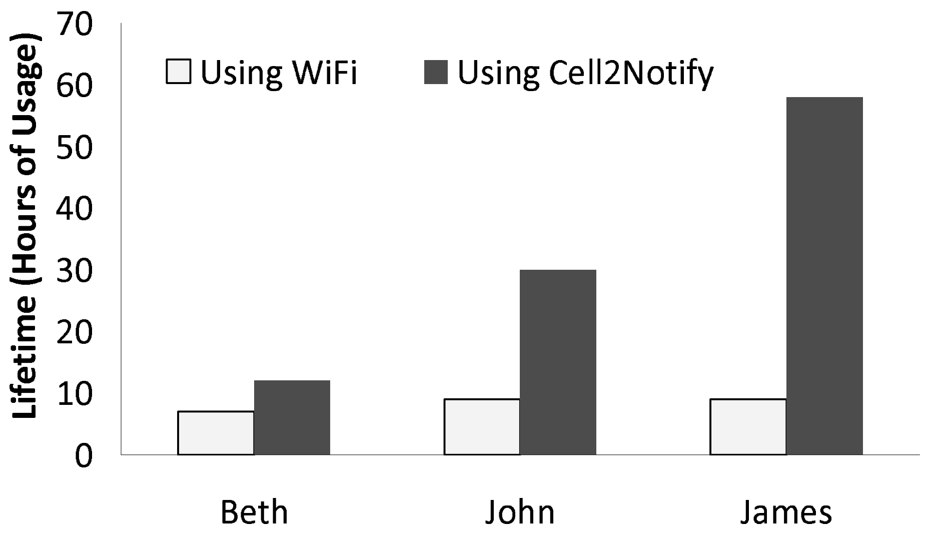


Figure 3.8: Energy consumption of a Cingular 2125 with and without Cell2Notify for three users. We assume that the user does not use the smartphone for any other purpose, but only for making and receiving VoIP calls.

are used, compared to the energy consumption when utilizing the Cell2Notify architecture. As can be seen even Beth, with a heavy usage pattern, can save up to 47% of the energy consumption compared to using the Wi-Fi cards in the Power Save Mode (PSM). John and James, who have lighter usage patterns end up saving 70% and 87% respectively of the energy consumed compared to using the Netgear MA701 in PSM mode.

In essence, lowering the energy consumption leads to longer battery lifetime of a smartphone. To quantify the effects of our scheme in terms of increased lifetime we measured the power consumption of a Wi-Fi enabled Smartphone (Cingular 2125) in various modes of operation. Using our detailed power measurements reported in Section 3.2.2 and the rated capacity of the phone battery (1150 mAH), we determine battery lifetime. Figure 3.8 shows the increase in battery lifetime for the three usage scenarios. Our base comparison is using the Wi-Fi in always on mode for the smartphone. As can be seen Beth experiences a 70% increase in battery lifetime by using Cell2Notify. John and James on the other hand experience a 230% and 540% increase in lifetime, primarily because of their light usage

Table 3.4: Standard deviation and maximum values for various steps of the Cell2Notify protocol. Note that the steps "Enable VoIP Client" and "Obtain IP Address" occur in parallel.

Cell2Notify Protocol Step	*Latency (in seconds)*	
	Standard Dev.	Max Value
Call on GSM	0.098	3.7
Enable Wi-Fi	0.265	1.7
Connect to AP	0.073	0.36
Enable VoIP Client	0.105	4.8
Obtain IP Address	1.08	4.44
SIP Operations	0.025	0.488

patterns.

3.5.2 End-to-End Latency

The reduction in energy consumption when using the Cell2Notify architecture has an associated tradeoff with respect to the added latency in connecting a VoIP call. Since the mobile device that is the end recipient of the VoIP call has its wireless interface switched off, there are multiple steps that have to be taken before the device can actually accept the call over Wi-Fi. Each of these steps has an associated latency overhead. In this section we evaluate these latencies for our prototype implementation. We also provide detailed measurements of these latencies for other platforms. Some of the latencies are fixed costs which are beyond our control, for example the time taken to connect a call over the cellular network, while some of the other latency components can be optimized. Using these measurements we can provide a reasonably accurate estimate of the lower bound on the total end-to-end latency that a device using our architecture experiences.

Figure 3.9 shows the breakdown of the call-setup latency introduced by various steps of the Cell2Notify system. The bar on the left in the figure shows the latencies measured on our prototype implementation using the combination of a Windows XP laptop and an SE-810i cellular phone. The column on the right shows the expected latency for the case of a final product implementation on a smartphone. Each latency value presented in the Figure is an average over a minimum of ten runs. We present the standard deviation and maximum values for

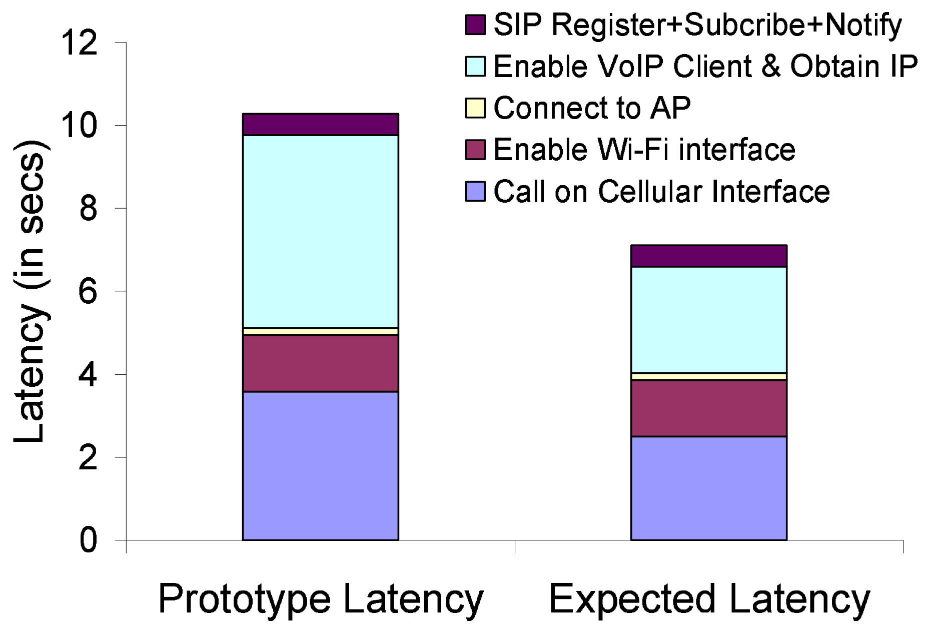


Figure 3.9: Breakdown of various steps of the Cell2Notify protocol in call-setup latency. The right bar shows the expected latency with our proposed optimizations. Even without optimizations, the extra delay is around ten seconds, which is *less than two rings*.

each of these steps in Table 3.4.

Our measurements show that the average added latency for our prototype implementation is around ten seconds. This extra wait is equivalent to two rings received by the caller. We believe this overhead is minimal and acceptable in most scenarios. Furthermore, we expect a real smartphone and enterprise deployment of Cell2Notify to incur an overhead of around seven seconds, which will provide a more seamless experience to users of Cell2Notify.

A large component of the overhead is the time taken by the SIP server to call the GSM interface of the Sony Ericsson cell phone. It is difficult to accurately quantify this overhead, since the caller (server) and callee (Sony Ericsson handset) are on two different machines. We used a stopwatch to measure this time for over 20 runs, but we are aware of possible inaccuracies due to human reaction times. However, we note that our reaction times will likely result in an overestimate of

Table 3.5: Distribution of time taken by the SIP server to "ring" a phone for various connection types. We present the latency to ring a land line number as a reference.

Client Phone (Signal)	*Latency (in seconds)*		
	Avg.	Std Dev.	Max.
GSM Cingular (Excellent)	3.6	0.074	3.6
GSM Cingular (Poor)	3.78	0.092	3.9
CDMA Verizon (Fair)	2.44	0.117	2.6
Landline Phone	2.41	0.074	2.5

the latency. As we see in Figure 3.9, the time taken for the Cell2Notify server to call the GSM interface of the Sony Ericsson phone is around 3.7 seconds in our prototype. A large portion of this overhead seems to be the time taken to call Junction Networks, and for Junction Networks to make a long distance call to our cell phone. To estimate the time it would take in a real prototype, we tested calling a local cell phone number using the Sipura [84] ATA that we have set up in our lab. We note that this time was only 2.5 seconds. Since most enterprises will have their private VoIP gateway, this seems to be a reasonable estimate in a real prototype.

We further explored the lower bound of this delay when the VoIP gateway is on the enterprise LAN. We placed calls from the SIP server to the client phone through the Junction Networks VoIP gateway, and for different types of client phone connections. We placed 10 calls for each connection type, and present the distribution of the time taken to place these calls in Table 3.5. We note that it takes an extra second to place a call to the GSM phone, and the connection quality of the phone does not add a significant latency. Furthermore, it takes much lesser time to place a call on the CDMA phone. This latency is comparable to the time taken for placing a call to the land line phone, which is around 2.4 seconds.

Our optimization of using cached Access Point BSSIDs gives good results. Our Cell2Notify client is able to associate with the AP in less than 200 ms. We used three different APs on three different frequency channels in our experiments. We disabled the card and randomly picked an AP to associate with in each run. We also measured the default time to connect to an AP without our optimization of caching the AP information resulting in a much higher overhead, between 3 and

4 seconds in each run. This latency is expected since without our optimization, the wireless card goes into scan mode. It stays for over 100 ms in each channel (all 802.11 a and g channels), and only then associate with the best AP.

Another significant latency in Cell2Notify is the time to obtain an IP address and bring up the softphone. As shown in Figure 3.9 this overhead is around 5 seconds in our prototype. Although it only takes about 2.5 seconds to obtain the DHCP address, the total time to start the X-Lite SIP client process takes around 5 seconds. As mentioned earlier, we had to restart the SIP client process to avoid modifications to the SIP client code. In an actual implementation over smartphone, we do not expect this artificial overhead of restarting the SIP client to be present. Instead, the only overhead should be the time required to obtain a valid IP address.

After the softphone has initialized and obtained a valid IP address it sends a SIP Register message, and a Subscribe message to the Cell2Notify server, which together take less than 0.5 seconds. Once the server receives the SIP register from the Windows XP SIP client, it connects the call. These steps have very low overhead.

Finally, we note that since most users are willing to tolerate up to five rings (25 seconds) after call connection to reach the voice mail, the less than ten seconds (2 rings) delay introduced by Cell2Notify is acceptable in most scenarios. Ideally a user study would be useful to estimate the actual impact of this increase in latency. We plan to investigate this as part of our future work.

3.6 Discussion

We now discuss various issues in the design and deployment of Cell2Notify. We first discuss the legality of our approach, and show how our system can be secured against spoofed Caller-IDs. We then discuss the concerns that cellular operators may have to the deployment of Cell2Notify. Finally, we discuss the deployability of our scheme.

3.6.1 Is Caller-ID Spoofing Legal?

There is no law in the US against Caller-ID spoofing [95]. Caller-ID over PSTN is sent using the SS7 signaling protocol. Before the days of VoIP, expensive equipment was required to spoof a Caller-ID. With VoIP, one can introduce fake Caller-ID information when passing the call from IP to PSTN. There are a number of commercial services [87, 88] that allow users to make calls from a spoofed Caller-ID. This has led to a few abuse cases of "pretext calls", where people pretend to be someone else to extract private information [95]. Therefore, in a recent development, the FCC is investigating the use of Caller-ID spoofing for fraudulent purposes. However, since Cell2Notify does not attempt any fraudulent activity, we do not expect it to be affected in the near future.

3.6.2 Handling Spoofed Caller-IDs

Given that Caller-ID spoofing is legal in some countries such as the US, we need to protect against attackers who might spoof the Caller-ID of the Cell2Notify proxy causing the smartphone to enable the Wi-Fi card and waste battery power. We can thwart this attack by authenticating the Cell2Notify proxy at the client using standard cryptographic techniques. One way to achieve this is to use the S/KEY system [34], which originated from Lamport's scheme [53] as follows. The Cell2Notify proxy shares a different secret key with each VoIP user, which is set up during secure registration. The first Caller-ID used by the proxy is the last nine digits of a one-way hash applied n times over the secret key, where n is a large number. The first digit of the Caller-ID is set to 0 to avoid collisions with a PSTN phone number. The subsequent Caller-ID is an $n - 1$ times one-way hash of the secret key, and so on. The Cell2Notify client authenticates the proxy by applying the one-way hash on the Caller-ID to see if it matches the previous Caller-ID. Given a strong hash function, this scheme can provide reasonable protection against a spoofed Caller-ID attack.

3.6.3 Concerns of Cellular Operators

Cellular operators have a valid reason for blocking the Caller-ID of the Cell2Notify server. After all, Cell2Notify only uses their network as a signaling channel. Consequently, cellular operators do not stand to gain by allowing Cell2Notify. We have several reasons to believe that cellular operators might be willing to allow Cell2Notify to make signaling calls over their network. Cell2Notify imposes little load on their network as for every incoming call to the VoIP phone, we make one signaling call over the cellular network which does not last more than a few seconds. Even assuming that the VoIP phone has similar usage characteristics as the cellular phone (in Section 3.5 we show in fact that an enterprise phone is used quite infrequently), a ring for every incoming call imposes little extra overhead. Furthermore, users might be willing to pay an extra "connection charge" to achieve longer battery lifetime. In some cases, the enterprise may be willing to pay a flat fee to cellular operators to support this service. We also believe this work is extremely timely given the launch of T-Mobile's UMA service [94]. Cellular operator's supporting UMA [48] can provide Cell2Notify service as an additional selling point. Finally, we note that it might be technically infeasible for cellular operators to block calls from the Cell2Notify server, since the proxy uses a different Caller-ID each time it sends a signal using the mechanism described in Section 3.6.2.

3.6.4 Deploying Cell2Notify

One obvious concern that arises is the practicality of our prototype as described in Section 3.4, given that we have emulated the Cell2Notify client rather than implementing it on a real smartphone. To address this limitation we have recently implemented Cell2Notify on Windows Mobile 6 (WM6), one of the various mobile platforms. This latest release of WM6 exports several well documented APIs that allow fine grained control of various Wi-Fi functions, such as powering the radio on and off from user space. Furthermore in the current release of the OS, we were also able to register for various call related events, such as trap an incoming call, detect its caller-ID and also know when the call has ended. In our current prototype, we do not suppress the momentary UI notification on the initial call over the cellular interface, and instead use it to alert the user of an incoming Cell2Notify VoIP call. Incidentally, we also leveraged the integrated SIP based VoIP stack on WM6 and as a result were able to use the same consistent user interface to display a VoIP call as a regular GSM call. We are currently looking to implement our architecture onto other smartphone platforms, such as the iPhone and Symbian based devices.

3.7 Related Work – Paging and Wakeup

Chapter 2 provided an overview of the prior work relating to energy management on mobile devices. We also discussed the techniques for reducing the power consumption of wireless interfaces, specifically for systems based on a single wireless interface. The approaches ranged from optimizations at various layers of the network protocol stack; at the application layer [27, 52], transport layer [11] and MAC Layer [101, 51]. However, all of these approaches show limited power reduction since they are only able to utilize the power saving modes (such as Wi-Fi PSM) of the single radios which is still quite significant. In this section we go over prior research that specifically looks at using multiple radio interfaces for paging or wakeup.

In the field of sensor networks, a passive wakeup scheme has been proposed [19]. A very low power receiver structure essentially fulfills the role of a separate wakeup channel. In addition, this approach requires custom built radio hardware, which is a viable option for sensor networks. Chiasserini *et al.* [13] suggest an approach to use a secondary mechanism to wake up a group of nodes by means of small range ID tags. They however look at analytical results only, while we focus on actual implementation, associated practical problems and power measurements. In STEM [80] the authors propose the use of a second duty-cycled radio to provide paging with their results based on analysis and simulation. Another similar duty-cycling based scheme is presented in [78]. These duty-cycling approaches could be utilized on top of our low power paging channel, as suggested in [80]. In contrast to these approaches, Cell2Notify targets existing commodity wireless access technologies that are present on current platforms.

Taking into account the high idle power of Wi-Fi, Wake-on-Wireless [81] proposes the use of a second special-purpose radio that serves as a wake-up channel for a Wi-Fi radio. The authors have proposed a PDA based phone usage scenario for their system, similar to Cell2Notify. However the choice of the short range custom radio necessitates multiple intermediate proxies and presence servers in order to notify the PDA-phone of an incoming call. On-Demand-Paging builds on the idea of [81], to use a commodity Bluetooth radio present on mobile devices to serve as a low power paging channel for Wi-Fi. The main contribution of On-Demand Paging was to address the range disparity between the Bluetooth paging radio and the Wi-Fi radio, thus not requiring a dense deployment of a paging infrastructure as in Wake-on-Wireless. Cell2notify in contrast to both Wake-on-Wireless and On-Demand-Paging, leverages the much *longer* range cellular radios compared to *short* range radios. This has two important advantages. First, since Cell2Notify uses cellular radios with almost ubiquitous coverage, the area of operation is much larger. Second, the infrastructure support needed for our scheme is minimal, with only minor software modifications needed at both the client device and an existing VoIP proxy in terms of call handling rules. Comparatively both Wake-on-Wireless [81] and On-Demand Paging schemes need substantial additional infrastructure support, while still limiting the area of operation to their region of deployment. A recent industry trend is the convergence of Wi-Fi and cellular services, using a technology called Universal Mobile Access (UMA). For example, chipset vendor Kineto [48] and mobile service provider T-Mobile [93] recently tested a service that allows a subscriber to make unlimited phone calls from the home hotspot or T-Mobile hotspots [94]. UMA increases coverage and reduces the cost for mobile operators. Our approach to energy savings are complimentary to UMA. Devices using UMA could use our Cell2Notify protocol to increase the battery lifetime of dual radio devices.

3.8 Summary

In this chapter we have described an energy saving architecture, called *Cell2Notify*, which leverages the cellular interface on a smartphone to reduce energy consumption of VoIP over Wi-Fi use scenario on these devices. We have shown that by using the long range cellular radio interface as a wakeup mechanism we can address the major limitation of previous wakeup approaches to significantly lower the bar to deployment. We also address the challenge of collaborating across two very diverse radios, a carrier controlled voice radio and a local area Wi-Fi data radio, by using a call on the cellular interface with a modified caller-ID to signal an incoming VoIP over Wi-Fi call. Since the higher power Wi-Fi radio can be powered off at all times, except during a VoIP call, Cell2Notify leads to substantial energy savings.

We quantify the performance of cellular data networks when used for VoIP and compare these results with Wi-Fi. We show that Wi-Fi consumes less power and delivers better performance than current cellular data networks for VoIP traffic. To the best of our knowledge, ours is the first work to present such measurements. We show that our system works with existing technologies and requires minimal changes to an enterprise's VoIP deployment. We have built a prototype of Cell2Notify and evaluated it in detail. We have shown that in most cases, Cell2Notify incurs less than two rings (10 seconds) of call setup latency while more than doubling the average battery lifetime of a smartphone.

Chapter 3, in part, is a reprint of the material as it appears in Proceedings of ACM Mobile Systems, Applications and Services (MobiSys '07), June 2007. Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl and Rajesh Gupta. The dissertation author is the primary investigator and author of this paper.

Chapter 4

Building a Switching Hierarchy using Collaborative Data Radios

In Chapter 3 we presented the idea of radio collaboration in mobile devices, where a lower idle-power radio was used purely as a means to wake up or page a higher idle-power radio. All data transfer takes place only over the higher power, higher bandwidth radio. The natural extension to radio collaboration that we explore in this chapter is to use all available radio interfaces for actual data transfer by building a hierarchy of radios. Effectively this allows us to choose the most appropriate radio interface that meets performance demands of applications running on the mobile device. Since radios that are not in use can be duty-cycled to save power, the battery lifetime of the mobile device can be improved significantly. For example, for applications with low network utilization the low-power/low-bandwidth interface can be used, and the system can dynamically switch to the high-power/high-bandwidth interface when application demands increase.

The multi-radio switching architecture that we describe in this chapter is called **CoolSpots**. A CoolSpot, analogous to a Wi-Fi hotspot, is a zone where a mobile device can switch to using a lower power radio rather than use the higher power Wi-Fi radio. Although switching between wireless interfaces to save power has been proposed earlier [68], CoolSpots is the first system that actually builds a radio switching framework and explores a suite of **switching policies** that guide the choice of the most appropriate radio interface to use while considering vari-

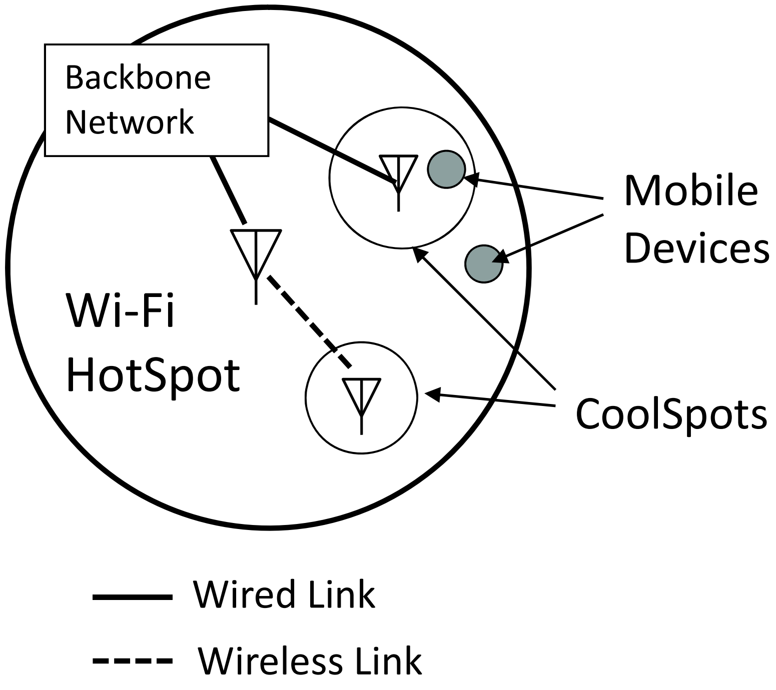


Figure 4.1: Multiple Bluetooth-enabled CoolSpots, inside of a traditional Wi-Fi HotSpot, allow mobile devices to connect other devices through the backbone network. CoolSpots are connected to the backbone network either directly (wired) or through the Wi-Fi network (wireless).

ous power and performance metrics. We show that the CoolSpots architecture is completely application agnostic and can be supported with minimal infrastructure changes. Our evaluation of the various CoolSpots policies demonstrates a 50% reduction in energy consumption, effectively doubling system battery life (depending on overall system behavior).

4.1 CoolSpots Architecture

CoolSpots provide energy-efficient communication capabilities when a mobile device is within a CoolSpot-enabled region (Figure 4.1). For example, if a home user is in their living room near the Access Point, appropriately enabled as a CoolSpot, the mobile device would exhibit lower-power consumption while

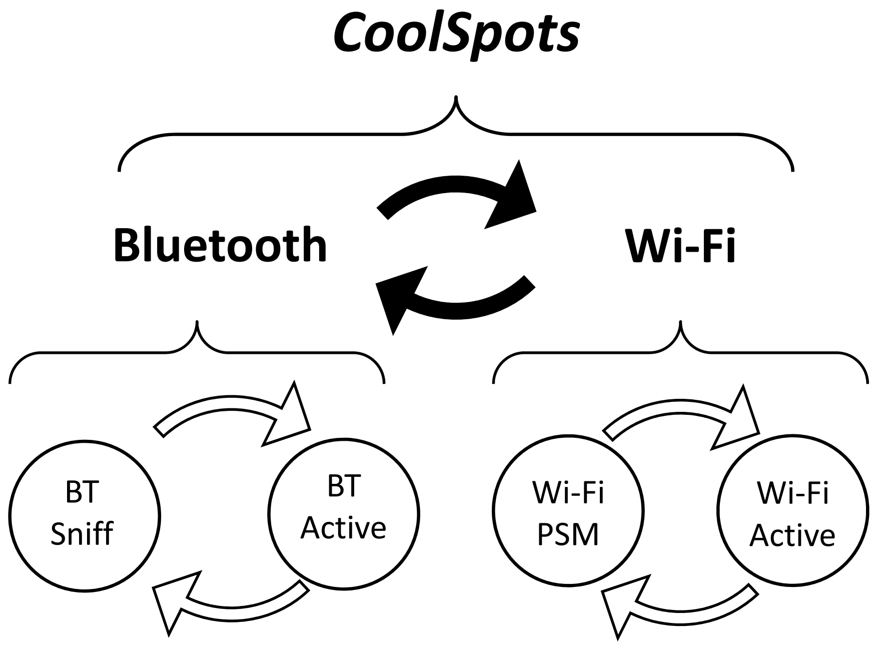


Figure 4.2: CoolSpots enables enables radio collaboration on top of individual radio power management techniques such as Bluetooth sniff mode and Wi-Fi PSM.

still supporting the desired application bandwidth. CoolSpots casts the generic concept of using two different classes of radios collaboratively into a concrete implementation that seamlessly integrates with existing applications, systems, and devices. In essence, CoolSpots implement multi-radio power management that directly takes advantage of the heterogeneity between different radio technologies. When you consider these radios interfaces individually, they each have different states of operation, for example Wi-Fi radios can be in the normal active mode or they can be in the Power Save Mode (Chapter 2). Similarly Bluetooth radios can be in the low power sniff mode or the active mode. In CoolSpots we can use these radios collaboratively, and as a result expand the number of available states at a system level(Figure 4.2). For example, a mobile device using the CoolSpots architecture can leverage the extremely low power Bluetooth sniff mode when it is idle, and use the Bluetooth active mode for low bandwidth applications such

Table 4.1: Power consumption for various wireless interfaces. Values marked with a * are measured values, while others are taken from data sheets.

Interface	**Low-Power Idle**	**Active Tx**
Wi-Fi Interfaces		
Cisco PCM-350*	390 mW	1600 mW
Netgear MA701	264 mW	990 mW
Linksys WCF12*	256 mW	890 mW
Bluetooth Interfaces		
BlueCore3	5.8 mW	81 mW
BlueCore3*	25 mW	120 mW

as browse web pages, and then switch to the Wi-Fi active mode for high bandwidth transfers such as downloading a file. Based on the idle power consumption of typical Wi-Fi and Bluetooth radios (Table 4.1), this concept has the potential to realize 10x reduction in power consumption for an idle system (from 256 mW down to 25 mW); however, the actual power savings depend on various factors as we will explore in this chapter.

From a networking perspective, the CoolSpots model is a simple addition to common networking infrastructure and does not require any extensive hardware changes. A CoolSpots enabled base station providing Bluetooth capability can simply be added into an existing Wi-Fi network, allowing nearby mobile devices reduced-power operation: by employing network routing changes, this base station can claim and route traffic to the mobile device, a technique similar to Contact Networking[14]. Bluetooth was designed as a low-cost addition to handheld devices, and so would be relatively inexpensive to add into the environment. From a technology perspective, this means that the AP is a dual-function device and is equipped with both a Wi-Fi radio and a Bluetooth radio. As a result all data traffic to and from a mobile device always passes through the same dual-function AP. If a mobile device is not sufficiently near a CoolSpot, and wants to save power, it always has the option to fall back to using Wi-Fi in PSM mode.

The core of the CoolSpots architecture, outside of the basic switching **mechanism**, are the switching **policies** that determine when to transition between the various radio technologies. The two decisions that are guided by the

Table 4.2: CoolSpot policies with switch up/down criteria.

Policy	**Switch-Up**	**Switch Down**	**Comments**
wifi-fixed	N/A	N/A	Only uses Wi-Fi
bandwidth	Static bandwidth	Static bandwidth	Can fail in bad network conditions
cap-static	Capacity detection	Static Bandwidth	Same as above
cap-dynamic	Capacity detection	Use Switch-Up bandwidth	Handles all network conditions
bluetooth-fixed	N/A	N/A	Only uses Bluetooth

policies need: when to "switch-up" to Wi-Fi to increase available bandwidth, and when to "switch-down" to Bluetooth and duty-cycle the Wi-Fi radio to conserve energy. The main penalty for switching is the latency and energy overhead of (de)activating the Wi-Fi network interface resulting in energy expended that is not doing any useful work. In fact, a poorly implemented policy would increase overall energy consumption by switching back and forth between wireless technologies without ever staying in any one state long enough to lower the overall energy consumption. The primary contributions of CoolSpots are the empirical measurements and evaluation of various policies that effectively manage multi-radio switching, thus resulting in an overall reduction in communication energy.

4.2 Switching Policies

As mentioned earlier, managing power used by multiple network interfaces requires the system to make two decisions: when to activate the higher-power Wi-Fi interface, and when to shut off the Wi-Fi interface and switch down to Bluetooth. Framed in terms of bandwidth, the question becomes when is there too little bandwidth available on the Bluetooth channel (switch up), or when is there too much unused bandwidth available (switch down). The simplest approach, using statically coded thresholds, does not work well given the variation in channel characteristics: the optimal bandwidth threshold changes as the distance between devices changes.

The various switching policies and the different criteria that guide the switch up/down decisions are enumerated in Table 4.2. These policies are executed by the mobile device and they are used to decide when to switch interfaces. Some switching decisions are made by measuring the active bandwidth on a given channel, and then activating a radio switch when a specified threshold is crossed. However, as mentioned earlier, this technique of employing bandwidth as the only criteria has problems as the available channel capacity changes with device range; this issue is addressed by two policies that use dynamic channel capacity detection for switch-up, and two different switch-down techniques.

A number of techniques to indirectly determine available channel capacity, such as indexing off of the measured channel RSSI, transmit power, or link quality, were investigated as a means to provide a dynamic switching threshold. None of these techniques, however, proved successful because the underlying metrics were either too unstable or not sufficiently correlated to actual channel capacity.

4.2.1 Switching Framework

The basic switching decision is made between a mobile device and a CoolSpot enabled base station (BS), both of which possess Bluetooth and Wi-Fi capabilities. The Bluetooth PAN profile is used to provide a standard IP channel. The mobile device is responsible for making the primary policy decision: it monitors the appropriate channel characteristics and (de)activates the Wi-Fi radio when necessary. Also, it is responsible for communicating the switch to the base station in order to alter traffic routing (i.e., route packets across either the Bluetooth or Wi-Fi link). The Bluetooth radio link is always kept active except when the device is out of range, but it is not used for communication while the Wi-Fi radio is active. Network traffic destined to the mobile device is managed by the BS using ARP and modifications to its local routing table. Initially an IP address on the local wired network is assigned to the mobile device, and packets meant for it are sent on either the Bluetooth or Wi-Fi radio link, as appropriate. Switching on the BS, therefore, entails a modification to the local routing table, and a similar adjustment is necessary on the MD end to send packets out the correct interface. To effect a switch, the mobile device simply sends a "change route" message to the BS, after setting up its own network interfaces. A limitation of our current implementation is the assumption that the BS is acting as both the Wi-Fi and Bluetooth base station, requiring dual function Access Points.

The switching setup assumes that a viable Bluetooth/PAN connection exists between the two devices, and that the mobile device knows the ESSID of the appropriate Wi-Fi network: although pre-configured in the experiments, it would be easy to communicate this information across the Bluetooth link. Alternatively, the device could start with a valid Wi-Fi connection, and then communicate the necessary Bluetooth connection information over the Wi-Fi link.

The CoolSpots switching framework is completely application agnostic: no application modifications are required to communicate bandwidth decisions to the underlying infrastructure. The CoolSpots framework can, therefore, work with any application, although it has to infer the optimal switching characteristics across a wide variety of application behaviors.

Each policy has a number of corresponding parameters that can be tuned to affect its sensitivity and responsiveness, such as the sampling interval, switch threshold, etc. Changes in these parameters will affect the sensitivity and reliability of the system: causing it to be more or less aggressive, which will ultimately affect system energy consumption.

4.2.2 Baseline Policies

The *wifi-CAM*, *wifi-fixed*, and *bluetooth-fixed* policies serve as baseline cases for measuring the basic system capability and performance. wifi-CAM, used as a baseline, operates the Wi-Fi radio in always-on mode, while all other policies operate Wi-Fi in power-save mode (PSM). For some of the benchmarks, one of either the wifi-fixed or bluetooth-fixed policies will often behave quite well; specifically, the Wi-Fi benchmark works well for bandwidth-intensive applications, while Bluetooth works better in low-bandwidth situations. The true strength of the switching policies is their ability to work well across the entire range of benchmarks, as well as to handle applications with dynamic workloads.

4.2.3 Bandwidth Policy

The *bandwidth-X* policies monitor the bandwidth of traffic going across the active wireless link, and trigger a switch when the measured bandwidth goes above or below the specified threshold (X). The same threshold is used for switching up and switching down. In an attempt to remove spurious transitions, the algorithm has some hysteresis: it periodically monitors the bandwidth and triggers the switch when it exceeds the threshold for a specified number of consecutive intervals. The evaluation section details a suite of static bandwidth tests that cover the range of switching thresholds.

Overall, a static bandwidth policy will perform quite well assuming it is properly tuned to the available channel. The real world is less ideal as the underlying channel capacity can change due to distance from the base station, interference, obstacles or other circumstances, and so it is hard to a priori pick the optimal static switching bandwidth. If the switching threshold is too high for a given channel, then the policy will never switch to the higher-capacity interface, limiting the system throughput to that of the current, less capable, radio. If the switching threshold is too low, then it will unnecessarily switch to the higher capacity interface, leading to wasted energy.

The bandwidth policies are primarily parameterized by the switching threshold, specified in terms of kB/s. If the measured bandwidth is above/below this value, the policy will trigger a switch to the other network interface. Implicit in this parameterization is a choice of the bandwidth measurement interval and a hysteresis component, in order to avoid switching on temporary or short spikes in measured bandwidth. For all evaluated cases, a 250 ms interval and a hysteresis constant of 6 subsequent intervals is used. These constants behave relatively well for the given workload, although we did not perform an exhaustive search on the parameterization space.

4.2.4 Cap-Static Policy

The *cap-static-X* policies use an active channel-capacity measuring technique to switch up, and then a static bandwidth threshold to switch down. A simple network ICMP echo-response (ping) round-trip time measurement is used to determine when the channel is "saturated" - a small round trip time (RTT) indicates that there is still available channel capacity, while a larger round trip times means that all the transmission slots are full, thereby delaying the ping packet. The ping RTT metric also works very well to detect other channel issues such as interference and obstacles. Switching down is accomplished just as with the static bandwidth benchmark, based on observed bandwidth across the higher capacity channel.

The basic asymmetrical nature of the algorithm is due to the asymmetry of the underlying network channels. A similar ping channel capacity detection technique can not be used for the Wi-Fi channel, in order to detect when to switch down, because the channel is likely to be under loaded even at bandwidths that are still too high for the Bluetooth channel. The primary weakness of the cap-static policy is similar to that of the bandwidth policies: the fixed switch-down threshold may not be optimal for the given channel.

The cap- policies' (including cap-dynamic, below) switch-up is parameterized by the check interval, ping latency threshold, and number of intervals checked. For all evaluated cases, the check interval is set to 250 ms, and two consecutive latencies greater than 750 ms will trigger a switch. The switch-down parameterization of cap-static is identical to the bandwidth policies.

4.2.5 Cap-Dynamic Policy

The cap-dynamic policy uses the same switch-up technique as cap-static, but instead uses a dynamically calculated threshold to affect the switch-down behavior. Specifically, it uses the measured bandwidth at the time of switch-up as the switch-down threshold. This technique dynamically captures the available channel capacity, implicitly taking into account the actual channel characteristics, such as range or interference. This dynamic capability prevents the system from either unnecessarily keeping Wi-Fi active when the Bluetooth channel would be sufficient, or erroneously switching back to Bluetooth only to find the channel is still congested. The policy does assume that the channel characteristics do not change

Table 4.3: Measured benchmark suite, with summary statistics (for a WiFi-only channel). Data transmitted is measured through the network interface, and so includes any protocol overhead.

Benchmark	**Time over Wi-Fi**	**Data Transmitted**	**Average Bandwidth**	**Data pattern**
idle	60s	0.0 MB	0 kbps	None
transfer-1	13s	6.6 MB	4482 kbps	Bulk transfer
transfer-2	27s	13.3 MB	4519 kbps	Bulk transfer
www-intel	176s	21.6 MB	1022 kbps	Intermittent data
www-gallery	150s	2.9 MB	158 kbps	Intermittent data
video150k	150s	3.1 MB	172 kbps	Real time streaming
video250k	150s	7.3 MB	402 kbps	Real time streaming
video384k	150s	8.5 MB	464 kbps	Real time streaming

significantly during the switch-up state: a shortcoming that may potentially place the system in sub-optimal configurations.

As an optimization, both the cap-static and cap-dynamic algorithms only actively measure the channel capacity when the measured network flow is above a small minimum threshold, thus avoiding the ping packets causing unnecessary network activity and energy wastage. This minimum threshold is small enough that it does not cause a switch.

Other than the ping RTT measurement frequency and threshold (described under cap-static), there is no additional parameterization of the cap-dynamic policy.

4.3 Benchmarks

A suite of representative benchmarks (Table 4.3), ranging from simple file transfer to web-browsing emulations and media streaming, provides the basis for the evaluation of the various switching policies. Based on the nature of the CoolSpot models, different benchmarks will have wildly different impacts on the dynamic switching policies: For example, the “idle” benchmark literally does nothing for an extended period of time and therefore will rely solely on the system’s Bluetooth capability, while an intensive file-copy benchmark should immediately trigger Wi-Fi for the duration of the transfer. Intermediate benchmarks with varied workloads, such as streaming media or web browsing, will provide a more accurate demonstration of the benefit for the various policies and their capabilities: these policies benefit applications that people are more likely to use on mobile devices.

Primarily, we use "Communications Energy" as the metric to compare and report the effectiveness of CoolSpots policies. This metric, which is the product of completion time and average communication power consumed, succinctly summarizes overall system characteristics: it directly represents changes in the system's behavior (power consumption) and performance (completion time). Evaluating communication energy does not make any measure of a subjective "user appreciation" time, i.e., how impatient they might become waiting for their file to download - but this analysis is beyond the scope of this paper.

Similarly, the calculations only consider the communications power of the system, that is the combined Bluetooth and Wi-Fi subsystem power, but not the power consumed by the entire device. The CoolSpots system is designed to reduce the power of the wireless subsystem, and as such is independent of the rest of the system. However, there are some indirect benefits. Streaming video to a PC/Digital Home TV to watch it there means that it is not being watched on the local LCD screen and the display can be shut down by the application (waiting for a key press to enable it), further extending battery life.

4.3.1 Baseline Benchmarks

Two basic benchmarks, *idle* and *transfer*, provide a baseline for evaluating the performance of the various algorithms, although they, in themselves, will not be indicative of real workloads. *Idle* causes no network traffic to be sent across the wireless link, while *transfer* represents a fast-as-you-can file transfer over TCP, which will consume all available bandwidth.

These two benchmarks are not indicative of a real system because they do not capture the system behavior surrounding the action itself: the process of starting the benchmark, or processing the results afterwards. Instead, they represent the asymptotic behavior of the communication channel, and assess how it would behave under extreme, and constant, workloads. Not surprisingly, idle and transfer correspond directly to the two basic wireless technologies, Bluetooth and Wi-Fi, respectively: Bluetooth was designed as a low-power always-on technology, while Wi-Fi was a high-bandwidth network replacement in which minimizing always-on power consumption was not a primary design constraint. Other benchmarks present a more realistic balance between these two extremes, something that will be more indicative of real workloads.

4.3.2 Streaming Benchmarks

The *streaming* benchmarks are a series of the same MPEG-4 video file transcoded to stream at various bit rates and transported using the Real Time Streaming Protocol (RTSP). The videos, coded at 128 kbps, 250 kbps, and 384 kbps (although the actual realized bitrates may vary), represent bit-rates suitable for mobile devices using a Bluetooth/PAN communication link. Higher bit-rates would be possible, of course, but would restrict the system to only using Wi-Fi - and would not be very feasible for a battery-constrained mobile device. Furthermore, higher bit-rates would not be necessary for watching a movie on a small-screen mobile device such as a cell-phone. The streaming benchmark is implemented using the Darwin streaming media server and VLC video player - both of which are open-source standard components.

The basic data pattern of a RTSP stream is an initial flurry of activity as the player buffers video data to smooth out jitter in the delivery time, and then is followed by a steady stream of data at the representative bit rate. The end of the movie, therefore, continues to play after data transfer has stopped, emptying out the buffer. Ideally, from an energy perspective, the system would use Wi-Fi upfront to initially fill the buffer, and then fall back to Bluetooth to handle the trickle of data and closing idle period. The video player program VLC will exit if it drops too many consecutive frames, indicating an underlying failure in the transport channel that represents an undesirable user viewing experience.

4.3.3 Web Traffic Benchmarks

The two *www* benchmarks represent a standard web browsing session, including downloading html pages, associated images, idle "think" time, and downloading large content files (such as a data sheet or other large document). The www benchmark was created by monitoring a typical web session, downloading the content locally, and then creating a script which mimics the traffic pattern for the content. Two versions of the *www* benchmark are derived from two different web sessions and have different traffic patterns, although the overall benchmark flows are similar.

The *www* benchmark comprises a variety of traffic patterns, which presents a good opportunity for a dynamic-switching algorithm to optimize overall energy consumption: The WiFi-only policy will behave poorly because it will consume a lot of power in the active state, and the Bluetooth-only policy will be very slow for downloading large images or data files. Furthermore, many individual web-pages are actually fairly small, making it more worthwhile to use Bluetooth to transfer them, instead of a higher-power Wi-Fi transfer. Overall, it is difficult to evaluate the end-user effectiveness of the *www* benchmark because changes in the download speed can have subjective effects on the user experience; therefore, the evaluation suite focuses only on the energy/power/latency evaluation of the system.

We now present an evaluation of the various CoolSpots policies for each of the benchmarks described earlier. We will first describe our experimental setup and then present energy consumption and latency results for each of the policies. We will then show the effect of location on the effectiveness of the individual switching policies.

4.4 Experimental Setup

Our experimental testbed, as shown in Figure 4.3, is designed to evaluate the behavior of the different policies across a variety of environmental conditions (i.e., distance between components). A Base Station (BS) device effectively acts as a wireless AP supporting both Bluetooth and Wi-Fi radios, and also allows

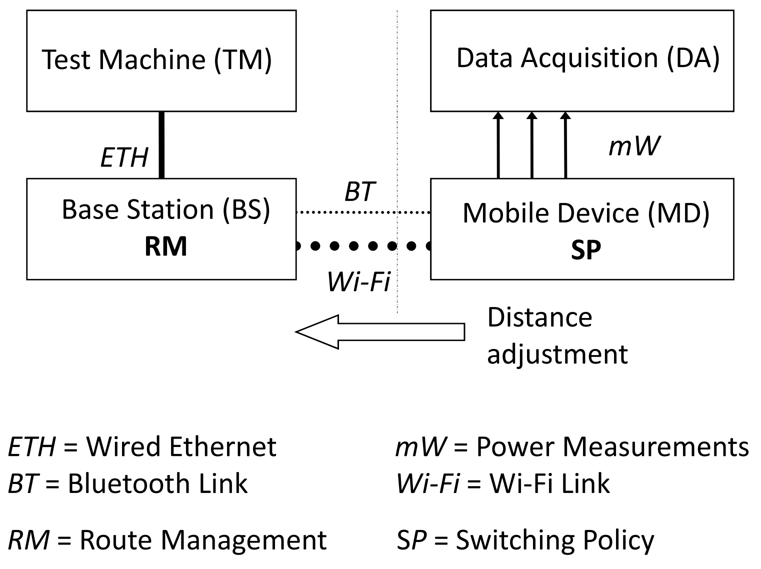


Figure 4.3: Experimental Setup. The Test Machine (TM) and the Base Station (BS) are on a cart which can be moved around to different locations.

dynamic switching between the interfaces. Likewise, the Mobile Device (MD) possesses both wireless interfaces and executes the switching policies. A Test Machine (TM) is responsible for running the test suite to send data traffic to the MD and also receive traffic from it, depending on the benchmark. The Data Acquisition (DA) machine uses specialized hardware to capture detailed power traces for the MD as described in Chapter 2. The effect of range on the CoolSpot system is measured by physically moving the test apparatus around on an equipment cart, placed at specific well-defined locations.

To exercise the test, the TM is given a file with a list of benchmarks (size N) and a file with the list of policy (size M), generating an MxN table of results. All the relevant data (including benchmark completion time) is captured by the DA. From the DA, the data can either be viewed graphically or exported to a file for processing. A post-processing script then processes the data to produce the duration of each benchmark and average power consumption for the various subcomponents for the MD: Wi-Fi, Bluetooth and total power. It does this for each benchmark/policy pair, generating an MxN table of results of time, Bluetooth power, and Wi-Fi power.

4.4.1 Hardware Specifications

The BS and MD are virtually identical hardware components based on the Stargate [21] research platform. The basic platform is based on the Intel XScale PXA255 processor, and runs a standard version of the Linux operating system. The TM is an IBM ThinkPad T42 laptop, also running Linux. The TM is connected to the BS through wired 10 Mbps Ethernet. The MD uses a Linksys WCF12 CF Wi-Fi card, that has been updated to run a more recent version of card firmware that supports PSM. The card is supported using the HostAP wireless drivers, a standard component of Linux. Unless otherwise noted, the Wi-Fi card is placed in PSM mode; the card firmware automatically switches to fully-active mode when there is significant data traffic present. Bluetooth is provided on the MD by the BlueCore3 module from CSR, supported by the Linux BlueZ Bluetooth stack. A Bluetooth/PAN profile connection provides standard TCP/IP link between the two devices. The BlueCore3 module supports multiple low-power modes, and the system is operating in a sniff mode with sleep enabled.

4.4.2 Energy Measurement

We use the energy measurement framework described earlier in Chapter 2. The individual power rails for Bluetooth and Wi-Fi are instrumented by placing separate 1%-tolerance ssense resistors in series with each radios power supply. We use energy, and not power consumption, for our evaluations since it captures both the power and time aspects of a particular benchmark. For example, if two benchmarks run and one consumes half as much power as the other, but takes twice as long, it will consume the same amount of energy. The energy numbers reported here only measure the communication components of the system (Wi-Fi and Bluetooth). Other components, such as processor, power regulators, memory, display, etc. are not included because, although they can be significantly impacted by the behavior of the wireless subsystem, they are not central to CoolSpots and

Table 4.4: Different location configurations. The bandwidth and power numbers represent the measured channel characteristics at the given range for full data transfer.

Locations	**Measured Bluetooth Bandwidth**	**Description**
Location-1	564 kbps	2 meters (line of sight)
Location-2	544 kbps	7 meters (line of sight)
Location-3	256 kbps	8 meters (through wall)

their contribution can vary widely between platforms.

4.4.3 Location Configuration

Several different locations, summarized in Table 4.4, are used to measure the impact of range on the CoolSpots system. Bluetooth only has a nominal range of only 10m, and although it is operational at this range its effective band-width is considerably reduced. The effect of distance is the primary motivator behind the cap-based policies, which can dynamically reveal channel quality. Although not directly measured, increasing the distance simulates the effect of other kinds of channel interference that reduces overall channel capacity. The range of Wi-Fi, which is on the order of 100m, is large enough that it does not factor into the measurements.

To make measurements at the various distances, the infrastructure side (TM and BS) of the test environment is located on a movable cart and manually positioned at the specified location. This measurement technique does not take into account the results of dynamic mobility, that is, movement during operation, but rather just migration of the device from spot to spot; however, this case is representative of typical usage models in a home or office environment, where people don't actually use computing very much while moving, but more commonly access computing statically in a few well-defined locations.

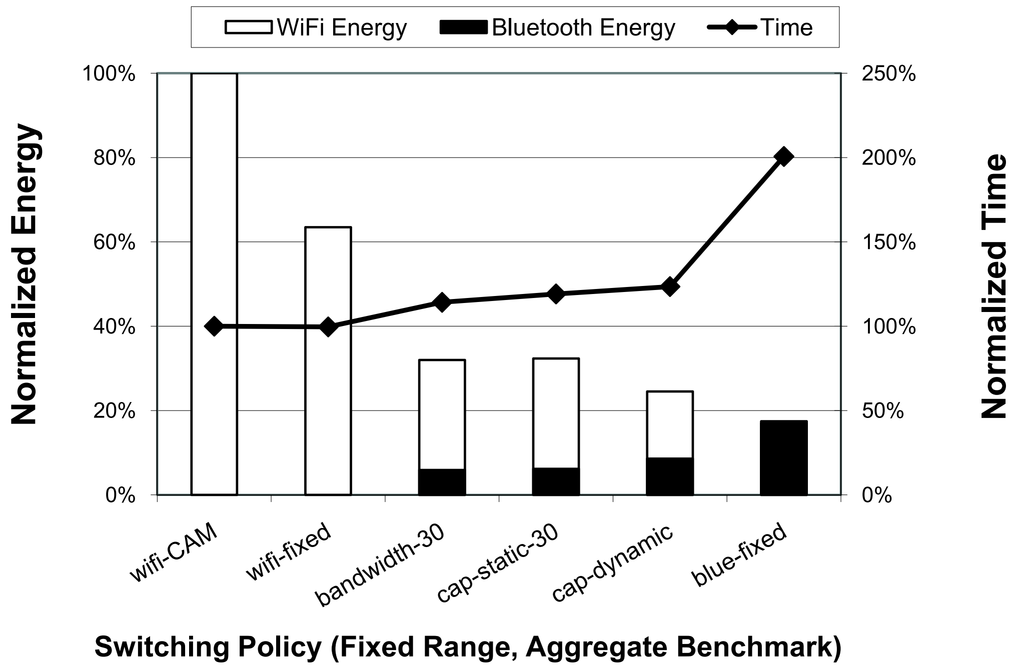


Figure 4.4: Average performance for a selection of CoolSpots policies at Location-2, across all benchmarks. Each bar summarizes the Wi-Fi and Bluetooth energy consumed, while the line represents the execution time - both normalized to Wi-Fi in fully active mode (without PSM).

4.5 Evaluation

Figure 4.4 shows an overview of the benefits provided by the CoolSpots system, using a geometric mean across the entire benchmark suite normalized to WiFi-CAM, showing both energy and time for a variety of policies. The geometric mean is a standard technique used by the SPEC [86] benchmark suite because it has desirable properties when combining results across a range of disparate benchmarks. These results clearly show how dynamic switching policies can reduce energy consumption of the wireless interface by as much as 75% (using cap-dynamic), without significantly increasing the overall delay. The Bluetooth-fixed policy has the lowest energy consumption, with a significant increase in time.

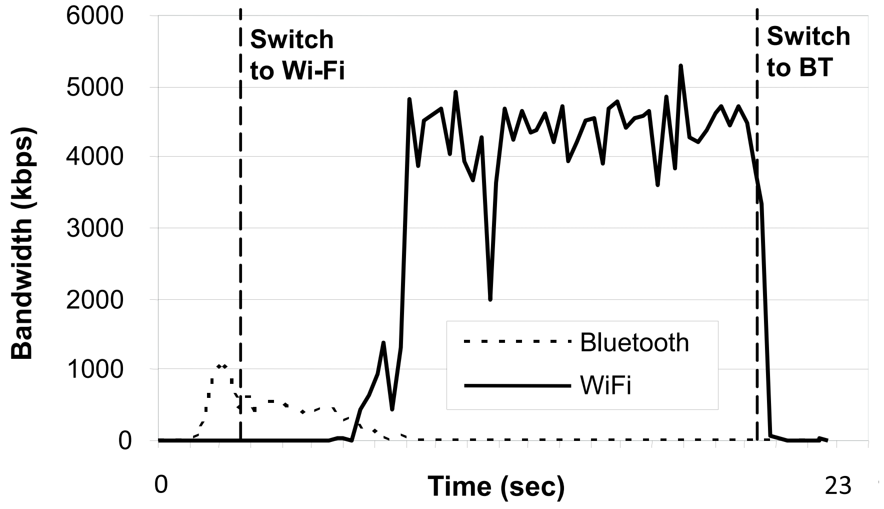
Parameterizations of the bandwidth and cap-static policies are shown as bandwidth-X, where X indicates the switching threshold, measured in kB/s, so bandwidth-30 would correspond to a 30 kB/s switching threshold. No parameterizations for the cap-dynamic, wifi-fixed, and blue-fixed policies are shown. A 30 kB/s threshold translates to a 240 kbps data stream, which is less than all the measured streaming benchmarks.

4.5.1 Characterizing Radio Switching

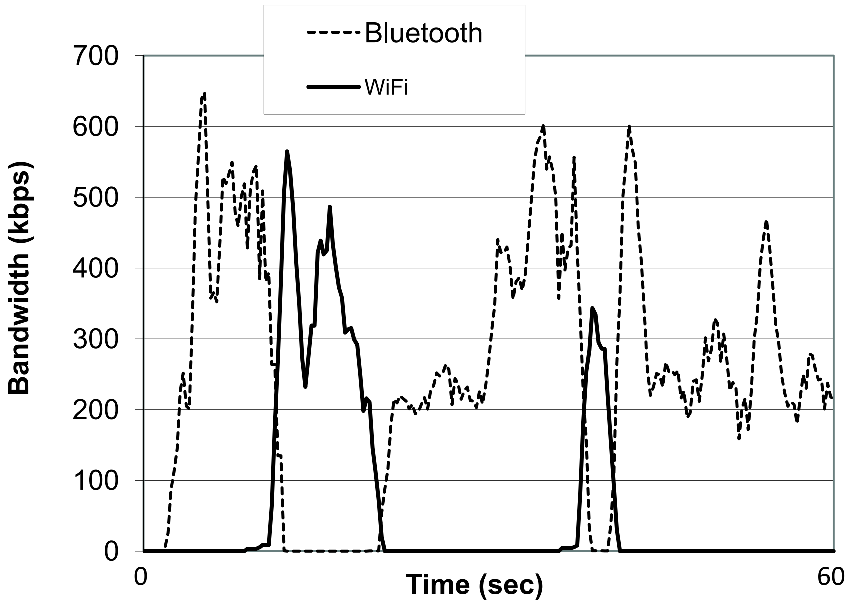
The basic switching mechanism encompasses a startup process for the higher-level radio that incurs some delay. The initial switching process can roughly be divided into four parts: pre-transfer, detection, power-up, and switched. Figure 4.5 shows the time line for both transfer-1 and video250k benchmarks using a simple bandwidth-only algorithm, showing the achieved data-rate a function of time. For transfer-1 (Figure 4.5(a), data transfer starts at 1s, and the switching decision is made just before 3s, and then Wi-Fi transfer starts around 7s. The video 250k graph (Figure 4.5(b)) shows multiple instances of switching, adapting to the dynamic load. The switch on/off events are not shown in the video 250k graph due to the compressed time scale. Since the Bluetooth channel is always available for communication, data transfer continues until the switch to use the Wi-Fi interface is complete.

The bandwidth traces shown are measured from the respective network interfaces, and so include all TCP/IP overhead (resulting in about a 7% increase over just the basic data traffic). Right before the switch-on decision there is a spike for the Bluetooth channel of up to 1000 kb/s, which is greater than the maximum Bluetooth channel capacity, which is only about 500 kb/s. This spike is caused by buffering in the transmit path, which temporarily gives the illusion that more bandwidth is available on the Bluetooth channel.

The video250k graph shows the variable data rate requirements, which are theoretically supported by Bluetooth, but occasionally trigger the Wi-Fi radio. Overall, the bandwidth-50 algorithm consumes 77% less energy than WiFi-fixed for this benchmark.



(a) *transfer-1* benchmark



(b) *video250k* benchmark

Figure 4.5: Bandwidth trace for the transfer-1 (a) and part of video250k (b) benchmarks using the bandwidth-50 policy.

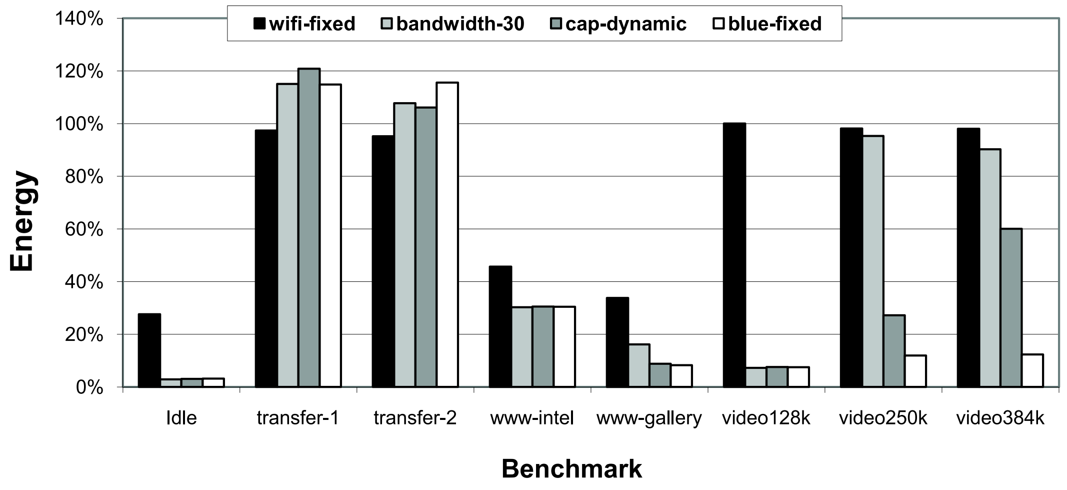


Figure 4.6: Breakdown across benchmarks for a selection of policies at Location-2, showing how the properties of the different benchmarks impact the various policies. Energy is percentage of WiFi-CAM.

4.5.2 Energy Savings for Individual Benchmarks

The effectiveness of the dynamic switching policies directly relates to the underlying benchmarks. Figure 4.6 shows a comparison of the performance of selected policies, in terms of energy consumption, across the range of benchmarks. Some, such as file transfer, offer little room for improvement since the channel is completely saturated; in fact, the dynamic policies slightly increase the overall energy consumption for file transfer as they incur the switching overhead without added benefit. In contrast, the idle benchmark is handled very well by the dynamic policies which can identify idle periods effectively and switch to Bluetooth.

The difference between the Transfer-1 and Transfer-2 benchmark shows the impact of the switching overhead for the dynamic policies. Transfer-2 handles exactly twice as much data, and so the overhead is approximately reduced by half; the WiFi-fixed and Bluetooth-fixed policies are unaffected, since they have no switching overhead. The www bench-marks provide an intermediate point between Idle and Transfer, with intermittent periods of bulk transfer combined with several idle periods.

The streaming video benchmarks are interesting to examine in the context

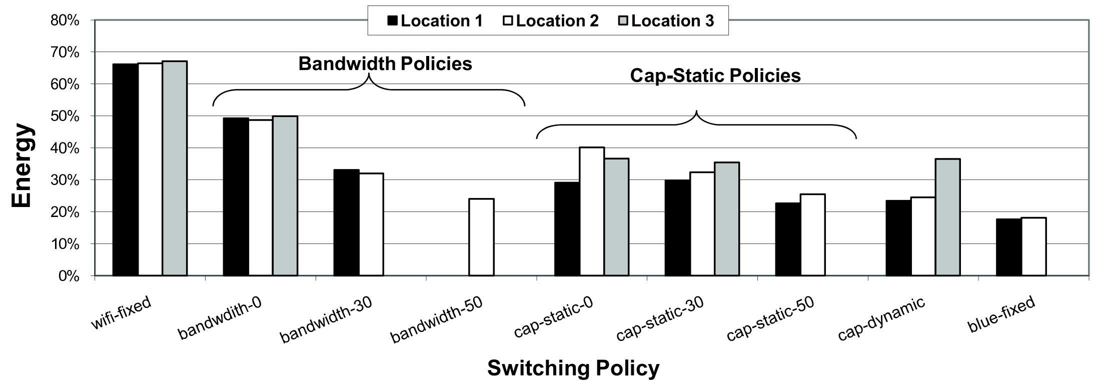


Figure 4.7: Location effect on benchmarks. Missing columns indicate that the given policy was not able to successfully handle every benchmark in the suite (at least one benchmark failed).

of CoolSpots because of their constant, unsaturated workload. Essentially, they trigger a failure point for WiFi-PSM because there is just enough data to keep the active mode of PSM activated - but there is not really enough data to warrant the Wi-Fi radio being used at all. All the dynamic policies strike a balance between the ideal case (Bluetooth-only) and worst-case (Wi-Fi): although not ideal, they successfully handle the majority of cases. The streaming bandwidth required determines how well the channel behaves: the high-bandwidth streams are more apt to trick the dynamic policies into using Wi-Fi when unnecessary.

4.5.3 Effect of Radio Ranges and Location

There are significant difference between the range of Wi-Fi and Bluetooth radios, and our results do indicate that the far end of the Bluetooth range does in fact cause significant problems for some policies. Figure 4.7 details the bandwidth benchmarks when applied to the suite of locations configurations, which shows either an increase in energy consumption or failure at some locations. The real problem with increased distances is unreliability: at distances further than Location-3, Bluetooth cuts out completely and is not a viable transport channel. This disparity highlights the benefit of the basic CoolSpots switching model, where the best available channel is used as conditions permit.

The hypothesis behind the cap-static and cap-dynamic benchmarks was that they would be able to better handle a variety of channel characteristics. As can be seen from the results, although the bandwidth policies behave very well at the short ranges, they are unable to usefully handle the longer ranges, and outright fail because they can not detect a necessary switch to Wi-Fi. The cap-static policies, although they can detect the necessary switch-up, have an inappropriate switch-down threshold and either consume excess energy, or eventually fail. The cap-dynamic policy does not succumb to this problem because it dynamically senses both the switch-up and down points based on measured channel capacity.

4.5.4 Discussion

Overall, the *Idle* benchmark highlights the necessity of incorporating a second radio channel into the system: An effective automatic switching policy will identify the idle state and power down the Wi-Fi radio, drastically lowering the overall power consumption. Alternatively, the user would need to manually activate the interface each time they wished to use it for data communication, something which not very compelling from a user experience standpoint. On the other hand, if the system is purely used for transferring bulk data then the Wi-Fi can be used exclusively and dynamic switching would not be necessary - but, it is unlikely that any general-purpose mobile device would be used in this fashion.

The *bandwidth-0* policy is conceptually very similar to the use of the second low-power radio purely as a wakeup channel [5]: as soon as any data transfer is necessary, the higher-power radio is activated. From this, it is easy to see how the more generic CoolSpots model realizes a large energy savings because it can sometimes utilize just the low-power radio for data transfer, without unnecessarily activating the higher power one. Technically, the bandwidth-0 algorithm still uses the low power radio for communication, until the higher power radio has been activated, a detail which only strengthens the argument.

Note that the measured results shown in this section focus on the communication subsystems of a device only, and do not include energy consumed by the rest of the system. One additional side effect of a slower policy like Bluetooth-fixed is that although it may consume less energy for communication, it will keep the entire device active for a longer duration, reducing overall battery life because of the associated system-level power drain.

All the policies are subject to parameterization; however, the more they are specifically tuned for a specific situation the less likely they are to work well in others. The bandwidth and cap-static policies can either be tuned more towards energy savings, or towards performance - but the problem with optimizing for performance is that they become unreliable under some circumstances. The strength of the cap-dynamic policy is that it is relatively free of tuning - there is only one ping threshold parameters - and therefore is better able to handle a wide variety of channel characteristics.

4.6 Summary

In this chapter we have presented the CoolSpots architecture that provides a seamless way for mobile devices to automatically reduce their power consumption during wireless communication. Without requiring any application modification, the system utilizes collaboration between multiple data radios to realize a greater than 50% energy savings across a representative suite of benchmarks when compared against standard WiFi-only power-saving techniques.

We have explored several switching policies that form the basis for switching between the wireless interfaces. The simplest policies, based on bandwidth monitoring, do very well under constrained channel conditions but have a difficult time adapting to greater communication ranges. A more adaptive algorithm (cap-dynamic), based on active channel measurements, is very effective at recognizing the appropriate instant to switch interfaces across a variety of channel conditions, yielding a robust and energy-efficient solution. For applications with real-time traffic patterns, as represented by streaming media benchmarks, the standard Wi-Fi power save mode (PSM) performs poorly as it keeps the radio active. The cap-dynamic policy in CoolSpots, however, proves very beneficial: saving between 40% and 92% power (over Wi-Fi CAM) for various streaming benchmarks.

This ability to adapt to steady-state low-bandwidth applications is a strength of a multiple-radio system.

Although CoolSpots is the first research effort to have a real implementation of a collaborative multi-radio architecture and to employ dynamic switching based on policies, it does have several limitations. First, the CoolSpots architecture necessarily requires a dual function AP integrating both Bluetooth and Wi-Fi radios. This increases the bar to deployment since existing Wi-Fi APs would need to change. Second, the CoolSpots architecture does not address the scalability of radio collaboration, specifically how well the policies behave when there are multiple devices, all dynamically switching radios. We address these issues, and other factors that must be considered when deploying a collaborative radio architecture, in the next chapter.

Chapter 4, in part, is a reprint of the material as it appears in Proceedings of ACM Mobile Systems, Applications and Services (MobiSys '06), June 2006. Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, Roy Want. The dissertation author is the primary investigator and author of this paper.

Chapter 5

Deploying a Collaborative Radio Infrastructure

In the previous chapter we introduced CoolSpots and explored switching policies that guide the selection of the appropriate radio interface in a collaborative multi-radio system. We also made several simplifying assumptions, for example availability of an Access Point that integrates multiple radios. Furthermore, the switching policies assumed a single client scenario and as a result the choice of the appropriate radio interface to use was purely client driven, without considering the effect on other clients which may also be using multi-radio switching.

In this chapter, we present a major generalization of our collaborative multi-radio architecture, which addresses the above limitations and as a result significantly lowers the bar to deployment. To do so, our proposed SwitchR system must overcome several challenges. First we must devise a switching policy that takes into account not only local knowledge of the wireless channel as seen by a communicating client, but also the traffic patterns of other simultaneously communicating clients. Second, given an existing wireless network infrastructure (such as Wi-Fi), we need to devise a switching architecture that allows incremental insertion of low-power access points that enables the clients to transparently switch networking technologies without any application-level modifications. Finally we must ensure that dynamic switching among radios is not only energy efficient when considering the overheads due to the switching decision, but that it also meets the quality of service requirements of diverse applications running on different clients.

There are practical aspects to these challenges: we must find ways to re-engineer the communications infrastructure while ensuring its easy adoption within existing wireless networks and using existing applications. In this chapter we show that indeed the envisioned multi-client switching policy that uses both local and global channel information can be implemented and leads to much more energy efficient switching decisions than can be taken by clients independently. This chapter highlights three primary contributions towards an effective multi-client multi-radio switching system:

1. The SwitchR energy-saving switching architecture, which utilizes *independent* low-power Bluetooth enabled APs that are incrementally deployable within an *existing* Wi-Fi infrastructure.
2. A multi-client switching policy and its detailed characterization and analysis that enables energy efficient communication and networking among multiple simultaneously communicating clients within a multi-radio environment.
3. Detailed analysis of how multi-client switching affects real-time media applications, including those based on the increasingly popular Voice-over-IP (VoIP) protocol, that have stringent QoS requirements.

5.1 SwitchR Architecture

The SwitchR architecture, shown in Figure 5.1, introduces a low-power Bluetooth Gateway (BTG) device into already existing Wi-Fi infrastructure networks. The BTG utilizes the Bluetooth PAN profile [30] to provide network layer (IP) connectivity to other Bluetooth devices. The Wi-Fi AP is connected to the backbone network over an Ethernet link, while the BTG can be connected to the backbone network either over Ethernet or over Wi-Fi. Individual mobile devices (MDs), initially connect to a Wi-Fi AP (WFAP), just as they would when accessing a Wi-Fi hot-spot, but then can optionally transition their connection to a BTG: enabling them to switch off their Wi-Fi radio as desired. In the scenario il-

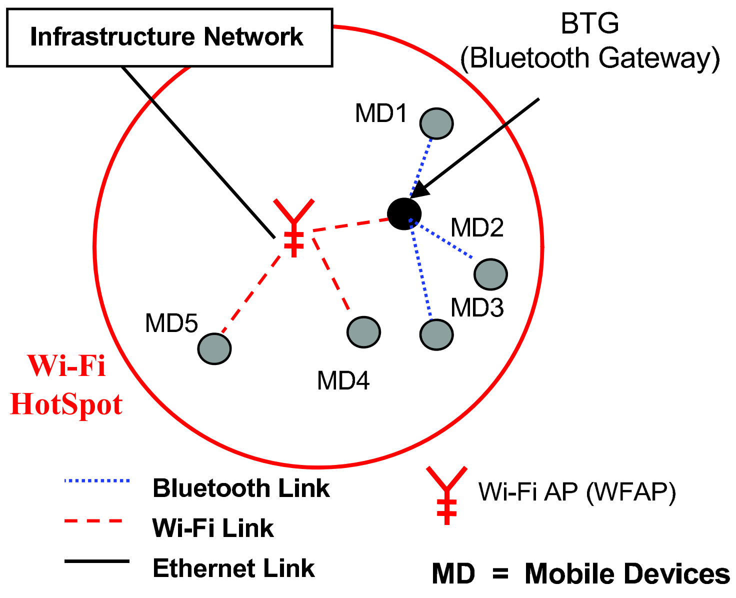


Figure 5.1: System Architecture

lustrated in Figure 5.1 MD1-MD4 are in range of the BTG and thus can connect to the network through the BTG (MD1,MD2,MD3), or the Wi-Fi AP (MD4). Since MD5 is not in range of the BTG it can only use Wi-Fi.

A key contribution of our system is the mechanism for switching between the two network access points/gateways. Switching in the SwitchR architecture is accomplished transparently for MDs with active network connections, minimizing both switching time and connectivity disruption. Since the WFAP and BTG are separate access points, additional care is needed to facilitate this transition and have packets efficiently routed to the appropriate MD. Details of the switching mechanism are discussed further in Section 5.2.

5.1.1 Separating the Wi-Fi AP and the Bluetooth Gateway

As mentioned earlier, one of the primary challenges of our switching architecture was to be easily deployable within existing Wi-Fi infrastructure. Thus, the two main components of our SwitchR architecture are regular Wi-Fi APs, which can be any 802.11 access point that is part of an existing infrastructure, and a Bluetooth Gateway (BTG) which is a device that functions as a Bluetooth AP. While a mobile device (MD) is communicating using its Bluetooth interface, its network traffic is routed through the BTG, which serves as a gateway to the infrastructure network, allowing the MDs Wi-Fi interface to be switched off. Subsequently, when an application executing on the mobile device requires a higher bandwidth connection, the MD can turn on its Wi-Fi interface and access the Wi-Fi infrastructure directly.

5.1.2 Handling Multiple Clients

In the previous chapter, we presented an evaluation of switching policies for a single client scenario with a co-located AP setup; i.e. both Wi-Fi and BT interfaces were attached to the same AP. When considering the scenario of multiple MDs communicating simultaneously in a multi-radio environment, making optimal switching policy decisions becomes much harder. For the single MD case, the MD had to determine whether the "quality" of the Bluetooth channel was satisfactory for its application requirements. In the case of multiple communicating MDs, the policies for switching between various interfaces must now take into account the dynamic nature of the Bluetooth channel as the presence of other MDs affects the total bandwidth available, in addition to the link quality of the Bluetooth channel between one particular MD and the BTG.

In Coolspots, an MD was able to estimate the Bluetooth channel condition as it was the only communicating client, with no other cross traffic. However, in the general case of multiple communicating clients there is only a limited amount of information that an MD can independently gather about channel utilization. Another alternative is for the BTG to control the switching decisions for the various MDs, since it has a global view of the Bluetooth network. The BTG however, has no knowledge of the communication needs of individual applications running on an MD. A hybrid approach that takes into account both the MDs application requirements and the effective capacity of the wireless channel is thus needed to design effective switching policies.

5.2 Switching Mechanism

Switching between the WFAP and BTG in the SwitchR architecture is accomplished by network level reconfiguration using Address Resolution Protocol (ARP) adjustments in the network and route-table updates on the MD as well as the BTG. In doing so we can ensure that the source and the destination IP addresses (Layer-3) of traffic to or from the MD remain the same irrespective of whether the MD is communicating over Wi-Fi or Bluetooth. As a result, the switch between interfaces is completely transparent to any application executing on the MD as well as any remote clients wanting to communicate with the MD. The setup assumes that the MDs, the BTG, and the WFAP are all on the same IP subnet.

The switching mechanism is similar to that used for handoff in managed Wi-Fi deployments with multiple physical Wi-Fi APs as part of the same logical wireless network. In these Wi-Fi deployments, seamless handoff is achieved using functionality provided by the Address Resolution Protocol (ARP) protocol, which provides a means to map an IP address (Layer-3) to the associated MAC address (Layer-2). In the case of Wi-Fi deployments when a mobile client performs a handoff and associates with a different Wi-Fi AP, the new AP sends out a "gratuitous ARP" to update all nodes on the local network (subnet).

5.2.1 Switching from Wi-Fi to BT

Switching from Wi-Fi to Bluetooth is a relatively quick operation that essentially relies on the BTG receiving packets for the MD over Wi-Fi and routing them through BT which can be accomplished by the BTG handling ARP requests in the case of traffic destined for the MD (called a proxy ARP). For traffic originating from the MD, as an optimization the ARP cache entries before the switch can be sent to the BTG. This "warming" of the ARP cache prevents unnecessary delays. The important steps for the switch to BT are:

1. Adjust MD routing table to outgoing traffic over BT
2. MD sends its ARP cache to BTG; set up proxy ARP on BTG and send out gratuitous ARPs

3. Delay to let Wi-Fi buffers on the MD drain
4. Power off Wi-Fi radio interface on the MD

5.2.2 Switching from BT to Wi-Fi

Switching from BT to Wi-Fi is dominated by the latency incurred by powering up the Wi-Fi interface and the subsequent association with the WFAP. After that point, it is a fairly quick process to switch traffic over to Wi-Fi. Similar to the switch to BT, it is necessary to warm the local ARP cache for the new Wi-Fi interface to prevent unnecessary delays. The important steps for the switch to Wi-Fi are:

1. Power on the Wi-Fi radio interface on the MD
2. Wait until MD can contact the WFAP over Wi-Fi. Then adjust MD routing table to send outgoing traffic over Wi-Fi
3. Fetch ARP table from BTG to warm local cache
4. Send gratuitous ARP to redirect MD-bound traffic through Wi-Fi
5. Release proxy ARP on BTG

5.2.3 Handling Mobility

Our use model assumes that MDs are typically *nomadic*, i.e. they are mobile however they remain in several well defined areas (where a BTG is available for example). When an MD moves out of coverage of the BTG, there is an implicit disconnection of the Bluetooth connection: The MD will switch its connection to Wi-Fi automatically to maintain connectivity. Effectively, a device moving out of range is handled using the same mechanism employed for handling a highly congested Bluetooth radio channel. In case the MD subsequently comes back in range of the BTG it can reestablish the Bluetooth connection and resumes its use of the SwitchR architecture to save energy.

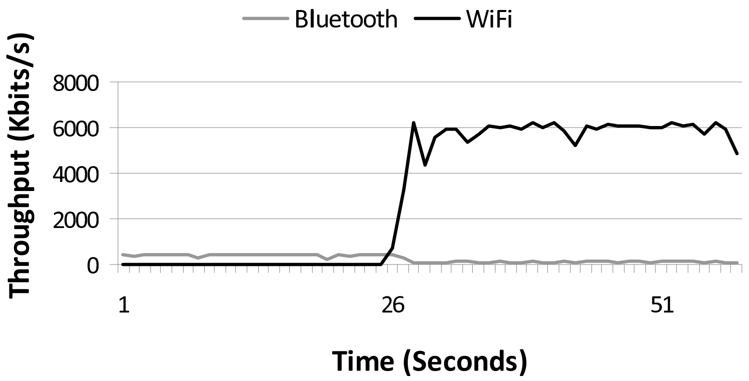
5.2.4 Baseline Switching Analysis

Figure 5.2 shows a basic characterization of the switching mechanism of the SwitchR architecture under various operating conditions. For streaming applications we measure jitter and packets loss, to quantify the effect of a mid-stream interface switch. Figure 5.2(a) illustrates a single TCP transfer session that starts off on Bluetooth with a switch to Wi-Fi is triggered, with an associated rise in observed TCP throughput once the switch to Wi-Fi is complete. Figure 5.2(b) illustrates a switch from Bluetooth to Wi-Fi and subsequently back to Bluetooth for a single 128Kbps UDP stream in an unloaded wireless environment (no cross traffic). As can be seen from Figures 5.2(a), 5.2(b) and 5.2(c), throughout the dynamic switching data transfer continues through at least one interface without interruption, highlighting the seamlessness of the switching mechanism.

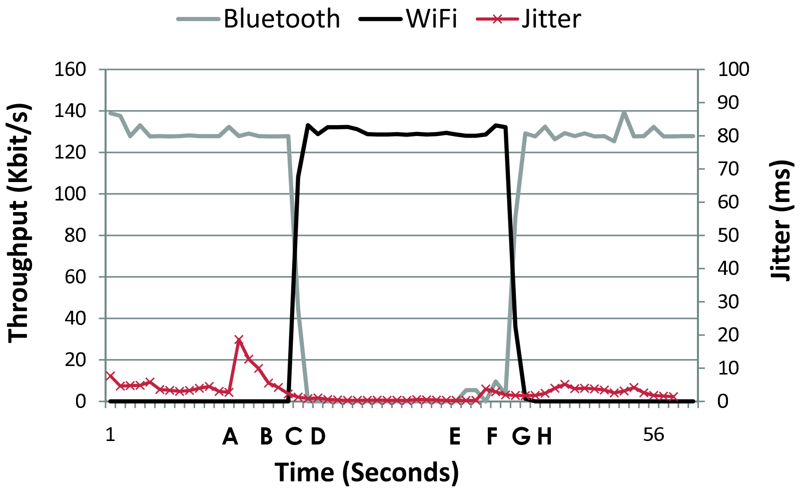
Figure 5.2(c) shows switching of a 128kpbs UDP stream when the wireless channels are loaded with other cross traffic: a 156Kbps UDP stream over Bluetooth, and a TCP transfer over Wi-Fi. This graph illustrates that in the case of a loaded channel the jitter of the UDP stream rises above 20ms for a short period, but stays below 50ms (jitter requirement for VoIP). The two spikes in the jitter curves appearing in Figures 5.2(a) and 5.2(c) are a result of the MD communicating with the BTG as part of the interface switch protocol. Additionally, the time taken for the loaded switch to complete is slightly longer because some of the phases of switching require sending a message to the BTG or WFAP, which takes more time when the network is loaded.

5.3 Switching Policies

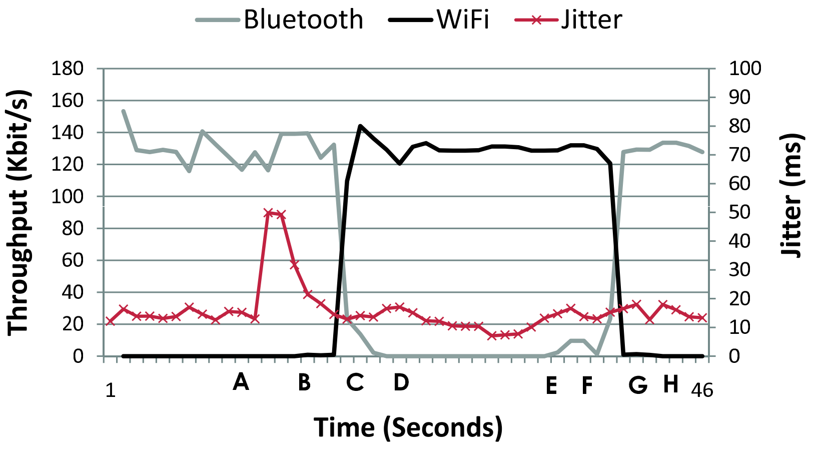
Similar to CoolSpots, the two main switching policy decisions to be made are:(a) When to switch on the high power, high throughput (Wi-Fi) radio, and (b) when to switch back down to the low power, low throughput (Bluetooth) radio. Excessive switching can potentially increase power consumption and adversely affect applications, on the other hand inadequate switching will lead to inefficient operation. Furthermore, since the wireless is a shared medium, the switching deci-



(a) TCP transfer



(b) 128Kbps UDP Stream (No Cross Traffic)



(c) 128Kbps UDP Stream (Under Load Conditions)

Figure 5.2: Dynamic Switching Profile.

sions indirectly affect other nodes in the system, calling for policies that are aware of other nodes in the network.

Several simple policies are included to create a baseline comparison, while a client-focused policy (cap-dynamic), represents the benefits of interface switching when a particular client only considers its own requirements. Finally, a multi-client policy, which considers all the nodes in the network, represents the added-value of the SwitchR architecture.

5.3.1 Baseline Policies

The *wifi-CAM*, *wifi-PSM* policies serve as baseline cases for evaluating the energy and performance behavior of the system. *wifi-CAM*, used as a baseline, operates the Wi-Fi radio in always-on mode. *wifi-PSM* and all the other policies use the Power Save Mode (PSM) of Wi-Fi, which essentially duty cycles the Wi-Fi radio as explained earlier in Chapter 2. We do not show results for a Bluetooth-only policy as Bluetooth bandwidth by itself is not enough to support multiple communicating clients and does not make for an interesting comparison.

5.3.2 Cap-Dynamic Policy

The cap-dynamic policy was the most energy-efficient switching policy from CoolSpots (Chapter 4), which looked at the current capacity of the Bluetooth channel in order to make its switching decision. It uses ping echo-responses as an active channel capacity measurement technique for switching up, and uses a dynamically calculated bandwidth threshold to effect the switch-down behavior.

The cap-dynamic policy works reasonably well for single client situations; however, in multi-client situations it has significant problems correctly predicting the available bandwidth since it assumes that Bluetooth channel conditions measured on switch up remain constant. If the channel subsequently becomes free, for example by some device switching to Wi-Fi or finishing its communication altogether, the other devices on the Wi-Fi channel have no way of knowing this fact. These devices will thus continue to use Wi-Fi believing the BT channel to still be congested and lose an opportunity to save energy by switching to Bluetooth.

5.3.3 Multi-Client Policy

A naive policy may cause the multiple MDs that are communicating at the same time, to independently conclude that the BT channel is busy and switch up to Wi-Fi. Unlike the cap-dynamic policy described earlier, an effective multiclient policy needs to take two metrics into consideration when switching down to BT from Wi-Fi. First, the policy needs to measure the quality of the BT channel as this places an upper bound on the total throughput the MD can possibly achieve given its location and range characteristics. The BT channel quality measurement does not capture the bandwidth that a client can get at a particular instant. Thus, the policy needs to determine whether there are other MDs actively using the BT channel at that time, and whether there is enough spare capacity on the BT to handle the MDs current application requirements. However, to estimate the spare capacity on the BT channel by the MD independently is difficult as the MD only has limited knowledge.

Taking into account these issues, the multi-client policy takes a hybrid approach to determine the appropriate switching points. For the switch-up case to Wi-Fi the multiclient policy uses the active channel quality measurement metric of multiple echo-response packets and the Received Signal Strength Indication (RSSI) of the BT link to estimate channel quality. If the average RSSI of the BT link degrades, and/or the echo-responses time increases substantially it signals a drop in channel quality. As soon as an application starts to transfer a large amount of data measured by an increase in echo-response time a switch-up to Wi-Fi is triggered.

The switch-down case to Bluetooth is a combined decision that involves the MD as well as the Bluetooth Gateway. At the BTG the maximum bandwidth $MAXBW_{bt}$ that the BT interface can support is estimated empirically and set up statically at the start of experimentation. For switching-down the policy (executing on the MD) periodically measures the average bandwidth on the Wi-Fi channel. If the average bandwidth observed on the Wi-Fi interface is greater than $MAXBW_{bt}$

then the policy reverts back to measuring the Wi-Fi channel as there is no point in switching down to BT given the current application requirements. However if the bandwidth measured on Wi-Fi is less than $MAXBW_{bt}$, then the multi-client policy performs multiple checks to determine whether it is optimal to switch down to Bluetooth.

First the policy checks the quality of the Bluetooth link by measuring the RSSI and the time for multiple echo response packets. If these parameters measured do not reflect a good enough channel the policy does not trigger a switch down to BT. In case the BT channel characteristics to the BTG are measured to be good, the multi-client policy queries the BTG and sends as a parameter the average application bandwidth requirements as measured on the Wi-Fi channel. The BTG continuously measures the total bandwidth it observes through its BT interface and in case there is some spare capacity (bandwidth $< MAXBW_{bt}$) it sends back the spare capacity to the particular MD that sent the query. Once the multi-client policy running on the MD gets this message from the BTG it can switch down to BT if the BTG reports spare capacity on the BT channel. If the BTG does not report spare capacity the policy reverts back to the state of measuring the bandwidth on the Wi-Fi interface and follows this decision process again.

Since it takes some time for the interface switch to happen, there may be a period during which the BTG has replied to a query by a particular mobile device (MD1), for available spare capacity, and meanwhile it receives another query from another device (MD2). If the BTG measures its spare capacity before the MD1 has actually switched to Bluetooth it will send an incorrect value to MD2 making both MDs switch to BT. This is only an issue if the BTG does not have enough spare Bluetooth capacity to handle both MD1 and MD2, in which case the MDs will immediately measure the channel to be congested and decide to switch back up to Wi-Fi, causing a thrashing effect. In order to prevent this, as part of the policy the BTG delays replying to any more queries from other MDs after sending a spare capacity message to an MD.

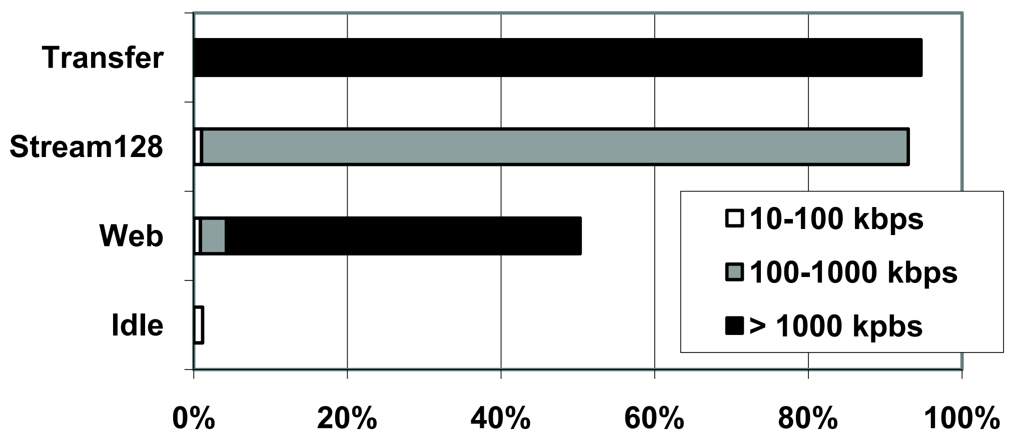


Figure 5.3: Benchmark Profile

5.4 Benchmarks

The benchmark set used to evaluate SwitchR includes media streams at various bit rates, VoIP sessions, and web browsing traces. Since our evaluation focuses on a multi-client scenario, we use a set of `n` benchmarks to constitute an application suite, where `n` corresponds to the number of MDs in our test setup. There is considerable interplay between the various benchmarks due to the shared nature of the wireless network channel, which is representative of use in real world situations. Thus it is important to consider the effects of a given benchmark in the context of other benchmarks. The important characteristic of each benchmark is the bandwidth of data transfer in each time-slice, depicted in Figure 5.3. The figure highlights the percentage of time each benchmark spends at a particular bandwidth range.

5.4.1 Idle and Transfer

The two baseline benchmarks we use are the *idle* and the *transfer* benchmarks. The *idle* benchmark is the state of the system in which there is no data transfer taking place, while the *transfer* benchmark represents a TCP stream that tries to send data as fast as it can over the wireless link.

5.4.2 Streaming

The streaming benchmark models viewing live video content on a handheld, or streaming audio MP3s. Another increasingly popular application is Voice-over-IP (VoIP), which uses either SIP or the H323 protocols [97]. All of these streaming type applications have real-time requirements and need QoS guarantees, which if not met can cause severe degradation in quality and result in bad user experience. Most media streams are sent over UDP and the two QoS metrics that are often used are jitter and packet loss. Media streaming applications can handle some packet loss by data buffering and interpolation, however large packet loss causes degradation in audio or video playback quality.

We use the standard *Iperf* tool is used to generate various sets of traffic patterns such as media streaming and VoIP. Iperf has various configurable parameters that allow customization of the UDP stream, such as a fixed data payload and a particular bandwidth, and is thus able to emulate a number of VoIP codecs. For our evaluation we emulate a commonly used VoIP codec, g711[97]. Furthermore, we use three streaming benchmarks: stream128, stream156, and stream200 with data rates of 128Kbps, 156Kbps and 200Kbps respectively. We have chosen these sample bit rates because these streams can be handled by our current BT v1.2 hardware (1Mbps). Recent Bluetooth v2.1 EDR+ hardware can provide even higher data rates (3Mbps) without any significant increase in power consumption.

5.4.3 Web Traffic

The web benchmark emulates the traffic pattern of a web browsing session. We monitored the web browsing traffic of a typical user and then downloaded the content that they visited locally. In addition, we measure the inter-arrival time between subsequent page requests capturing the user "think" time. To be consistent with our overall experimental setup, we used Iperf, which allows transferring data over a TCP connection to a remote device by reading data from a representative file. Our goal in creating this benchmark was to emulate a session with sporadic data transfer characteristics, i.e. periods of small transfers and a large transfer, interspersed with various idle intervals. This benchmark demonstrates the oppor-

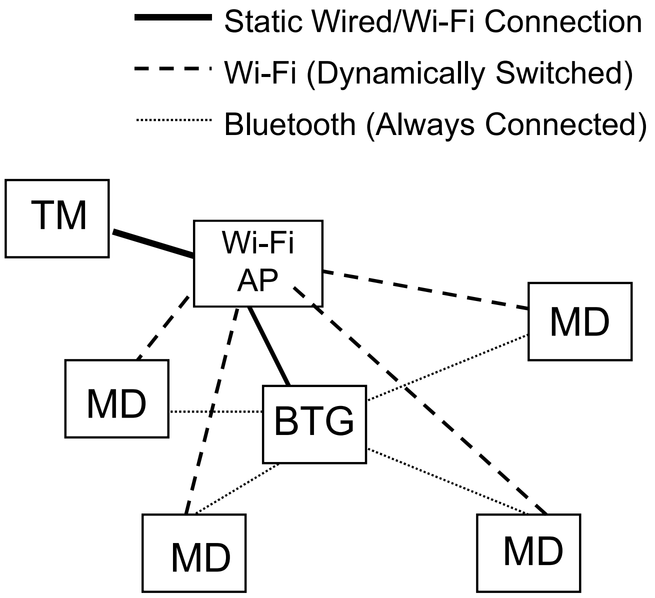

Static Wired/Wi-Fi Connection
Wi-Fi (Dynamically Switched)
Bluetooth (Always Connected)
TM
Wi-Fi AP
MD
BTG
MD
MD
MD

Figure 5.4: Experimental Setup

tunity for energy saving, especially during the “think” time, when the low power Bluetooth radio is most efficient. Another purpose of this benchmark is to evaluate the effects that the bursty and sporadic nature of web requests have on the other benchmarks, such as media streaming and VoIP.

5.5 Experimental Setup

To evaluate SwitchR, we set up an experimental test bed consisting of multiple mobile nodes placed at various fixed locations in a moderately sized laboratory (8m by 12m). Each mobile node is instrumented with an integrated power measurement capability and also monitors its own network traffic to log the amount of data transferred. Using this distributed power measurement and data logging capability, we can simultaneously measure the energy consumption for all of the mobile devices to get a detailed characterization of the overall system power consumption.

The test setup we use for our evaluation, depicted in Figure 5.4, consists of four mobile devices (MD), a Bluetooth Gateway (BTG), a Wi-Fi Access Point (WFAP), and a Test Machine (TM). The MDs and the BTG are based on the Stargate2 [43] research platform, an updated revision of the original Stargate platform [21]. The SG2 platform has an onboard Bluetooth radio (Bluecore3) and supports a compact flash slot for inserting a wireless card (Netgear MA701). The WFAP that we have used is an off the shelf wireless router from Linksys (BEFW11S4), operated in AP mode. The BTG, Test Machine (TM) and the Wi-Fi AP are all connected to a separate local subnet for the sake of performing controlled experiments and to prevent any spurious cross traffic effects. A local DHCP server leases out dynamic IP addresses to the mobile devices on this subnet. The Wi-Fi AP was set to use one of the non-overlapping frequency channels available in our building. All the other Wi-Fi APs in our part of the building are on orthogonal channels, thus minimizing any interference effects from other Wi-Fi clients.

The various MDs are spread across a laboratory room (8m by 12m), with the WFAP placed in one corner of the room while the BTG is placed near the center. The devices remain in their respective fixed locations during the experiment to ensure that the conditions are similar across multiple runs of different policies. In a mobile (pedestrian) environment the channel conditions will vary, of course. Our switching mechanism handles MDs moving in and out of range of the BTG as described earlier in Section 5.2.3.

5.5.1 Energy Measurement

To evaluate the effectiveness of our switching architecture under various switching policies and load conditions, we measure the energy consumed by the communication subsystem of each mobile device in our setup, which essentially means the Bluetooth and the Wi-Fi radios. We do not include the power consumed by other components of the SG2 platform, such as memory, CPU, etc. as our primary goal is to reduce the energy consumed for communication. The power consumed by the other components is considered the "base" power consumed by the platform, under the observation that it can vary significantly across platforms.

Although this base power is important to consider from an overall system power minimization perspective, it is not central to the concept of utilizing multiple-radios for reducing communication energy which as shown earlier constitutes a major portion of the total energy budget of a mobile device (Chapter 2).

We measure the energy consumption for all the devices simultaneously so that we can correlate the effects of energy consumption on each device with the traffic imposed by other MDs. Since our power measurement setup (Section 2.2.1 did not have the functionality of measuring the power of multiple devices *simultaneously*, we had to update our setup. We instrumented the SG2 [43] devices in our testbed to have an on-board power measurement subsystem with an integrated Analog to Digital (A-to-D) converter. We have placed sense resistors in series with all the power rails supplying its operational subsystems, including the Wi-Fi and the BT radios. To measure the energy consumption at any particular instant each device measures and logs the average power consumption of both BT and Wi-Fi at regular intervals. At the start of a test the power logs are annotated with the test parameters, and when the tests have completed the logs are collected from all the mobile devices. The energy consumption for a particular test run is then calculated in a non time critical fashion using our laboratory PCs. Using this novel capability we are able to measure the energy consumption of all the MDs simultaneously, giving us an accurate energy profile for all the mobile devices in our testbed.

5.5.2 Experimental Design

Our experimental design consists of four benchmark tests running on the four mobile devices; where in any run each mobile device executes a different benchmark. We ensure that each benchmark executes at least once on each device, factoring out any hardware variance between individual devices. In any run, all devices use the same policy; each benchmark suite is replicated for each of the four policies, resulting in 4 (benchmarks) x 4 (devices) x 4 (policies) = 64 benchmark runs for a set of results. The benchmark themselves execute in a continuous loop (since they are not necessarily the same length), and an individual result consists of a fixed-length sample of different statistics (e.g. power consumed) consisting of at least two complete benchmark executions.

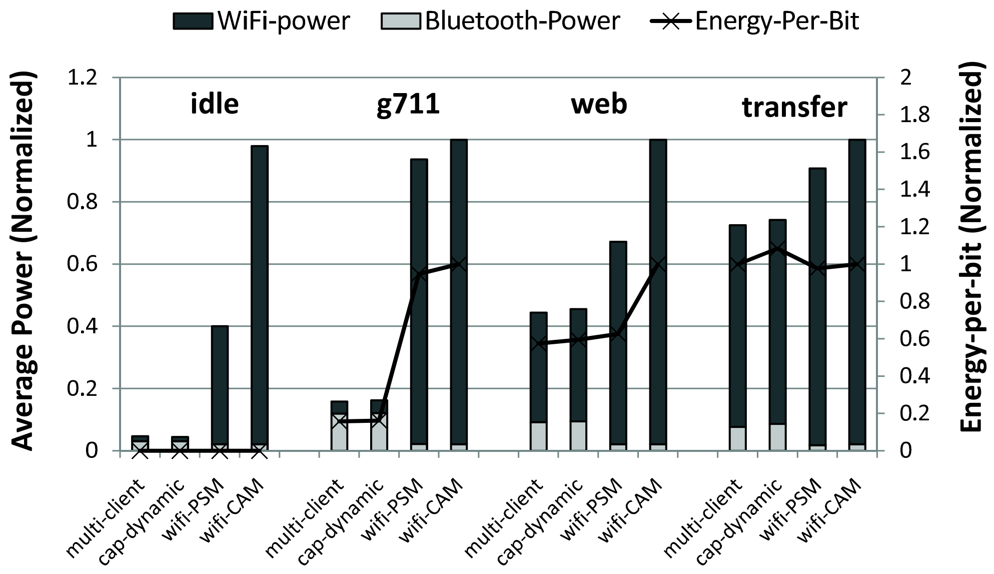
Statistics are collected independently on each device, consisting of power measurements, benchmark results (e.g., data transferred, packet jitter), and switching events. Results are post-processed by a script, which collects data from similar runs (same policy/benchmark) across the various mobile devices and aggregates results. The energy-per-bit values are calculated from the base power consumption (shown as individual Bluetooth and Wi-Fi components), and total data transferred (not shown).

5.6 Evaluation

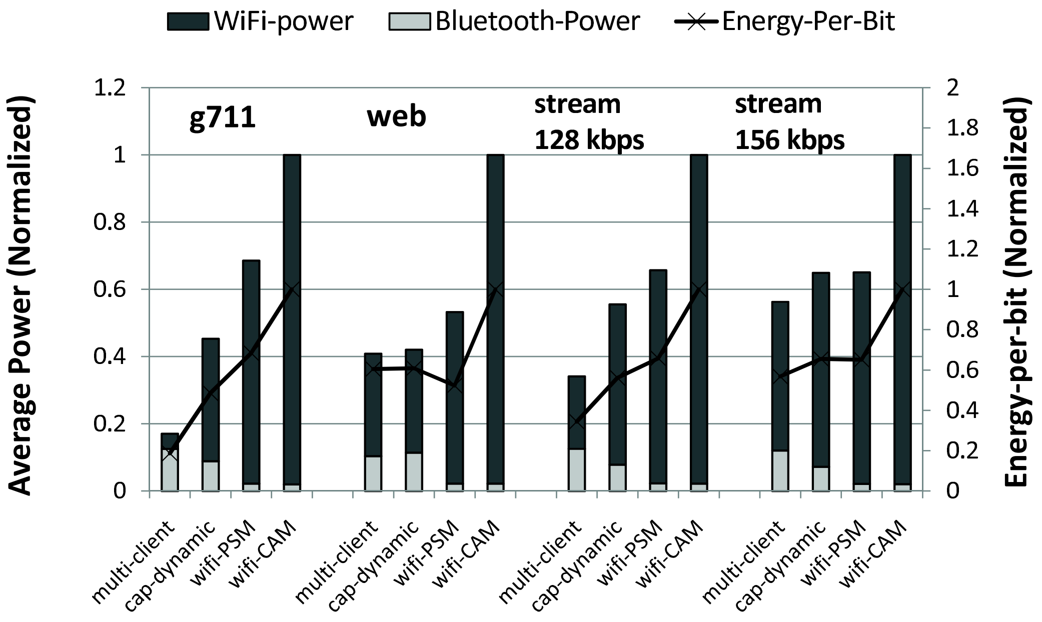
Figure 5.5 summarizes the impact of each policy for two separate benchmark suites. Figure 5.5(a) considers the four basic benchmark types, and highlights the overall effectiveness of the multi-radio switching concept, while Figure 5.5(b) considers a more loaded scenario that highlights changes introduced in the multi-client policy. For all graphs, the impact of using the 802.11 Power Save Mode (*wifi-PSM*) as compared to using Wi-Fi in the Awake Mode (*wifi-CAM*) shows how a single-radio optimization technique can impact power consumption by entering a low-power state when there is no data to transfer. The overall results are not surprising: Idle shows large energy savings, transfer shows very little savings, and the streaming media and web benchmarks show varied savings depending on context.

Note that measuring only power consumption can be misleading for some instances (such as base data transfer) because it ignores the amount of data transferred, which is captured in the calculated energy-per-bit value. So, although the dynamic and PSM policies consume less *power* for a straight transfer operation, they also decrease system throughput, resulting in a near-constant energy-per-bit value. Therefore, in most cases, a successful power-saving policy will show a reduction in the energy-per-bit along with overall power consumption.

As illustrated in Figure 5.5(b), the multi-client policy saves up to 62% over



(a) Basic Benchmark Suite



(b) Loaded Benchmark Suite

Figure 5.5: Comparing various switching policies for two benchmark suites

the cap-dynamic policy and up to 72% energy over the wifi-PSM, depending on the application. The normalized energy-per-bit for the multi-client policy for the web benchmark in Figure 5.5(b) is slightly higher than that for wifi-PSM. The reason for this increase is that the web benchmark is active a lot of time and does not exhibit a lot of contiguous idle-time; therefore, there are not enough opportunities for the dynamic switching policies to switch down to BT to save power. Other web-browsing sessions that might contain more "idle-think" time, will lead to the switching policies performing much better than the wifi-PSM policy, which keeps the Wi-Fi radio turned on.

The multi-client policy shows its main improvement for the VoIP and streaming media benchmarks, as shown in Figure 5.5(b). These workloads are relatively constant, and the corresponding switching decision is dictated primarily by the behavior of the other nodes in the system (e.g., a change in workload by the web benchmark). The primary drawback with the cap-dynamic policy is that it only considers the data traffic through the respective device itself, and ignores other traffic on the wireless channel: when the web benchmark stops transferring data, the cap-dynamic policy does not adjust to make use of the now free Bluetooth channel.

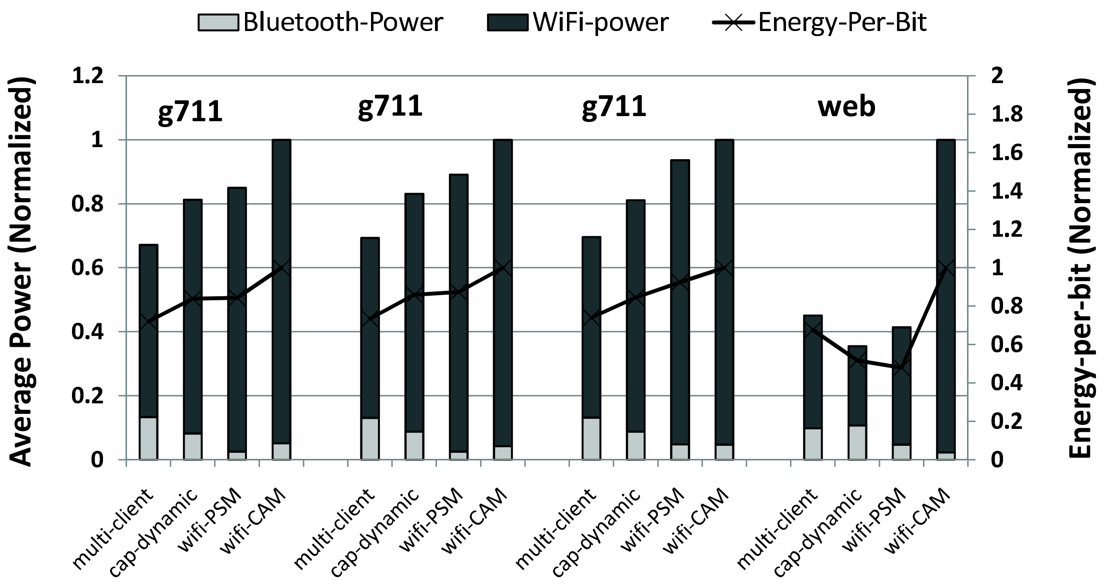
5.6.1 Media Streaming Applications

As discussed earlier, streaming media applications such as audio, video and VoIP are important for emerging mobile devices. In this section we evaluate the effect that the SwitchR architecture has on streaming media, specifically with regards to the effect that multi-radio switching has on the QoS parameters associated with such real-time traffic.

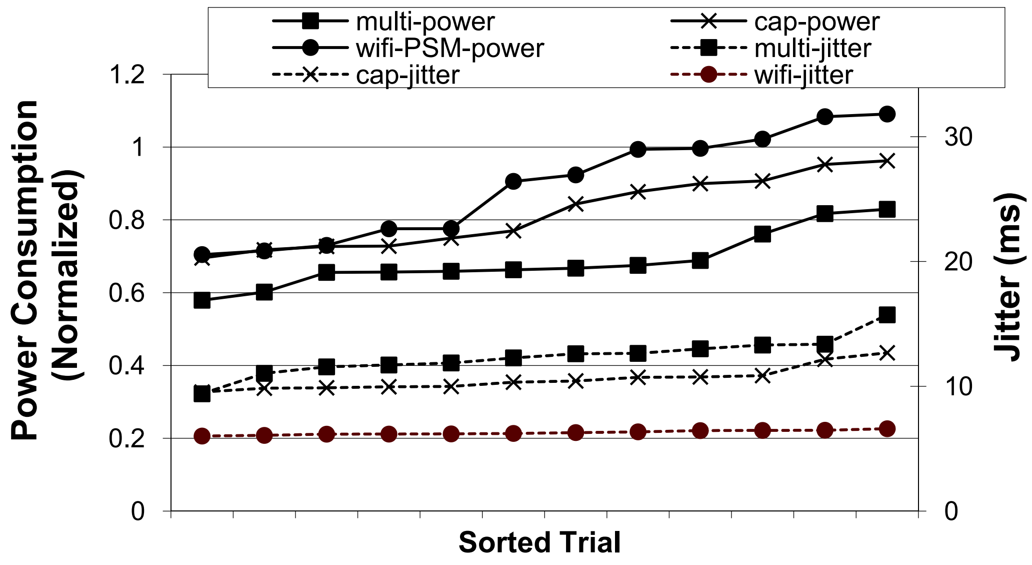
The actual bandwidth required by a VoIP session is usually quite low and depends on the codec used (8Kbps for g729 and 64Kbps for g711). The inter-packet arrival time is usually around 20-30ms, with each packet data payload being between 20 60 bytes. A VoIP session thus sends and receives a large number of packets per second, although each packet is relatively small, resulting in an overall low bit-rate. Unlike normal media traffic which can be buffered a priori to reduce the effect of jitter, voice traffic has a strict jitter requirement, which is set to be less than 50ms for continuous speech. (All VoIP characteristics are taken from[97].) We use Iperf to emulate several parallel VoIP streams to the mobile devices, for two commonly used voice codecs: g711 and g729 respectively[97]. g711 is an uncompressed codec with a bandwidth of 64 kbps x 2 (bi-directional) while g729 is compressed and uses 8kbps x 2 (bi-directional).

Figures 5.6, 5.7 and 5.8 outline results that focus on the behavior of media streaming and VoIP benchmarks. Figure 5.6(a) shows the aggregate results in a similar format as the previous section. It is important to note that any switching policy that utilizes the low-power channel for low-bandwidth traffic tends to benefit greatly in a VoIP scenario, given its low-bandwidth requirements. The Power Save Mode of Wi-Fi, which is based on duty cycling the radio during periods of inactivity, is thus not efficient due to the short inter-packet arrival time between subsequent VoIP packets. The switching policies however perform much better in terms of power consumption as compared to the baseline wifi-CAM and wifi-PSM policies since they are able to switch to the lower power radio. From the graphs (Figure 5.6(a) and Figure 5.7), it is easy to see that the multi-client policy benefits by allowing some of the VoIP streams to drop down to the lower bandwidth radio, while the cap-dynamic policy does not effectively enable this switch. However the channel capacity of the current Bluetooth hardware (400 kbps) in our testbed is less than the combined requirements of three bi-direction g711 VoIP streams (Figure 5.6(a)), thus requiring some of the streams to transition up to the Wi-Fi radio.

Figure 5.7 considers a suite of two low bandwidth g729 VoIP streams, a high bandwidth g711 stream and a web benchmark. Although the Bluetooth channel capacity should be able to handle these three VoIP streams, the occasional additional traffic induced by the web benchmark causes some of the VoIP streams to switch up to Wi-Fi. The advantages of the multi-client policy can be clearly seen as it allows some of the streams to switch down to the low power radio. The multi-client policy thus results in substantial energy saving ranging from 18% to 45% as compared to cap-dynamic policy, and from 41% to 65% compared to Wi-Fi-PSM



(a) Energy Savings



(b) Power Consumption and Jitter Distribution

Figure 5.6: Energy Savings and QoS for a VoIP Benchmark suite.

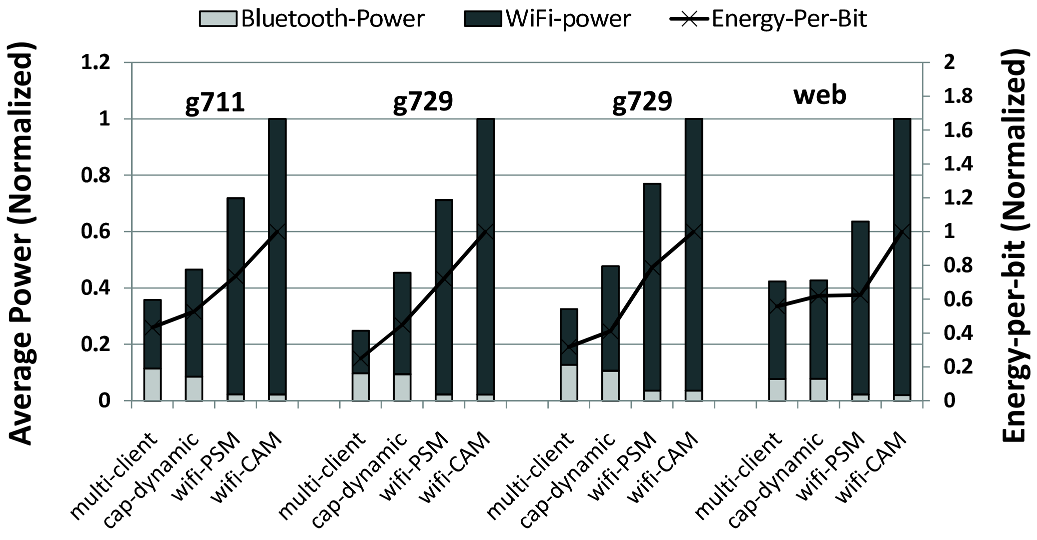


Figure 5.7: VoIP Benchmark Suite (2 X g729 and 1 X g711 VoIP streams)

for the various VoIP streams (for the g711 and g729, respectively).

Figure 5.6(b) shows a power and jitter distribution curve for the various policies applied to g711 VoIP. Each data set is sorted from low to high, showing the resulting power and jitter distribution. This data represents the individual runs of three identical g711 VoIP benchmarks (with one web benchmark also running, not shown in the figure). The 12 data samples represent four executions of three simultaneous g711 VoIP streams. As can be seen from the graph, the jitter values for all the VoIP streams are less than 20ms, well within the QoS requirements of a standard VoIP session (50ms jitter tolerance). The jitter values for the lower bandwidth g729 codec benchmark (Figure 5.7) are not shown: they were all measured to be less than 20ms. Figure 5.8 similarly illustrates a suite of three simultaneous media streams and a transfer benchmark. In the case of the multi-client policy both 128kbps media streams switch back down to Bluetooth after an initial period, thus reducing energy-per-bit by almost 40% compared to cap-dynamic and by over 52% compared to wifi-PSM.

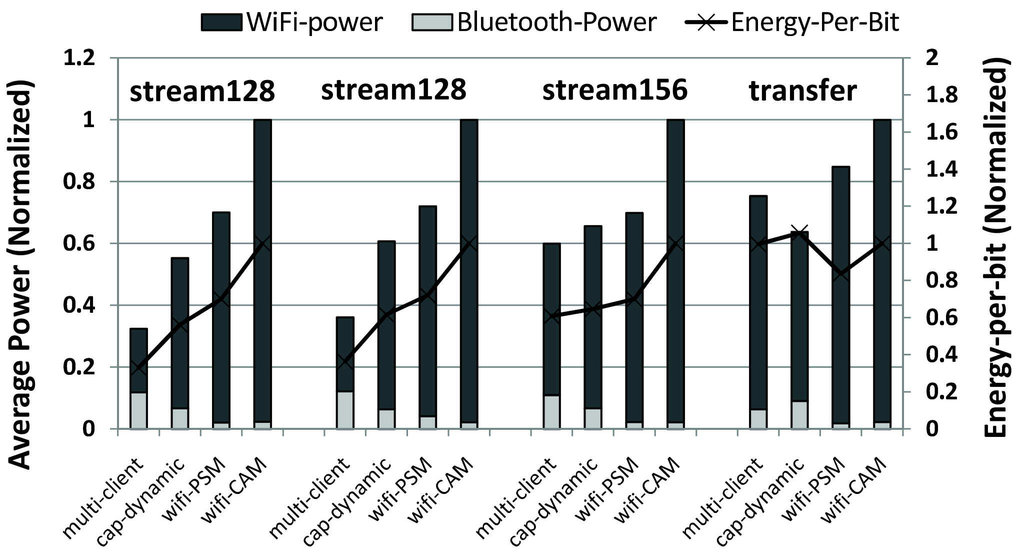


Figure 5.8: Loaded Media Streaming Benchmark Suite

5.7 Related Work – Radio Collaboration

In Chapter 2 we provided an overview of the related work in energy management on mobile devices, including techniques that are based on optimizing energy consumption of single radio systems. In Chapter 3 we then discussed prior work relating to radio collaboration, where one radio is used purely to wakeup another radio. In this section we go over prior research that particularly looks at extending radio collaboration where all available radios can be used for active data transfer.

Seamless handoff handoff between heterogeneous local-area and wide-area networks [16], has been explored extensively within the perspective of overlay networks to provide ubiquitous coverage [91, 98] and to provide bandwidth aggregation across multiple links [35]. Stem et al. [91] for example, investigate vertical handoffs between local-area and wide-area networks to provide ubiquitous coverage, but do not address the issue of power consumption. Their system design is oriented towards providing a user with the best connectivity and minimizing the disruption during a handoff latencies between heterogeneous wireless networks. Wang et al. [98] build on this work and propose policy enabled handoffs across different wireless networks. They propose a formal model to specify the policies needed to guide the handoff decision, primarily done by user interaction. The authors propose using a 3-tuple for the policy specification (bandwidth, power and cost of access), values of which are assumed to be static and specified at the start. The decision to handoff is based on the cost function of each network, based on which the "best" network is selected. Neither of these handoff schemes however, consider energy minimization as a goal to switch interfaces.

Most of these vertical handoff schemes are based on using the functionality of Mobile-IP [39, 45] to provide the underlying switching and handoff between wireless networks. Mobile-IP is an IETF communication protocol designed to provide node mobility within the Internet. There have also been proposed light-weight extensions to Mobile-IP to provide more efficient localized networking support [14, 102], but again they have not considered utilizing multiple radio interfaces for energy savings.

The idea of using multiple radios for various purposes, including a scheme called *data-on-lpr*, which suggests using the low power radio for active data transfer was proposed earlier [68]. However, the authors do not present any evaluation of the benefits of such a scheme. Multi-radio systems can also be controlled using application-level hints to determine which wireless channel would be most efficient [73, 83], unlike our collaborative radio architectures, which requires no application hints; these other systems, however, do not provide much detail in terms of their evaluations and experimental set-up. Furthermore, neither work considers the impact of range, location, scalability or other communcation clients on the system. In contrast we have built two collaborative radio architectures and extensively evaluated the switching policies that guide switching decisions between radios. In addition, we have demonstrated significant energy savings across a wide variety of applications and location configurations in real deployed testbeds.

5.8 Summary

In this chapter of the dissertation we have presented SwitchR, a collaborative radio architecture that enables mobile devices to use standard wireless applications yet significantly increase their battery operating time. A major advantage of our SwitchR architecture is that it is incrementally deployable within existing Wi-Fi infrastructure, and that it can be used without modifying client applications. Furthermore, SwitchR performs well even with multiple simultaneous communicating clients, and reduces the energy requirements of all participating devices substantially. We have shown that for a suite of representative benchmark applications, the multi-client policy enables energy savings up to 72% over Wi-Fi Power Save Mode (PSM), and up to 60% compared to the single-client policies presented in the previous chapter.

In this chapter, we have also characterized the effect that radio collaboration has on media streaming applications and real-time VoIP traffic. We show that these applications can benefit substantially by using the SwitchR architecture in terms of energy savings, even while maintaining the stringent QoS requirements placed on VoIP traffic.

We have shown the benefits of radio collaboration using SwitchR for Wi-Fi and Bluetooth radios since they are currently-available technologies that are commonly found in existing platforms. Going forward, it is important to investigate emerging technologies such as 802.11n, which is a higher-bandwidth version of the earlier Wi-Fi a/b/g standards, and Ultra Wide-Band (UWB), which is a very high-bandwidth, short-range technology. Although these technologies will present different power and performance profiles than current technologies, their different design targets (computer networking and consumer electronics, respectively), will most likely result in similar opportunities for power savings.

Chapter 5, in part, is a reprint of the material as it appears in Proceedings of IEEE International Symposium of Wearable Computing (ISWC '08), July 2008. Yuvraj Agarwal, Trevor Pering, Roy Want and Rajesh Gupta. The dissertation author is the primary investigator and author of this paper.

Chapter 6

Processor Collaboration - Energy Saving for PCs

In the previous chapters we explored ways to make mobile device more energy efficient and as a result improve their battery lifetime. We applied collaboration to duty-cycle radio interfaces more efficiently since they are the dominant power consumers in these advanced mobile platforms given their communication centric usage models.

In this chapter, we look at improving the energy efficiency of a completely different class of devices: Personal Computers (PCs), which includes mains-powered desktops and battery powered laptop platforms. Many PCs remain switched on for much or all of the time, even when a user is not present [75], despite the existence of low power modes, such as sleep or suspend-to-RAM (ACPI state S3) and hibernate (ACPI state S4) [1]. The resulting electricity usage wastes money and has a negative impact on the environment.

PCs currently can be either in an "awake" mode or in a power saving mode such as ACPI [1] states S3/S4. In the awake mode PCs are fully functional and can respond network events, but as a consequence consume significant power even when they are idle and not actively in use. In the power saving modes PCs consume much less power than when they are in the awake mode (usually over two orders of magnitude lower), but are essentially inactive and cannot respond to network events or do any processing. Utilizing these power saving modes is conceptually equivalent to duty-cycling PCs to save energy.

However, often the use models of these PCs require them to always be reachable and to always maintain their presence on the network, which implies keeping the PC in the awake mode continuously. Some of these network facing applications include ensuring remote access to local files, maintaining the reachability of users via incoming email, instant messaging (IM) or voice-over-IP (VoIP) clients, file sharing and content distribution, and so on. Unfortunately, these are all incompatible with current power-saving modes such as sleep (S3) and hibernate (S4), in which the PC does not respond to remote network events. Ideally what is needed is a hybrid mode of operation, in which PCs consumes power similar to that of power saving modes while providing some of the functionality of the awake mode.

In this chapter, we present an energy saving architecture for PCs, called Somniloquy[1], that supports continuous operation of many network-facing applications, even while a PC is in a sleep (S3) state. The key idea we are exploring is to augment PCs with a separate secondary processor that can masquerade as the host PC and respond on its behalf whenever it is asleep. The primary processor (i.e. the PC) and the secondary processor can then be used **collaboratively**, in order to duty-cycle PCs more efficiently. For this processor collaboration to work and be useful, two requirements must however be met. First, the secondary processor must be **functionally similar** to the host such that it can respond on its behalf. Second, the secondary processor must be **heterogeneous** as compared to the PC, i.e. have much lower power consumption than the PC itself.

The Somniloquy architecture proposes augmenting the Network Interface Card (NIC) of PCs to have this secondary processor. This secondary processor includes a low power CPU, small amount of memory, non-volatile flash storage and executes an embedded operating system. We show that many applications can be supported, either with or without application-specific code or “stubs” on the secondary processor. Applications simply requiring the PC to be woken up on an event can be supported without stubs, while other applications require stubs

[1]somniloquy: the act or habit of talking in one’s sleep.

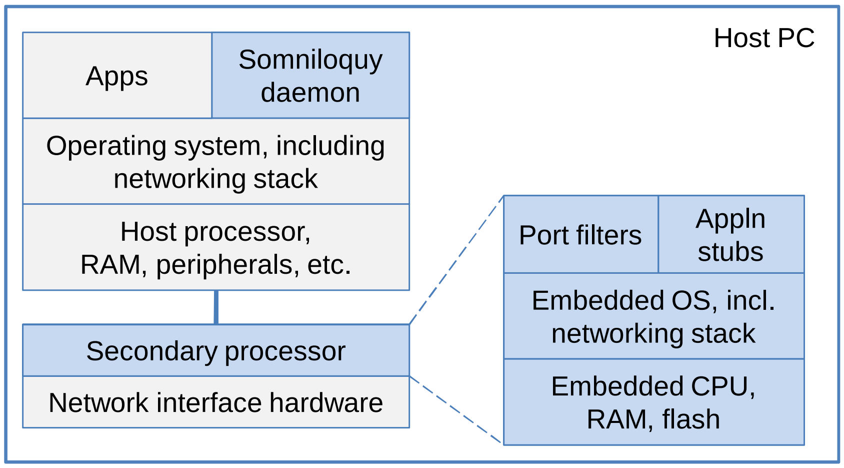

Host PC
Apps
Somniloquy daemon
Operating system, including networking stack
Host processor, RAM, peripherals, etc.
Secondary processor
Network interface hardware
Port filters
Appln stubs
Embedded OS, incl. networking stack
Embedded CPU, RAM, flash

Figure 6.1: Somniloquy Architecture.

but in return support greater levels of functionality during the sleep state.

Somniloquy does not require any changes to the operating system, to network hardware (e.g. routers and switches), or to remote application servers. We have prototyped Somniloquy using a USB-based low power network interface and implemented support for several common applications including remote desktop access, SSH, telnet, VoIP, IM, web downloads and BitTorrent. We also show that our system can be extended to support other applications. Our evaluation of Somniloquy in various settings show that a PC in Somniloquy mode consumes 11x to 24x less power than in the idle state. For commonly occurring scenarios this translates to significant energy savings of 60% to 80%.

6.1 Somniloquy Architecture

Our primary goals during the development of Somniloquy were:

- to allow an unattended PC to be in low power S3 state while still being available and active for network-facing applications as if the PC were fully on and active;
- to do so without changing the user experience of the PC or requiring modi-

fication to the network infrastructure or remote application servers.

We accomplish these goals by augmenting the PC's network interface hardware with an always-on, low power embedded CPU, as shown in Figure 6.1. The shaded portions in the figure indicate the elements introduced as part of the the Somniloquy architecture. This *secondary processor* has a relatively small amount of memory and flash storage(our prototype had 64 MB DRAM and 2 GB of flash) which consumes much less power than if it were sharing the larger disk and memory of the host processor. It executes an embedded operating system with a full TCP/IP networking stack, such as embedded Linux. The flash storage is used as a temporary buffer to store data before the data is transferred in a larger chunk to the PC. A larger flash on the secondary processor allows the PC to sleep longer. This architecture has a couple of useful properties. First, it does not require any changes to the host operating system, and second, it can be incrementally deployed on existing PCs using a peripheral network interface as will be described later (Section 6.2).

The software components of Somniloquy and their interactions are illustrated in Figure 6.2. The high-level operation of Somniloquy is as follows: When the host PC is powered on, the secondary processor does nothing; the network stack on the host processor communicates directly with the network interface hardware. When the PC initiates sleep, the Somniloquy daemon on the host processor captures the sleep event, and transfers the network state to the secondary processor. This state includes the ARP table entries, IP address, DHCP lease details, and associated SSID for wireless networks i.e. MAC- and IP-layer information. It also includes details of what events the host should be woken on, and application-specific details such as ongoing file downloads that should continue during sleep. Following the transfer of this information to the secondary processor, the host PC enters sleep state.

Although the host processor is asleep, power to the network interface and the secondary processor is maintained [1]. To maintain transparent reachability to the host while it is asleep, the secondary processor impersonates the host by using the same MAC and IP addresses, host name, DHCP details, and for wireless, the

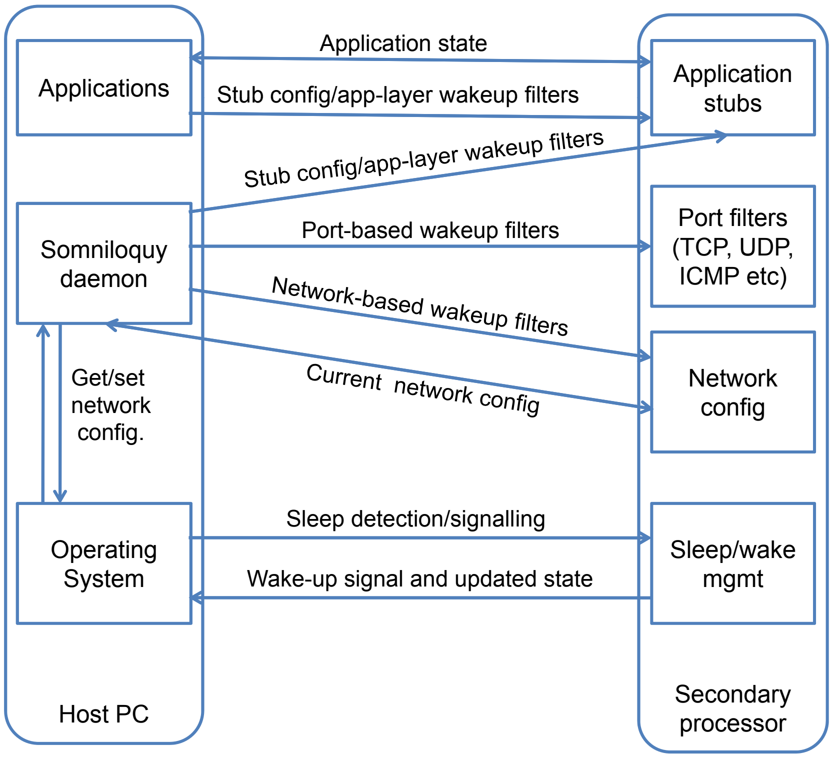

Applications
Application state
Stub config/app-layer wakeup filters
Application stubs
Stub config/app-layer wakeup filters
Somniloquy daemon
Port-based wakeup filters
Port filters (TCP, UDP, ICMP etc)
Network-based wakeup filters
Current network config
Network config
Get/set network config.
Operating System
Sleep detection/signalling
Wake-up signal and updated state
Sleep/wake mgmt
Host PC
Secondary processor

Figure 6.2: Somniloquy Software Components.

same SSID. It also handles traffic at the link and network layers, such as ARP requests and pings – thereby maintaining basic presence on the network. New incoming connection requests for the host processor are now received and handled by the network stack running on the secondary processor. In this way the PC's transition into sleep is transparent to remote hosts on the network.

To ensure that the host PC is reachable by various applications, a process on the secondary processor monitors incoming packets. This process watches for patterns, such as requests on specific port numbers, which should trigger wake-up of the host processor. Although, this simple architecture [6, 18, 32] supports several applications with minimal complexity, Somniloquy can get much greater energy savings for some applications by not waking up the host processor for simple tasks, for example, to send instant messenger presence updates. To perform these tasks on the secondary processor, we require the application writer to add a small amount of application specific code ("stubs") on the host and secondary processor. In the rest of this section we describe in more detail how we handle various applications – with and without application stubs.

6.1.1 Supporting Stateless Applications: Wakeup Filters

Stateless applications are applications for which the operating system on the host PC does not need to maintain any active state or connection information when the PC goes to sleep. The behavior of these applications is such that they require the resources of the PC directly and as a result the PC needs to be woken up on an incoming request. Somniloquy supports these applications using configurable filters. The Somniloquy daemon on the host processor specifies these packet filters, i.e. patterns on incoming packets, on which the secondary processor should wake up the host processor from a sleep state.

The Somniloquy daemon creates these filters at various layers of the network stack. At the link layer and network layer, the secondary processor can be told to wake the computer when it detects a particular packet, analogously to the magic packets used by Wake on LAN, though not requiring the MAC address to be known by the remote host (see further discussion in Section 6.5). Trigger conditions at the transport layer may also be specified, for example, wake on TCP port 22 for SSH requests.

Although the host PC will wake up within a few seconds, it will not receive the packet(s) that triggered the wake-up. One way to solve this problem is to buffer the packet on the secondary processor and replay it on the network stack of the host processor once it has woken up. However, since the time to wake up is just a few seconds, most sources can be relied upon to retry the connection request. For example, any protocol using TCP as the transport layer will automatically retransmit the initial SYN packet. Even UDP-based applications that are designed for Internet use are designed to cope with packet loss using automatic retransmissions.

This simple packet filter based approach to triggering wake-ups has the inherent advantage that application-specific code does not need to be executed on the secondary processor. Nonetheless, it is sufficient to support many applications that get triggered on remote connection requests, such as remote file access, remote desktop access, telnet and ssh requests to name a few.

6.1.2 Supporting Stateful Applications: Stubs

Several applications maintain active state on the PC even when it is idle, and hence prevent a PC from going to sleep. For example, a movie download client on a home PC (e.g. from Netflix) will require the host PC to be awake for a few hours while downloading the movie. An instant messenger (IM) client will require the PC to be on in order for the user to stay "online" (reachable) to their contacts. These are all examples of *stateful applications*, which need application specific code running on the secondary processor to be supported.

Somniloquy provides a way for these applications to consume significantly less power. The secondary processor can perform some lightweight operations on behalf of the host processor, for example send and receive presence updates to/from the IM server, while the host processor is asleep. During a large unattended download, the secondary processor can download portions of the file, putting the host processor to sleep opportunistically in the meantime.

The challenge in supporting stateful applications on the secondary processor is that by design it is limited in resources (low power CPU, small amount of RAM) and as a result cannot execute the full application that runs on the host processor. The key to supporting these applications is the use of application **"stubs"** that run on the host and the secondary processor. We have implemented stubs for three popular applications – IM (MSN, AOL, ICQ), BitTorrent, and web download. Here, we will describe the general guidelines for writing these stubs, and describe the specific implementations for the three applications in Section 6.2.

Writing application stubs: When implementing an application stub, the first step is to understand the subset of the application's functionality that needs to run when the PC is asleep. This is implemented as a stub on the secondary processor. For example, for an IM stub, the functionality to send and receive presence updates is essential to maintain IM reachability. However, the stub need not include any UI-related code – such as opening a chat window.

The next step is to decide when to wake up the host processor. Triggers can be user-defined and are application specific, for example waking up on an incoming call from a specific IM contact. Other trigger triggers conditions include waking up when the secondary processor's resources are insufficient, for example when the flash is full or more CPU resources are needed. In all of these cases, the stub wakes up the host processor.

To interface with the application on the host PC and the Somniloquy daemon, the application stub needs to have a component on the host processor. This component registers two callback functions with the Somniloquy daemon — one that is called just before the PC goes to sleep and the other just after it has woken up. The first function transfers the application specific state information to the stub on the secondary processor, and also sets the trigger conditions on which to wake the host processor.

The second callback function, which is called when the host resumes from sleep, checks the event that caused the wakeup — whether it was caused by a trigger condition on the secondary processor or due to user activity. It handles these events differently. If the wakeup was caused by user activity, the stub transfers state from the secondary processor, and disables it. However, if the wakeup was caused by a trigger condition on the secondary processor, the application stub handles it as defined by the user. For example, for an incoming VoIP call, the stub engages the incoming call functionality of the VoIP application.

Having determined what functionality needs to be supported by the application stub and host-based callbacks, and what state must pass between them, individual stubs can be implemented. We have used two manual approaches to doing this. For the web download stub, we built all the functionality ourselves based on detailed knowledge of the application protocols, and for the BitTorrent and IM stubs, we trimmed down existing application code to reduce memory and CPU footprint. An alternative could be to automatically learn protocol behavior to build these application stubs. However, we believe that this is an extremely difficult problem. There are parts of the application that are difficult to infer, and any inaccuracy in the application stub will make it unusable. For example, knowledge of how BitTorrent hashes the file blocks is necessary for the stub to successfully share a file with peers. We are unaware of any automatic tool that can learn such application behavior. Therefore, we believe that the best (although perhaps not the most elegant) approach to building these stubs is to modify application source code and remove functionality that is not required by the secondary processor.

We realize that partial application stubs might be created using tools such as the Generic Application-Level Protocol Analyzer [12] and Discoverer [22], which automatically learn the behavior and message formats for a range of protocols. As part of future work, we plan to explore how the knowledge of the protocol can be augmented with application-specific behavior to ease the development of application stubs.

When should application stubs be used? Not all applications are conducive to low-power operation via application stubs. A CPU intensive application, such as a compilation job, will be very slow on the secondary processor since it has a less powerful CPU and low memory. Similarly, an I/O intensive application, such as a disk indexer, will need to read the disk very often and will therefore need the PC to be awake. We assume that for these applications, the PC will remain awake since it is not in an idle state, and cannot use Somniloquy. Download and file sharing applications are an interesting exception, because portions of a file can be transferred by the secondary processor whilst the host sleeps.

Even for an application stub that saves energy for a given application, it is not always useful to offload the application to the secondary processor when the host PC is going to sleep. Several other applications may also want to run their application stubs on the secondary processor. This might overload the CPU of the (weaker, low power) secondary processor. In this case, it might be beneficial to keep the host PC awake.

There are a few ways to address this issue. First, users can decide and specify what application stubs they are interested in and only those stubs are executed. Second, we can detect when the secondary processor is overloaded and wake up the host processor. We take a combined approach, wherein we monitor the CPU utilization on the secondary processor and if that goes over 90% for a specified interval (>30 seconds) we wake up the host processor. The user can also override this setting and elect to keep using the secondary processor even in this state. In our Somniloquy deployment the need to move applications arose when running multiple application stubs on the secondary processor, such as two concurrent 8 Mbps web downloads and two concurrent BitTorrent downloads.

6.1.3 Quantifying Energy Savings

The amount of energy saved through adoption of Somniloquy is easy to estimate; it depends on the relative power consumption of the awake and sleep states, and the proportion of time that a machine can be kept asleep when it would previously have been awake. This ratio of the awake time of the host PC to the total time (awake+sleep) time is effectively the duty-cycle. For applications without stubs, this proportion is largely dependent on the actions of a remote user - how frequently a remote ssh session is initiated for example, and for how long. On the other hand, for applications with stubs the secondary processor may regularly wake up the host to perform some task of the other. We quantify the energy savings for an application with different wake-up intervals in Section 6.3.4.

More formally, suppose the host is woken up once every T_{sleep} seconds, whereupon it stays awake for T_{awake} seconds. T_{awake} includes the time it takes to transfer data between the PC and the secondary processor. Also assume that d is sum of the time to wake up the host plus the time to transition to sleep. Suppose:

- P_a is the power consumption of the PC when it is awake (in W)
- P_s is power consumed in sleep mode (in W), and
- P_e is power consumed by the secondary (embedded) processor (in W)

The energy (E) consumed during Somniloquy operation is given by:

$$\begin{aligned} E_{somniloquy} &= E_{PCinSleepMode} + E_{PCinAwakeMode} + E_{SecondaryProcessor} \\ &= T_{sleep} * P_s + (T_{awake} + d) * P_a + (T_{awake} + d + T_{sleep}) * P_e \quad Joules \end{aligned}$$

In the absence of Somniloquy, the amount of energy consumed by the host PC in the same time is $E_{host} = P_a * (T_{awake} + T_{sleep})$ Joules. Therefore, the ratio of energy consumed by Somniloquy compared to the host PC being always on is given by:

$$\frac{E_{somniloquy}}{E_{host}} = \frac{T_{sleep} * (P_e + P_s) + T_{awake} * (P_a + P_e) + d * (P_a + P_s)}{P_a * (T_{awake} + T_{sleep})}$$

Typically, as we show using measurements in Section 6.3, P_e and P_s are two orders of magnitude less than P_a for a desktop computer, and d is around 10 seconds (to wake up the host, and put it back to sleep). Therefore, for most energy savings, we would want T_{awake} to be much less than T_{sleep}, i.e. if $T_{awake} \ll T_{sleep}$, then the ratio $E_{somniloquy}/E_{host}$ is approximately $(P_e + P_s)/P_a$. We will present the approximate energy savings for different applications in Section 6.2.

Of course, Somniloquy could save more energy by disabling the secondary processor when the PC is awake. This would require the PC to enable the secondary processor before going to sleep, and disable it when the PC has woken up. We were unable to fully implement this functionality in our prototype, but we expect this to be easily fixable in a final productized version.

6.2 Prototype Implementation

Although the most appropriate implementation of Somniloquy would be on the network interface card of a PC itself, current NICs do not have the necessary components (low power CPU, RAM, flash storage) that we need. Instead we have prototyped Somniloquy using *gumstix*, a low power modular embedded processor platform manufactured by Gumstix Inc that support a wide variety of peripherals.

6.2.1 Hardware and Software Overview

An important goal when prototyping Somniloquy was to have it work with existing unmodified desktops and laptops, and for both wired and wireless networks. Furthermore, we required the platform to be low power, have a small form

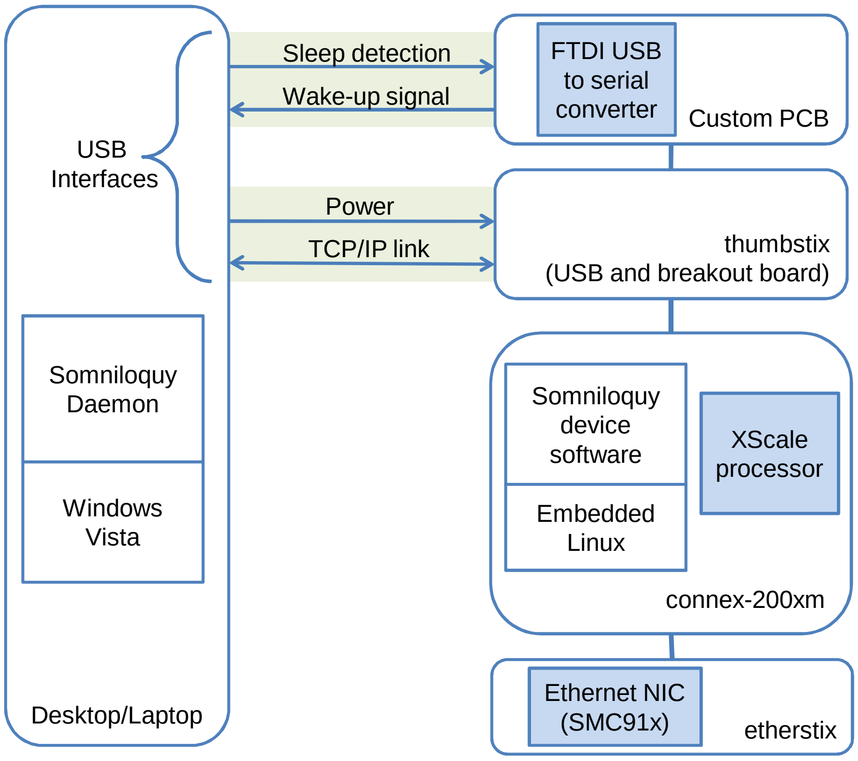


Figure 6.3: Block diagram of the Somniloquy prototype *(Wired-1NIC version)*.

factor, and be well supported for development. The gumstix platform served all these design requirements well. The specific components we use for Somniloquy include a connex-200xm processor board, an etherstix network interface card (NIC) (for wired Ethernet), a wifistix NIC (for Wi-Fi), and a thumbstix combined with a custom USB interface/breakout board that we designed. The connex-200xm employs a low power 200 MHz PXA255 XScale processor, with 16 MB of non-volatile flash and 64 MB of RAM. The etherstix provides a 10/100BaseT wired Ethernet interface plus an SD memory slot to which we have attached a 2GB SD card. We have also tested against the connex-400xm (400 MHz XScale processor) and the connex-600xm (600 MHz XScale processor). The thumbstix provides a USB connector, serial connections and general purpose input and output (GPIO) connections from the XScale.

To enable Somniloquy we needed mechanisms to wake-up the host PC, and also to detect its state (awake or in S3). To achieve this we added a custom designed circuit board that incorporates a single chip — the FT232RL from FTDI. The FT232RL is a USB-to-Serial converter chip supporting functionality such as sending a resume signal to the host and detecting the state of the host, both over the USB bus. This board is attached to the computer via a second USB port and to the thumbstix module (and thence to the XScale processor) via a two-wire serial (RS232) interface plus two GPIO lines. One GPIO line is connected to the FT232RL's 'ring indicator' input to wake up the computer. The second GPIO line is connected to the FT232RL's 'sleep' output which can be polled by the gumstix to detect whether the host PC is active or in S3.

As mentioned above (and shown in Figure 6.3), the computer is connected to the secondary processor via two USB connections. One of these provides power and two-way communications between the two processors. It is configured to appear as a point-to-point network interface ("USBNet"), over which the gumstix and the host computer communicate using TCP/IP. The second USB interface provides sleep and wake-up signaling, and a serial port for debugging purposes. The use of two USB interfaces is not a fundamental requirement, it is simply for ease of prototyping and debugging. In a productized version a single USB port would suffice.

The inherent advantage of using standard USB ports for interfacing with the host for communication, power supply and sleep signaling, is that our prototype works on any recent desktop or laptop that supports USB. We run an embedded distribution of Linux on the gumstix that supports a full TCP/IP stack, DHCP, configurable routing tables, a configurable firewall, SSH and serial port communication. This provides a flexible prototyping platform for Somniloquy with very low power operation.

We have implemented the Somniloquy host software on Windows Vista. We leverage the standard sleep states that the OS supports, specifically sleep or (ACPI state S3) since the resume time from this state is of the order of seconds. Although hibernate (ACPI state S4) support does exist in Vista, we do not use it since the resume time is substantially higher from this state. More importantly, in most PCs USB ports are powered off completely in hibernate, thereby being unable to power the gumstix in this state.

We use the OS power management APIs to trap a suspend event, and to invoke the Somniloquy daemon before the host goes into the S3 state. The Somniloquy daemon then transfers the network state (MAC address, IP address, and in the case of the wireless prototype, the SSID of the AP) and other information about the wakeup triggers and the application stubs as discussed in Section 6.1.

6.2.2 Three different prototypes

We have prototyped three different Somniloquy designs to explore different aspects of operation. The first design follows from the initial vision of an augmented network interface, as described in Section 6.1. However, in our prototype this has some performance limitations so we have also implemented a second design which uses the gumstix in cooperation with the existing high-speed Ethernet interface of a PC. Finally, we have a Wi-Fi version for use with laptops. All three prototypes are described in further detail below.

Augmented Network Interface: We call this implementation the *Wired-1NIC* version. The architecture is shown in Figure 6.3, with a photograph of the prototype with various components marked shown in Figure 6.4. In this prototype, we disable the NIC of the host, and configure the PC to use the USBNet interface (USB connection between the gumstix and the host) as its only NIC. The gumstix is connected to the network using its Ethernet connection. To enable the host PC to be on the network, we set up a transparent layer-2 software bridge between the USBnet interface to the host and the Ethernet interface of the gumstix. This bridge is active when the host is awake. When the host transitions to sleep, the gumstix disables the bridge, and resets the MAC address of its Ethernet interface to that of the USBNet interface of the host. The gumstix thus appears to the rest of the network as the host itself, since it has the same network parameters (IP, MAC address). When the host wakes up, the gumstix resets its MAC address to its original value and starts bridging traffic to the host again.

Although our *Wired-1NIC* prototype hardware supports a 100 Mbps Eth-

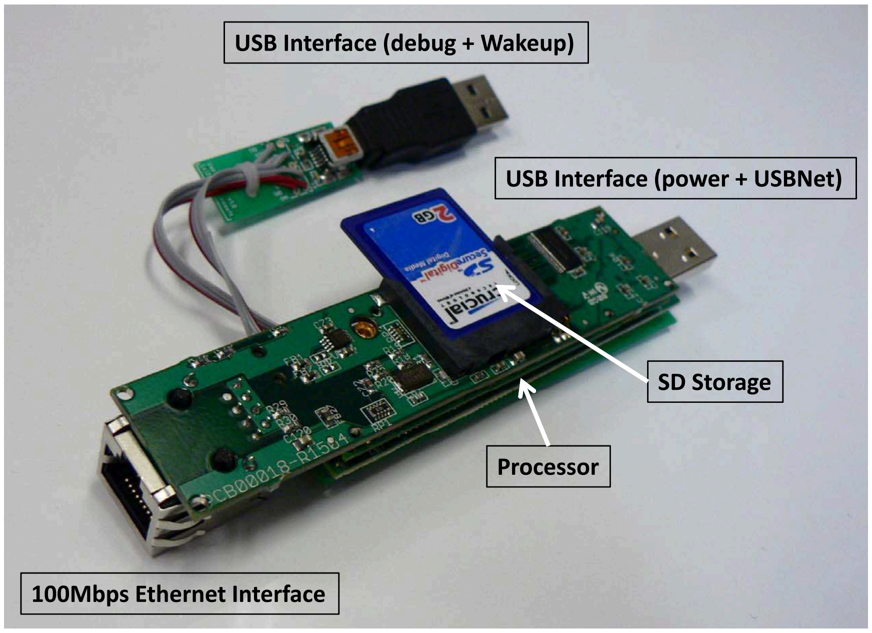

USB Interface (debug + Wakeup)
USB Interface (power + USBNet)
SD Storage
Processor
100Mbps Ethernet Interface

Figure 6.4: Photograph of the gumstix based *Wired-1NIC* prototype.

ernet interface, we are limited to a throughput of 5 Mbps due to the bandwidth supported by the current USBNet interface driver. There is also a slight overhead of bridging traffic on the gumstix. Although this limits bandwidth to the host significantly in our prototype, we note that in a final integrated version, this overhead of bridging can be avoided by allowing both the host and the low power secondary processor to access the NIC directly.

Using Existing Network Interface: Somniloquy can coexist with an existing NIC. On such systems, the overhead of bridging is avoided by using the existing Ethernet interface on the host PC for data transfer when it is awake, with the gumstix using its own Ethernet interface (while still impersonating the host PC) when the host is asleep. We have built this version where the gumstix does not perform Layer-2 bridging, and call it the *Wired-2NIC* prototype.

Using Wi-Fi: We have also implemented a wireless version of Somniloquy. We were unable to implement a one-NIC version since the Marvell 88W8385 802.11 b/g chipset present on the wifistix does not currently support layer 2 bridging. We have however implemented a *Wireless-2NIC* version.

6.2.3 Applications Without Stubs

We have implemented a flexible packet filter on the gumstix using the BSD raw socket interface to support applications that do not require stubs, e.g. RDP, SSH, telnet and SMB connections. Every application in this class provides a regular expression matched against incoming packets to decide whether to trigger host wakeup. For example, handling incoming remote desktop requests requires the host to be woken up when the gumstix receives a TCP packet with destination port 3389.

We note that waking up the host computer is not enough; the incoming connection request must somehow be conveyed to the host. We accomplish this by using the `iptables` firewall on the gumstix to filter any response to TCP or UDP packets that the gumstix does not handle itself. Thus trigger packets are not acknowledged by the gumstix and the remote client sends retries. After the host has resumed, one of the retries will reach it (since it is still using the same IP and MAC addresses) and it will respond directly. We opted for this approach of relying on retries since it is much simpler, rather than the complicated approach of buffering packets on the gumstix and inserting them on to the host PCs network stack when it has resumed. Using port-based filtering, we have implemented wake-up triggers for four applications: remote desktop requests (RDP), remote secure shell (SSH), file access requests (SMB), and Voice over IP calls (SIP/VoIP).

6.2.4 Applications Using Stubs

To demonstrate how modest application stubs can enable significant sleep-mode operation in Somniloquy, we have also implemented application stubs for three applications that were popular in an informal survey[3] we had conducted: background web download, peer to peer content distribution using BitTorrent, and instant messaging.

Background Web Downloads: We developed the web download stub for `wget` which works as follows: When the host PC transitions to sleep, the status of active downloads is sent to the stub running on the gumstix. The status includes the download URL, the offset of how much download has taken place, the buffer space available, and the credentials (if required for the download). Most popular web servers (e.g. IIS and Apache) allow these byte ranges to be specified using the HTTP ‘Accept-Ranges’ primitives [74]. The web download stub then uses the information passed to resume the downloads from the respective offsets of the files, and stores the data on the flash storage of the gumstix. If the flash memory fills up before the downloads complete, the stub wakes up the host PC and transfers the downloaded files from flash storage to the host PC, thereby freeing up space. The host PC then goes back to sleep while the stub continues the downloads. At the end of a download, the gumstix wakes up the host PC, and transfers the remaining part of the file.

The download stub consumes significantly less energy to download a file than keeping the PC awake to download it. The overhead is a slight increase in latency. We can quantify the savings and overhead using the model described in Section 6.1.3. If flash storage is F MB and the download bandwidth is B MBps, then the host PC is woken up every F/B seconds, and it is awake for F/T seconds, where T is the transfer rate between the host and the gumstix. Therefore, using the formula in Section 6.1.3, Somniloquy gives most energy savings at low B and high T. We empirically validate this observation in Section 6.3.4. When T is of the same order as B, Somniloquy may not save much energy. This can happen if the NIC supports very high rates (e.g. 1 Gbps), while the secondary processor can only support lower data rates (up to 100 Mbps) or if the transfer rate T is limited. However, we anticipate the download stub to be primarily used in scenarios where the download speeds are limited by the last mile connection of at most a few tens of Mbps – here, this stub is nearly always beneficial.

BitTorrent: For the BitTorrent stub we customized a console-based client, *ctorrent*, to run on the gumstix with a low CPU utilization and memory footprint. Prior to suspending to S3, the host computer transfers the ‘.torrent’ file and the portion of the file that has already been downloaded to the gumstix. The BitTorrent stub on the gumstix then resumes download of the torrent file and stores it temporarily on the SD flash memory of the gumstix. When the download completes, the stub wakes up the host and transfers the file.

When only downloading content, the energy saved by using this stub is similar to that of the web download stub, i.e., frequency of waking up the PC and the duration for which it is woken up depends on the download bandwidth B, the transfer speed T and the flash size F. However, when uploading/sharing (which is key to altruistic P2P applications), the energy savings are much more. The same file chunk can be uploaded to many peers, and hence the PC can sleep for much longer – implying more energy savings using the formula in Section 6.1.3.

Instant Messaging: For the IM stub, we used a console-only IM client called *finch* that supports many IM protocols such as MSN, AOL, ICQ, etc. On the PC, we used the corresponding GUI version of the IM client. To ensure our goal of a low memory and CPU footprint we customized finch to include only the features salient to our aim of waking up the host processor when an incoming chat message arrives. This only requires authentication, presence updates and notifications; we disabled other functionality. The host processor transfers over the authentication credentials for relevant IM accounts before going to S3. The gumstix then logs into the relevant IM servers, and when an incoming message arrives it triggers wakeup. The energy saved by the IM stub is thus similar to applications that are handled using packet filters (e.g. SSH/RDP), where the duration for which a host can sleep depends on the frequency of occurrence of wake-up triggers.

6.3 Evaluation

We now present the evaluation of our Somniloquy prototype. First, we present detailed power measurements and characterization of the gumstix hardware and show that it consumes much less power than a PCs by profiling standalone desktops, laptops in different power states. Second, we measure the energy saved (and latency introduced) by Somniloquy when used on an "idle" host processor. Third, we show how Somniloquy affects the performance of various applications, with and without application stubs. Finally, we quantify Somniloquy's energy savings — monetary and environmental cost for an enterprise and battery lifetime increase for laptops.

Table 6.1: Power consumption and S3 suspend/resume times for two desktop PCs.

Condition	Optiplex 745	Dimension 4600
Normal idle state	102.1 W	72.7 W
Lowest CPU frequency	97.4 W	N/A
Disable multiple cores	93.1 W	N/A
'Base power'	93.1 W	72.7 W
Suspend state (S3)	1.2 W	3.6 W
Time to enter S3	9.4 s	5.8 s
Time to resume from S3	4.4 s	6.2 s

Power and Energy Measurements: To measure the power consumption of laptops and desktop PCs, we used a commercially available mains power meter, *Watts-Up* [2]. To measure the power consumption of the standalone gumstix, we built a USB extension cable with a 100 mΩ 0.1% sense resistor, which was inserted in series with the +5 V supply line, and we used this cable to connect the gumstix to the computer. We use the power measurement setup described in Chapter 2 earlier to measure the power draw of the gumstix. All power numbers presented in this section are averaged across at least five runs.

6.3.1 Microbenchmarks – Power, Latency

Desktops: Table 6.1 presents the average power consumption for two Dell desktop machines: an Intel dual core (2.4 GHz Core2Duo) OptiPlex 745 with 2 GB RAM running Windows Vista, and a 2.4 GHz Pentium 4 Dimension 4600 with 512 MB RAM running Windows XP. The display is turned off in these experiments, and only the essential system processes are left running. In all cases the processor is idle and the hard disk is spun down. The power consumption of the desktop in S3 is two orders of magnitude less than when it is awake. This is consistent with prior published data on the power consumption of modern PCs [18]. We use the term 'base power' to indicate the lowest power active mode that a PC can be in and still be responsive to network traffic (without using Somniloquy). To get this number, we further scaled down the CPU to the lowest permissible frequency

[2]`http://www.wattsupmeters.com/`

Table 6.2: Power consumption, battery lifetime and S3 suspend/resume times for various laptop PCs.

Condition	Lenovo X60	Toshiba M400	Lenovo T60
Normal idle state	16.0 W	27.4 W	29.7 W
Backlight minimum	13.8 W	22.4 W	24.7 W
Screen turned off	11 W	18.3 W	21.3 W
'Base power'	11 W	18.3 W	21.3 W
Suspend state (S3)	0.74 W	1.15 W	0.55 W
Battery capacity	65 Wh	50 Wh	85 Wh
Base lifetime	5.9 h	2.7 h	4.0 h
Suspend lifetime	88 h	43 h	155 h
Time to enter S3	8.7 s	5.5 s	4.9 s
Time to resume from S3	3.0 s	3.6 s	4.8 s

on these desktops. Furthermore, we disabled the multi-core functionality using the system BIOS to effectively use only one core and verified that the system was actually doing so by using a processor ID utility supplied by Intel. The time taken for the desktops to resume from S3 and reconnect to the network is of the order of a few seconds (Table 6.1).

Laptops: Table 6.2 presents the average power consumption of three popular laptops: a Lenovo X60 tablet PC with 2 GB RAM running Windows Vista, a Toshiba laptop with 1 GB RAM running Windows XP, and a Lenovo T60 laptop with 1 GB RAM running Windows Vista. For all power measurements, the processor is set to the lowest speed and is idle, the hard disk is spun down and the wireless network interface is powered on. The base power is between 11 W and 22 W, resulting in a battery lifetime of around 4 to 5 hours with the batteries that are present on these laptops. Using the sleep/S3 state can dramatically extend the battery lifetime, to between 40 and 150 hours for the laptops we tested, although the laptop is unreachable in this state.

Gumstix: Table 6.3 shows the average power consumed by the gumstix (with both etherstix and wifistix) in various states of operation. The gumstix has a base power of approximately 210 mW when no network interface is present (row 1). A gumstix with an active network interface typically consumes approximately 1070-1300 mW (rows 2 and 9), however with an associated Wi-Fi interface in power save

Table 6.3: Power consumption for the gumstix platform in various states of operation.

	Gumstix state	Power
	Wired version	
1	gumstix only - no Ethernet	210 mW
2	gumstix + Ethernet idle	1073 mW
3	gumstix + Ethernet bridging	1131 mW
4	gumstix + Ethernet + write to flash	1675 mW
5	gumstix + Ethernet broadcast storm	1695 mW
6	gumstix + Ethernet unicast storm	1162 mW
	Wireless version	
7	gumstix only – no Wi-Fi	210 mW
8	gumstix + Wi-Fi associated (PSM)	290 mW
9	gumstix + Wi-Fi associated (CAM)	1300 mW
10	gumstix + Wi-Fi broadcast storm	1350 mW
11	gumstix + Wi-Fi unicast storm	1600 mW

mode it consumes only 290 mW (row 8). The power consumption of the gumstix when its network interface is active and the downloaded data is being written to flash is around 1675 mW (row 4). Broadcast and unicast 'storms' (continuous traffic) increase the power consumption by a few hundred milliwatts[3]. Importantly, the power consumption of the gumstix is approximately one tenth that of an awake laptop, and approximately 50 times less than an idle desktop, both in their **lowest** power states (denoted as "base power").

6.3.2 Somniloquy in Operation

We now report the power consumption of Somniloquy in operation. For these measurements we use two testbed systems: a desktop (Dell OptiPlex 745 with 2 GB RAM running Windows Vista) with the Wired-1NIC prototype of Somniloquy, and a laptop (Lenovo X60 tablet PC running Windows Vista) with the Wireless-2NIC version of Somniloquy. Thus, our tests span both Ethernet and Wi-Fi networks, and both the integrated single network interface, and the higher performance versions which uses the existing internal network interface. The test

[3]Wi-Fi broadcasts are sent at 6 Mbps while unicasts are sent at 54 Mbps in our setup. Consequently a unicast storm consumes more power than a broadcast storm.

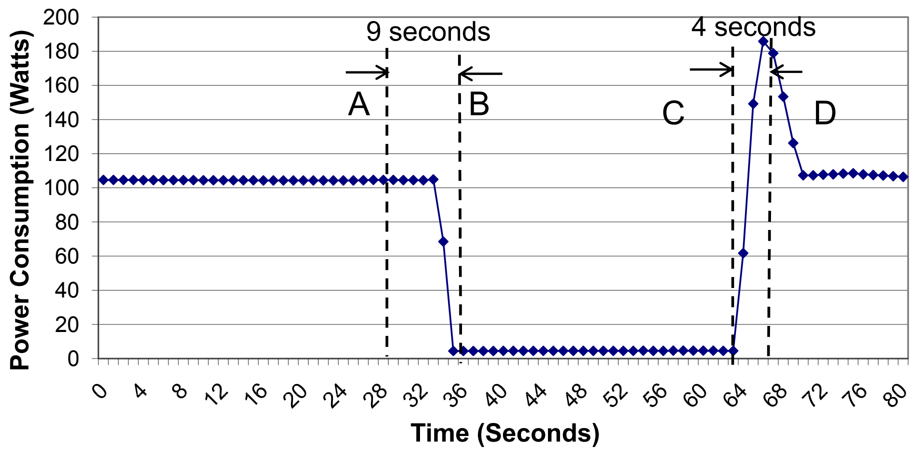


Figure 6.5: Power consumption and state transitions (desktop testbed).

traffic is generated using a standard desktop machine running on the same (wireless or wired) LAN subnet as the testbed machine.

Figure 6.5 shows the power consumption of our desktop testbed. Initially the desktop's host processor is awake and uses the gumstix for bridging, and the whole system draws 104 W of power. At time 'A' a state change to S3 is initiated by the user. This request completes at time 'B' after which the power draw of the system is approximately 4.4 W, i.e. 24x less. This power is split between the gumstix, the DRAM of the PC, and other power chain elements in the PC. Subsequently at time 'C' the gumstix, which has been actively monitoring the network interface, wakes up the host in response to a network event. This request completes at time 'D' when the host system has fully resumed. As the figures illustrate this resume event takes about 4 seconds. Figure 6.6 similarly shows a power trace for our testbed laptop (Lenovo X60). It looks very similar to the desktop trace with a starting power of 16 W with the screen on (which drops to 11 W if the screen is turned off), a power draw of 1 W when using Somniloquy (11x less than the screen-off case) and a resume time of 3 seconds.

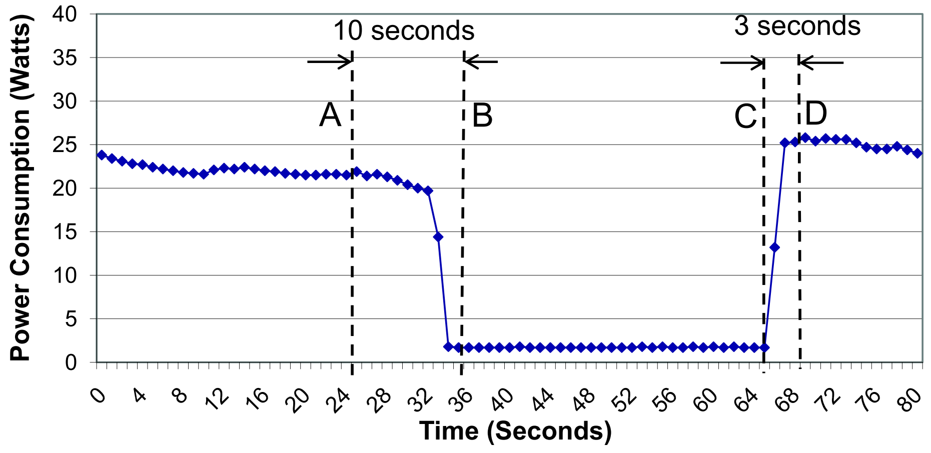


Figure 6.6: Power consumption and state transitions (laptop testbed).

6.3.3 Application Performance

As described earlier there are two classes of applications that are supported by Somniloquy: first, a large class of stateless applications that do not require application stubs, and second a smaller class of stateful applications that can be supported using application stubs running on the gumstix. We now evaluate the performance of both these classes of applications.

Stateless applications

We now quantify the end-to-end latency (as perceived by users) incurred by the applications that are handled by Somniloquy without using application stubs. For these experiments, we use the same two testbeds as above, with the addition of a third testbed based on the Wired-2NIC prototype (using same desktop machine as the Wired-1NIC case), providing a direct comparison between the 1NIC and 2NIC cases. In each case the latency reported is the mean over five test runs.

Figure 6.7 reports the time taken to satisfy an incoming application-layer request for four sample applications. For each application, we show the latency for "awake" operation (i.e. when the host is on and directly responds to the request) and when the host is in S3 and Somniloquy prototype receives the incoming packet and triggers wake-up of the host. The four applications we tested are:

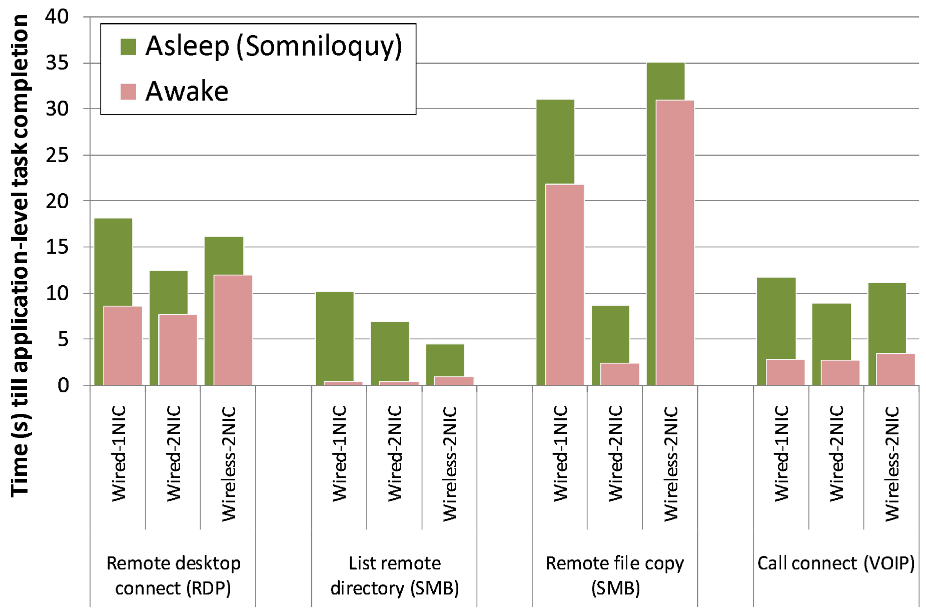


Figure 6.7: Application-layer latency for various Somniloquy prototypes.

Remote desktop access (RDP) — Here we used a stopwatch to measure the latency between initiating a remote desktop session to the host and the remote desktop being displayed. A stopwatch was used to ensure that true user-perceived latency was measured. The gumstix was configured to wakeup the main processor on detecting TCP traffic on port 3389 (the RDP port).

Remote directory listing (SMB) — A directory listing from the Somniloquy testbed was requested by the tester machine (via Windows file sharing, which is based on the SMB protocol). The time between the request being initiated and the listing being returned was measured using a simple script. The secondary processor was configured to initiate wake-up on detection of traffic on either of the TCP ports used by SMB,i.e. ports 137 and 445.

Remote file copy (SMB) — The SMB protocol was used again, but this time to transfer a 17 MB file from the Somniloquy testbed to the tester machine.

VoIP call (SIP) — A Voice-over-IP call was placed to a user who had been running a SIP client on the Somniloquy laptop before it had entered S3. On receipt of the incoming call the SIP server responded with a TCP connection to the testbed, causing the gumstix to trigger wakeup. A similar procedure was used in [2]. Once again, the latencies were measured using a stopwatch to measure true user-perceived delay.

As Figure 6.7 shows, Somniloquy adds between 4-10 s latency in all cases. As described earlier, part of this latency is attributed to resuming from S3, i.e. 4-5 s for the desktop and 2-3 s for the laptop, and is independent of Somniloquy. Further latency is due to the delay for TCP to retransmit the request, and for the host to respond to the request (which may take longer since it has just resumed). Note that the Wired-1NIC prototype shows higher latency than the Wired-2NIC prototype. This is purely an artifact of our prototype caused by the overhead of MAC bridging and largely the slower speed of the USBNet IP link between the gumstix and the host. The latter is particularly obvious in the file copy test, where the file copy time with the Wired-2NIC case is much faster than for Wired-1NIC (although the Wired-1NIC speed is still faster than Wireless-2NIC).

While Somniloquy does result in 4-10 s additional application-layer latency, these delays are acceptable for real usage (including VoIP [2]) in exchange for the substantial benefit of 20x-50x power savings. Additionally it is important to note that this additional latency is purely an initial startup latency that can easily be amortized over the length of the ensuing application session.

Stateful Applications

In this section we present evaluations for stateful applications that require stub support on the gumstix, primarily looking at the overhead in terms of memory consumption and processing capabilities that they impose on the gumstix. For all the measurements in this section, the total memory for the gumstix is 64 MB. We have implemented application stubs for three common applications — background downloads using the http protocol, P2P file sharing using BitTorrent, and maintaining presence on IM networks — as described in Section 6.2.

To study the overhead of IM clients, we run the corresponding application stub using up to three different IM protocols simultaneously — MSN Messenger, AOL Messenger and ICQ Chat. Table 6.4 shows the processor utilization and

Table 6.4: Processor and memory utilization for the IM stub for various configurations.

Accounts	Processor 95th percentile	Memory 95th percentile
None	0.0%	5.9 MB
MSN only	10.0%	6.5 MB
MSN+AOL	21.6%	6.7 MB
MSN+AOL+ICQ	26.0%	6.9 MB

memory footprint of the Wired-1NIC prototype when running these IM clients. Since the behavior of the IM stub is such that it maintains presence of the user on various networks and on receipt of an appropriate trigger (IM from someone) wakes up the host, the latency values are similar to those of the VoIP application as reported in Figure 6.7.

To evaluate the overhead of P2P file sharing using the BitTorrent stub on the gumstix, we initiated downloads using a torrent from a remote website[4] into the 2 GB SD card of the Wired-1NIC gumstix. We varied the memory cache available to the stub while conducting a single download, and then tested two simultaneous downloads. The results in Table 6.5 show that the memory footprint of the stub increases proportionally to the cache size as expected, while the processor utilization remains constant. When there are two simultaneous downloads, each instance of the stub uses memory proportional to its specified 4 MB cache.

Finally, to evaluate the web-download stub on the gumstix we initiate download of a large (300 MB) file from a local web server. We varied the throughput of the downloads and measured the processor utilization and the memory consumption of the gumstix, and experimented with two simultaneous downloads. As shown in Table 6.6, the processor utilization increases as the download rate increases although the memory footprint for each download remains constant.

The above results show that using application stubs, we can support fairly complex tasks and applications, including background web downloads and P2P file sharing using relatively modest resources on the gumstix. It is important to note that the power consumption of the gumstix did not exceed 2 W in all of these

[4]http://www.legaltorrents.com/

Table 6.5: Processor and memory utilization for the BitTorent stub for various configurations.

Configuration	Processor 95th percentile	Memory 95th percentile
Single download		
4MB cache	16.0%	6.5 MB
8MB cache	16.0%	10.6 MB
16MB cache	16.1%	18.9 MB
Two simultaneous downloads (4 MB cache)		
1st download	16%	6.5 MB
2nd download	24%	7.0 MB

Table 6.6: Processor and memory utilization for the web download stub for various configurations.

Configuration	Processor 95th percentile	Memory 95th percentile
Single download		
2Mbps	9.2%	1.8 MB
4Mbps	21%	1.8 MB
8Mbps	50%	1.8 MB
Two simultaneous downloads (4 Mbps each)		
1st download	31%	1.8 MB
2nd download	26.3%	1.8 MB

experiments.

6.3.4 Energy Savings using Somniloquy

In addition to evaluating the operating performance of our Somniloquy prototypes, it's also important to assess whether we satisfy the higher level goal of this work, namely reduce the energy consumption of PCs. In this section we present data which demonstrates the potential of Somniloquy to reduce both desktop and laptop energy usage in general terms. We also verify the energy saving model presented in Section 6.1.3, which allows the specific savings in a given application scenario to be calculated. Unless otherwise noted, we are using the Wired-1NIC version of our prototype for the desktop energy measurements and the Wireless-2NIC version for the laptop energy measurements.

Reducing Desktop Energy Consumption

Our testbed desktop PC consumes 102 W in normal operation and <5 W in S3 with Somniloquy. Somniloquy therefore saves around 97 W. On this basis, if Somniloquy were to be deployed in an environment where a PC is actively used for an average of 45 hours each week (i.e. 27% of the time), this would result in 620 kWh of savings per computer in a year. Assuming 0.61 kg CO_2/kWH[5] and US$ 0.09/kWH[6], this means an annual saving of 378 kg of CO_2 (to put it in perspective, the average US residents annual CO_2 emissions are 20 metric tonnes as compared to a worldwide average of 4 metric tonnes per person[7]) and US$ 56 per computer. We believe this is significantly higher than the bill of materials cost of the components required to implement a commoditized Somniloquy-enabled network card. In this case, deployments of Somniloquy-enabled devices would pay for themselves within a year.

Desktop Energy Savings for Real Workloads

We now estimate the energy savings enabled by Somniloquy under realistic workloads. We use an existing dataset [65] that provides the use patterns of twenty two distinct desktop PCs; each of which is classified as being either idle, active, sleep or turned off. We then compute the energy consumed by each of the PCs with and without Somniloquy using the formula of Section 6.1.3. For ease of exposition, we bin the data into three different categories: PCs that are idle for <25% of the time (7 machines), idle for 25%-75% of the time (6 machines) and finally those that are idle for >75% of the time (9 machines). The average energy savings for these twenty two PCs when using Somniloquy is 65%, as compared to normal operation without Somniloquy. The average energy savings for the PCs in the individual categories are 38%, 68% and 85% respectively. As expected, the most energy savings are for the PCs with larger idle times since they have more opportunity to use Somniloquy.

[5] http://www.eia.doe.gov/cneaf/electricity/page/co2_report/co2report.html
[6] http://www.eia.doe.gov/cneaf/electricity/epa/epa_sum.html
[7] http://www.sciencedaily.com/releases/2008/04/080428120658.htm

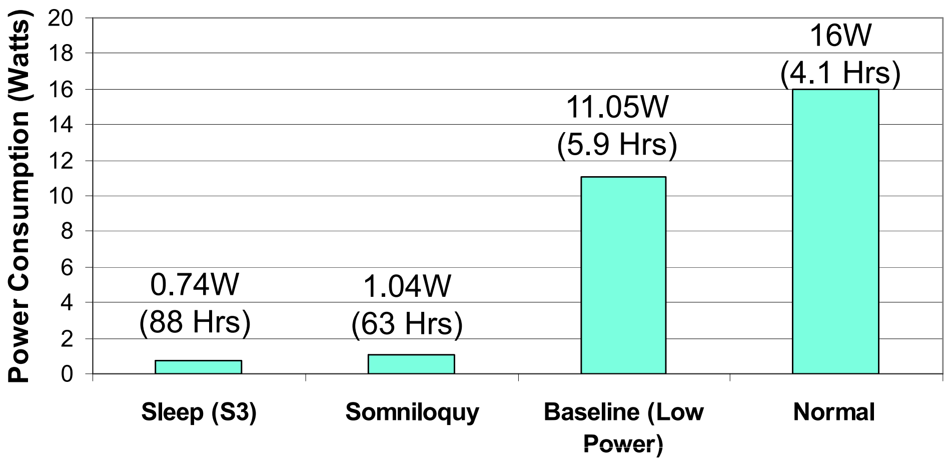


Figure 6.8: Power consumption and resulting estimated battery lifetime of a Lenovo X60 using Somniloquy.

Increasing Laptop Battery Lifetime

Figure 6.8 shows the average power consumption of the laptop testbed when operating normally (i.e. no power saving mechanisms), with standard power saving mechanisms in place (the baseline power), when Somniloquy (Wireless-2NIC) is operational, and in the standard S3 mode (without the gumstix attached). Somniloquy adds a relatively low overhead of 300 mW to S3 mode, resulting in a total power consumption which is close to just 1 W, as compared to the 11 W of the idle laptop. This means that when the laptop needs to be attached to the network and available for remote applications but is otherwise idle, it can be put into Somniloquy mode to enable an order of magnitude decrease in power consumption and a resulting increase in battery lifetime from 5.9 hours to 63 hours (using the standard 65 Watt-Hour battery).

Energy Savings for Specific Applications

The basic analysis of energy consumption and battery lifetime presented above is very generic; for a given usage scenario it should be possible to use the energy saving model presented in Section 6.1.3 to predict savings much more accurately. In order to validate this model we ran experiments downloading content

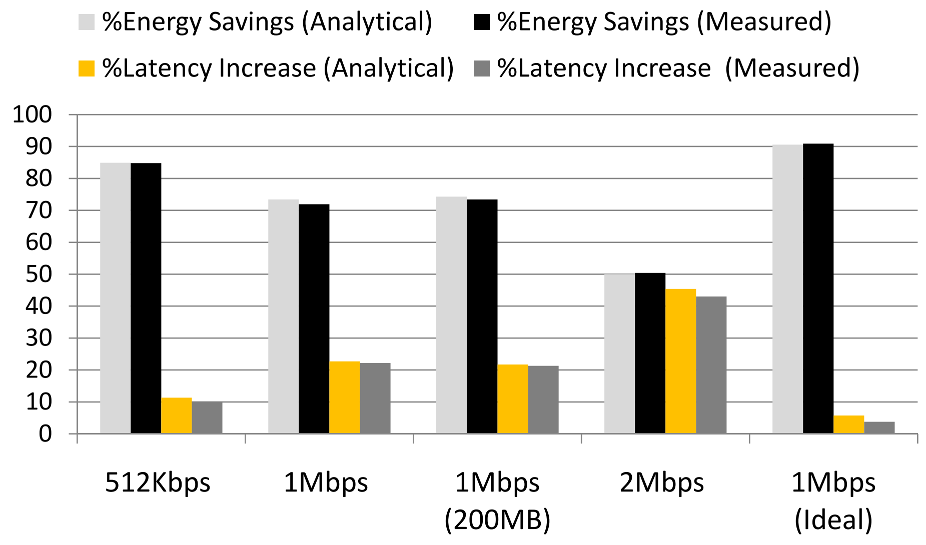


Figure 6.9: Validating analytical energy model with measured values for the download stub.

from a remote web server, and measured both energy consumption and latency so as to compare them with their corresponding analytical values. Note that we only measure the energy consumption for the duration of the application.

The web download stub was chosen since it was relatively easy to change the duty-cycle of the host, i.e. the duration for which the host can sleep (T_{sleep}) after which it needs to be woken up to transfer data from the gumstix (T_{awake}). As discussed in Section 6.1.3, T_{sleep} depends on the download bandwidth and the amount of flash storage on the gumstix, while T_{awake} depends on the amount of flash storage on the gumstix and the transfer rate between the gumstix and the host. We downloaded a 300 MB file at various link bandwidths ranging from 512 Kbps to 2 Mbps, and used two different flash storage sizes at the gumstix - 100 MB and 200 MB, effectively varying T_{sleep} from approximately 1600 seconds down to 400 seconds. We measured the power consumed during the download using the methodology described in the beginning of this section. In Figure 6.9, we present the measured energy savings and the corresponding predicted values using our model for four different data points. The flash storage available on the gumstix is set to 100 MB, unless stated otherwise. As we can see from the figure, the predicted energy savings and the increased latency closely match the measured values (within 1.5%). The values do not exactly match since the actual measured power values vary over time, and the time taken to suspend and resume also varies across runs. We used a fixed value for these in the formula.

Figure 6.9 also illustrates that increasing the bandwidth from 512 Kbps to 2 Mbps reduces the energy savings from 85% to 50%, and increases the latency from 11% to 43%, although a larger amount of flash storage improves the energy saving and latency. As explained earlier this is due to the limited transfer speed of the USBnet interface in our prototype (<5 Mbps), because of which the PC is awake for longer periods of time while transferring the data from the gumstix (T_{awake}= 181 seconds to transfer 100 MB of data). In Figure 6.9 we have also plotted an ideal case (1 Mbps-ideal) where the host can read the flash storage of the gumstix directly. For the ideal case the duration for which the host needs to stay awake to transfer data from the gumstix reduces considerably (T_{awake}= 23 seconds). This improves energy savings to 91% and limits the increase in latency when using Somniloquy to less than 5%.

6.4 Discussion

We now discuss some issues pertaining to the design of Somniloquy. In particular we consider security issues with the introduction of a separate secondary processor. Then we discuss the feasibility of another possible approach to implement Somniloquy using multi-core processors commonly available on modern PCs.

6.4.1 Handling Security Implications

A common requirement of corporate IT departments is that all PCs should be up to date with the latest OS and application patches. Somniloquy can ensure that this constraint is met even when PCs are asleep. This is achieved using a port-based trigger to wake up the host PC when the SMS (Systems Management Server) contacts the host PC to install updates.

Somniloquy ensures that the secondary processor is secure by patching its OS whenever security updates become available. Also, it prevents attackers from replacing the secondary processor by requiring that it be a physically part of the PC (as part of the network interface). In some cases however, the functionality that Somniloquy provides could be misused to conduct attacks that spuriously wake up the PC and waste energy. This kind of denial-of-service attack would be particularly effective for mobile devices where a drained battery might result. One way to address this issue is to disable wakeup filters, and instead exclusively use application stubs which ensure that only authenticated remote hosts are allowed to trigger wakeup.

Another concern is that application stubs, and hence the use of extra code, increases the PC's attack surface. To mitigate the impact of this vulnerability we use a few techniques. First, the secondary processor only listens on ports that have been opened by applications on the host PC and drops traffic on all other ports. Second, we require the PC and the secondary processor to be on the same administrative domain.

6.4.2 Alternative Design of Somniloquy

With the increasing prevalence of multi-core PCs, one idea to alleviate the need for the additional secondary processor introduced by Somniloquy would be to use one of the cores of the host CPU instead. Running just one core at the lowest possible clock frequency would minimize energy consumption and obviate the need for a separate low power processor in the NIC.

However, it turns out that such an approach is not useful without significant modification to today's PC architecture. Our measurements (see Section 6.3.1) show that the power consumption of a multi-core PC with only one core active (all other cores disabled), running at the lowest permissible clock speed is still approximately 50 times that of our low power secondary processor, even with all other peripherals in their lowest power modes – e.g. disk spun down. This is because of the lack of truly fine-grained power control of PC components such as the Northbridge and Southbridge chipsets, memory buses, parts of the storage hierarchy and various peripherals. Even if fine-grained control were available, the base power consumption of individual components (NIC, hard drive, etc) is significant (see Table 6.2). This has been confirmed by Mahesri et al. [60] in their power measurements of a laptop PC. One way to reduce this base power draw would be to have a separate and relatively simple core with a small amount of associated memory running from a separate power domain so that it can function without powering on other components. Such an architecture is very similar to Somniloquy, and most of our design principles can easily be adopted.

6.5 Related Work – Processor Collaboration

There have been several proposals to reduce the energy consumption of desktop PCs and laptops. Prior work can largely be grouped in three categories: reducing the active power consumption of devices (when awake) [4, 7, 26, 27, 52, 55], reducing the power consumption of the network infrastructure (e.g. routers and switches) [32, 33, 66], and opportunistically putting devices to sleep. Somniloquy falls in the third category. Since a machine in sleep state consumes significantly less power than in lowest power active state [32, 85] (verified by us in Section 6.3), significant energy savings are possible by putting the machine to sleep whenever possible.

For opportunistic-sleep systems, the biggest challenge is to ensure connectivity when the host is asleep. Prior techniques to solve this problem either use advanced functionality in the NIC [56] or use extra network interfaces [81, 85]. We now compare and contrast Somniloquy to both these classes of work.

Among schemes that do not use an extra network interface, the most well-known are Wake-on-LAN (WoL) [56] and its wireless equivalent, Wake-on-WLAN (WoWLAN). In both these schemes, the NIC parses incoming packets when the host is asleep. It wakes up the host PC whenever an incoming "magic" packet is received. According to the specification [56], the magic packet payload must include 6 characters of a wakeup pattern that is set by the host PC, followed by 8 copies of the NIC's MAC address. In WoWLAN, the only difference is that this packet is sent over the Wireless LAN. Although most modern NICs implement WoL functionality, few deployed systems actually use this functionality[6], due to four main reasons. First, the remote host must know that the PC is asleep and that it must wake it up before pursuing application functionality. Second, the remote host must have a way of sending a packet to the sleeping PC through any firewalls/NAT boxes, which typically do not allow incoming connections without special configuration. Third, the remote host must know the MAC address of the sleeping PC. Fourth, WoWLAN does not work when laptops change their subnet because of mobility. In contrast, Somniloquy does not require the extra configuration of firewalls/NAT boxes, and is transparent to remote application servers. It can handle mobility across subnets since the secondary processor can re-associate with services such as Dynamic DNS (to redirect a permanent host name to the PC's new IP address), and re-log-in to servers such as IM servers. In addition to these differences, Somniloquy also allows applications to be offloaded to the low power processor. There is no such concept in WoL, which instead wakes up the host when any pattern is matched.

Intel recently announced its Remote-Wake' [41] chipset technology (RWT) that claims to extend WoL on new motherboards by allowing VoIP calls to wake up a system, although its general applicability to other applications is not known. The details of this technology are not published. In contrast, Somniloquy goes beyond just WoL or RWT. It allows low power operation for various applications other than VoIP. Furthermore, Somniloquy does not require modifications to application end points or servers. RWT requires applications to first contact a server, which then sends a special packet to the PC to signal a wake up.

Another approach is to use additional "low-power" network interfaces to maintain connectivity to a device that is asleep. This approach has been proposed for use with mobile devices[81] or for laptop PCs [85]. Wake-on-Wireless [81] wakes up the mobile device on receiving a special packet on the low power network interface. Turducken [85] uses several tiers of network interfaces and processors with different power characteristics, and wakes up the upper tier when the lower tier cannot handle a task. In contrast to these schemes, Somniloquy requires only a single network interface, and presents the paradigm of a single PC to users rather than a multi-tiered system, preserving the current user experience and therefore requiring less training to use. Somniloquy also gives the impression to remote application servers that a device remains awake all the time even though it is actually asleep, since the same MAC and IP addresses are used. This level of transparency is not provided either by Wake-on-Wireless or Turducken. Futhermore, Turducken is designed around applications with a "polling" model, for example periodically checking for email updates. Applications that are inherently asynchronous, such as remote access (RDP, SSH), incoming VoIP calls and push email, cannot be handled by a periodic wake-up and polling scheme such as Turducken. In contrast, Somniloquy can handle both asynchronous and polling based applications using stubs. Finally, we have gone into more detail than previous work on ways of supporting applications that require interactions among the secondary and the host processor to perform offload – such as IM, BitTorrent and web downloads.

To reduce the power consumed by desktop PCs, some early proposals have suggested the use of proxies on the subnet that function on behalf of the desktop PC when it is asleep [6, 18, 32]. The proxy monitors incoming packets for the PC, and wakes it up using WoL when the PC needs to handle the packet. We are not aware of any published prototype implementations of such systems. Recently, Sabhanatarajan et. al. [77] propose a smart NIC that can act as proxy for a host to save power. However, the authors focus primarily on the design of a high speed packet classifier for such an interface. In comparison, Somniloquy has much wider applicability than the above schemes. It can be used in homes and small offices where it might be infeasible to deploy a dedicated server to handle processing for another PC.

A contemporaneous effort to Somniloquy is the idea of a Network Connection Proxy (NCP) [44, 65], which is a network entity that maintains the presence of a sleeping PC. In [44], the authors define the requirements of an NCP and propose modifications to the socket layer (similar to Split TCP) for keeping TCP connections alive through a PC's sleep transitions. In [65], the authors extend these APIs to support other protocols as well. Somniloquy is similar in spirit to NCP, and

NCP's socket APIs can reduce Somniloquy's overhead when waking up from sleep (Section 6.1.1). Furthermore, to the best of our knowledge, Somniloquy is the first published prototype of any proxying system.

We note that the concept of adding more processing to the network interface has also been explored earlier. Existing products offload processing to the NIC to improve performance (TCP offload [63]) and remote manageability (Intel AMT [40]). Somniloquy uses a similar offloading paradigm, but to conserve energy instead of improving performance or manageability.

6.6 Summary

In this chapter, we presented a new architecture, called Somniloquy, that utilizes processor collaboration to reduce energy consumption in commodity PCs. Somniloquy proposes augmenting the network interfaces of PCs with a secondary processor, to allow them to be duty-cycled opportunistically, without sacrificing functionality. The secondary processor on the augmented NIC acts collaboratively with the host PC, and masquerades on the behalf of the host when it is in a low power sleep state. This collaborative architecture enables several new energy saving opportunities. First, PCs can be put to sleep while maintaining network reachability, without special network infrastructure as needed by previous solutions (e.g. WoL). Second, some applications can be executed on the lower power secondary processor while the host is in a sleep mode thereby requiring much less energy. We have shown the feasibility for three such applications: BitTorrent, instant messaging, and web downloads.

Somniloquy achieves these energy savings without requiring any modifications to network, to remote application servers, or to the user experience of the PC. Furthermore, Somniloquy can be incrementally deployed on legacy network interfaces, and does not rely on any changes to the operating systems of PCs to implement this functionality.

Our current prototype implementation, based on a USB peripheral, includes support for waking up the PC on network events such as incoming file copy requests, VoIP calls, instant messages and remote desktop connections, and we have also demonstrated that file sharing/content distribution systems (e.g. BitTorrent, web downloads) can run in the augmented network interface, allowing for file downloads to progress without the PC being awake. Our tests show power savings of 24x are possible for desktop PCs left on when idle, or 11x for laptops. For PCs that are left idle most of the time, this translates to energy savings of 60% to 80%.

Chapter 6, in part, is a reprint of the material as it appears in Proceedings of USENIX Symposium on Networked System Design and Implementation (NSDI) 2009. Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Paramvir Bahl and Rajesh Gupta. The dissertation author is the primary investigator and author of this paper.

Chapter 7

Conclusions

Reducing energy consumption is an active area of research within the context of mobile devices, aiming to not only improve battery lifetime but also fulfill the promise of "all-day computing". Of late, attention has shifted to improving energy efficiency of desktops, laptops and servers in data centers that burn power continuously. This change in focus has been brought about by the growing awareness of the impact of global warming, and the increasing contribution of computing equipment power to overall energy demand.

Recognizing this trend, computer and device manufacturers are constantly innovating by introducing new processes and technologies that further enable power efficient operational states. This dissertation is based on a key observation that despite these advances in building low power components, duty-cycling or powering off components that are not in use remains by far the most effective way to save energy. Therefore, the thesis put forward in this dissertation is that computing platforms can in fact be duty-cycled aggressively by using "collaboration" amongst heterogeneous, but functionally similar subsystems. To validate this thesis, we have architected and implemented several systems based on this approach of collaboration within two specific contexts. First, for mobile devices we have identified radios as dominant power consumers and implemented multiple forms of radio collaboration to improve energy efficiency; on average our techniques enable a 2x - 3x improvement in battery lifetime, and in some cases 8x. Second, for desktops and laptops we show that entire platforms can be duty cycled by using processor collaboration, resulting in energy savings ranging from 60% to 80%.

In the rest of this chapter, we briefly review the specific contributions of this dissertation with a note about future research directions.

7.1 Contributions

This dissertation makes three contributions. First, we devise the notion of collaboration in building platforms with capabilities for duty-cycling components. Second, we concretize and demonstrate collaborative duty-cycling on various prototypes that target both mobile as well as desktop systems; addressing both computing and communications subsystems. Finally, we provide evaluations and detailed energy measurement of the various collaborative architectures presented in this dissertation.

7.1.1 Improving Duty-Cycling using Collaboration

There are several key insights that this dissertation provides that are fundamental to building aggressively duty-cycled systems. First, heterogeneity is essential for achieving energy efficient operation using collaboration. In some platforms, this system heterogeneity already exists and is part of the system architecture, arising from functional needs. As an example, consider the multiple heterogeneous radio interfaces available on existing mobile devices, as used by the various collaborative radio architectures presented in Chapters 3, 4 and 5. Each of these radios is designed for a specific purpose and use case. Differences in choice of radios, therefore, reflect differing needs as well as the regimes (bandwidth, standby power used, latency) in which they are most efficient. The presence of such heterogeneous but abstractly similar (in terms of functionality) components makes it easier to devise strategies that enable collaboration across these alternatives for improved duty-cycling.

In other cases, heterogeneous components must be introduced explicitly to achieve energy efficiency. This is exemplified by our collaborative processor architecture as described in Chapter 6, where we explicitly introduced a heterogeneous secondary processor. There is little tradition of including widely different processors on the same platform, because of intellectual property reasons and also due to the differences in the software stack and memory system architectures on different processors. Based on our results, however, we believe that hybrid heterogeneous architectures will be important in future energy efficient computers.

Second, the heterogeneous subsystems must be designed so as to be functionally similar but have very dissimilar energy/power and performance characteristics in order to allow collaboration to be useful from a duty-cycling standpoint. The main purpose of heterogeneity, from an energy efficiency point of view, is to provide very different operating points on the quality/performance versus cost (power/energy) curve. The larger this difference, the more effective the energy states that can be built. Third, when building duty-cycled platforms it is essential that user experience remains largely unaffected, and that duty-cycling does not cause any usability issues.

7.1.2 Deployable Prototypes

All of the collaborative systems described in this dissertation have been implemented on actual platforms. In our experience taking initial architectures and designs to final prototypes brought forward various systems and scalability issues that would otherwise not be apparent in simulations on analytical models. For example, although our initial Cell2Notify (Chapter 3) prototype was based on a laptop as a proof of concept, we recently implemented it on Windows Mobile (WM6) based smartphones. Our experience with Windows Mobile provided us with useful insights allowing us to reduce the latencies observed by Cell2Notify, such as optimizing the user interface delays and reducing the Wi-Fi association times.

Similarly, we have built an immediately deployable Somniloquy prototype (Chapter 6) which interfaces to a PC over the USB interface, and as a result can be used with any recent desktop or laptop PC. Furthermore, basing our prototype on an existing gumstix platform allows us to quickly build inexpensive prototypes of Somniloquy. We are currently in the process of distributing Somniloquy wired prototypes widely to users of desktop PCs in our department. In addition, we have installed energy measurement meters, allowing us to compare energy usage with and without Somniloquy enabled on these PCs.

7.1.3 Evaluation Results, Platforms and Power Measurements

We have evaluated radio collaboration in various settings, taking into account differences in radio ranges, varying application requirements, mobility, and changes in wireless channel conditions. Furthermore we have evaluated various possible degrees of radio collaboration, ranging from wakeup techniques, to a hierarchy of collaborating radios. Our evaluations show that using the techniques proposed in this dissertation battery lifetime of mobile platforms can be significantly increased, in some cases even up to eight times, without adversely affecting latency or causing loss in functionality.

Similarly, we have evaluated our collaborative processor architecture for both desktop and laptop PCs. We have presented energy saving results for different applications, ranging from applications that require PCs to be woken up, to applications that can be supported even while the PC remains asleep. We have evaluated the overhead in terms of added latency imposed by our prototype. Our evaluation shows that the energy consumption of both desktop and laptop PCs can be reduced significantly using processor collaboration, without requiring any changes to user behavior or any loss of functionality on the PCs. For example, a PC using processor collaboration consumes 11x to 24x less power than a PC in a normal idle state. For commonly occurring scenarios this translates to energy savings of 60% to 80%.

The detailed power measurements and component breakdown for mobile platforms, smartphones, laptops and desktops provides useful data for research to others working in this area. Furthermore, the Stargate2 [43] platform together with the novel in-situ power measurement functionality that we have built provides a useful platform to the community for further research in energy management of mobile devices.

7.2 Future Work

While this work provides a useful starting point to think about how energy efficient computing architectures and communication infrastructures can be built, there are many more potential energy saving opportunities. These opportunities exist both at a micro-level, e.g. dynamic power management for individual components such as multi-core processors, and at a macro-level such as energy management for entire enterprises.

7.3 Deploying Somniloquy in Enterprises

For individual users with desktops and laptops, Somniloquy (Chapter 6) is immediately useful since it enables significant cost savings. However, for large enterprises the need to add a separate piece of hardware to every desktop can become challenging due to the added administration and hardware cost. Thus, in our current research we are exploring ways to achieve Somniloquy-like functionality without needing to add hardware to each desktop machine by using a dedicated Sleep-Server machine, or even a network router, that in effect serves as a collection of many individual Somniloquy hardware pieces. We are currently working on a full scale implementation and deployment of Sleep-Servers within our CSE department that will allow 750+ desktop PCs and hundreds of servers to save energy by transitioning to low power modes (Standby) when idle, while remaining responsive to network traffic, leading to an estimated US$ 50,000 annual savings in direct energy costs.

7.3.1 Extending Processor Collaboration

At the fundamental level, Somniloquy demonstrates that many forms of computers can benefit significantly by embedded co-processing functions. While we show improvements in energy efficiency, the architectural design principles of combining low power processing and communication with high power and performance components can also be applied to build more available, reliable, secure or higher performance systems. In particular, the advent of multiple cores on a single processor presents a great opportunity for energy savings. Since individual cores may be heterogeneous, the operating system and applications can be modified such that they can dynamically scale down their memory footprint and run on lower power, albeit lower performance, cores to save energy and scale back up as and when needed. Our preliminary work has shown that such systems can indeed be built; for example, by employing techniques from programming languages and computer architecture such as program slicing and dynamic binary instrumentation it is feasible to synthesize different versions of applications – a fully functional variant and a reduced functionality application stub – even automatically.

7.3.2 Detailed Energy Accounting by In-situ Measurements

Most of the prior work in power and energy management, including that presented in this dissertation, assumes that the power consumption of devices is fairly predictable, and depends on the mode of operation. For example, for a Wi-Fi radio this translates to knowing whether the radio is in Active Mode or Power Save Mode and what the power consumption in these individual states is. As a result, most techniques for reducing the energy consumption of computing platforms are based on using power consumption measurements of the various states of devices in a static testbed setting.

Although this static power measurement and characterization works well for particular instances of hardware, it does not work well given different hardware types, nor when the power consumption of devices changes dynamically depending on workload or external effects. Take for example the power consumption of a radio, which may depend on the distance from the Access Point, or the amount of contention in the network. There are also significant differences in power consumption between hardware from different manufacturers, for example Wi-Fi interfaces can have widely varying power characteristics (Chapter 2). Additionally, in certain cases energy efficient decisions may only be possible at run time based on actual energy measurements.

The next significant advances in reducing system energy consumption will only be possible by having detailed **in-situ** energy measurement and management capabilities on platforms themselves. The currently available interfaces for quantifying energy use, such as detecting the residual capacity of batteries in laptops, are very coarse-grained. Ideally, we need more instrumentation on the platforms, allowing detailed energy breakdown and accounting on a per process basis, per application basis, and even based on individual hardware subsystems. In collaboration with Intel, we have built such an energy measurement capability into the Stargate2 [43] research platform; allowing both detailed in-situ energy measurements and fine grained control to power down individual subsystems. We used this novel capability in our collaborative radio architecture, SwitchR (Chapter 5), to characterize the energy consumption of all the devices simultaneously. Recently, similar designs have been proposed to instrument sensor platforms [89, 29] with similar energy measurement capabilities.

This energy measurement and management capability in platforms potentially opens up several interesting research opportunities. For example, this capability can be used to build a generic radio collaborative architecture, where the actual energy efficiency of every radio can be measured directly by the underlying hardware and be used to guide radio selection policies. In another example, using this capability the operating systems of platforms can determine the highest power consumers and use that information to make better power management decisions. Often device drivers do not switch power states as expected, and these in-situ measurements can serve to detect and alleviate these conditions. Applications can also benefit from these in-situ energy measurements by dynamically adapting their behavior, similar to that proposed in the Odyssey framework [27]. Finally, power management specifications such as ACPI [1] can be extended to include standard interfaces to report detailed power measurement from devices.

Bibliography

[1] ACPI. Advanced Configuration and Power Interface Specification, Revision 3.0b. `http://www.acpi.info`.

[2] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 179–191, New York, NY, USA, 2007. ACM.

[3] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*. USENIX Association Berkeley, CA, USA, 2009.

[4] Y. Agarwal, T. Pering, R. Want, and R. Gupta. "SwitchR: Reducing System Power Consumption in a Multi-Clients, Multi-Radio Environment". In *Proceedings of IEEE International Symposium on Wearable Computing (ISWC)*, 2008.

[5] Y. Agarwal, C. Schurgers, and R. Gupta. Dynamic Power Management Using On Demand Paging for Networked Embedded Systems. In *ASP-DAC '05: Proceedings of the 2005 Conference on Asia South Pacific Design Automation*, pages 755–759, New York, NY, USA, 2005. ACM Press.

[6] M. Allman, K. Christensen, B. Nordman, and V. Paxon. Enabling an Energy-Efficient Future Internet Through Selectively Connected End Systems. In *6th ACM Workshop on Hot Topics in Networks (HotNets)*. ACM, November 2007.

[7] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning Wireless Network Power Management. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189, New York, NY, USA, 2003. ACM Press.

[8] Asterisk. The Open Source PBX. `http://www.asterisk.org/`.

[9] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of Design Techniques for System-level Dynamic Power Management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, 2000.

[10] L. Benini and G. D. Micheli. *Dynamic Power Management: design techniques and CAD tools*. Kluwer Academic Publishers, 1998.

[11] D. Bertozzi, A. Raghunathan, L. Benini, and S. Ravi. Transport protocol optimization for energy efficient wireless embedded systems. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10706, Washington, DC, USA, 2003. IEEE Computer Society.

[12] N. Borisov, D. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo. A generic application-level protocol analyzer and its language. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS)*, 2007.

[13] Carla F. Chiasserini and Ramesh R. Rao. Combining Paging with Dynamic Power Management. In *INFOCOM*, pages 996–1004, 2001.

[14] C. Carter, R. Kravets, and J. Tourrilhes. Contact Networking: A Localized Mobility System. In *MobiSys '03: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, 2003.

[15] R. Chandra, V. Padmanabhan, and M. Zhang. WiFiProfiler: Cooperative Fault Diagnosis in WLANs. In *MobiSys*, 2006.

[16] L.-J. Chen, T. Sun, Y. Guang, and M. Gerla. USHA: A Simple and Practical Seamless Vertical Handoff Solution. In *IEEE Consumer Communications and Networking Conference (CCNC'06)*, 2006.

[17] G. Chinn, S. Desai, E. DiStefano, K. Ravichrndran, and S. Thakkar. Mobile PC Platforms Enabled with Intel® Centrino. *Intel Technology Journal*, 2003.

[18] K. Christensen, C. Gunaratne, and B. Nordman. The Next Frontier for Communication Networks: Power Management. *Computer Communications*, 27(18):1758–1770, 2004.

[19] Chunlong Guo and Lizhi C. Zhong and Jan M. Rabaey. Low-power Distributed MAC for Ad-Hoc Sensor Radio Networks. In *Globecom*, pages 2944–2948, 2001.

[20] COUNTERPATH. X-Lite 3.0 telephony client. http://www.xten.com/.

[21] Crossbow and Intel. Stargate Reseach Platform. `http://platformx.sourceforge.net`.

[22] W. Cui, J. Kannan, and H. J. Wang. Discoverer : Automatic Protocol Reverse Engineering from Network Traces. In *Proceedings of the USENIX Security Symposium*, 2007.

[23] G. Dhiman and T. S. Rosing. Dynamic Power Management Using Machine Learning. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 747–754. ACM New York, NY, USA, 2006.

[24] G. Dhiman and T. S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of the 2007 international symposium on Low power electronics and design*, pages 207–212. ACM New York, NY, USA, 2007.

[25] F. Douglis, K. P., and B. Bershad. Adaptive Disk Spin-down Policies for Mobile Computers. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, pages 121–137, 1995.

[26] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic Performance Setting for Dynamic Voltage Scaling. In *MobiCom '01: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 260–271, 2001.

[27] J. Flinn and M. Satyanarayanan. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, 2004.

[28] FMA. FloAt's Mobile Agent Online. `http://fma.sourceforge.net/`.

[29] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking Energy in Networked Embedded Systems. In *Proceedings of 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, 2008.

[30] B. S. I. Group. "Bluetooth Core", Specification of the Bluetooth System. v1.1, 2001.

[31] GSM World. TW 09 Battery Life Measurement Technique. `http://www.gsmworld.com/documentgs/index.shtml`.

[32] C. Gunaratne, K. Christensen, and B. Nordman. Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed. *Int. J. Netw. Manag.*, 15(5):297–310, 2005.

[33] M. Gupta and S. Singh. Greening of the Internet. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 19–26, New York, NY, USA, 2003. ACM.

[34] N. Haller. The S/KEY One-Time Password System. RFC 1760, February 1995.

[35] H.-Y. Hsieh and R. Sivakumar. A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts. *Wireless Networks*, 11(1):99–114, 2005.

[36] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *Proceedings of the USENIX Annual Technical Conference 2003*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[37] IEEE 802.11b/D3.0 Wireless LAN Medium Access Control(MAC) and Physical(PHY) Layer Specification. High Speed Physical Layer extension in the 2.4Ghz band, 1999.

[38] IEEE 802.1x-2001. IEEE Standards for Local and Metropolitan Area Networks, 1999.

[39] IETF. Mobile-IP. `http://www.ietf.org/ids.by.wg/mobileip.html`.

[40] Intel. Intel Active Management Technology (AMT). `http://www.intel.com/technology/platform-technology/intel-amt/`.

[41] Intel. Intel Remote Wake Technology. `http://www.intel.com/support/chipsets/rwt/`.

[42] Intel and Microsoft. Advanced Power Mangement BIOS Interface Specification, 1996.

[43] Intel-Research. Stargate2 and iMote2 Reseach Platforms. `http://embedded.seattle.intel-research.net/wiki/`.

[44] M. Jimeno, K. Christensen, and B. Nordman. A Network Connection Proxy to Enable Hosts to Sleep and Save Energy. In *IEEE International Performance Computing and Communications Conference*, 2008.

[45] D. B. Johnson and D. A. Maltz. Protocols for Adaptive Wireless and Mobile Networking. *IEEE Personal Communications*, 3, 1996.

[46] Junction Networks. SIP, IAX, IAX2 and Asterisk VoIP Service for Business. `http://www.junctionetworks.com/`.

[47] K. Kawamoto, J. G. Koomey, B. Nordman, R. E. Brown, M. A. Piette, M. Ting, and A. K. Meier. Electricity Used by Office Equipment and Network Equipment in the US. *Energy*, 27(3):255–269, 2002.

[48] Kineto Wireless. How Mobile and Wi-Fi Converge. `http://www.kinetowireless.com/`.

[49] A. Klaiber. The Technology behind Crusoe Processors. Technical Teport, Transmeta Corporation, 2000.

[50] J. Koomey. Estimating Total Power Consumption by Servers in the US and the World. *Final Report. LBNL. February 2007*, 2007.

[51] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 119–130, New York, NY, USA, 2002. ACM Press.

[52] R. Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. *Wireless Networks*, 6(4):263–277, 2000.

[53] L. Lamport. Password Authentication with Insecure Communication. *Communications ACM*, November 1981.

[54] K. Li, R. Kumf, P. Horton, and T. Anderson. A Quantitative analysis of disk drive power management in portable computers. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 22–22, 1994.

[55] X. Li, R. Gupta, S. V. Adve, and Y. Zhou. Cross-Component Energy Management: Joint Adaptation of Processor and Memory. *ACM Trans. Archit. Code Optim.*, 4(3):14, 2007.

[56] P. Lieberman. Wake-on-LAN technology. `http://www.liebsoft.com/index.cfm/whitepapers/Wake_On_LAN`.

[57] J. Lorch. A complete picture of the energy consumption of a portable computer, 1995.

[58] Y. Lu, T. Šimunić, and G. De Micheli. Software controlled power management. In *Proceedings of the seventh international workshop on Hardware/software codesign*, pages 157–161. ACM New York, NY, USA, 1999.

[59] Y.-H. Lu and G. de Micheli. Adaptive hard disk power management on personal computers. In *GLS '99: Proceedings of the Ninth Great Lakes Symposium on VLSI*, page 50, Washington, DC, USA, 1999. IEEE Computer Society.

[60] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop. *Lecture notes in computer science*, 3471:165, 2005.

[61] B. Marsh, F. Douglis, and P. Krishnan. Flash Memory File Caching for Mobile Computers. In *System Sciences, 1994. Vol. I: Architecture, Proceedings of the Twenty-Seventh Hawaii Internation Conference on*, volume 1, 1994.

[62] T. L. Martin. Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing. *PhD Thesis, Department of ECE, Carnegie Mellon University, Pittsburgh*, 1999.

[63] J. C. Mogul. TCP Offload Is a Dumb Idea Whose Time Has Come. In *HotOS*, pages 25–30, 2003.

[64] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the intel core duo processor. *Intel Technology Journal*, 10(2):109–122, 2006.

[65] S. Nedevschi, J. Chandrashekar, B. Nordman, S. Ratnasamy, and N. Taft. Skilled in the art of being idle: reducing energy waste in networked systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[66] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 323–336. USENIX Association Berkeley, CA, USA, 2008.

[67] B. Nordman. Networks, Energy, and Energy Efficiency. Cisco Green Research Symposium, March 2008. `http://efficientnetworks.lbl.gov/pubs/cisco-nordman-03-08.pdf`.

[68] P. Bahl and A. Adya and J. Padhye and A. Wolman. Reconsidering Wireless Systems with Multiple Radios. *ACM CCR*, Jul 2004.

[69] G. Parsons. Real-time Facsimile (T.38) - image/t38. RFC 3362, August 2002.

[70] 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*, pages 220–232, New York, NY, USA, 2006. ACM.

[71] T. Pering, T. Burd, and R. Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *Power Driven Microarchitecture Workshop, attached to ISCA98*, 1998.

[72] T. Pering, V. Raghunathan, and R. Want. Exploiting Radio Hierarchies for Power-Efficient Wireless Device Discovery and Connection Setup. In *Proceedings of the 18th International Conference on VLSI Design (VLSID'05)*, pages 774–779, Washington, DC, USA, 2005. IEEE Computer Society.

[73] W. Qadeer, T. S. Rosing, J. Ankorn, V. Krishnan, and G. D. Micheli. Heterogeneous wireless network management. In *Proceedings of the Workshop in Power Aware Computer Systems (PACS)*. Springer, 2003.

[74] R.Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, and T. Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[75] J. Roberson, C. Webber, M. McWhinney, R. Brown, M. Pinckard, and J. Busch. After-hours Power Status of Office Equipment and Energy use of Miscellaneous Plug-load Equipment. *Lawrence Berkeley National Laboratory, Berkeley, California. Report# LBNL-53729-Revised*, 2004.

[76] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[77] K. Sabhanatarajan, A. G.-R. M. Oden, M. Navada, and A. George. Smart-NICs: Power Proxying for Reduced Power Consumption in Network Edge Devices. In *ISVLSI '08*, 2008.

[78] A. Salkintzis and C. Chamzas. An Outband Paging Protocol for Energy-Efficient Mobile Communications. In *IEEE Transactions on Broadcasting*, pages 246–256, 2002.

[79] H. Schulzrinne and J. Rosenberg. A comparison of sip and h.323 for internet telephony, 1998.

[80] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Optimizing Sensor Networks in the Energy-Latency-Density Space. In *IEEE Transactions on Mobile Computing*, pages 70–80, 2002.

[81] E. Shih, P. Bahl, and M. J. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *MobiCom '02: Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pages 160–171, New York, NY, USA, 2002. ACM Press.

[82] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic Power Management for Portable Systems. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19, New York, NY, USA, 2000. ACM.

[83] T. Simunic, W. Qadeer, and G. D. Micheli. Managing heterogeneous wireless environments via Hotspot servers. In *Proceedings of the SPIE*, volume 5680, pages 143–154, 2005.

[84] Sipura. SPA-3000 Analog Telephony Adapter. `http://www.sipura.com/products/spa3000.htm`.

[85] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical Power Management for Mobile Devices. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, 2005.

[86] SPEC. Standard Performance Evaluation Corporation (SPEC). `http://www.spec.org/`.

[87] SpoofCard. Be Who You Want To Be. http://www.spoofcard.com/.

[88] Spooftel. The Worlds Leader In Caller ID Spoofing. `http://www.spooftel.com/`.

[89] T. Stathopoulos, D. McIntire, and W. Kaiser. The Energy Endoscope: Real-time Detailed Energy Accounting for Wireless Sensor Nodes. In *Proceedings of the IEEE/ACM Information Processing in Sensor Networks (IPSN '08)*, 2008.

[90] M. Stemm and R. H. Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. *IEICE Transactions on Communications*, E80-B(8):1125–31, 1997.

[91] M. Stemm and R. H. Katz. Vertical Handoffs in Wireless Overlay Networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

[92] Sukjae Cho. Power Management of iPAQ, USC ISI. `http://pads.east.isi.edu/presentations/misc/sjcho-pm-report.pdf`.

[93] T-Mobile. Stick Together with T-Mobile. `http://www.t-mobile.com/`.

[94] The New York Times. T-Mobile Tests Dual Wi-Fi and Cell Service, October 2006. `http://www.nytimes.com/`.

[95] Thomas J. Navin, Chief, Wireline Competition Bureau, FCC. H.R. 5126, the Truth in Caller ID Act of 2006, May 2006. `http://www.fcc.gov/ola/docs/navin051906.pdf`.

[96] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing Power in High-Performance Microprocessors. In *Proceedings of the 35th Annual Conference on Design Automation*, pages 732–737. ACM New York, NY, USA, 1998.

[97] VoIP-Info. Wiki. `http://www.voip-info.org/wiki/`.

[98] H. J. Wang, R. H. Katz, and J. Giese. Policy-enabled handoffs across heterogeneous wireless networks. In *Second IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99*, pages 51–60, 1999.

[99] R. Want, T. Pering, G. Danneel, M. Kumar, M. Sundar, and J. Light. The personal server: Changing the Way we Think about Ubiquitous Computing. *Lecture notes in computer science*, pages 194–209, 2002.

[100] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. *Kluwer International Series in Engineering and Computer Science*, pages 449–472, 1996.

[101] H. Woesner, J.-P. Ebert, M. Schlager, and A. Wolisz. Power saving mechanisms in emerging standards for wireless lans: The mac level perspecitve. *IEEE Personal Communications*, 5(3):40–48, June 1998.

[102] X. Zhao, C. Castelluccia, and M. Baker. Flexible Network Support for Mobile Hosts. *Mobile Networks and Applications*, 6(2):137–149, 2001.