
Dissertations, Theses, and Masters Projects

Theses, Dissertations, & Master Projects

2013

Network Traffic Aware Smartphone Energy Savings

andrew Joseph Pyles

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Pyles, andrew Joseph, "Network Traffic Aware Smartphone Energy Savings" (2013). *Dissertations, Theses, and Masters Projects*. Paper 1539623617.

<https://dx.doi.org/doi:10.21220/s2-wy0s-7w61>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

Network Traffic Aware Smartphone Energy Savings

Andrew Joseph Pyles

Williamsburg, VA

B.S. Computer Science, The Ohio State University, 2009
M.S. Computer Science, College of William and Mary, 2010

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
May 2013

COMPLIANCE PAGE

Research approved by

Protection of Human Subjects Committee


Protocol number(s): PHSC-2012-11-12-8276

Date(s) of approval: 2012-11-18

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy



Andrew Joseph Pyles

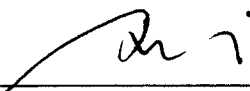
Approved by the Committee, May 2013



Committee Chair
Gang Zhou, Computer Science
The College of William and Mary



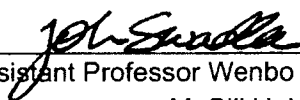
Professor Weizhen Mao, Computer Science
The College of William and Mary



Associate Professor Qun Li, Computer Science
The College of William and Mary



Associate Professor Haining Wang, Computer Science
The College of William and Mary



Assistant Professor Wenbo He, Computer Science
McGill University

ABSTRACT

In today's world of ubiquitous Smartphone use, extending the battery life has become an important issue. A significant contributor to battery drain is wireless networking. Common usage patterns expect Smartphones to maintain a constant Internet connection which exacerbates the problem.

Our research entitled *A Network Traffic Approach to Smartphone Energy Savings* focuses on extending Smartphone battery life by investigating how network traffic impacts power management of wireless devices. We explore 1) Real-time VoIP application energy savings by exploiting silence periods in conversation. WiFi is opportunistically placed into low power mode during Silence periods. 2.) The priority of Smartphone Application network traffic is used to modify WiFi radio power management using machine learning assisted prioritization. High priority network traffic is optimized for performance, consuming more energy while low priority network traffic is optimized for energy conservation. 3.) A hybrid multiple PHY, MAC layer approach to saving energy is also utilized. The Bluetooth assisted WiFi approach saves energy by combining high power, high throughput WiFi with low power, lower throughput Bluetooth. The switch between Bluetooth and WiFi is done opportunistically based upon the current data rate and health of the Bluetooth connection.

Our results show that application specific methods for wireless energy savings are very effective. We have demonstrated energy savings exceeding 50% in generic cases. With real-time VoIP applications we have shown upwards of 40% energy savings while maintaining good call quality. The hybrid multiple PHY approach saves more than 25% energy over existing solutions while attaining the capability of quickly adapting to changes in network traffic.

TABLE OF CONTENTS

| | |
|---|-----|
| Acknowledgments | iv |
| Dedication | v |
| List of Tables | vi |
| List of Figures | vii |
| 1 Introduction | 1 |
| 1.0.1 Contributions | 3 |
| 1.0.2 Dissertation Organization | 5 |
| 1.1 Related Work | 5 |
| 1.1.1 Real-time Smartphone Application Data | 5 |
| 1.1.2 Smartphone Application Data Priority | 7 |
| 1.1.3 Mixed Radio Data Driven Energy Savings | 8 |
| 2 SiFi: Silence prediction based WiFi energy adaptation | 10 |
| 2.1 SiFi-Background | 13 |
| 2.2 Silence Modeling & Prediction | 14 |
| 2.2.1 Lightweight Silence Detection | 14 |
| 2.2.2 ECDF Based Silence Prediction | 16 |
| 2.2.2.1 Determine the Runtime Training Length | 18 |
| 2.2.2.2 The Observed Silence Length α | 20 |

| | | |
|---------|--|----|
| 2.2.2.3 | The Confidence of Prediction | 21 |
| 2.3 | SiFi: Silence Prediction based WiFi Energy Adaptation | 22 |
| 2.4 | Android Phone Implementation | 25 |
| 2.4.1 | Application Level Modifications | 26 |
| 2.4.2 | System Level Modifications | 26 |
| 2.5 | Performance Evaluation | 28 |
| 2.5.1 | Evaluation Setup | 29 |
| 2.5.2 | Evaluation Method | 30 |
| 2.5.3 | SiFi Energy Savings | 32 |
| 2.5.4 | SiFi Application Fidelity | 33 |
| 2.5.5 | SiFi Robustness for Longer Calls | 35 |
| 2.6 | Conclusions | 36 |
| 3 | SAPSM: Smart Adaptive 802.11 PSM | 37 |
| 3.1 | Background and Motivation | 40 |
| 3.1.1 | Background | 40 |
| 3.1.2 | Sprint HTC Hero Adaptive PSM | 41 |
| 3.1.3 | Adaptive PSM Behavior of Different Smartphones (and other Handheld Devices) | 42 |
| 3.2 | SAPSM Design | 45 |
| 3.2.1 | Architecture | 47 |
| 3.2.2 | SAPSM Core | 48 |
| 3.2.3 | Application Priority Manager | 49 |
| 3.3 | Implementation | 51 |
| 3.4 | Evaluation | 52 |
| 3.4.1 | Evaluation Method | 53 |
| 3.4.2 | Low Priority Application Behavior | 54 |

| | | |
|---------|--|----|
| 3.4.2.1 | Traffic with no listening socket | 54 |
| 3.4.2.2 | Low Priority Energy Inversion | 55 |
| 3.4.3 | Energy Savings of Typical Applications | 55 |
| 3.4.4 | SAPSM High Priority Networking Performance | 59 |
| 3.5 | User Study | 60 |
| 3.6 | Conclusions & Future Work | 64 |
| 4 | Mixed Radio Data Driven Energy savings | 65 |
| 4.1 | Introduction | 65 |
| 4.2 | Background and Motivation | 68 |
| 4.2.1 | WiFi PSM | 69 |
| 4.2.2 | Bluetooth | 70 |
| 4.2.3 | Motivation | 71 |
| 4.3 | Bluesaver Design | 74 |
| 4.3.1 | Architecture | 75 |
| 4.4 | Implementation | 79 |
| 4.5 | Evaluation | 81 |
| 4.5.1 | Evaluation Method | 82 |
| 4.5.2 | Energy Comparison | 82 |
| 4.5.3 | Network Adaptation | 86 |
| 4.6 | Conclusions | 90 |
| 5 | Conclusions & Future Work | 91 |
| | Bibliography | 93 |
| | Vita | 98 |

ACKNOWLEDGMENTS

This writer wishes to express his appreciation to Professor Gang Zhou, under whose guidance this investigation was conducted, for his patience, guidance and criticism throughout the investigation. The author is also indebted to Professors Li, Mao, Wang, Liu and He for their careful reading and criticism of the manuscript.

This Ph.D. is dedicated to my wife Jill. Without her dedication and support, none of this would be possible.

LIST OF TABLES

| | |
|--|----|
| 2.1 Prediction with Different β | 21 |
| 2.2 Baseline Average Power | 28 |
| 2.3 Adaptive PSM Settings in Sprint HTC Hero | 29 |
| 2.4 SiFi Fidelity | 35 |
| 2.5 Multiple Longer Call Tests: VAD + SiFi | 35 |
| 3.1 Smart devices tested | 44 |
| 3.2 SAPSM high priority TCP Performance results with associated standard deviation. | 60 |
| 3.3 Majors. | 60 |

LIST OF FIGURES

| | |
|---|----|
| 2.1 VAD and RTP | 13 |
| 2.2 ECDF of Silence Length | 17 |
| 2.3 KL Divergence vs. Training Length | 19 |
| 2.4 Observed Silence Period Length α vs. Predicted Silence Period Length Δ | 20 |
| 2.5 SiFi Architecture | 23 |
| 2.6 Android WiFi Architecture | 27 |
| 2.7 Energy Measurement Setup | 29 |
| 2.8 Adaptive PSM State Transitions in Sprint HTC Hero | 30 |
| 2.9 VAD + Adaptive PSM vs. VAD + SiFi | 31 |
| 2.10 non-VAD + Adaptive PSM vs. non-VAD + SiFi | 31 |
| 3.1 Sprint HTC Hero Adaptive PSM implementation; initial state is PSM, timer is set to one second. | 42 |
| 3.2 Adaptive PSM implementation response to UDP network traffic. . . | 43 |
| 3.3 SAPSM Architecture; the packet flow is diverted through the SAPSM core kernel module. | 45 |
| 3.4 The pop-up window to assist user decision. | 50 |
| 3.5 Comparison of power consumed from ingress traffic with no listening socket. | 53 |
| 3.6 Energy sdf inversion: 10MB file downloaded w/varying data rates. . | 54 |

| | | |
|------|---|----|
| 3.7 | Application energy comparison. | 57 |
| 3.8 | RSS Reader with Adaptive PSM | 57 |
| 3.9 | RSS Reader w/ low priority SAPSM | 58 |
| 3.10 | Application classification result. | 63 |
| 4.1 | WiFi Energy Latency tradeoff. Measurements reflect recently published measurements on Smartphone WiFi PSM. | 70 |
| 4.2 | Youtube typical broadband connection | 73 |
| 4.3 | Skype audio typical broadband connection | 74 |
| 4.4 | Skype video typical broadband connection | 75 |
| 4.5 | Lab Setup. | 76 |
| 4.6 | Bluesaver Architecture. | 77 |
| 4.7 | Bluesaver Design. | 78 |
| 4.8 | Bluetooth vs Adaptive PSM power consumption. Adaptive PSM consumes between 20% more and 35% power than Bluetooth. . . | 83 |
| 4.9 | Bluesaver vs WiFi streaming video Energy comparison. Bluesaver saves up to 25% energy for bitrates ranging from 64 to 512 kbps. . | 85 |
| 4.10 | Bluesaver vs WiFi streaming video power comparison. Bluetooth consistently uses less power than WiFi. | 86 |
| 4.11 | Rate Adaptation: Bluesaver switches from Bluetooth to WiFi while sequentially downloading a 10MB file followed by a 100MB file without interrupting the download. | 87 |
| 4.12 | Room Layout showing lab environment for connection adaptation evaluation. | 88 |

4.13 Connection Adaptation: Bluesaver switches from Bluetooth to WiFi when Bluetooth connection issues are detected. A constant ping from the Server to the phone was done for the duration of the test. When the switch to WiFi occurs at around 30 seconds, the ping test is unaffected. 89

Network Traffic Aware Smartphone Energy Savings

Chapter 1

Introduction

Over the past five years, Smartphones have exploded in popularity. More and more functionality is added with each new release. New phones are coming out with new features such as faster processors and advanced networking capabilities. These features make the use of Smartphones for everyday tasks very compelling. People now have instant access to a wealth of information that was inconceivable several years ago. Access to this wealth of information, one of the most compelling aspects to Smartphones, does not come without cost however. One of the key challenges behind the design of Smartphone systems is that of limited battery capacity. While the traditional PC with a wall outlet has virtually unlimited power constraints, the Smartphone must be designed to be completely functional for a variety of uses, while at the same time minimizing the consumption of the battery. One of the largest power consumers on Smartphone, is networking traffic. For instance a recent Smartphone measured as part of this work [54], shows that the screen consumes approximately 50% less power than the WiFi networking adaptor actively transmitting data. It has also been shown that in [11] that 3G networking consumes even more power than WiFi. Identifying areas where network traffic can be made more efficient can make a big impact on the usefulness and lifespan of the device.

When Smartphones were first introduced, consumers were commonly given unlimited

data usage plans. In the past year, however, some providers have pulled away from this business model. For instance, At&T recently raised eyebrows when it put usage caps on all new plans. To justify the new policy, it was explained that data usage exploded 20K percent between 2007 and 2011. This enormous growth is also shown in Mobile App stores. As of this writing, both the Android Play store and the iPhone app stores have over 500K applications [6]. Clearly, lots of network activity is prevalent on these devices with no signs of slowing down.

One of the most common Network interfaces on Smartphones is the WiFi adaptor. While WiFi is more efficient than 3G [57], it also is not without its challenges. On the one hand, WiFi can enter into Power Save Mode (PSM) to be more energy efficient and adding significant delay to all network traffic. On the other hand, the WiFi interface can enter into Constantly Aware Mode (CAM) which has a significantly higher power cost while minimizing network delay. In order to switch between these modes, the WiFi driver switches between power saving modes completely oblivious to the actual type of traffic currently being transmitted through the radio.

Lots of related work [22] [59] [41] covered in section 1.1, focuses on research to make WiFi more efficient. However, in actuality, most vendors that we systematically measure as part of this work [53] use a fairly simple scheme of switching power savings modes based upon the combined data rate that travelling through the interface. Although this catch-all approach is effective, we demonstrate that there is lots of room for improvement.

Our research is entitled a Network Traffic Aware Approach to Smartphone Energy Savings. Our approach is to examine all types of network traffic and determine the most energy efficient method of transmitting packets through the phone. This research is broken up into three main categories: *Real-time delay sensitive network traffic*: We examine real-time network which is particularly challenging where any delay will affect the QoS of the application. For instance, this covers real-time video or audio calls.

By identifying delay tolerant periods we show that we can save significant energy over existing solutions as well as ensuring that QoS is not impacted. *Application aware prioritization of network traffic*: Since all network traffic can equally impact the Power consumption of the device. We explore the concept of Application prioritization. Important Application traffic can be identified over non-important traffic. Important traffic is allowed to modify the power saving mode of the WiFi driver, while unimportant traffic is not resulting in significant energy savings over existing solutions. *Hybrid multi-phy approach to energy savings*: Certain types of streaming traffic which have a low data rate is particularly inefficient with WiFi. The rate is high enough that the driver will enter into CAM, but still low enough where other radios can be more efficient. We identify this type of traffic and using a hybrid MAC based approach combining Bluetooth and WiFi, we show that significant energy savings can be saved without adding significant delay.

1.0.1 Contributions

The main contributions of this work are summarized as follows:

- *Real-time Smartphone Application Data*: Since one-third of a Smartphone's battery energy is consumed by its WiFi interface, it is critical to switch the WiFi radio from its active or Constantly Awake Mode (CAM), which draws high power (726mW with screen off), to its sleep or Power Save Mode (PSM), which consumes little power (36mW). Applications like VoIP do not perform well under PSM mode however, due to their real-time nature, so the energy footprint is quite high. The challenge is to save energy while not affecting performance. In this dissertation we present SiFi: Silence prediction based WiFi energy adaptation. SiFi examines audio streams from phone calls and tracks when silence periods start and stop. This data is stored in a prediction model. Using this historical data, we predict the length of future silence periods and place the WiFi radio to sleep during these periods. We

implement the design on an Android Smartphone and achieve 40% energy savings while maintaining high voice fidelity.

- *Smartphone Application Data Priority:* The ability for 802.11 power management on Smartphones to distinguish traffic priority is an important issue. Low priority background traffic unnecessarily causes the client to switch to the high power Constantly Awake Mode (CAM) wasting unnecessary energy upwards of 50% in some cases. The priority of network traffic flows is dependent upon application intent. Wide availability of application stores such as the Android Market with thousands of applications makes application prioritizing a challenge. In this dissertation we introduce SAPSM: Smart Adaptive Power Save Mode. SAPSM gathers network usage metrics per application. It unobtrusively requests per-application priority setting confirmation from users with the aid of a lightweight machine learning algorithm. Only high priority applications affect the client's behavior to switch to CAM, while low priority traffic is optimized for energy efficiency. Our implementation on an Android Smartphone improves energy savings by up to 56% under typical usage patterns.
- *Mixed Radio Data Driven Energy Savings* The limited battery life on Smartphones makes energy conservation an important issue. Wireless networking efficiency is particularly important since it consumes a large percentage of total energy. In recent studies, WiFi and 3G have been shown to consume more than double the amount of power than powering on the screen. WiFi power save mode saves energy by trading added latency for less power consumption. This latency is caused by packet buffering at the Access Point. Minimal latency but maximum power on the other hand, is consumed with WiFi Active mode. WiFi effectively has two extremes: low power consumption and high latency or low latency and high power consumption. While research has advanced in mitigating these extremes, certain types of network traffic such as constant bitrate streaming make the contrast

unavoidable. We introduce Bluesaver which provides low latency and low energy by maintaining a Bluetooth and WiFi connection simultaneously. Bluesaver is designed and implemented at the MAC layer and is able to opportunistically select the most efficient connection for packets while still assuring that latency is acceptable. We implement Bluesaver on an Android phone and Access Point and show that we can save more than 25% energy over existing solutions and attain the capability of quickly adapting to changes in network traffic.

1.0.2 Dissertation Organization

The rest of this Dissertation is organized as follows. In Section 1.1 we review related work that is closely related to our work. Section 2 our work in *Real-time Smartphone Application Data* is covered. Section 3 covers our work in *Smartphone Application Data Priority*, while in Section 4 we cover our work in *Mixed Radio Data energy savings*. Finally, we conclude in Section 5.

1.1 Related Work

1.1.1 Real-time Smartphone Application Data

A large amount of research attention has been recently paid to the problem of WiFi energy saving in mobile devices. Here we discuss the work most relevant to SiFi.

Exploiting Idle opportunities. Considerable work has been done finding idle opportunities within WLAN to exploit for power savings. In Bounded Slowdown [35], idle periods between TCP establishment are exploited to switch WiFi between CAM and PSM modes. Time periods between slow start and between Web transactions are used. Beacon intervals are dynamically adjusted to minimize PSM overhead.

Micro Power Management [39] exploits small time periods (μ seconds) between MAC frames to save energy. Micro power sleep periods are effective since the speed of WLAN's are typically much greater than the WLAN up-link speed. This approach is complimentary to SiFi. SiFi examines the payload of RTP packets and based on the prediction methods, puts the radio to sleep. Micro power management could be combined with SiFi to gain additional energy savings.

Another approach, Catnap [22] stores data into blocks that are combined at the AP, then the blocks can be transmitted efficiently to the STA allowing for additional power savings. This approach is not applicable to real-time applications such as VoIP due to the added delay incurred.

Self-Tuning [1] provides an API to application developers to identify network traffic that is considered background or foreground traffic. Using these hints, the kernel driver can schedule background traffic into PSM mode while foreground traffic can be scheduled at a higher priority. Self-Tuning is not a good fit for the real-time nature of VoIP for two reasons. First, in the case of non-VAD RTP traffic, the same traffic pattern exists for voice and silence RTP packets which is difficult for Self-Tuning to attach different traffic hints. Second, unlike SiFi that directly controls the WiFi driver, the Self-Tuning kernel module may introduce delay that impacts VoIP performance.

VoIP Specific Approaches w/o Idle Exploitation. There are several other VoIP specific approaches that do not exploit idle opportunities for saving WiFi energy. The GreenCall algorithm [47] uses a deadline approach that does not consider silence periods. By examining the inter-arrival times of the packets, they determine the play-out deadline. As long as the play-out deadline has not been reached, the WiFi radio is put into PSM mode. Our approach is complimentary to this approach with the following distinction. We place special emphasis on putting the radio to sleep particularly during silence periods. The play-out deadline for silence packets is not as important as for voice packets.

AP Modification. Some AP centric approaches are also developed for WiFi energy. Napman [59] focuses on modifying the AP scheduler so that PSM traffic and CAM traffic are treated fairly. Specific care is given to Adaptive PSM implementations on the latest smart phones such as the iPhone and Android based approaches. By adjusting the TCP window size, PSM-Throttling [64] increases the burstiness of multimedia streaming traffic causing energy saving to be realized.

IEEE 802.11e U-APSD. IEEE 802.11e also introduces U-APSD [32] to provide an extra layer of QoS while saving energy. When the AP receives a frame from the STA, the AP will send all buffered data to the client without requiring the PS-POLL mechanism. This works when upstream and downstream RTP streams are complimentary as in the non-VAD case. However, in cases such as VAD, the upstream and downstream RTP streams can be asymmetric and the down-link frames will not be triggered.

In [18] an exponential back-off scheme is employed during silence periods of VoIP calls to determine the maximum period to put the radio into sleep mode. For long silence periods, this scheme has the potential problem of over sleeping causing delay. By using training data, SiFI can more accurately determine the length of the silence period, to prevent extended over-sleeping.

1.1.2 Smartphone Application Data Priority

A number of prior solutions have been proposed to reduce energy consumption on mobile devices. We focus on these most closely related to SAPSM.

A number of solutions have been proposed for optimizing PSM behavior on 802.11 clients for greater efficiency. PSM-Throttling [64] uses traffic shaping on the client to add burstiness to TCP traffic which contributes to more efficient PSM behavior. Sleep durations are modified [35] in reaction to current traffic levels maintaining a bounded

RTT for more efficient adaptive PSM behavior. μ Power management [30] uses prediction to determine microsecond sleep periods during busy intervals to save energy. Down-clocking the radio during idle listening periods to reduce energy consumption is explored in [66]. Most closely related to SAPSM, STPM [1] adapts PSM behavior by combining the monitoring of current traffic and knowledge of application intent provided by an API made available to developers. SAPSM is complementary; application intent can be combined with application priority to achieve further energy savings.

Other solutions propose modifications to network infrastructure. In [22] the difference in bandwidth between Wireless and WAN connectivity is exploited. Napman [59] reduces contention by staggering beacon intervals per client and the use of a fair scheduling algorithm. In [41] network contention reduction is explored: beacon periods are staggered by eavesdropping on AP's in close proximity. SAPSM is complimentary in reducing energy with these solutions. Any improvements reducing network contention will result in additional energy savings.

A number of projects have considered idle periods to duty cycle the radio into a minimal power state to save additional energy. In [54] silence periods are exploited in VoIP calls to save additional energy. Coolspots [51] uses multiple radios to only enable the radio with highest energy cost when required.

1.1.3 Mixed Radio Data Driven Energy Savings

Although there is a significant body of research focused on mobile energy efficiency, we focus primarily on those closely related to Bluesaver.

Client modifications: Numerous client side research exists addressing the problem of making the client more energy efficient. Micro power management [38] predicts and utilizes microsecond sleep periods to save energy. While others [1] [53] focus on

application specific traffic shaping based upon priority or sensitivity to delay; traffic not sensitive to delay or given a low priority is forcibly sent through PSM. Others [54] examine application specific idle periods to save energy by switching to PSM during idle periods. PSM-throttling [64] uses traffic shaping to streamline TCP traffic for PSM efficiency. E-Milli [66] uses a downclocking scheme which reduces the voltage to the WiFi driver during idle listening times. Bluesaver is complementary to these approaches; client side enhancements can be extended with the use of multiple radios.

Infrastructure modifications: Others have modified the wireless infrastructure to be more efficient. Napman [59] uses a fair energy-aware scheduling algorithm to increase WiFi efficiency. Sleepwell [41] reduces client collisions with the use of multiple APs staggered at different time intervals while Catnap [22] exploits the difference between WLAN and WAN speeds. Others [8] explore the use of proxies to reduce client-side polling. All of these approaches save energy by streamlining the wireless traffic to and from the Client. These approaches are complimentary to Bluesaver; any enhancements in the infrastructure to make WiFi more efficient will also enhance Bluesaver.

Multiple PHY: Other research focuses on combining multiple radios for best efficiency. ZiFi [67] and Blue-Fi [2] use low power radios to detect the presence of WiFi access points. Coolspots [51] uses a combination of Bluetooth and WiFi to save energy. Only one radio is active at the same time causing serious delay and dropped connections when switching between radios. BlueStreaming [60] also uses WiFi and Bluetooth. Both radios are on at the same time with 2 separate IP addresses. Applications therefore have to be specifically designed to bind to one of the IP addresses. Bluesaver uses Both WiFi and Bluetooth. Since it is implemented at the MAC level, it is able to seamlessly switch between radios without impacting applications.

Chapter 2

SiFi: Silence prediction based WiFi energy adaptation

As demonstrated in [56], there is a tradeoff between the cellular data network and WiFi communication that both exist in currently widely used smart phones. Compared to the WiFi interface, the cellular data network incurs a lower penalty to stay connected, but much higher energy price per MB of data transfer. In addition, the cellular data network has higher latency and may incur additional airtime charges. Towards improving battery lifetime and enhancing user experience and productivity [57] [51] [12], many newly developed smart phone related applications choose to involve the WiFi radio communication [56] [43] [9] [40] [65].

Even though WiFi radio is energy efficient when communicating a large amount of data, it is expensive just to stay connected. For example, the Sprint HTC Hero [63] consumes 726mW (with screen off) in the Constantly Awake Mode (CAM). So, it is essential to put WiFi to the Power Save Mode (PSM) which consumes 20 fold less energy (36mW in Sprint HTC Hero). In the research community, a large number of efforts have been proposed to reduce energy consumption. For example, [35] [22] [1] propose to

save WiFi energy by exploiting periods of idleness, while [59] improves scheduling on the WiFi Access Point (AP). [32] shows a general purpose 802.11 protocol changes to save energy for general real-time applications, and [47] analyzes the VoIP network traffic playout deadline to determine when to put the radio to sleep. Also, the Adaptive PSM [59] commonly deployed saves energy by monitoring throughput through the WLAN. The radio stays in PSM by default, but switches to CAM when traffic is observed.

However, none of the aforementioned existing work examines the RTP payload to exploit silence period for WiFi energy savings during a VoIP call, even though it has been shown that up to 60% [23] of a typical human conversation is made up of silence. Therefore, we are motivated to address the research challenges towards exploiting silence period for energy saving during a smart phone VoIP call: (1)how to model and predict VoIP silence periods of a phone call at runtime? and (2)how to apply silence prediction to the existing WiFi infrastructure of an Android phone to save energy?

To address the first challenge, we first developed a light weight silence detection algorithm that is able to distinguish silence from voice in RTP packets during a phone call. Then, once an adequate set of silence periods are obtained during runtime, a running statistical model is built to characterize the silence periods distribution in an accurate and energy efficient way. Finally, when the statistical model is observed stable enough, runtime prediction of the future silence period length is conducted based on statistical analysis.

To address the second challenge, we propose SiFi, a silence prediction based WiFi energy adaptation framework for smart phones. SiFi is carefully designed so that it fits well with existing mobile phone constraints as discussed in [42], namely OS limitations, API and operational limitations, energy management limitations, etc. SiFi is able to incorporate the statistical modeling and prediction theory and realize it in the limited smart phone WiFi infrastructure and demonstrates more than 40% energy saving.

Our main contributions can be summarized as follows:

- With lightweight digital signal processing, and statistical modeling and prediction, we successfully exploit during runtime the silence periods of a VoIP call which forms a solid base for runtime WiFi energy saving. To ensure high accuracy and low cost for the runtime silence exploitation, different modeling and prediction techniques including empirical cumulative distribution function and time series analysis are compared, and important statistical issues like runtime training length and confidence of prediction are thoroughly explored.
- To apply the statistical silence exploitation technique we develop, we propose a silence prediction based WiFi energy adaptation framework, called SiFi, for smart phones energy saving. By making modifications to the low level system architecture as well as application components, SiFi is able to directly control the WiFi power save mode based on silence prediction, and fits well with the limited Android phone infrastructure.
- We deploy SiFi on a real system with Sprint HTC Hero that runs VoIP application and obtain more than 40% power savings during runtime. We achieve high call fidelity by sleeping during silence periods and being active during voice periods. Our real system evaluation also demonstrate SiFi's resilience to network congestion, and its robustness in different phone call scenarios like with different phone call lengths, different languages, different number of speakers, and different genders of speakers.

The rest of the work is organized as follows. We explain background knowledge followed by related work. Next, we present details of detecting, modeling and predicting VoIP silence periods of a phone call at runtime. Then, a detailed design of the SiFi framework is given followed by details of the SiFi implementation on an Android phone.

Real system performance evaluation with Sprint HTC Hero is illustrated next. Finally, we present the conclusions.

2.1 SiFi-Background

Session Initiation Protocol (SIP) is widely used in Internet Telephony. It is used to establish and tear down calls where Real Time Protocol (RTP) media is streamed. To illustrate how SIP works, imagine two phones A and B, where A would like to call B. Once a call has been established, RTP packets are sent bi-directionally until a call is terminated with a BYE message. Detailed description of SIP is outside the scope of this work, and interested readers are referred to [58]. When a call is initiated with an INVITE packet, phone A notifies phone B which UDP port it is listening for RTP packets. Phone B responds soon after with a 200 OK message. The 200 OK message will notify phone A which UDP port it will listen for RTP packets. During this negotiation phase, the two parties also agree on the codec to use and also whether or not Voice Activity Detection (VAD) and Comfort Noise (CN) is supported.

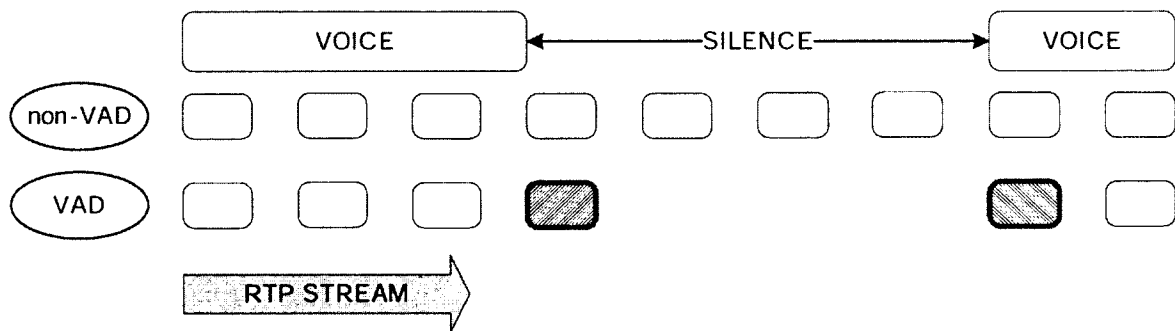


Figure 2.1: VAD and RTP

RTP packets are usually sent at evenly spaced intervals that the two parties agree upon during the codec negotiation phase. For instance, if two parties use the G.711 codec, typically a 20ms RTP interval is used. Figure 2.1 shows a typical RTP flow. RTP

packets that do not use VAD, that is silence packets are transmitted the same as voice packets, we define as non-VAD. However during silence periods, VAD has a different behavior. When the silence period starts at the sender end, the next scheduled RTP packet is a Comfort Noise (CN) packet. When the CN packet is received at the receiver side, it knows that a silence period has begun. When the period of silence has stopped, a normal RTP packet with the Marker bit is set. When the this RTP packet is received, the silence period has ended and a new voice period has begun.

2.2 Silence Modeling & Prediction

In this section, we use a lightweight threshold based algorithm to detect silence RTP packets from voice RTP packets. Then, we compute the length of silence periods between consecutive voice packets and also use statistical analysis to characterize the silence data. We finally present an algorithm to predict the future silence period length based on observed silence history. Such prediction will be used in the next section for saving WiFi energy in smart phones.

2.2.1 Lightweight Silence Detection

It has been shown that approximately 60% of a typical conversation is made up of silence [23]. Intuitively, during a silent period of a conversation we should not need to transmit any packet and therefore maximize the energy savings of the WiFi radio used. Our approach is to look for mutual silence where at time t , all RTP streams are silent from both parties. That is, the payload in the RTP packets have no meaningful data. This can correspond to a short silence period between two consecutive words in a sentence or a pause in a conversation, for example.

Although some codecs such as g.729b and G.273.1 3GPP provide Silence suppression [25], these codecs are not always available. By implementing a lightweight silence detection on the phone, this allows greater flexibility and allows SiFi to be used with any voice codec.

In order to find the start of a period of mutual silence, we have two cases to deal with. The first case is that the remote end of the conversation supports VAD. In this case the silence detection is dealt with by the sender. By examining the RTP packets, we can easily determine when the silence starts. For the second case where VAD is not in use, we implemented a simple Digital Signal Processing (DSP) algorithm. This algorithm and its associated problem, which is well studied in the Speech and Language Processing research area, is known as Endpoint Detection. However, most recent Endpoint Detection algorithms [28] [37], although highly accurate, are very computational expensive. Since the end result of our research is to save energy, we decided upon a lightweight threshold based algorithm that relies upon the amplitude of the audio stream which from our analysis performs reasonably well.

The DSP algorithm we use is as follows. We set a threshold th . th is configured manually to the noise level of a normal conversation. When the average audio level (over the past k samples) are less than th , $Silence = true$, where k is the number of audio samples in a single RTP packet. When the average audio level equals or exceeds th , $Silence = false$.

For the purpose of simplification, we assume the VAD case in the following discussions unless explicitly stated otherwise. Since we can observe the start of the mutual silence period we need to determine how long the silence period will be so that we can maximize the radio sleep time.

2.2.2 ECDF Based Silence Prediction

In this subsection, we discuss in detail how to design the silence length prediction. An accurate but simple (less computationally intensive) silence length prediction algorithm is important as this helps achieve better conversation quality while at the same time saves more energy.

To this end, we first analyze 7 Skype call traces which last 5 hours and 41 minutes in total and contain 52,929 silence periods (65% of the call time). These Skype calls are conducted by 3 different groups of participants. Group 1 and Group 2 each has three participants using two Skype clients: one participant at one end, and two at the other end. We collected 3 traces from Group 1, and 2 traces from Group 2. In Group 3, 4 participants use 4 Skype clients respectively for conference calls, and we collected 2 traces. The silence period lengths from the traces are used to build the empirical cumulative distribution function (ECDF) as shown in Figure 3. From the ECDF, we observe that the length of the silence period can range from 20ms to more than one minute, with variation as large as more than 900ms. With this observation, we propose a staged prediction algorithm that utilizes the conditional probability. The call traces were gathered from conference calls with students as well as personal calls provided by student volunteers.

Let X denotes the length of the silence period, and $P(\alpha) = P(X \leq \alpha)$ be the probability that the silence period X lasts less than time α . Then, the probability of the silence period lasting longer than α is $P(X > \alpha) = 1 - P(\alpha)$. Assume that the silence period has already lasted for time α , the conditional probability that it will last longer than $(\alpha + \Delta)$ is:

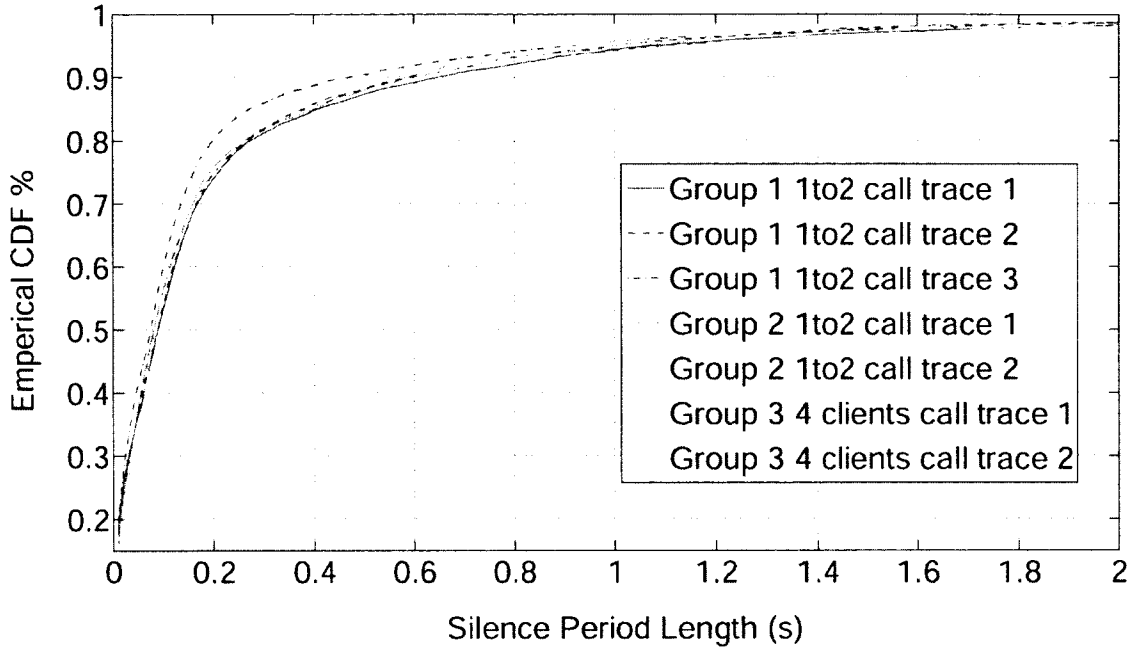


Figure 2.2: ECDF of Silence Length

$$\begin{aligned}
 P(X > \alpha + \Delta | X > \alpha) &= \frac{P(X > \alpha, X > \alpha + \Delta)}{P(X > \alpha)} \\
 &= \frac{1 - P(\alpha + \Delta)}{1 - P(\alpha)}
 \end{aligned} \tag{2.1}$$

Input: confidence interval β , observed silence length α , *ECDF*

Output: predicted silence length Δ

on event that a silence period has lasted for time α , find the maximum Δ that satisfies

$$P(X > \alpha + \Delta | X > \alpha) \geq \beta$$

if no Δ can be found then

$$\Delta = 0$$

else

$$\alpha = \alpha + \Delta$$

return(Δ)

Algorithm 2.1: ECDF Based Silence Prediction

With the ECDF, our prediction algorithm is able to look for appropriate value of Δ (i.e. new increment in time) that has conditional probability $P(X > \alpha + \Delta | X > \alpha)$ larger than

or equal to a given confidence interval β . If such Δ can be found, the algorithm predicts that if the silence period has lasted for time α , it would stay silence for the next period of Δ with high confidence. By the end of each predicted period, the algorithm will predict again if it detects that the silence period continues. The prediction loop breaks when the silence period ends or no Δ value can be found with the confidence bound. This algorithm is shown in algorithm 1.

We use R^2 error value to evaluate the accuracy of the prediction, which is commonly used to measure how well statistical models can predict the future outcomes. Let f_i be the predicted value, y_i be the real value, and \bar{y} be the mean of y_i . R^2 is computed using Equation 2.2. With R^2 value closer to 1, the predictions are more accurate.

$$\begin{aligned} SSE &= \sum_i (y_i - f_i)^2 \\ SST &= \sum_i (y_i - \bar{y})^2 \\ R^2 &= 1 - \frac{SSE}{SST} \end{aligned} \quad (2.2)$$

We apply the prediction algorithm to the 7 Skype traces with the first half of the traces as training data to build the ECDF, and predict for the second half and test its accuracy. Given $\beta = 65\%$ and α initiated as 50ms, 6 of the prediction results have R^2 value above 0.9 and the other one over 0.8.

2.2.2.1 Determine the Runtime Training Length

To determine the proper length of the training period, we compute the Kullback-Leibler divergence [36] of the silence period length distribution. When a new call starts, the system begins to collect the training data of silence periods in groups of 50 (usually included in about 20s calling time). If the Kullback-Leibler divergence between the current training set including and excluding the new group of silence periods is larger than a

predefined threshold KL_{thres} , the new group of 50 silence periods is added to the training data set, and the training continues. Otherwise, we know that the current training data is enough to build a stable ECDF. When the training period ends, the prediction period begins. KL_{thres} is a design parameter. For example, with $KL_{thres} = 0.02$, in one of our traces the training periods stops after 500 silence periods, which lasts 193s in call time, see Figure 2.3.

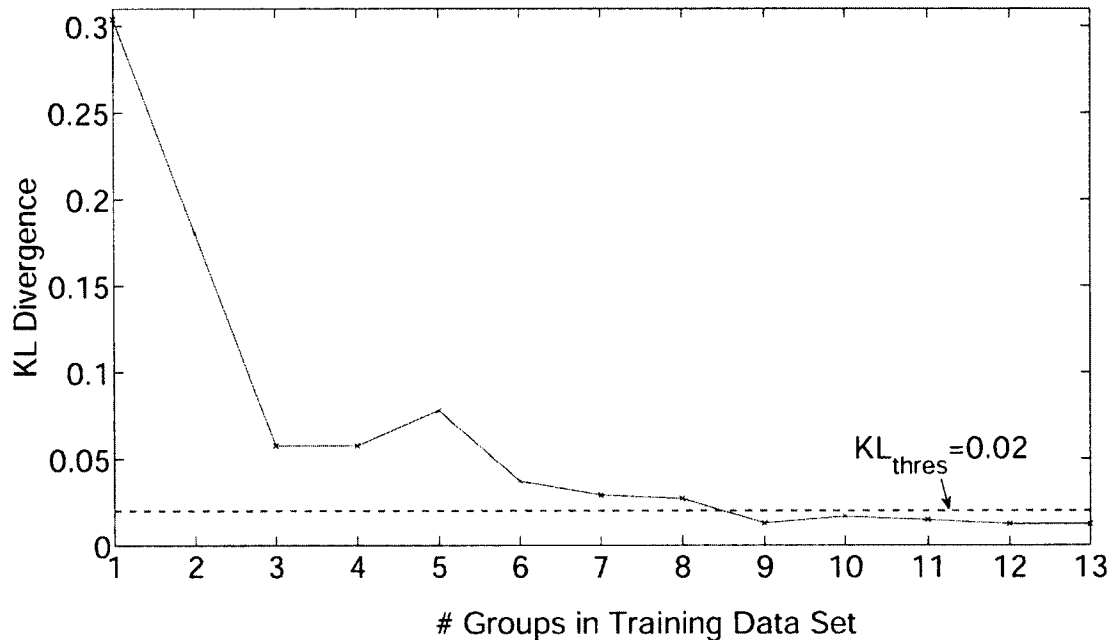


Figure 2.3: KL Divergence vs. Training Length

After the training period, we also check whether the ECDF needs to be updated with every 50 new silence periods. If the Kullback-Leibler divergence raises above KL_{thres} . The system stops predicting and updates the ECDF with new silence periods. When the Kullback-Leibler divergence is below KL_{thres} , the prediction resumes.

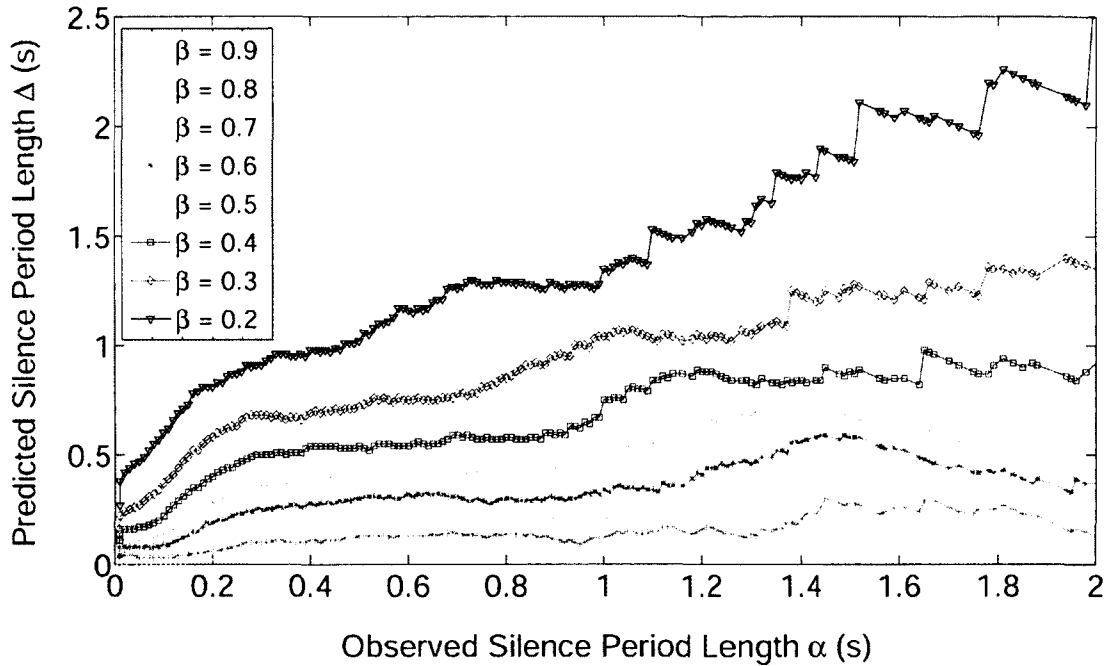


Figure 2.4: Observed Silence Period Length α vs. Predicted Silence Period Length Δ

2.2.2.2 The Observed Silence Length α

Figure 2.4 show the analysis of the predicted silence period length with different observed silence period lengths. For different values of confidence interval β , we observed that the predicted silence period length Δ generally increases when α increases. When a silence period starts, Algorithm 1 waits for an initial period of α before prediction, with the phone on the CAM mode. If the setting of the initial α is too large, some silence periods may be too short to predict. For longer silence periods, less energy can be saved with the long initial waiting period. If the initial period α is too short, however, the predicted silence period may be too short to save energy. Also, the lightweight silence detection is not very accurate in classifying very short silence periods, even though it is very accurate for longer silence periods.

Therefore, we decide to use a larger value of initial α and do not predict for very short silence periods. We configure the initial value of α so that the first predicted Δ (a

sequence of Δ predictions are possible within the same long silence period) is larger than 20ms. In our trace data, the corresponding initial α value is 50ms.

2.2.2.3 The Confidence of Prediction

| β | R^2 | Mean Iteration | Mean Δ |
|---------|--------|----------------|---------------|
| 0.15 | 0.0237 | 1.1155 | 0.7846 |
| 0.20 | 0.2894 | 1.1636 | 0.6215 |
| 0.25 | 0.5778 | 1.2084 | 0.4998 |
| 0.30 | 0.6332 | 1.2641 | 0.4192 |
| 0.35 | 0.8044 | 1.3362 | 0.3509 |
| 0.40 | 0.8448 | 1.4246 | 0.3039 |
| 0.45 | 0.8902 | 1.5262 | 0.2612 |
| 0.50 | 0.9047 | 1.6576 | 0.2260 |
| 0.55 | 0.8965 | 1.8258 | 0.1925 |
| 0.60 | 0.9149 | 2.0046 | 0.1664 |
| 0.65 | 0.9251 | 2.2376 | 0.1425 |
| 0.70 | 0.9187 | 2.5303 | 0.1214 |
| 0.80 | 0.8946 | 4.0082 | 0.0679 |
| 0.75 | 0.9387 | 3.1789 | 0.0905 |
| 0.85 | 0.8750 | 5.5128 | 0.0472 |
| 0.90 | 0.7861 | 9.3979 | 0.0259 |

Table 2.1: Prediction with Different β

For each silence period, our prediction algorithm iterates until the silence period ends or no Δ can be found with the given confidence interval β . We analyze the effect of β on the prediction accuracy in Table 2.1. R^2 is used to measure the prediction accuracy. We observe that when the confidence interval β increases from 0.10 to 0.65, R^2 increases because each Δ is calculated with higher accuracy. But R^2 fluctuates when β increases after 0.65, and when β equals 0.90, the R^2 drops to 0.7861. This is because the algorithm can not find a Δ prediction for very large α with the given confidence interval.

In order to save more energy, large silence prediction Δ is preferred with a lower confidence interval β , but the prediction also risks large errors which may degrade the quality of the phone call. We can estimate the expected error for each prediction of Δ . If the silence period stops at some time x before the predicted time ($\alpha + \Delta$), the

voice packets arrived after x will be delayed. Also considering the beacon interval I in the Power Save Mode, the delayed packets won't be received until the next beacon. Equation 2.4 defines the expected error E of prediction which is also the expected delay.

$$d_i = \begin{cases} \alpha + \Delta - x_i, & \text{if } \Delta \leq I \\ I - (x_i \bmod I), & \text{if } \Delta > I \end{cases} \quad (2.3)$$

$$E = \sum_{x_i \in (\alpha, \alpha + \Delta]} \frac{P(X > x_i) - P(X > x_{i-1})}{P(X > \alpha)} * d_i.$$

We find the value of β so that for all possible prediction Δ the expected error is below a threshold. Besides the prediction accuracy, we also consider the cost for the algorithm to wake up and check for each iteration. For each silence period, the prediction algorithm should not wake up too many times. From Table. 2.1, we can see the mean iterations for predictions within one silence period increases as β increases. Considering both accuracy and cost, in our experiments, we limit β values within the range of 0.25~0.6. We found imperically that the phone call fidelity was of good quality for β values in that range.

Time series models such as Autoregressive moving average (ARMA) and Autoregressive Integrated Moving Average (ARIMA) [17] were also examined, but the drawbacks were many. We found a weak data correlation and low prediction accuracy, not to mention the higher computation cost. Our ECDF based prediction algorithm is both efficient and can predict the silence period length with high accuracy.

2.3 SiFi: Silence Prediction based WiFi Energy Adaptation

We propose SiFi: silence prediction based WiFi energy adaptation. As shown in figure 2.5, SiFi is composed of the following main components: Modeling and Prediction, WiFi Manager, and the Silence Classifier. The RTP Media Server is an optional

component included here to handle the case of VAD. Each of these components has been added to the SIP user agent on Android.

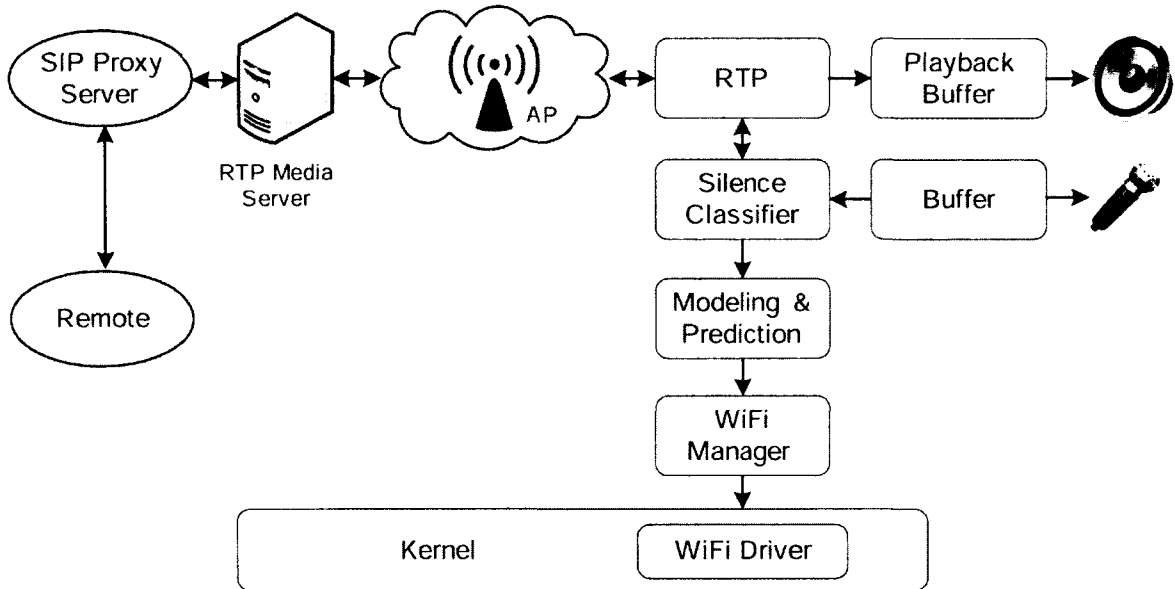


Figure 2.5: SiFi Architecture

Modeling and Prediction. Based upon the call history, we can determine how long we can sleep for. Modeling has two modes: training and running mode. During training mode, the modeling component is fed the observed silence period lengths from the silence classifier. The cumulative silence period length data is stored in an internal data structure. We transition to the running mode as soon as we have enough silence period lengths. Immediately after entering the running mode, we compute the empirical cumulative distribution function (ECDF) based on this training data. We also calculate an appropriate α based on the ECDF.

The prediction component is idle during the training period. As input, the prediction component receives two arguments: the silence period start event and β . We immediately sleep for the predefined α milliseconds. Then we follow Algorithm 1. Based on the value of β we predict the length of the sleeping period.

WiFi Manager. The WiFi manager is responsible for putting the WiFi radio to sleep. When the current WiFi power mode is switched to active we override the Adaptive PSM and forcibly place the WiFi driver into PSM mode for a specified time period. Once the time period expires, the driver is switched back to Adaptive PSM.

Silence Classifier. This component is responsible for detecting silence in both directions: inbound RTP packets and outbound RTP streams. When RTP packets are received, the silence classifier observes the silence start and silence stop events. We compute the average amplitude level over each received RTP packet. By using a known threshold value, we can detect if the current packet is a start or stop silence event.

Inbound RTP packet processing needs to deal with two separate cases. The first case is when silent packets are embedded into the RTP stream or the non-VAD case. In the non-VAD case, the silence classifier checks every packet. The silence classifier also handles the case when VAD is enabled for a call. In this case, it has the same responsibility for detecting silence, with an additional layer of complexity. In addition to checking every received packet for the average amplitude level as before, the call state is implicitly received through Comfort Noise (CN) packets and those packets with the *marker* bit set. Silence starts when a CN packet is received while silence stops when the Marker bit is set. The silence classifier still computes the amplitude level of every packet, in case the remote VAD implementation is not aggressive enough at detecting silence.

For the silence classifier to handle outbound RTP streams, we sample input from the microphone and detect the average amplitude level. The sample interval is determined by codec negotiation. As before, when the amplitude level drops the threshold th , a silence start event occurs. Similarly, when the amplitude level exceeds the pre-defined threshold, a voice event occurs. If VAD is enabled, we do not send silence packets. Only voice RTP packets are transmitted.

The voice and silence states from both outbound and inbound are combined to

produce a joint event we call mutual silence. Mutual silence occurs when there is silence both with the sender and receiver RTP streams. These mutual silence start and mutual silence stop events are sent to the Modeling and Prediction component. To illustrate mutual silence, we consider the following scenario: the receiver RTP stream is silent for three seconds. Starting at time 0, the sender RTP stream alternates between one second of silence, then one second of voice and so on until three seconds. This means the first mutual silence period is between time zero and one. The second is between time two and three.

RTP Media server. Finally, our last component is the RTP Media server. This is an optional component included here to handle the case of VAD. We found an open source RTP Media Server [27] that allows us to relay any RTP traffic including embedded silence in the RTP payload. The media server has its own DSP algorithm and detects when silence occurs. When silence starts, a CN packet is inserted. When voice starts, an RTP packet with the marker bit is sent.

2.4 Android Phone Implementation

We implemented our design on the Sprint HTC Hero [63] with root access. This version contains version 2.1 of Android with HTC enhancements. The Hero has the TI WLAN 1251 driver which is part of the Android source repository that is freely available on top of the 2.6.29 Linux Kernel. The implementation is comprised of two parts: the WiFi system modifications and the application modifications done within the Android virtual machine layer. We first start with the application, followed by system level modifications.

2.4.1 Application Level Modifications

There are a number of VoIP clients available on the Android market. For our research, we wanted one that is well established on the Android Market. A big plus was also to find a package that was open source since we need to make a number of changes. One such SIP User Agent [61], hereafter referred to as SIPua, is available for Android with over 250K downloads from the Android Market and has high user ratings. It provides the ability to make and receive VoIP phone calls using WiFi or a mobile data plan. It is able to handle video and audio and a number of codecs are supported. It is comprised of third party SIP and RTP stacks.

The silence classifier component is shared between the SIPua's RTP sending and receiving threads. We analyze the energy level of the payload of each RTP packet. We first implemented the component in Java. We quickly realized, however, that the energy calculation was too computationally expensive and the performance was unacceptable. Using the Android Native Development Kit (NDK) [3] for a native code implementation, we were able to obtain acceptable performance with a minimal performance penalty. We measured the performance penalty to be a 3% energy overhead. The penalty was realized by comparing a call with the silence classifier enabled for both inbound, outbound streams to a call made with the silence classification disabled.

2.4.2 System Level Modifications

We introduce the WiFi Manager implementation. The WiFi Manager has two components: the low level system implementation and the Application API. The latest Android system (as of this writing version 2.2) is missing the functionality to forcibly switch the WiFi radio from adaptive PSM to static PSM from the Android environment. We modified the Android Virtual Machine to include this functionality, but quickly realized this broke other parts

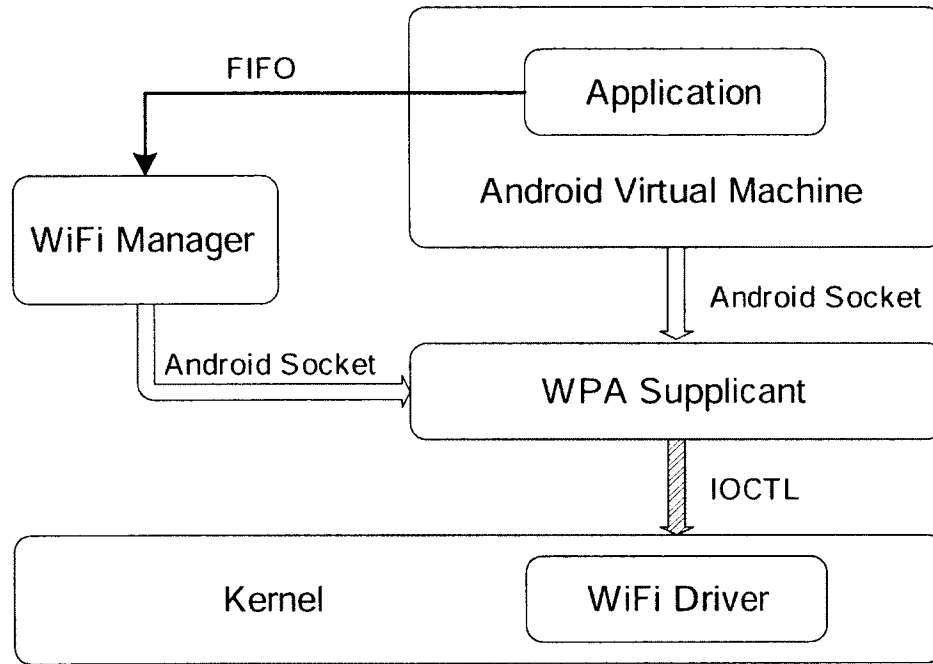


Figure 2.6: Android WiFi Architecture

of the system. As we discovered, although Android is open source, contributors can add to the system various components that are not. By modifying the Android VM, we would not be able to have a fair comparison since our modified VM would be missing key components. Instead we needed a simpler design that would not require modifications to the Android VM.

Figure 2.6 describes how the current WiFi subsystem works within Android. When an application makes an API call to the WiFi sublayer, it passes through the VM and connects to the *WPA_Supplicant* daemon running on the system. To save energy, when the WiFi radio is inactive, it runs for 15 minutes and then the driver is disabled in the Linux kernel. When the Android system receives an indication that network activity starts again, the driver is then loaded again. During the loading and unloading phase, the *WPA_Supplicant* daemon is also started and stopped. The *WPA_Supplicant* listens for events through an Android Socket interface. The Android socket is essentially a UNIX socket with some modifications.

We created a separate daemon process called the WiFi Manager. This daemon simply listens on a FIFO interface which our application has access to. Then, when the Android application wants to send a special WiFi command, it can send a FIFO message to the WiFi Manager. We used the Android NDK to implement a native code interface to interact with the WiFi Manager. The WiFi Manager then communicates to the *WPA_Supplicant* daemon through the same Android Socket as the VM. We modified the init scripts on the phone such that whenever WiFi is enabled and disabled, WiFi Manager is also started and stopped, respectively.

Once the aforementioned system was in place, it was trivial to implement the `sleepforMS()` WiFi function. This function an integer millisecond argument and sends a request to the WiFi Manager through the FIFO interface.

We also modified the *WPA_suppllicant* daemon. When it receives the `sleepforMS()` command, it forcibly puts the radio into PSM mode. Then it pauses for the specified milliseconds before changing the radio back to Adaptive PSM.

Finally, we modify the RTP Media Server. When a silence start event is observed, a CN RTP packet is sent. Prior to our modifications, the RTP Media Server would only set the marker bit when a voice start event occurred. With this change, the Silence Classifier component can easily detect when silence starts and stops when VAD is enabled.

2.5 Performance Evaluation

| Radio Status | Screen ON | Screen Off |
|--------------|------------|------------|
| CAM | 1070.00 mW | 726.05 mW |
| PSM | 381.40 mW | 36.5 mW |
| Disabled | 345.94mW | 4.9 mW |

Table 2.2: Baseline Average Power

With the implementation in Sprint HTC Hero phone, we present system performance evaluation in the Android platform. Our results demonstrate that our SiFi solution achieves more than 40% energy savings in the smart phone.

2.5.1 Evaluation Setup

| Threshold | Metric |
|-----------------------|---------------|
| AutoPowerModeActiveTh | 8 packets/sec |
| AutoPowerModeDozeTh | 4 packets/sec |
| Transition δ | 1.5 sec |

Table 2.3: Adaptive PSM Settings in Sprint HTC Hero

Our evaluation setup consists of a Linux server that runs a media server [27] and runs hostapd to serve as a WiFi AP. The media server is configured to host audio recordings. When a specific dial string is called the recording is played back. The phone is approximately one meter from the AP. The only delay incurred is isolated to the wireless leg.

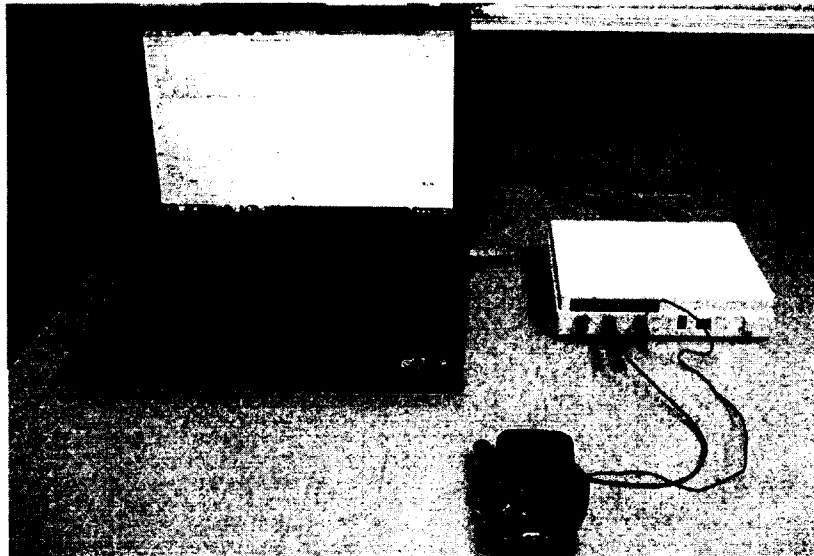


Figure 2.7: Energy Measurement Setup

We measure the energy use of the phone in real-time using a power meter from Monsoon technologies. Figure 2.7 shows the setup for energy measurement. A simple circuit is configured such that the positive terminal on the battery is insulated with electrical tape and the positive feed from the power meter is instead connected to the phone’s battery terminal. When the phone is powered on, we can measure in real-time the energy usage.

Table 2.2 shows the baseline power usage of the phone. We measured the power usage by placing the phone in ‘airplane mode’ which disables all the network interfaces including Bluetooth and mobile radio. Clearly CAM mode is very expensive and should be avoided if possible. Powering the screen is also quite expensive as well. The measurements indicate the power consumed when the radio is idle.

2.5.2 Evaluation Method

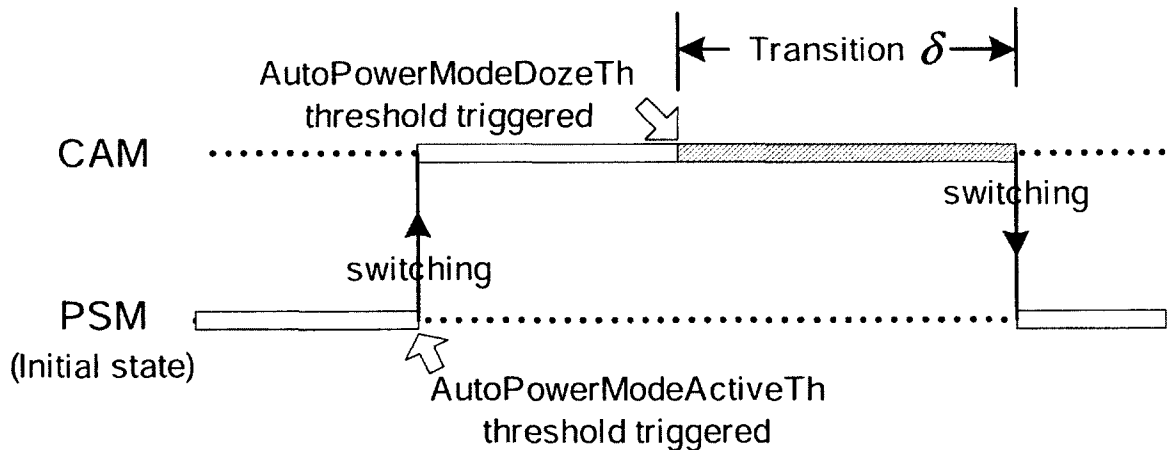


Figure 2.8: Adaptive PSM State Transitions in Sprint HTC Hero

In the following sections we show the results on how SiFi compares with Adaptive PSM in the VAD case and the NON-VAD case. The Sprint HTC Hero phone uses Adaptive PSM to save energy [35]. Adaptive PSM is completely controlled by the WiFi

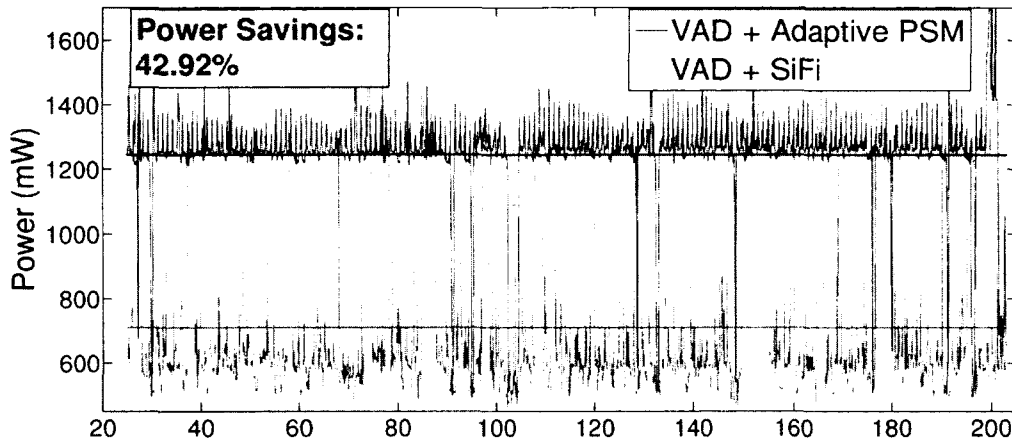


Figure 2.9: VAD + Adaptive PSM vs. VAD + SiFi

kernel driver. The Adaptive PSM sets the default power mode to PSM. When the network interface starts to receive packets, it transitions to CAM mode by sending a NULL:Awake packet to the AP. When the wireless network interface is idle and the driver desires to switch to PSM, it sends a NULL:Sleep packet to the AP.

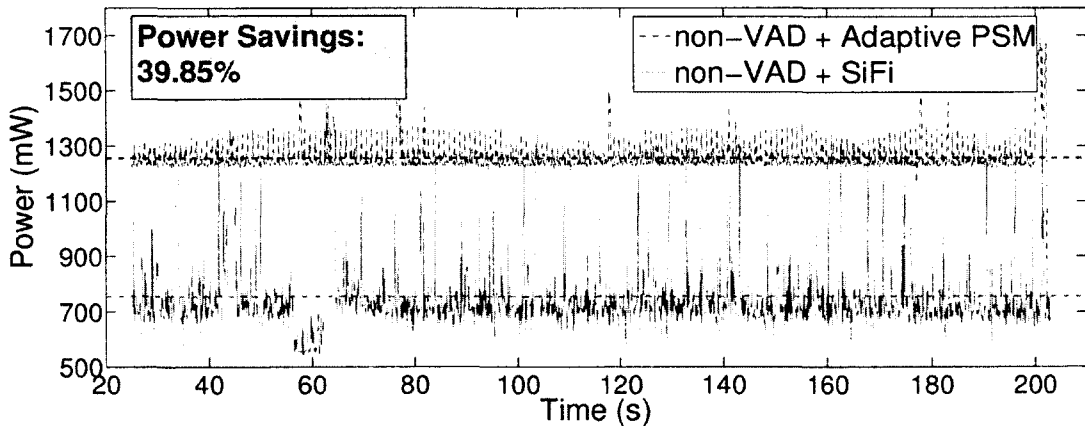


Figure 2.10: non-VAD + Adaptive PSM vs. non-VAD + SiFi

In Table 2.3 we see the settings for Adaptive PSM for the Sprint HTC Hero. The *AutoPowerModeActiveTh* parameter refers to the number of packets necessary to trigger the phone to switch from PSM to CAM mode, while the *AutoPowerModeDozeTh* parameter shows the threshold value necessary to switch back to PSM mode. We

note these settings are the default settings for this phone. The transition δ parameter is not configurable (without hacking the driver, as we do in the next section). We observed this parameter by sending a small packet burst to the phone that exceeds the *AutoPowerModeActiveTh* threshold. By watching the kernel logs on the WiFi Access point, we recorded the time difference from when the STA returned to PSM after switching to CAM. We also determined that no other network traffic was sent on the interface.

Although the observed transition δ value might be unique to this phone, others have also observed similar behavior. [59] cites several variations in implementation among common Smartphones. Aggressive Adaptive PSM, is when the transition δ is small. Default Adaptive PSM is when the transition δ is longer such as the Sprint HTC Hero. When we refer to Adaptive PSM, we are referring to the default case.

There is tradeoff between the aggressive and default adaptive PSM. As noted in [59], Aggressive Adaptive PSM can in some cases lose packets, and ultimately consume more energy when the AP is under heavy use and the transmit buffer is full. This is best illustrated by the following example. Suppose the phone is in CAM mode due to a high packet receive rate. Some time later, the AP transmit buffer becomes full from some other traffic. If it takes longer than the transition δ to process the packet through the transmit buffer, the phone will go to sleep and the packet may be lost.

2.5.3 SiFi Energy Savings

In this section we show how SiFi improves both the VAD and non-VAD cases. SiFi is able to efficiently save 40~43% average power over Adaptive PSM.

Figure 2.9 shows the results of VAD with SiFi enabled. For this call, β was set to .5 and α to 50ms. A 43% power savings was achieved over Adaptive PSM. At certain extended voice periods of the call, for example at 150 seconds, the power levels of

Adaptive PSM and SiFi are the same. Similarly when extended silence periods occur, Both Adaptive PSM and SiFi can take advantage. For instance, during the periods right before and after the 100 second mark we see the energy use drop significantly. Adaptive PSM performs poorly due to the long transition δ .

We present the results of non-VAD + SiFi in Figure 2.10 . β was set to .25 and α to 50ms. When SiFi detects silence and puts the radio into PSM mode, RTP packets continue to queue up at the AP. This results in extra overhead since the phone has to PS-POLL every packet queued at the AP. Therefore, even though β is set to half of the setting used for VAD, the overall power savings are slightly less. Compared to Adaptive PSM, SiFi has 39% power savings. Since RTP packets are always sent even during silence periods, Adaptive PSM never switches to PSM. Once a call is established, there will always be a packet rate that exceeds the *AutoPowerModeActiveTh* threshold.

To be fair, since the Android phone has a very long transition δ setting, we modified the WiFi driver such that the transition δ was as close to α as possible. Without major modifications to the driver, the lowest setting we were able to maintain was approximately 70ms. Adjusting the transition δ to anything lower than 70 caused stability issues(exceeding the Active threshold did not always keep the driver in CAM). We compared SiFi vs. Aggressive Adaptive PSM and a non-VAD call. In this case, SiFi clearly wins out because the amount of RTP traffic does not change during silence periods, so it will never switch to PSM. Secondly, we compare a VAD call with SiFi vs. a VAD call with Aggressive Adaptive PSM. In this case, SiFi has a 34% improvement (1070 vs 710 mW).

2.5.4 SiFi Application Fidelity

We evaluated the application fidelity of SiFi by using the industry standard for evaluating Voice Quality, the Mean Opinion Score (MOS). The MOS scale ranges from 5-1, with

(5) best quality, (4) High quality, (3) Medium quality, (2) Low quality and (1) Completely unusable. VoIP is particularly susceptible to packet loss and delay. The E-model [19] can be used to calculate the MOS rating of a call.

The MOS score was originally designed to be a subjective measurement of call quality. The E-model can be used to estimate the MOS score based on the observed packet loss, one way delay and codec metrics from the codec in use. The E-model computes the R-factor which can then be used to calculate the MOS score as follows:

$$1 + 0.035R + 7 * 10^{-6}R(R - 60)(100 - R).$$

The R-Factor can be calculated as $R \sim 100 - I_s - I_d - I_{ef} + A$. Where I_d is the delay impairment, I_{ef} the loss impairment, I_s the signal-to-noise impairment and A the expectation factor. The latter is a subjective measurement that is higher when users expect higher call quality. We set A to zero as in [10, 19] since it is not easily quantified. Since we are using the g.711 codec, the R-Factor can then be simplified to [19]: $R \sim 94.2 - 0.024d - 0.11(d - 177.3)H(d - 177.3) - 30\ln(1 + 15e)$. Where d is the mouth-to-ear delay comprised of the codec delay, the delay due to the jitter buffer and the network delay. e is the error rate comprised of total packets lost and late packets dropped by the jitter buffer. H is a heavy-side function where $H(X) = 0, X < 0$ and $H(X) = 1, X \geq 0$.

The g.711 coding delay is 20ms and we assume the jitter buffer delay is 60 ms. Finally, the network delay is calculated as follows: We assume the one way delay across the US is 40ms to account for VoIP calls over the Internet. d is then equal to $d_{net} + 120ms$, where d_{net} is the delay caused by WiFi. d_{net} is computed by measuring the one-way delay from the media server to the phone. We measure the one-way delay by using the same method in [10]. We assume that the delay is identical in both directions and remove the clock skew according to [45].

We calculate the e_{jitter} by assuming any packet with jitter higher than the jitter buffer,

| Description | Length | μ OWD(ms) | e_{jitter} | MOS |
|-------------|------------|---------------|--------------|------|
| non-VAD | 1 hour | 85 | 1.4% | 3.76 |
| VAD | 10 minutes | 63 | 1.4% | 4.03 |

Table 2.4: SiFi Fidelity

that is 60ms is lost. We made a number of 10 minute calls comparing both the non-VAD and VAD and show the results in Table 4. The overall MOS score on the VAD call slightly outperforms the non-VAD call since only voice packets are transmitted. The MOS is calculated by adding the extra 40ms Internet delay to the shown values. The overall results show that SiFi causes minimal call quality degradation.

| Call | SiFi | #People | Duration (min:sec) | μ Power (mW) | μ OWD (ms) | e_{jitter} | MOS |
|------|------|----------|--------------------|------------------|----------------|--------------|------|
| #1 | y | 2 people | 26:38 | 615 | 68.6 | 0.63% | 4.28 |
| #2 | y | 3 people | 48:53 | 618 | 69.3 | 2.03% | 4.15 |
| #3 | y | 4 people | 50:00 | 675 | 69.3 | 1.14% | 4.25 |

Table 2.5: Multiple Longer Call Tests: VAD + SiFi

2.5.5 SiFi Robustness for Longer Calls

In this section, we show the robustness of SiFi for longer calls. Different call scenarios are explored: longer call lengths, different languages, different genders, and different numbers of people in the phone call. Table 2.5 describes the following scenarios: Call #1 consists of 2 people (both females & using Chinese) conversing for 26 minutes. Call #2 is a 3 person (all males & using English) conference lasting for 48 minutes. Call #3 is a 4 person (all males & using English) conference lasting for 50 minutes. In all three combinations, SiFi shows consistent high power savings. For reference, the average power consumption with VAD + Adaptive PSM enabled on Call #2 and SiFi disabled was recorded as 1263mW. For longer calls, SiFi performs 51% better than VAD + Adaptive PSM. While saving energy, the quality does not suffer as well, the MOS in all cases is well above 4.

We also tested the stability of the training period. First, training was enabled for call #2 as reflected in Table 2.5. Call # 3 was made using the training data from call #2. Less than a 10% difference in is apparent between the two calls. As was shown in Figure 2.2, the silence period distributions observed in the traces are similar. This shows that if training from a previous call is re-used, significant energy savings can still be realized while maintaining high fidelity.

2.6 Conclusions

In this work, with lightweight digital processing, and runtime modeling and prediction, we exploit during runtime the silence periods of a VoIP call. Thorough statistic analysis is conducted to ensure high modeling and prediction accuracy and low cost for the runtime silence exploitation. To apply silence period exploitation to save smart phone WiFi energy, we also propose the design, implementation, and real system evaluation of a silence prediction based WiFi energy saving framework called SiFi. By making modifications to the low level system architecture and also application components, SiFi is able to directly control the WiFi power save mode based on silence prediction. Our real system evaluation running VoIP application demonstrates that SiFi saves more than 40% energy compared to the standard Adaptive PSM solution deployed in Sprint HTC Hero. We achieve high call fidelity by sleeping during silence periods and active during voice periods. Our real system evaluation also demonstrate SiFi is resistant to network congestion and robust in multiple phone call scenarios.

Chapter 3

SAPSM: Smart Adaptive 802.11 PSM

WiFi on Smartphones is a significant source of energy consumption. Constantly Awake Mode (CAM) consumes 20 times more power than Power Save Mode (PSM) when idle [54]. PSM consumes little power at the cost of added latency of up to 300ms. Latency can cause performance issues for interactive applications such as web browsers [35] and real-time VoIP applications [47]. CAM consumes high power but delivers high performance and low latency.

There are a number of existing PSM and adaptive PSM mechanisms [31] [35] [32] that utilize PSM to save energy (described in the background section). Adaptive PSM alternates between PSM and CAM based solely upon network activity thresholds. Clearly, the choice between PSM and CAM should be done carefully. But what criteria should be considered when the switch is made? Existing Adaptive PSM behavior on Smartphones, we find through controlled experiments, has a simple threshold based approach reacting to aggregate traffic volume. Unimportant traffic bears the same weight as foreground highly interactive traffic.

How can Adaptive PSM on Smartphones be improved? Clearly the Adaptive PSM implementations we evaluate here are not adequate. Certain applications have varying

delay tolerances. Background traffic with a high delay tolerance should use a strategy optimized for energy conservation. On the other hand, interactive traffic should be optimized for performance. STPM [1] is a pioneering work that considers the priority of different traffic flows. However, it requires application developers to indicate traffic intent through the use of a custom API. With the prevalence of application stores like the Android Market with thousands of developers of varying skill levels, it is impractical to rely upon developers to accurately provide application intent through an API. For instance, developers generating revenue from advertising may not be motivated to indicate their application is low priority background traffic. Recently, [50] has shown that 65-75% of energy consumed by free Smartphone apps is spent on downloading ads and uploading user tracking information.

A key challenge is *how to determine which applications are high priority without assistance from application developers*. Non-technical users may not be able to determine which applications should be high priority. Once an application's priority is established, *an additional challenge is how to ensure an application's priority is tracked through the system in an efficient and energy conscious manner*.

In this work we present SAPSM, a Smart Adaptive PSM solution that prioritizes network traffic based on application priority, which we now define. Each application is tagged with a priority. When an application is set to high priority, the application's network traffic is permitted to adaptively switch to CAM. A low priority setting for an application means that application's network traffic is not permitted to switch to CAM, but will instead remain in PSM¹. Therefore, SAPSM works with any Android application without any modifications. Additionally, any traffic not associated with an application is considered low priority to save more energy. SAPSM is able to handle both high

¹Low priority traffic remains in PSM for most traffic flows. We experimentally determine that data rates exceeding 3Mb/sec while in PSM consume more power than in CAM. When high data rates are observed, Low priority traffic always chooses the path with lowest energy consumption. See the Evaluation section for more details.

priority and low priority traffic simultaneously. Only high priority applications are permitted to switch the WiFi driver to CAM by incrementing a high priority traffic counter, which can only be incremented by high priority traffic. SAPSM unobtrusively determines an application's priority by observing network usage patterns and assisting users with a short list of applications that might benefit from high priority Adaptive PSM behavior. Each application's priority is stored within a kernel module. SAPSM tracks network traffic priority by comparing the owner of the active listening socket associated with the current traffic flow to a list of known high priority applications. If a match occurs, treat as high priority. If no match is found, the app is treated as low priority.

In summary, motivated by current Adaptive PSM implementation's inability to distinguish network importance, We contribute SAPSM with the following features:

- In SAPSM, we propose the Core component to augment Adaptive PSM behavior by favoring application *priority* compared to aggregate network traffic. We maximize energy savings by staying in PSM for low priority applications when it is expedient to do so, while switching to CAM for high priority applications.
- In SAPSM, we propose the Application Priority Manager (APM). The APM is responsible for observing network behavior. By unobtrusively gathering feedback from users it provides an easy to use mechanism for setting application priority.
- We implement SAPSM on Android. Using extensive experiments performed on an Android HTC Hero Smartphone, we show that SAPSM provides significant energy savings for Android applications that have background low priority traffic: 56% with an RSS reader application and 44% with a popular streaming audio application.

3.1 Background and Motivation

In this section, we first explain the background knowledge of CAM, PSM, and Adaptive PSM. Then, we take two steps to examine Adaptive PSM implementations on a multitude of Smartphones and also a few other devices: in the first subsection, we report on the internals of the Sprint HTC Hero driver; next, we analyze Adaptive PSM implementations by sending spurious network traffic unassociated with any application. By observing Adaptive PSM behavior of different Smartphones, we show that no commercial implementation of Adaptive PSM that we test is able to differentiate between important and unimportant traffic. Every implementation switches to high power CAM when spurious traffic is received. This wastes energy and shows the need for improvement.

3.1.1 Background

First standardized in 1999 [31], the static PSM approach was developed for the client to conserve energy. The client and the access point (AP) agree upon a beacon period. Between the beacon periods, the AP buffers packets. Right before the beacon period, the client awakes and listens for the beacon. The beacon contains a Traffic Indication Map (TIM), which tells the client if packets are currently being buffered. If packets are buffered by the AP, the client will send a PS-POLL message to the AP. The AP will send a data frame back to the client. The data frame includes a MORE field which indicates if more packets are buffered. The client continues to PS-POLL until no further packets are buffered. The static PSM approach saves energy by keeping the radio off except during the beacon period where it briefly wakes up to communicate with the AP.

The static PSM approach saves energy by minimizing the amount of time that the WiFi radio is active. However, there is a tradeoff. Static PSM adds a delay of 100-300ms. In between beacon periods when the client's WiFi radio is off, any incoming packets

are buffered at the AP. This added delay impacts real-time applications such as VoIP and interactive applications like web browsers [35]. Because of this issue, Adaptive PSM [35] [59] is now commonly used to switch between CAM and PSM. The client switches to CAM based upon aggregate traffic volume. To switch between the two modes, the client sends a NULL frame with the POWER field disabled. When the AP receives the NULL frame, it will cease to buffer packets for the client. To switch back to PSM, the client sends a NULL frame with the POWER field enabled. In this case, the AP is now aware the client is in PSM and will start to buffer packets for it.

After the client switches operation to CAM, it will stay in CAM for an idle timeout period before switching back to PSM. This tail energy cost, also prevalent in 3G networks [10] differs between implementations. Our results show a 1.5 second idle timeout period in the HTC Hero, while [59] shows a 20-25ms timeout period for an iPhone 3GS. In the following subsections, we show the results from our survey of various devices.

3.1.2 Sprint HTC Hero Adaptive PSM

Now we examine the source code of the Sprint HTC Hero WiFi driver and describe its Adaptive PSM implementation. Due to the open-source nature of Android, we were able to obtain the complete source of the driver, later modified in the implementation Section. The driver is built as a kernel module and is loaded on-demand by Android.

The behavior is illustrated in Figure 3.1. During a one second interval, which is not configurable, both ingress and egress frames are counted. If the frame count exceeds a configurable *UP* threshold, the driver will switch to CAM. When the frame rate drops below a configurable *DOWN* threshold, it will switch back to PSM.

Since CAM consumes much more energy than PSM, we contend that certain packets should not impact this decision. To illustrate this point, in the next section we send

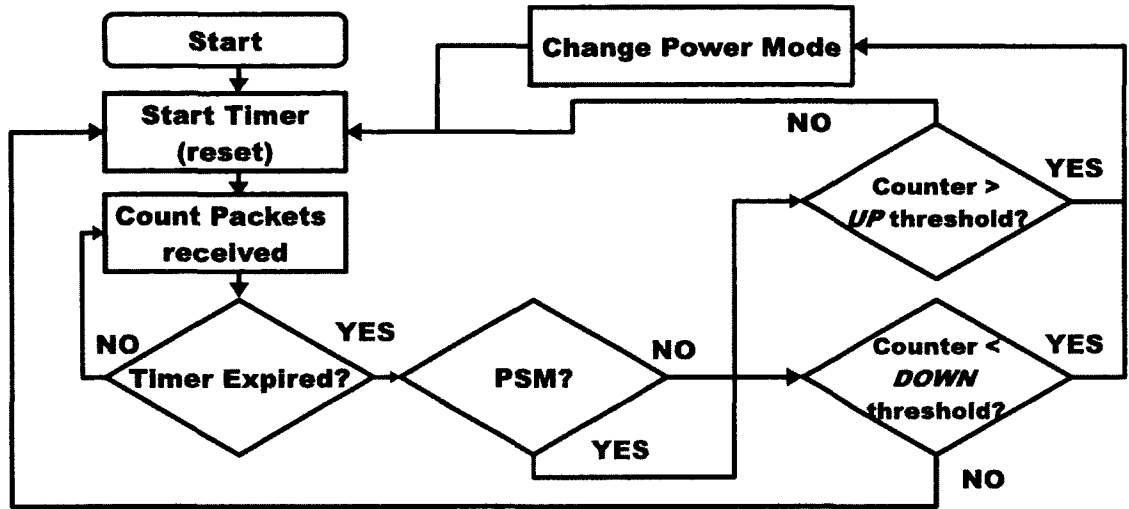


Figure 3.1: Sprint HTC Hero Adaptive PSM implementation; initial state is PSM, timer is set to one second.

unwanted traffic to a number of Smartphones. In all cases, each Adaptive PSM implementation switches to CAM unnecessarily. In contrast to our approach, which emphasizes the priority of network traffic based upon Application priority, existing implementations place equal weight on all network traffic.

3.1.3 Adaptive PSM Behavior of Different Smartphones (and other Hand-held Devices)

How different WiFi driver implementations react to various network traffic is discussed in this section. The methodology for testing the Adaptive PSM behavior is as follows. First, a Linux server is setup running hostapd to operate as an Access Point. The devices were placed less than one meter away to minimize delay and retransmissions. Second, all applications accessing the network were stopped to ensure the only source of network traffic to/from the device is due to the packets transmitted during the test. Finally, all network interfaces besides WiFi, such as 3G/4G and Bluetooth were disabled.

We test the Adaptive PSM behavior of a Blackberry Curve, three Android phones

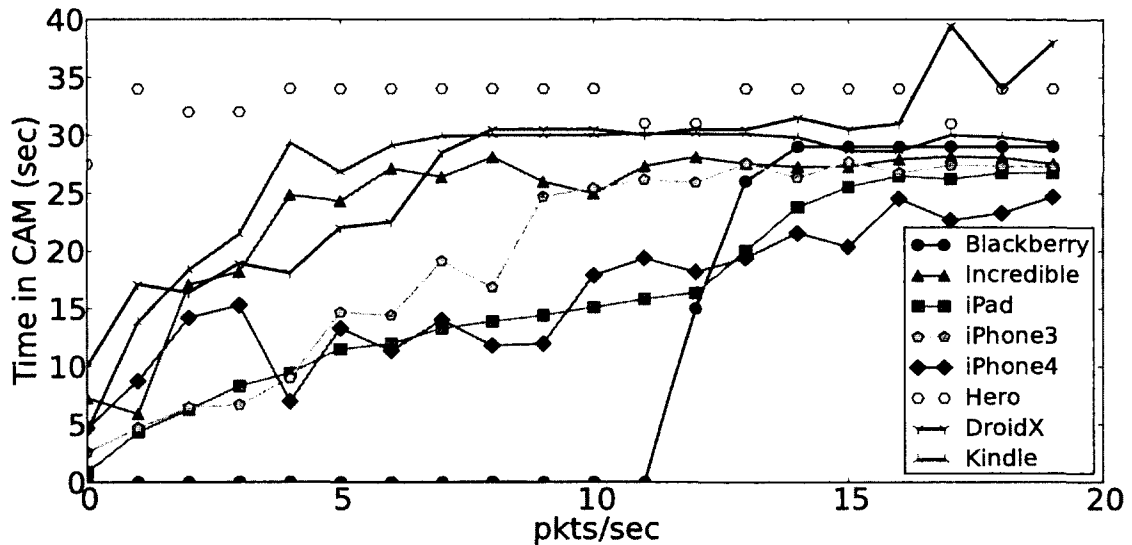


Figure 3.2: Adaptive PSM implementation response to UDP network traffic.

from various manufacturers, a Kindle, an iPhone3, iPhone4 and iPad. The methodology of these tests is described in this section.

By observing 802.11 management frames Adaptive PSM behavior can be observed. Recall that the client utilizes the POWER field in NULL frames as an indication to the AP when to switch between CAM and PSM. The difference in timestamps between NULL frames with opposing POWER settings indicates how long the client stayed in CAM or PSM.

To determine if the test is successful, we measure the packet rate (UP threshold) at which the phone will remain in CAM for an extended amount of time. First, one packet per second of a given traffic type is sent to the device. Gradually the rate is increased up until 20 packets per second. Each test lasts for 30 seconds and is repeated for each traffic type.

Different Adaptive PSM implementations have varying Adaptive PSM bandwidth timer windows. As described previously, the HTC Hero has a timer of one second. Other devices, like the iPhone, have smaller timer windows sometimes referred to as aggressive Adaptive PSM timeout [59]. To compensate for this variation, if the device stays in CAM

for at least 50% of the time we assume the *UP* threshold has been reached.

To illustrate the problem we send Multicast, UDP, ICMP and TCP based traffic to gauge each device's Adaptive PSM response to unwanted traffic. Unwanted traffic in this context means that traffic is generated where there is intentionally no listening socket on the device. For consistency, we keep the packet size to 512bytes across all tests, except for TCP² which is 60 bytes.

| Smart devices | | | Traffic type reaction | | | |
|------------------|-----------|---------------------|-----------------------|-----|-----|------|
| Model | Version | <i>UP</i> threshold | MCAST | UDP | TCP | ICMP |
| iPhone4 | 4.2.1 | 10pkts/sec | N | Y | N | Y |
| iPad1 | 3.2.2 | 12pkts/sec | N | Y | N | Y |
| Curve-8530 | 5.0.0.973 | 12pkts/sec | N | Y | N | Y |
| Droid Incredible | 2.2 | 2pkts/sec | N | Y | Y | Y |
| DroidX | 2.2.1 | 1pkt/sec | N | Y | Y | Y |
| HTC Hero | 2.1 | 1pkt/sec | Y | Y | Y | Y |
| Kindle | 3.1 | 2pkts/sec | Y | Y | N | Y |

Table 3.1: Smart devices tested

The results are shown in Table 3.1. The *UP* threshold is measured for each device. The next several columns show what traffic types can trigger the device to switch to CAM. In most cases, MCAST packets were ignored and did not trigger the device to switch to CAM. All devices were susceptible to the ICMP and UDP variations, while three devices were vulnerable to the TCP traffic.

Figure 3.2 shows the response of each device to the UDP traffic test. Due to the connectionless nature of UDP datagrams, it is easy to understand why all devices tested are susceptible to unwanted UDP traffic. Although some devices have a higher *UP* threshold than others, all devices react to this kind of traffic once the packet rate is increased. By observing this figure, we can determine what the *UP* threshold for each device.

As shown, all adaptive PSM implementations are triggered by unwanted network

²Since TCP requires a connection to be established first and since there is no listening socket, we generate a SYN packet to a non-listening TCP port on the device. The host responds with an RST packet. A SYN packet is 60 bytes: IP 20 bytes + TCP 40 bytes.

traffic. Adaptive PSM behavior is implemented within the WiFi driver which is responsible only for the MAC layer. Any unwanted traffic that is detected by examining layer 3 and above will not be detected. Therefore, current implementations are not able to determine which packets should influence Adaptive PSM behavior without additional information from the TCP/IP stack. We discuss our solution to this problem in the next section.

3.2 SAPSM Design

In order to address the challenge of what traffic is permitted to influence Adaptive PSM behavior, we present SAPSM (Smart Adaptive 802.11 Power Save Mode). SAPSM is designed with the following constraints in mind: (1) Minimal user interaction; even non-technical users can use the system. (2) Performance must not be impacted; the critical-path is respected. (3) Any hints from either the Android system or individual applications are honored.

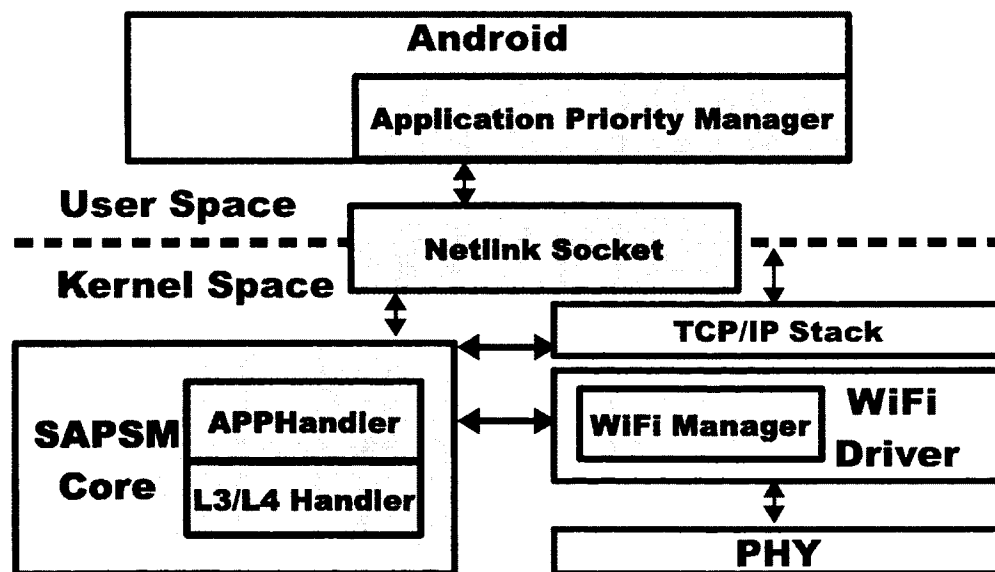


Figure 3.3: SAPSM Architecture; the packet flow is diverted through the SAPSM core kernel module.

The first step in the SAPSM system is to observe individual usage metrics of a specific app running on the phone. This entails recording low level network characteristics and usage patterns, such as the data rate an application uses, which can be used to train a classifier. We develop a classifier which is trained with an assortment of applications specifically chosen which have diverse network patterns. We conduct a user study where users interact with these applications and set the priority of applications. From these results we create a classifier. This classifier can then be used to compare new applications to the training done in our user study.

After the individual usage patterns are observed for an app, these usage patterns are compared with the classifier. By comparing the usage patterns of the existing application with known patterns learned from the classifier, the priority of each application can be estimated with minimal user intervention. Once the priority is determined, the kernel component of the SAPSM system will ensure that the system stays in PSM when low priority applications transmit network data or adaptively switch to CAM based upon the amount of traffic for high priority applications.

The SAPSM system is designed to work autonomously without assistance from the Operating System. However, any hints given are freely utilized. When the screen is off, we assume that the phone is not currently actively used. We then set the entire system to low priority to save energy. This is a reasonable assumption, since by turning the screen off, Android will disable the WiFi driver after 15 minutes. To get around this issue, some applications such as Pandora and Skype keep the screen on, but very dim, when network traffic is anticipated for long periods of time.

Background and Foreground Traffic: The Android API provides developers with several options for retrieving data in the background. Background data be used used to enable a push notification background service [16]. Android provides multiple ways of running data in the background: Background threads and Android services [7] are just two

such methods. Android services or background threads, however, are not necessarily an indication of low priority intent. For instance [61] uses a background service for receiving delay-sensitive RTP packets. Therefore, we cannot rely upon this factor alone to indicate low priority intent.

3.2.1 Architecture

In summary, the SAPSM system saves energy on the device with smart WiFi Power management. By receiving hints from the Operating system, and confirmation from end-users, the SAPSM system is designed to intelligently save energy on Smartphones.

The SAPSM system architecture is described in Figure 3.3. In order to address all of the design criteria mentioned previously, the SAPSM system includes components running at the Kernel level and within Android. These modules are the WiFi driver modifications *WiFi Manager*, the kernel component *SAPSM core*, and an Android application component, the *Application Priority Manager*.

WiFi Manager is a component of the Smartphone's WiFi driver that is responsible for exposing the internals of the Adaptive PSM implementation to be controlled by the SAPSM Core kernel module.

SAPSM Core is a kernel module responsible for determining the priority of packets traversing through the network stack. It performs a check if the current packet either originated from or is destined to an application running on the device. If the socket is currently paired to an application, it determines the priority of the application by performing a lookup in the high priority application table. If the current packet is paired to a high priority application, it will update the Adaptive PSM traffic counter in the driver. Otherwise, the traffic counter will not be updated.

Application Priority Manager is an Android application that runs as a background service. It gathers usage metrics of Android applications running on the phone. Based on inferences gathered from these metrics, hints are provided to the end-user regarding the priority of the application. Finally, it is responsible for communicating application priority settings to the SAPSM core.

3.2.2 SAPSM Core

The SAPSM Core module functionality is detailed in this section. We describe how we track each application's network traffic and how applications priority is enforced.

Inbound Packets: When a packet first enters the WiFi interface, the driver does checking on the MAC header and passes the packet on to the networking stack. Before the packet is processed by the networking stack, the packet is intercepted by the kernel module. We check the destination port number and determine if there is a process listening on that port. If a valid process is found we compare the UID of the process against the list of known high priority applications.

Android pairs each application to a unique UID [4] allowing efficient matching of Android Application to sockets. If the packet is deemed to be high priority, the driver's Adaptive PSM traffic counter is updated. This allows traffic only from High priority applications to trigger the WiFi driver to switch to CAM.

Outbound Packets: When an outbound packet is sent by an application, the packet is intercepted before being processed by the networking stack. As with inbound packets, checks for socket validity and valid UID are also done. We then rewrite the IP header by setting the TOS bit and recalculate the IP header checksum. Then the packet is returned to the networking stack for normal processing. When the packet eventually arrives at the

WiFi driver, if an IP header with the TOS bit set exists, the Adaptive PSM traffic counter is updated.

The UID validity check is performed to check the priority of the userid of the socket's owning process. The High-Priority list contains userids of high priority applications. The High-Priority list is updated over a netlink socket by the Application Priority Manager.

To summarize, the SAPSM Core kernel module permits only high priority packets to switch the WiFi driver into CAM. The WiFi driver looks only at the high priority traffic counter to adaptively switch to CAM. This design allows both traffic types to occur simultaneously, since only the high priority packets can increment the high priority traffic counter.

3.2.3 Application Priority Manager

To facilitate non-technical users setting priority for each application, we design the Application Priority Manager (APM) that is implemented as an Android service. When the WiFi interface is active, it polls the Kernel using the TrafficStats API for all available per application statistics as described in the implementation section. The TrafficStats API provides us with the total amount of bytes each application has transmitted and received at a given time. By polling every few seconds (one second in our evaluation) we can determine the data rate that each application is using. Each individual poll performs a low impact atomic read() from the /proc file system which has a low energy consumption. By polling every second we can determine the current data rate.

Through the TrafficStats API, APM collects the following four statistics. *RXBytes*: the total received bytes by the WiFi driver. *TXBytes*: the total transmitted bytes by WiFi driver. *RXRate*: receiving data rate in KBytes/sec. *TXRate*: transmitting data rate in KBytes/sec. For each application, *RXBytes* and *TXBytes* reflect the total traffic while

RXRate and *TXRate* reflect instantaneous traffic. These four statistics together capture each application's ingress and egress traffic.

With the collected information of each application's WiFi usage, APM uses an offline-trained classifier (trained through our user study data detailed in the next section) to classify each application into either high priority or low priority. While this information is being collected each application is set to low priority. We select low priority by default. This allows users to save energy on newly installed applications during the data gathering phase. If the latency introduced by PSM noticeably places the usability of an application in question, the user can manually set the application's priority to high.

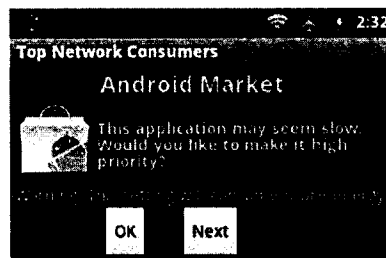


Figure 3.4: The pop-up window to assist user decision.

The application priority decision is then offered for confirmation to the end-user. While this process could be completely automated, we require confirmation from the end-user because this decision can impact the battery life of the device. Also, this feedback can eventually be used to train an individualized classifier which we reserve for future work. If an application is classified as high priority, APM pops up a window and asks the user whether this application should be set to high priority (Figure 3.4). APM then stores the user's decision in a database and updates the SAPSM Core kernel module with the userid of the application in question.

In APM design, we choose Support Vector Machine (SVM) to differentiate applications into high priority and low priority based on the collected information of each application's WiFi usage. SVM is one of the best classifiers and has been successfully applied

in many real-world classification problems, including text categorization [34], image recognition [52], hand-written digit recognition [20], and bioinformatics [33]. In general, SVM has four advantages over other classifiers [13] [21]: (1) the optimization problem involved in SVM is a convex optimization problem, whose local solution is also a global solution; (2) it is able to achieve high accuracy with a relatively small number of training examples; (3) it scales well with data dimensionality; (4) it is fast to execute at runtime. These four nice properties of SVM make it a perfect fit for our Smartphone application priority management.

3.3 Implementation

We implement SAPSM on the Sprint HTC Hero [63]. The phone comes with the TI 1251 WiFi chipset which is capable of 802.11 b/g. The driver is freely available and is part of the Android Linux Kernel tree. The Android platform is a natural choice because the source code is freely available. The implementation consists of the SAPSM core system, implemented completely at the kernel level and the Application Priority Manager, which is developed as an Android application.

SAPSM Core System. The SAPSM Core system is implemented as a Linux kernel module and is dependent upon the WiFi driver, also implemented as a module. Android loads and unloads the WiFi driver on demand. The user has the ability to load and unload the module at will. The SAPSM core design requires access to the WiFi module to update the traffic counter. This causes dependency issues if the WiFi driver needs to be unloaded. To address this problem, we implement (un)register() functions in the SAPSM module and export them so that they are available to the WiFi driver. When the WiFi driver is loaded it registers the UpdateTrafficCounter() function with the module. When the WiFi driver is unloaded, SAPSM is notified with the unregister() function. The SAPSM kernel module is loaded at boot time to avoid any further dependency problems.

We listen on a netlink socket within the SAPSM module. In order for the Application Priority Manager to connect to the netlink socket, we require the use of raw sockets which requires root access. Since Android applications do not run with root privileges for security reasons, we developed a “netlink manager” system service with root privileges. “netlink manager” listens for packets with a FIFO socket interface and then relays the packet through the netlink socket to the SAPSM kernel module.

We use the Linux Netfilter API [48] for packet interception in the SAPSM module which can be used to register a hook for inbound and outbound processing.

We store the list of high priority applications in a linked list that is kept persistent after a system reboot. The Application Priority Manager maintains an internal list of high priority applications for persistency. When the system is rebooted, the list is pushed to the kernel module through a netlink socket.

Application Priority Manager. The Application Priority Manager uses the Android TrafficStats API for the periodic check of network statistics per application. The TrafficStats API retrieves information from the Android specific location of `/proc/uid_stat/%UID/` directory. Each time a packet is transmitted or received, this `/proc` directory is updated on a per user basis. This allows a detailed snapshot of each application’s network usage. The Android system supports both UDP and TCP packets per application [5].

3.4 Evaluation

In this section we evaluate the SAPSM solution by answering the following questions: (1) *Do low priority applications save energy over high priority applications?* We address this by measuring the power consumed by Adaptive PSM, static PSM and SAPSM while conducting the load tests explained in the motivation section. (2) *How does the SAPSM solution save energy with a typical use case?* We address by comparing energy use of

low priority applications using SAPSM and those same applications with Adaptive PSM.

(3) *Does general networking performance suffer for applications placed into high priority?*

We determine that the SAPSM implementation does not impede high priority applications by comparing Adaptive PSM to SAPSM high priority using a networking benchmark application.

3.4.1 Evaluation Method

For performance testing, we use an off the shelf access point and router. A laptop is configured with to the router via Gigabit Ethernet. The Smartphone is connected to the router via WiFi.

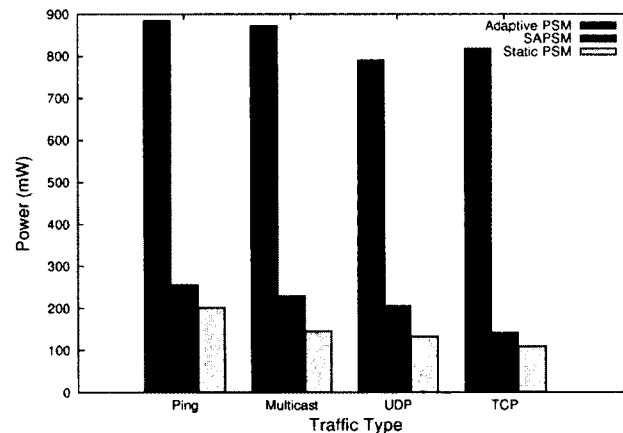


Figure 3.5: Comparison of power consumed from ingress traffic with no listening socket.

We use the Monsoon Solutions Power Monitor [44] to measure the energy consumption on the Hero Smartphone. The Power Monitor is configured by blocking the positive terminal on the phone's battery with electrical tape. The voltage normally supplied by the battery is supplied by the Power Monitor. The Monsoon records voltage and current with a sample rate of 5 kHz. We disable all radio communication except for WiFi.

3.4.2 Low Priority Application Behavior

In this section we evaluate SAPSM behavior with low priority applications. We first evaluate behavior with network traffic with no listening socket (Multicast,UDP, TCP, ICMP). Next, we experimentally determine what point is it expedient to switch to CAM if the goal is to maximize energy savings.

3.4.2.1 Traffic with no listening socket

In this section we evaluate the behavior of SAPSM, Static PSM and Adaptive PSM when subjected to traffic not associated with a listening socket. We repeat the load tests described in the motivation section. The applications were placed in low priority for the SAPSM case. All categories of traffic were transmitted at 20pkts/sec for 30 seconds.

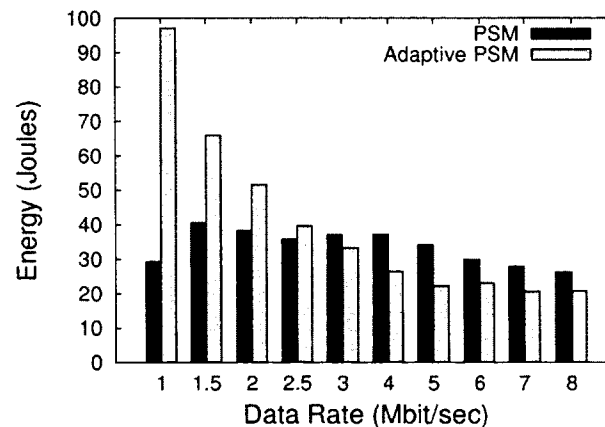


Figure 3.6: Energy sdf inversion: 10MB file downloaded w/varying data rates.

Adaptive PSM switches to CAM for the duration of the test due to the high traffic levels and results in significantly higher power consumption (340%) compared to SAPSM as shown in Figure 3.5. Adaptive PSM has no way to distinguish unwanted traffic from necessary traffic. This test shows the potential for unnecessary excessive power consumption. Since traffic not associated with a listening socket is treated as low priority

traffic by SAPSM, SAPSM is able to save significantly more energy than Adaptive PSM. SAPSM has an overhead compared to Static PSM of approximately 20% due to the listening socket check we perform on each packet. A faster listening socket check could improve these results. If all applications were low priority, Static PSM would be a good fit. However, in reality there is a mix of high priority and low priority applications which Static PSM is not able to address.

3.4.2.2 Low Priority Energy Inversion

In this section we determine experimentally the data rate for which static PSM network traffic ultimately consumes more energy than CAM. To conduct this test, we configure traffic shaping on a web server. Traffic shaping permits us to accurately limit the data rate for files downloaded from the server. We limit the data rate ranging from 1Mbit/sec to 8Mbit/sec. Our results, which also coincide with [35], show that as the data rate exceeds 3Mbit/sec, a PSM energy inversion occurs: static PSM consumes more energy than Adaptive PSM. For data rates less than this threshold the energy savings can be significant. As shown in Figure 3.6, data rates of 1Mbit/sec potential energy savings of over 500% are possible. Exceeding the 3Mbit/sec threshold for sustained periods with Internet traffic is unlikely. As shown in [22], WLAN data rates typically far exceed that of WAN connection speeds. This shows that low priority applications can achieve significant energy savings, especially when the data rates are low.

3.4.3 Energy Savings of Typical Applications

In this section we evaluate the energy use of several typical applications which consume a significant amount of network traffic. The applications we selected are a *streaming audio* application that allows users to stream audio over the Internet, an *offline map*

application which downloads in-advance maps of a new area you are traveling to with limited network coverage and an *RSS reader* application that retrieves RSS feeds from the Internet and caches them on the SD card. Finally, we evaluate *social networking applications*, including email, Facebook and Twitter, running in the background while the screen is off.

After each application is installed, we perform the following steps. First, we allow the application to run for approximately 10 minutes. During this time the APM gathers each application's network statistics described in the design section. Next, APM classifies these measured results with the classifier trained by the user study data. After running this process for these applications that we selected, we discovered that the RSS reader and the offline map applications are correctly classified as low priority. The streaming audio application was incorrectly classified as high priority.

Further investigation into why the streaming audio application was incorrectly classified produced a surprising result. We would expect the application to have a higher receiving data rate and a very small transmitting data rate since the primary function of the application is to stream audio from a remote server. However, we discovered that RXrate and TXrate are identical. It is not clear why the application transmits and receives simultaneously. This is most likely why the classifier incorrectly classified this application. In this case we manually set the application to low priority.

Each application's behavior is compared with Adaptive PSM and low priority SAPSM, Since the applications we selected were determined to be *low priority*. The Adaptive PSM results indicate the default behavior on Android Phones without SAPSM enabled. Figure 3.7 shows the comparison. The energy savings range from 13% for social networking applications up to 56% energy savings for the RSS reader application.

Streaming Audio. The streaming audio (XiiaLive) application has a wide selection of streaming radio stations where users can play different music styles. This application is

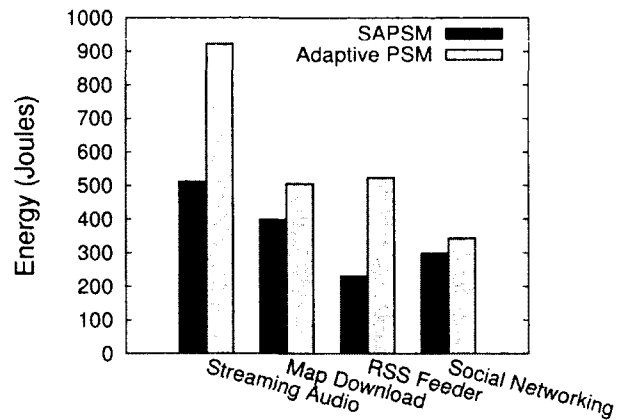


Figure 3.7: Application energy comparison.

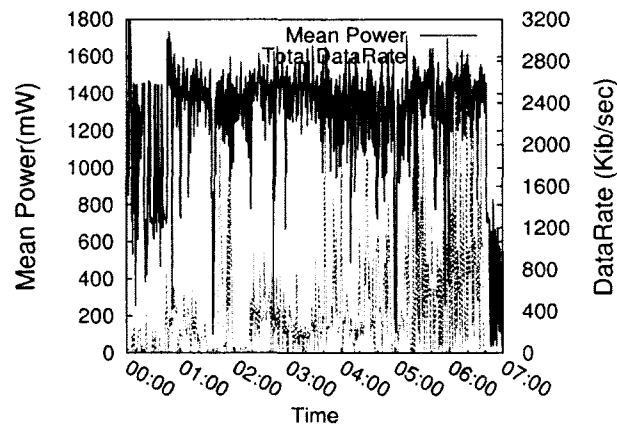


Figure 3.8: RSS Reader with Adaptive PSM

very popular with millions of installations. We select a station at random with a 128kbps stream. The added delay of PSM does not affect the quality of the playback since the application is able to buffer the audio stream as noted in [1]. We play the same audio stream for 10 minutes. SAPSM saves 44% energy compared to Adaptive PSM. This kind of traffic is clearly low priority; there is no noticeable effect if this traffic runs in CAM or in PSM. SAPSM is able to make this distinction over Adaptive PSM and save considerable energy in the process.

Map Offline Download. The offline map application (MapDroyd) has an extensive collection of free maps that can be downloaded. For this test we download a map of a US

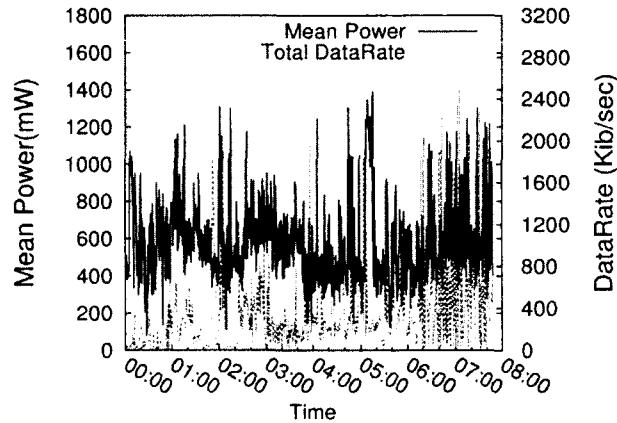


Figure 3.9: RSS Reader w/ low priority SAPSM

state which is 70MB. After running the test several times, SAPSM saves 18-22% energy compared with Adaptive PSM. As we described previously, since the data rate is small approximately 1.5Mbit/sec SAPSM clearly save more energy at the expense of taking more time. Since delays of this type are acceptable, SAPSM clearly wins in this case.

RSS Reader. The RSS reader application (RssDemon) has a default install of 16 feeds dispersed over categories of general interest. These include sports, news, technology and Entertainment. Periodically, feeds are updated by retrieving the latest from the Internet. Figures (3.8 and 3.9) show the plot of power consumption and combined data rate during an update. The RSS reader downloads a number of XML RSS files at the first part of the update period, the first 3-4 minutes. At approximately 4.5 minutes several larger files are downloaded. The *UP* threshold is triggered quickly at the beginning and stays for the duration of the test. SAPSM saves 56% energy over Adaptive PSM in this case.

Combined Social Networking. Applications using push technology are often run in the background. Even with the screen off, new data is actively pushed to applications. This type of application is clearly low priority. We select the following popular applications: Gmail, Facebook and Twitter. For the test we subscribe to the Linux kernel mailing list

and subscribe to a number of popular Twitter feeds. Facebook, which does not use push-based notifications, is set to update every 30 minutes (the lowest setting). We conduct a test in the early morning for one hour. During this period we received a total of 12 tweets and 30 emails while observing no Facebook activity. We record the timestamps when each email and tweet is received and then replayed each for comparison.

The Adaptive PSM results show that the data rate during this test is quite low yet high enough to switch the radio in to CAM at several occasions. SAPSM results show that the radio stays in PSM the entire time. Even with light traffic, SAPSM results in a 13% energy savings over Adaptive PSM.

3.4.4 SAPSM High Priority Networking Performance

While saving energy is important, having solid networking performance for high priority traffic is equally important. We use Netperf [49] to evaluate the performance of SAPSM. We install the Netperf server component on the laptop and the client component on the phone. We measure the maximum sustained TCP throughput rate that the device can tolerate.

We repeat the Netperf test 10 times and the results are shown in Table 3.2. SAPSM incurs a 3% general networking performance penalty compared to Adaptive PSM. The critical section is the lookup function as described in the implementation section. The use of a better data structure, like a hash table, for UID lookup would make a small improvement. To address this issue we test a version of SAPSM with the lookup disabled, which is labeled SAPSM1 in Table 3.2. In this case the data rate shows a slight improvement, very similar to the Adaptive PSM result.

Additionally, we also perform another test where we download a 10MB file from a local web server and record the time. This test, repeated 10 times per implementation, shows

no significant differences per implementation. Interestingly, No improvements are seen from the SAPSM1 results, suggesting that the optimization results seen with Netperf are of negligible impact outside of extreme performance tests. Overall, SAPSM high priority performs on-par with Adaptive PSM.

| Type | Throughput | | Time Delay | |
|--------------|------------|--------------|------------|--------------|
| | (u) | (σ) | (u) | (σ) |
| Adaptive PSM | 8.34Mbps | 0.25 | 14.83s | .50 |
| SAPSM | 8.08Mbps | 0.10 | 14.70s | .47 |
| SAPSM1 | 8.48Mbps | 0.13 | 15.04s | .50 |

Table 3.2: SAPSM high priority TCP Performance results with associated standard deviation.

3.5 User Study

| Major/Minor | Count |
|---------------------|-------|
| Computer Science | 7 |
| Economics/Finance | 1 |
| Government/Math | 1 |
| Kinesiology | 1 |
| Math/Physics | 1 |
| Neuroscience | 1 |
| Rhythmic Gymnastics | 1 |
| Sociology | 1 |

Table 3.3: Majors.

In order to train an SVM classifier and also to evaluate whether it is able to provide accurate classification results for different users, we conduct a user study. A random mixture of fourteen technical and non-technical users participate in the user study. In the study, each application is set to low priority, which is the default WiFi configuration for each application in SAPSM. Each participant is required to use each of six applications for ten minutes. We selected a number of applications that have a diverse array of network behavior. We select applications that are interactive (Android Market and the Android web browser), while also addressing those with a low degree of interactivity (Tanks and Turrets game). Social networking applications with ambiguous priority depending

on usage are also selected (Gmail, Facebook and Twitter). These applications are ambiguous because, on the one hand, they can be used interactively, i.e. clicking on every link or, on the other hand, run in the background non-interactively.

During the user study, we assign each participant a set of instructions that they should follow. Each phone was configured with static PSM enabled. As mentioned in the background section, static PSM adds approximately 100-300 ms of network delay or latency. We vary the degrees of interactivity among all participants' instructions and ask them to determine if they feel the observed latency is acceptable. The answers from participants are used as labels for the applications. If a participant feels the observed latency is unacceptable, this application is labeled as high priority. Otherwise, the application is labeled as low priority, which means that any perceived latency does not impact the users experience with the application. At the end of the user study, we do a brief survey to collect participants' basic information such as their majors and their experience about Smartphone. The major distribution of all the participants is summarized in Table 3.3.

In the background, APM collects four statistics (described previously in the Application Priority Manager) that measure WiFi usage for each application. APM groups the statistics for each application and extracts a set of features: (i) the maximum, mean, median and variance of RXRate and TXRate; (ii) RXBytes, TXBytes as well as the ratio of RXBytes / TXBytes. These features measure different statistical characteristics of WiFi usage.

RXBytes / TXBytes can reflect an Application's network interactivity much better than non-network features like the touch screen rate. If a user is regularly touching the screen, this does not always mean that network traffic is occurring; video games for example are very interactive with respect to the user and the screen, but typically non-interactive with respect to the network.

The accuracy of an SVM classifier depends on the input features. We use the

Sequential Forward Search based feature selection algorithm [29] to select the best features from these 11 features. The algorithm returns two optimal features - the maximum and mean of RXRate.

These two features are meaningful in terms of predicting whether any perceived latency added to a given application by static PSM is noticeable by end-users. The reasons are: (1) comparing to TX related features, RX related features are more important because Smartphones are more receivers rather than producers of information. The time spent on receiving is usually much longer than that of transmitting; (2) the maximum of RXRate reflects the participant's experience at the network traffic peak and the mean of RXRate reflects the participant's long-term network experience.

With these two features, we select the optimal parameters for the SVM classifier from the user study data following the routine of 6-folds cross validation to avoid overfitting and estimate the runtime accuracy. In each round of cross validation, data is divided into 6 subsets, 5 of which are used for training and the remaining 1 is used for testing, so that the testing data is different from the training data. This process is repeated 6 times and each of the 6 subsets is used exactly once as testing data. The accuracy is the average accuracy over 6 rounds [13]. The parameters with the maximal accuracy during cross validation are selected. Then, the resulting classifier is trained with the selected parameters and achieves 88.1% accuracy.

Figure 3.10 describes the classification results. There are three zones. The top and bottom zones reflect low priority applications. While the middle of the figure reflects high priority applications.

First, from the classification results, we observe that all participants label the Tank game as low priority. It is because the Tank game is offline except for periodically fetching advertisements in a background thread. The SVM classifier accurately classifies all the Tank game data points in the user study data as low priority.

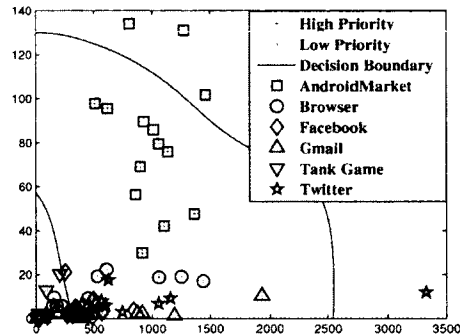


Figure 3.10: Application classification result.

Second, we observe that the Browser, Facebook, Gmail and Twitter are ambiguous in terms of user-defined priority. This is due to different users have different degrees of interactivity. For instance, repeatedly clicking on a URL leads the user to label the browser as high priority while simply updating the Twitter status leads the user to label the Twitter or Facebook as low priority. For these applications, the SVM classifier is able to accurately classify 47 out of 56 user study data points.

Third, the most interesting observation is that some applications with either high maximum or high mean values of RXRate are labeled as low priority. For instance, three users label the Android Market as low priority as depicted in the top zone of Figure 3.10. These applications contradict the common intuition that an application with a high RXRate will have unacceptable latency and should be labeled as high priority. One possible explanation is the network data is received in a background thread so the latency is not noticeable. The SVM classifier is robust to these applications and accurately classifies all of them.

To sum up, the SVM classifier achieves satisfiable classification accuracy for different users with different backgrounds. Integrating with the SVM classifier, APM is adequate to assist even non-technical users in configuring application priority.

3.6 Conclusions & Future Work

Effective WiFi power management is an important issue. We have shown that by labeling each application with a priority, the overall system is able to save energy by allowing only the traffic from high priority applications to impact WiFi power management behavior. By using a user study trained classifier we demonstrate that even non-technical users can effectively label the priority of applications. Our evaluations of real scenarios show energy savings from 13% to 56% depending on the application.

The user study demonstrates that the SVM classifier used by APM can achieve satisfying classification accuracy. In future work, we plan to develop an optional individualized classifier. Newly acquired data such as personal application priority decisions and corresponding network traffic measurements can be offloaded to a server and can be used to adapt to an individual's usage patterns. When the server receives a retraining request, the server will train a new classifier which can then be used to replace the default classifier that is trained through our user study. Finally, we plan to extend SAPSM to other smart handheld devices.

Chapter 4

Mixed Radio Data Driven Energy savings

4.1 Introduction

Energy efficiency on Smartphones is a driving factor because of limited battery life. Due to the always-connected nature of Smartphones, the efficiency of Internet access is particularly important. Wireless networking choices for Smartphones typically consist of either WiFi or 3G/4G networking. When the mobile device is in a fixed location such as a home or business, WiFi is faster and more energy efficient than 3G networking [56]. Additionally, mobile plans typically place data usage limits.

Although WiFi networking is more energy efficient than 3G, considerable research has been done to make it more efficient. The WiFi standard includes Power Save Mode (PSM) which saves energy by sleeping during idle periods. Then periodically the radio wakes up to detect if packets are waiting at the Access Point (AP). While this is generally energy efficient, the buffering of packets at the AP adds additional delay. Previous work includes enhancing sleep periods during periods of inactivity [38] [53]. While this body of

work has made significant progress, still the high power requirements for the WiFi radio still allows room for improvements, especially for low bitrate traffic.

WiFi is more efficient on a per-bit basis [51] than other radios such as Bluetooth. An obvious question arises: shouldn't we always use WiFi? When the WiFi Radio is predominately idle, ironically, this is also when it can be most inefficient. WiFi drivers on Smartphone's come equipped with Adaptive PSM [53], the ability to switch the current power mode between sleep and active based upon the current data rate. When the WiFi radio data rate is high enough where it triggers the Adaptive PSM threshold, it will switch from sleep to active mode. Active mode in-turn can consume up to 20 times more energy than Sleep mode when idle [54].

Others investigate the use of multiple radios [51] [39] to be more energy efficient. For low bitrate traffic the low power radio can be used, then when network traffic conditions change, the schemes can adapt to the other radio. A major challenge to this approach is that the act of switching between radios can be an expensive operation. In [51], only a single radio is powered on at the same time. However, when network conditions change, the other radio has to be powered on and configured which can cause a delay of several seconds, consumes extra energy each time a switch occurs and also terminates all active sockets. A key challenge is to allow the use of multiple radios without disrupting existing socket connections and allow rapid adaptation to changing conditions all while saving energy.

In this work we also investigate using Bluetooth and WiFi with the goal of saving energy. In order to address existing challenges of previous work, we focus on the ability to switch between multiple radios without disrupting existing socket connections and have the ability to switch between radios immediately. We do this by implementing our solution at the MAC layer. Recent developments in low power WiFi radios and Bluetooth allow us to keep both radios active at the same time. The inactive radio is kept in the low

power mode.

Applications for which high network throughput is desirable should use WiFi due to its superior speed (802.11n handles speeds of 300Mb/sec). But for *typical* Smartphone Internet traffic is high speed always necessary? According to a recent study [24], data rates for mobile video are considerably less than 1Mbit/sec. In fact fast data speeds may not be as common as might be expected. Certainly for LAN applications high throughput is expected and should be used. Internet connections, however, are magnitudes of order slower than the WiFi router speed, throttled by [22] slower Internet routers and the broadband connection speed.

Bluetooth, explained further in the background section, has a max data speed of about 2-3Mbit/sec and a range of between 10-50m. Constant bitrate traffic is a special case where the WiFi connection is over utilized. The WiFi radio has to stay on for the duration in order to minimize latency. However if the bitrate is also below the maximum Bluetooth speed, then Bluetooth is more energy efficient while maintaining an acceptable latency. For certain types of traffic Bluetooth is a viable alternative.

Significant WiFi network traffic exists that under utilizes the WLAN connection. Are there other alternatives that will not impede network performance and still save energy? While WiFi PSM is energy efficient by sleeping during idle periods, the added latency is unacceptable for many applications. Bluetooth is an acceptable alternative and Bluetooth devices are present in virtually all Smartphones. Although Bluetooth can handle a much smaller data rate than WiFi, Bluetooth power consumption even in its highest power state is significantly less than WiFi in Active mode.

To address these concerns we introduce Bluesaver: A Multi PHY Approach to Smartphone Energy Savings. Bluesaver combines Bluetooth and WiFi together both at the Phone and at the Access Point. When the Phone is in range of the Bluetooth radio on the AP it can efficiently send and receive packets over Bluetooth. When out of range

or the requirement for a higher data rate is requested, the phone uses WiFi. Bluesaver is implemented on a Motorola Razr Android phone and can save 25% energy over existing solutions.

The individual traffic patterns of Smartphones are difficult to predict. While some applications such as Skype and web-browsing may be ideal for Bluetooth, other applications such as Youtube clearly benefit from WiFi. If the goal is to save energy, how can you switch between the radios with minimal impact to the user?

To address this problem, Bluesaver provides a mechanism to seamlessly switch between WiFi and Bluetooth without impacting current applications in such a way that will save energy. Therefore, if current network traffic can be more efficiently transmitted over Bluetooth, then the Smartphone can seamlessly switch between WiFi and Bluetooth for best efficiency.

4.2 Background and Motivation

In this section we cover the background section specifically related to the Bluesaver architecture. Since Bluesaver covers both WiFi and Bluetooth, we give a brief overview of both WiFi power saving mode (PSM) and Bluetooth. While WiFi PSM does save energy, it has the downside of adding considerable delay. Bluetooth has the advantage of a low power solution when low data rates can be used. We show that combining elements from both Bluetooth and WiFi PSM, we can potentially save more energy while minimizing delay.

4.2.1 WiFi PSM

WiFi PSM is part of the original 802.11 spec first standardized in 1999 [31]. WiFi clients connecting to an Access Point (AP) can negotiate a low power state. In this way, the client's radio will remain off, while incoming packets are buffered at the AP. At pre-determined beacon intervals, the client will poll the AP for any queued packets. The AP will respond with the queued packets. While this approach works well for power saving applications, it adds an approximate 100-300ms of delay caused by the buffering of packets during the beacon intervals. When a packet is buffered at the AP, the AP will set the TIM bit. When the TIM bit is set, the client knows to poll the AP for the buffered packets.

Recently, there have been several alternatives to PSM. Most deal with switching between Active mode and PSM. Active mode requires the WiFi radio to remain active, requiring significantly more power. Adaptive PSM as described in [53], [59] use an approach to adaptively switch to active mode based upon the observed data rate. When the data rate drops, the WiFi radio switches back to PSM. The "switching" occurs by sending a NULL management frame from the client to the AP. The client sets the power management bit according to whether active or PSM mode is desired. If switching from PSM to active, the buffer on the AP is first cleared using a PS-POLL management frame, also initiated by the client.

As shown in figure 4.1, still the underlying trade-off with latency and power remain as WiFi research challenges. The data in the figure published in [54] shows that WiFi is suited for high speed operations. Network traffic with moderate data transfer speeds will either suffer high latency or consume extra power. Therefore, it is logical to investigate the use of other means of transmitting data that falls into this category.

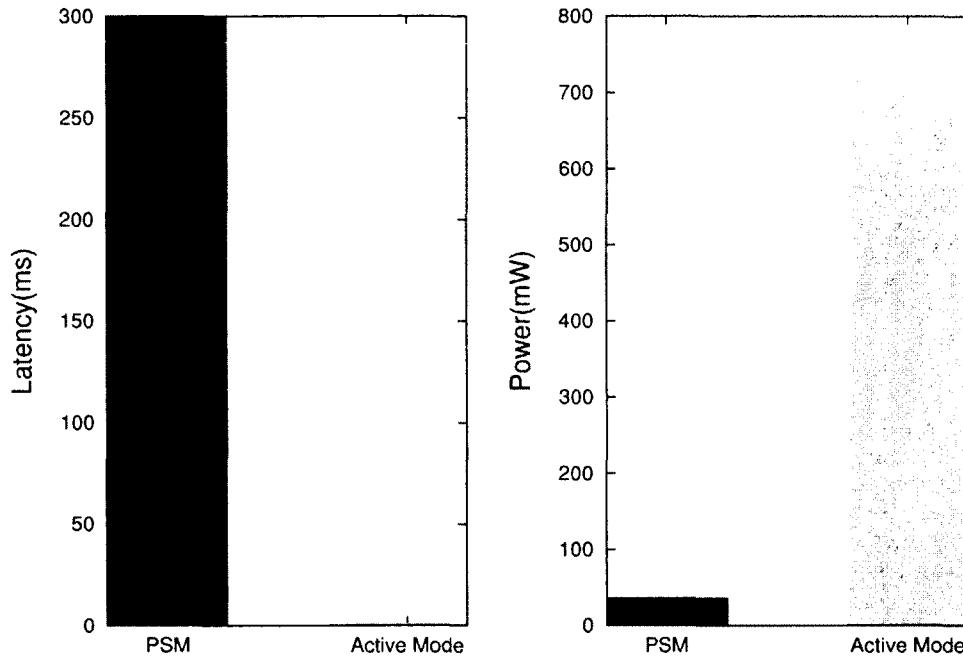


Figure 4.1: WiFi Energy Latency tradeoff. Measurements reflect recently published measurements on Smartphone WiFi PSM.

4.2.2 Bluetooth

Bluetooth, in contrast to WiFi, is designed with low energy and small distance in mind. Data Rates have an upper bound between 1Mb/sec to 3Mb/sec with version 2.0 [14] enhanced data rate. Additionally the range is limited to around 50m with the BT 4.0 specification, compared to 100m WiFi range, while older versions are limited to approximately 30m. Bluetooth is effectively used for a variety of close range applications such as streaming audio to headsets to peer to peer data sharing applications.

One disadvantage of sharing peer data via Bluetooth is that the slower speed can take a significant amount of time when large files are transmitted. In order to address this concern, the 3.0 specification has included the High Speed (*HS*) [15] specification. This specification allows files to be transmitted at high speed by utilizing the high speed capabilities of a co-existing WiFi card. The connection is established with Bluetooth and

then the file can be transmitted to the peer (which also must support HS) via WiFi.

The 3.0 *HS* specification provides the capability for Bluetooth connections to ultimately be more energy efficient for the transmitting of large files. WiFi is more efficient at transmitting large files, since the higher throughput of WiFi can send files in less time. When a constant stream based connection such as a Bluetooth headset for streaming audio, Bluetooth is more efficient than attempting to use the higher energy cost of WiFi.

WiFi has nothing to compare with the *HS* option. When transmitting data over WiFi when sending large files over a LAN connection, WiFi is an ideal solution. However, when it comes to streaming audio or other applications that require small data rates, WiFi is forced to use the high speed, high power radio for this task.

4.2.3 Motivation

In this subsection we investigate cases where WiFi alone can use some improvement from an energy savings perspective and look to see how prevalent such cases are. Specifically, we examine cases where one of these cases exist:

- bandwidth is limited.
- streaming applications such as video or audio.

In Figure 4.2, we see the bandwidth required over time as a Youtube video is watched on a recent Motorola Android smartphone. As can be seen, the bandwidth tends to spike and then quickly drop off. Static video content, typically delivered in chunks can take advantage of the speed and efficiency of a WiFi connection. In this case, WiFi handles the bursty nature of static video content providers such as Youtube efficiently. The WiFi driver is suited to quickly switch to Active mode and download the next available

chunk of video. As can be seen the broadband connection allows connections of up to approximately 3Mb/sec, and so WiFi is a good fit.

In Figure 4.3 we see a different story. In this case, instead of bursty traffic, we see the bandwidth of a constant bitrate live streaming Skype call. Figure 4.3 shows a trace of a Skype audio call between two users. In this case, the trace shows a constant bandwidth of slightly less than 100KB/sec which ends approximately 2.5 minutes later. The bandwidth is high enough that the WiFi driver will switch to active mode, thus minimizing latency. However, due to the higher power requirements of the WiFi radio, the lower bandwidth requirements of this particular applications could just as easily be fulfilled by another radio such as Bluetooth. This wastes unnecessary energy and can use some improvement.

Figure 4.4 shows a Skype video call that was set to high quality and again is at a constant bitrate. In this case, the bandwidth requirements are approximately ten times higher than the previous audio call slightly below 800Kib/sec. Note that since the constant bitrate never exceeds 1Mbit/sec, the WiFi radio again is under-utilized. Even at the highest available video streaming rate, Bluetooth is a viable alternative for video streaming. Bluetooth, with it's lower power draw, could be used to save energy. At the same time since the connection is below one megabit per second, the QoS requirements are not impacted.

As we have shown, for constant bitrate network traffic, WiFi is under utilized. In order to minimize delay, the WiFi radio must be kept active most if not all of the time. Therefore, the use of an alternative radio such as Bluetooth could easily be utilized to save energy.

For video streaming at HD the video streaming is peaked at 1.5 Mb/sec [62] which is available for premium members only; not available as an option for the Android client. The more likely high quality video streaming is at 500Kb/sec. Facetime has been measured [26] at less than 400Kb/sec. Clearly in these cases the WiFi radio is mostly idle.

Limited Bandwidth How realistic is the case of limited bandwidth for WiFi connections? Clearly WiFi Wireless LAN connections are getting anything but slower. WiFi 802.11n for instance supports speeds up to 300Mbit/sec. However, research conducted recently by Dogar [22] shows that the bottleneck is not the highspeed WiFi connection between the AP and the client, but rather between the AP and the Internet. Therefore it is entirely plausible to have a Smartphone with a 300Mb connection to the AP and a 1Mb connection to the Internet.

By finding opportunities where traffic patterns meet the criteria above where Bluetooth consumes less energy than WiFi, we can exploit these periods to save energy on Smartphones.

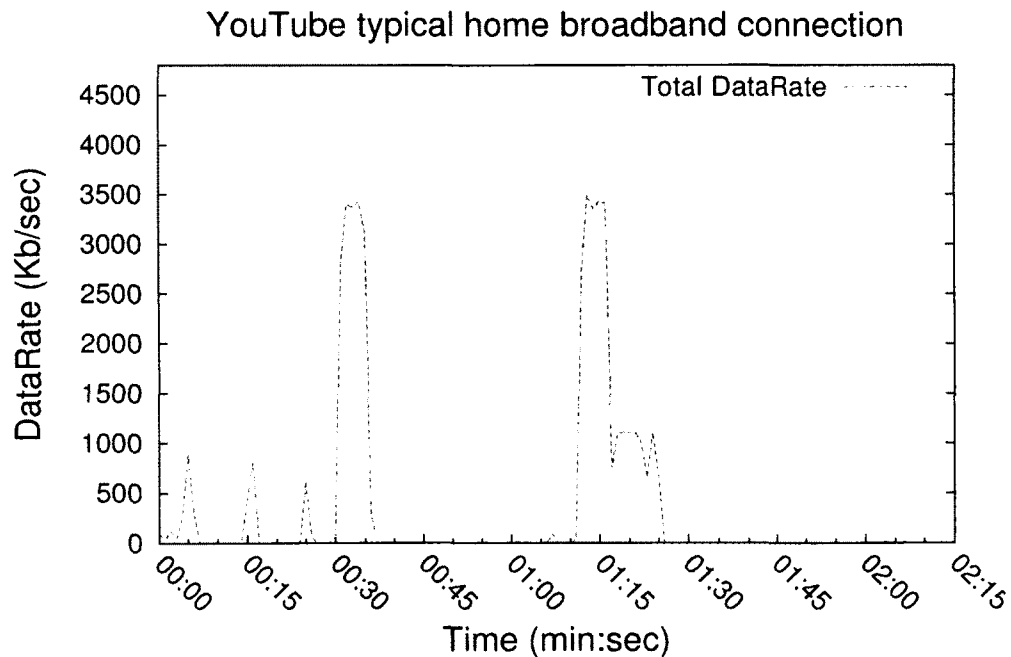


Figure 4.2: Youtube typical broadband connection

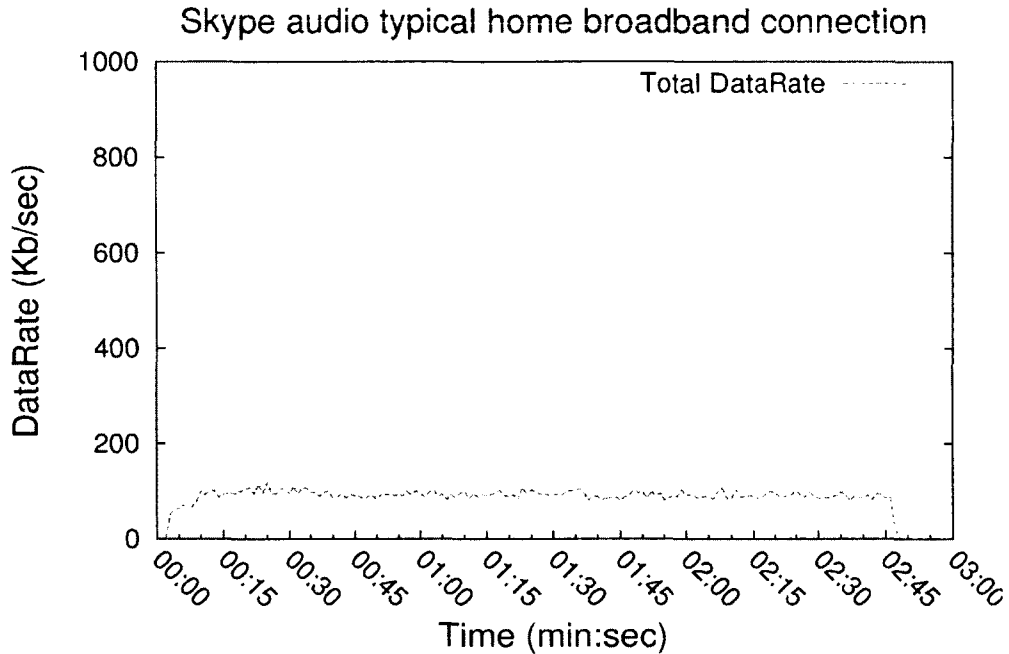


Figure 4.3: Skype audio typical broadband connection

4.3 Bluesaver Design

We have described the challenges facing WiFi clients. To address these challenges, we introduce Bluesaver: Multi-PHY approach to smartphone energy savings. In this section we describe the system architecture and discuss the design of the Bluesaver system. Bluesaver has been designed to function at the MAC layer. Both radios are kept on simultaneously. This way, packets can be sent either via Bluetooth or via WiFi. To save energy, both WiFi and Bluetooth connections are kept in a low power state when idle to save energy.

The Bluesaver design consists of a modified WiFi AP which also includes a Bluetooth adaptor. The client is an Android Smartphone which also includes WiFi and Bluetooth. The lab setup is shown in figure 4.5. While our lab setup is used with a Smartphone, Bluesaver can be used with any system that has multiple radios. Bluesaver could be easily extended to laptops and tablets.

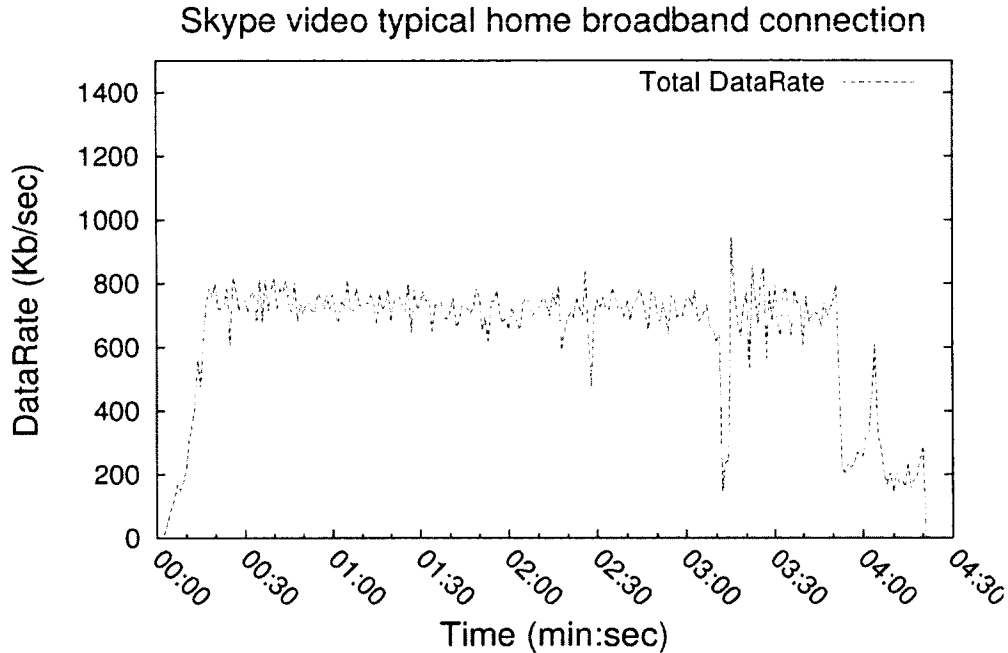


Figure 4.4: Skype video typical broadband connection

4.3.1 Architecture

The Bluesaver architecture is spread out between the client portion running on the Smartphone and the WiFi AP shown in figure 4.6. It is comprised of three main components: The Health Monitor(HM), The Bluesaver Connection Manager (BCM), and The Sending Decision Manager (SDM). All of these separate components used together are responsible for switching packets over the best available PHY interface.

The HM component is responsible for tracking the health of each Bluetooth connection. When a Bluetooth connection with a peer is established, the HM monitors traffic going through the device. Specifically, for each connection the HM component is responsible for monitoring the current data rate, connection status, packet loss and delay. Once this information is gathered, information can be passed onto the SDM in order to determine which interface the packet should be sent.

A crucial part of the HM component is the Bluetooth Availability Manager (BAM).

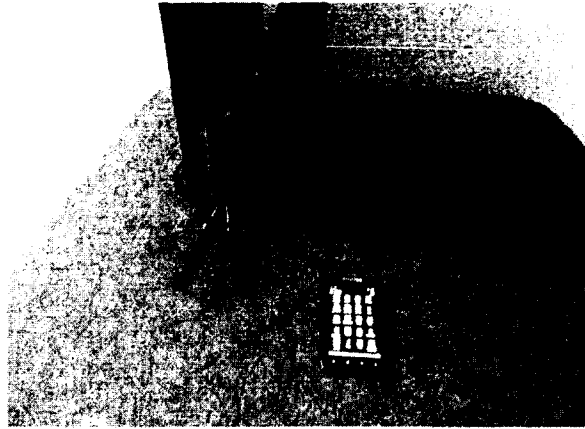


Figure 4.5: Lab Setup.

This component is responsible for checking the connection status of the peer. The BAM determines the health of the peer, using *I2ping* which is similar to ICMP ping but instead uses Bluetooth I2cap packets. To save energy, the BAM operates periodically, currently once per second and only when network traffic is observed.

The bulk of the HM operation occurs within an asynchronous timing thread that re-occurs every second. If a new packet has been transmitted in the past second, that is if any network traffic has been observed, the current connection quality and the current data rate are updated. If the data rate exceeds the threshold of what Bluetooth can handle (1.5Mb/sec) then *UseBluetooth* is set to *false*. Additionally, if the results of the RTT observations retrieved by the BAM using I2ping either fails (unable to connect) or shows an unacceptable latency (greater than 100ms), then *UseBluetooth* is set to *false*.

The HM notifies BAM to refresh the latest health statistics through a netlink socket. The BAM (which is running in userspace) sends an I2cap ping to the peer only when current traffic is detected. We use an I2cap ping operation because Bluetooth devices have I2ping operation enabled by default as part of the firmware. We send 4 packets and record the average RTT (the firmware of most devices closes the socket after 4 packets). If the delay is determined to be more than 100ms, then we assume that the connection is unsuitable, and BAM then sends a notification through the netlink socket. The BAM

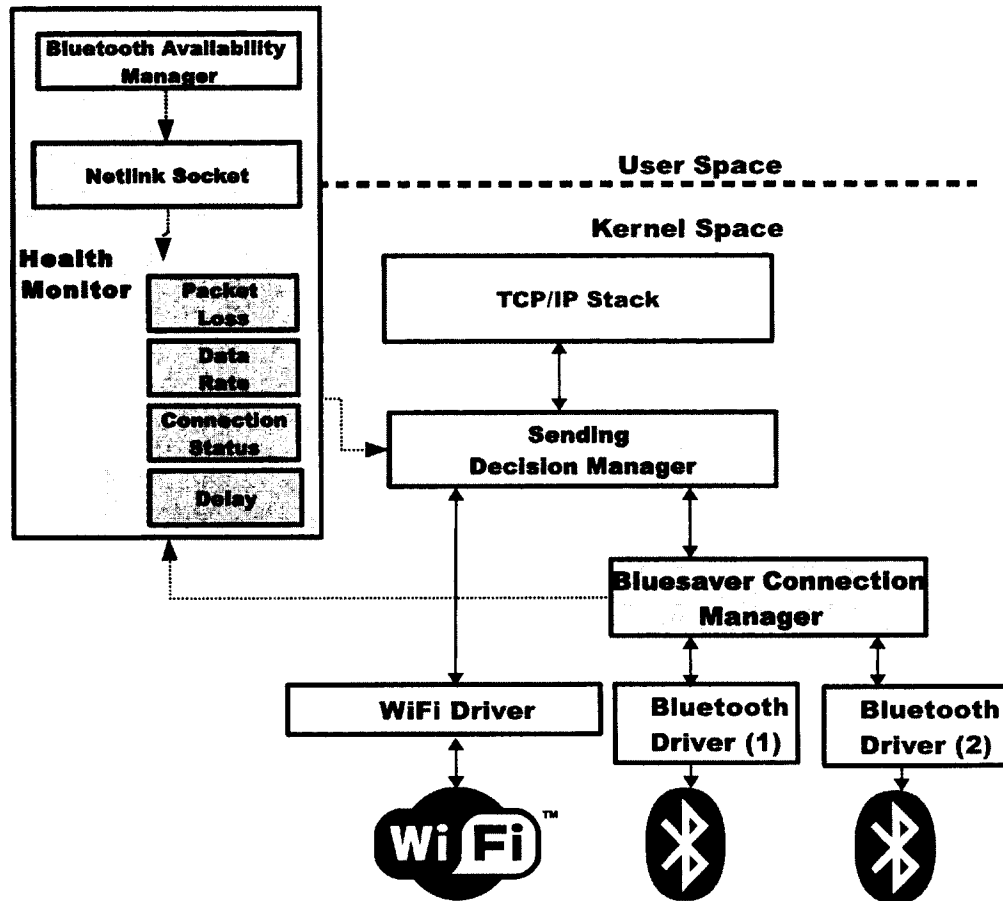


Figure 4.6: Bluesaver Architecture.

currently only runs on the AP.

When a packet is ready to be transmitted, the SDM determines which interface will be used. Figure 4.7 describes the interaction. When the host OS sends a packet, it will pass the packet onto the driver. Bluesaver will intercept that packet and determine which interface to use. When the packet is placed in the transmit queue, the PHY interface to use is determined via the *UseBluetooth* variable. Then the packet will be either transmitted through the WiFi driver or through Bluetooth.

Once these parameters have been obtained, a decision is made whether the connection is suitable for Bluetooth operation. If a threshold is crossed for either latency, packet

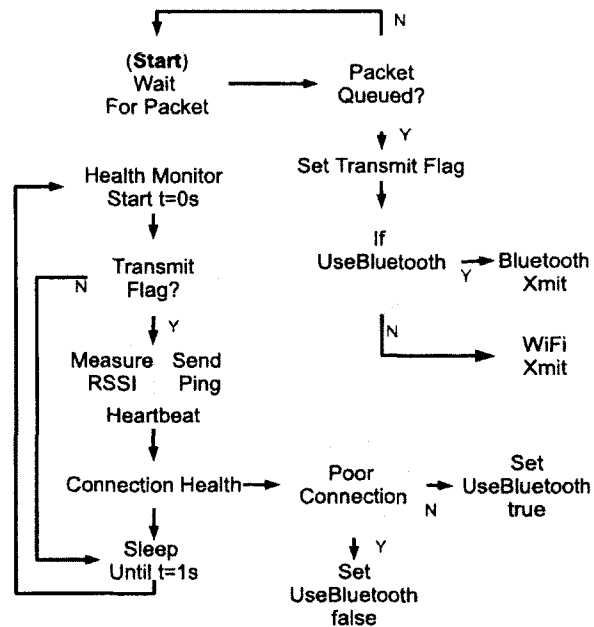


Figure 4.7: Bluesaver Design.

loss or data rate, the variable *UseBluetooth* is set accordingly. At this point when an outbound packet destined for the WiFi interface is queued in the driver transmit queue, the WiFi driver will either transmit the packet directly if set to *true* else the BCM will transmit the packet over the bluetooth interface.

The AP is responsible for checking the health of each client through the HM component as described earlier. When a decision is ultimately made to send via a specific PHY interface, an important consideration is to keep this decision in sync with the client. That is, when the AP sends packets over WiFi, the phone should also send over the same medium. In order to accomplish this, we have a much simpler design on the client. The client's initial setting is set to *UseBluetooth*. The AP will determine the overall health of the system using the method previously described and the appropriate PHY will be selected. On the client, when a packet is received on a different interface than what is expected, the local *UseBluetooth* will be set accordingly. For instance if the client is sending packets over Bluetooth, but then receives a packet over WiFi, *UseBluetooth* will be set to false.

The Bluetooth Connection Manager (BCM) is responsible for transmitting Bluetooth packets through the system. It does this by opening a Bluetooth l2cap socket to the peer. When l2cap packets are received over this socket, the packet will be passed to the host so that the host OS cannot differentiate it from a WiFi packet. In order to accomplish this, the Bluetooth MAC addresses are replaced with the WiFi source and destination MAC addresses in an 802.3 header that the host OS is expecting. When packets are transmitted over Bluetooth, they are taken from the WiFi driver transmit queue and sent over the l2cap socket.

On the AP, the BCM requires one Bluetooth adaptor for a single client. A single Bluetooth adaptor can also scale to multiple clients, since our implementation requires a separate l2cap port for each client. Depending on the traffic loads from each client, the combined bandwidth of multiple clients could overwhelm a single adaptor. In this case, additional Bluetooth hardware can be used to support multiple users. Our initial design is to dedicate a single Bluetooth adaptor to each client. In future, we plan to extend this to make this more extensible by extending the BCM to have multiple socket connections to the Bluetooth adaptors on the AP. Then utilize an appropriate load balancing scheme to determine which destination interface to which to transmit.

By operating at the MAC layer, Bluesaver is able to quickly adapt to adverse network connections and switch quickly between WiFi and Bluetooth.

4.4 Implementation

We implement Bluesaver on the the Motorola Razr [46]. The Razr comes with Android version 2.3.5 and Linux kernel version 2.6.35.7. The WiFi AP is implemented on a PC equipped with Ubuntu 12.04, an ath9k WiFi driver and an ath3k Bluetooth driver. The

implementation consists of a Linux kernel modules on both the phone and the AP. The Bluetooth Availability Manager (BAM) is implemented as a user level daemon process.

One of the challenges faced with the implementation is that the Smartphone comes with a locked bootloader. This makes it nearly impossible to modify the kernel. Therefore, all of our modifications had to be done within the confines of a kernel module. Due to this limitation, we were unable to access health related statistics from the Linux kernel from the Bluetooth device because the symbols were not exported. Also due to differences in the Linux kernel between the AP and the phone, the Bluetooth kernel API's were slightly different.

The BCM component requires modification to the WiFi drivers on both the phone and AP. When a packet is about to be transmitted over WiFi in the driver, we modify the transmit portion of the WiFi driver to check if the packet should be sent over Bluetooth. If so, the packet is placed in a transmit queue and transmitted over the l2cap socket. The receive functionality on the driver is not modified.

Phone Challenges:

Another subtle difference between the Android implementation and the AP is the issue of Wakelocks. The Android kernel supports the concept of entering a deeper sleep when a wakelock is not held. When packets are transmitted or received, we hold a wakelock. Each time a packet is received we set a flag. We have a timer that runs every second. If a packet is received during that time window, we hold a wakelock for one second. In this way, the maximum amount of time we hold a wakelock is one second when idle.

The client component is implemented with a queue data structure. When new outbound packets need to be transmitted, they are inserted into the transmit queue. A separate thread is run periodically whenever the queue length has at least one packet in it.

Bluetooth Availability Manager. The BAM is a userlevel process that is responsible for determining the status of the Bluetooth connection. Every 500ms the BAM sends a BT heartbeat packet to it's peer. If a heartbeat packet is not received within one second then the connection is bad. This could be caused from the phone being outside of the range of bluetooth. The connection status is then transmitted to the Health Monitor via a netlink socket.

BAM uses a *select()* loop to respond to incoming packets. In order to precisely send packets at a given interval, we use the Linux *timerfd()* system call. However, Android's bionic libc does not support this particular system call. Therefore, we had to explicitly add support for this system call with *system()*.

Bluetooth Notes. We made the best effort to obtain the highest Bluetooth throughput possible. Recall from the Background section, that Bluetooth supports up-to 3Mbit/sec. In our tests, we were able to achieve 1.7Mbit/sec, which is close to the theoretical maximum of 2.1Mbit/sec. We used l2cap based sockets with default options for the implementation.

4.5 Evaluation

In this section we evaluate the Bluesaver system. To evaluate the system correctly. We must show that it first saves energy over exisiting solutions. Second, we must show that Bluesaver can adaptively switch between between Bluetooth and WiFi due to changing network conditions.

This section is organized by our evaluation method followed by the energy comparsion section. Finally, we conclude with an evaluation of how Bluesaver responds to dynamic network conditions.

4.5.1 Evaluation Method

We implement and evaluate the phone component of the Bluesaver system on a Motorola RAZR [46] Android phone. The WiFi AP is implemented on a workstation running Ubuntu 12.04. The workstation is equipped with a Qualcomm-Atheros reference design PCI card [55] that includes both Bluetooth 4.0 and WiFi capabilities.

To measure the power consumption, we use the Monsoon [44] power monitor. The Monsoon bypasses the existing battery and provides power to the phone. It measures the instantaneous voltage and current with a sample rate of 5kHz. We can then determine the overall system power draw over a given time interval. In order to isolate the power consumption specifically to the test in process, we enable "Airplane" mode which disables all PHY interfaces. Then the interface that is about to be tested is explicitly enabled. Additionally, we make a best effort attempt to disable all services and background processes running on the phone during the test.

4.5.2 Energy Comparison

To assess the energy comparison between Bluesaver and Wifi adaptive PSM, we compare power consumption levels between WiFi and Bluetooth first for data rate throughput testing. Second, we compare the power consumption of video streaming at incrementally increasing data rates between the AP and the phone. In this section we are ultimately comparing the power and energy consumption of Bluetooth vs. WiFi. Although Bluesaver can handle both cases, it can really save the most power and energy when using Bluetooth.

We measure the throughput by sending ICMP ping. By varying the packet size and packet sending rate, we were able to accurately measure throughput and data rate. The reason ICMP ping was used for throughput testing, is that it is an accurate bi-directional

throughput testing tool. Since the payload size for ping packets is identical for sending and receiving, the sending and receiving operations are equally tested. This is especially important for WiFi. Recall from the Background section that when WiFi is in PSM mode, receiving packets that are queued at the AP has an added delay of several hundred milliseconds. ICMP ping therefore places equal weight on sending and receiving packets.

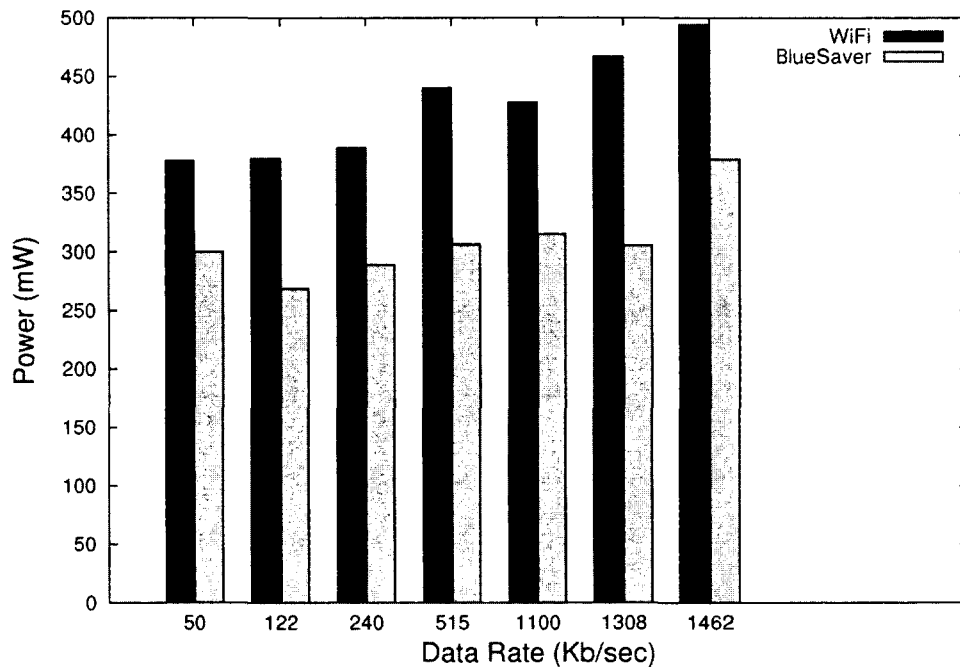


Figure 4.8: Bluetooth vs Adaptive PSM power consumption. Adaptive PSM consumes between 20% more and 35% power than Bluetooth.

Using ICMP ping for throughput testing, we test bitrates from 50kb/sec up to 1400kb/sec, as can be seen in Figure 4.8. We initiate the test from the phone to the AP. For the entire bitrate range that we test, the Adaptive PSM implementation within the WiFi driver switches to CAM for a majority of the time resulting in extra power consumption. Clearly the WiFi driver is performing the best it can under the circumstances. Recall from the Background section that staying in PSM will result in unacceptable delay, while staying in CAM results in the higher power consumption. Within the bounds of the bitrate range that we test, Bluetooth is more efficient than WiFi. Once the rate exceeds

1500kb/sec, we start to see packet loss and added RTT delay over the Bluetooth radio. Therefore, as part of the Bluesaver design, when the rate exceeds 1500kb/sec, we switch to WiFi to minimize delay.

As can be seen, Bluetooth consistently saves between 20% and 35% power consumption compared to WiFi adaptive PSM. This shows that when data rates are within this range, Bluetooth should be used to extend the battery of Smartphones.

This result also shows an interesting trend. In [54], the power consumption of CAM mode on an Android smartphone from 2010 was measured to approximately 20 times higher than that of PSM around 720mw when idle. Note that with the Motorola RAZR, which came on the market less than two years later, we find that the WiFi driver is much more efficient approximately 375mw with a light load of 50kb/sec. The reasons for this improvement are not that clear, however, it could be that the Adaptive PSM implementation has been improved as well. Even with these improvements, Bluetooth is still a better alternative with lower bitrate network traffic.

Video Streaming

According to [24], 80% of videos for smart phone traffic use a bitrate of less than 256kbps. In order to evaluate the energy efficiency of Bluetooth, we setup a streaming server located on the same LAN as the AP. We set the video streaming server VLC to stream via RTSP and streaming at video bitrates from 64kbps to 512kbps. We measure the total energy consumed and compare to Adaptive PSM.

We install a popular RTSP streaming application called MoboPlayer on the phone. This application is capable of playing back RTSP stream over the internet. We measure the average power consumption during the test and measure the total time taken to play back the entire video at the various bitrates. This particular application requires time to buffer the video stream before it is played back. Figures 4.9 and 4.10 show the results

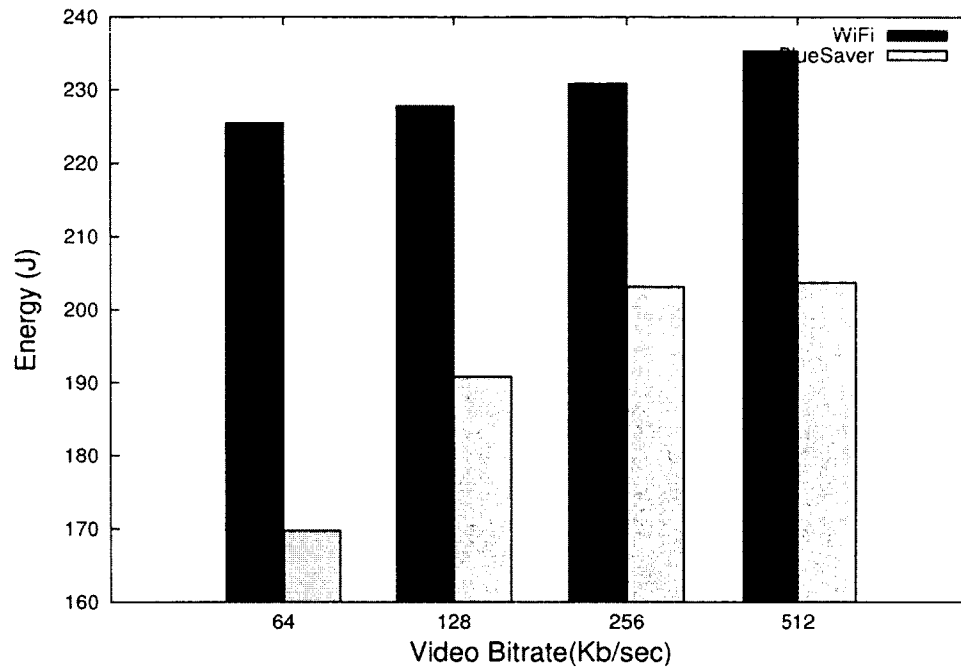


Figure 4.9: Bluesaver vs WiFi streaming video Energy comparison. Bluesaver saves up to 25% energy for bitrates ranging from 64 to 512 kbps.

of this test. In Figure 4.10, it is quite clear that Bluetooth consumes much less power than WiFi in all tests.

Since we are using the RTSP protocol for streaming, there are several seconds of buffering that occurs before the stream actually begins playing. In this case, WiFi clearly has the advantage due to its superior speed. Therefore, the Bluetooth energy results, shown in Figure 4.9 reflect a more modest energy savings ranging from 25% for video codec streaming at 64kbps to 13.5% at 512kbps. This shows that even if a streaming protocol requires extra time for buffering, Bluetooth is still a better solution for streaming video.

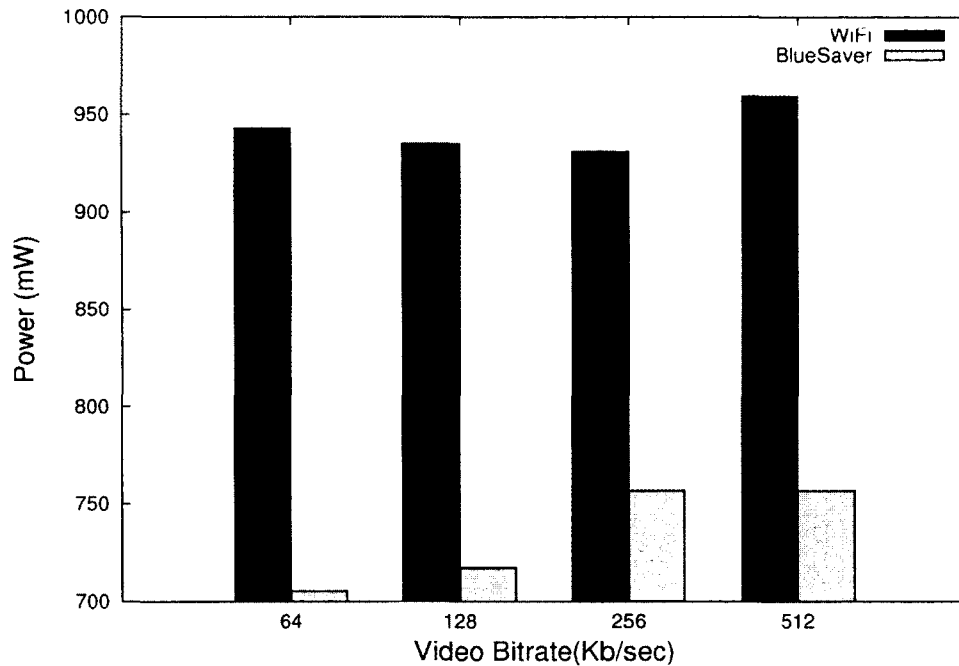


Figure 4.10: Bluesaver vs WiFi streaming video power comparison. Bluetooth consistently uses less power than WiFi.

4.5.3 Network Adaptation

In this subsections we demonstrate that Bluesaver has the ability to quickly switch between Bluetooth and WiFi. A key characteristic of the Bluesaver architecture is to nimbly switch between radio types with minimal delay. In order to thoroughly test this aspect of Bluesaver design, we test two key components. How quickly does Bluesaver adapt to fluctuations in data rates. Second, we address connection quality adaptation. That is, how quickly and how does Bluesaver adapt when the phone is outside of the useful range of Bluetooth and still within the useful range of WiFi.

Data Rate Adaptation

We measure the responsiveness, or how quickly the system can detect changes in data rate and respond accordingly. When the AP first starts to send packets to the phone the data rate spikes above the Bluetooth data rate threshold of 1.5Mbit. When the data

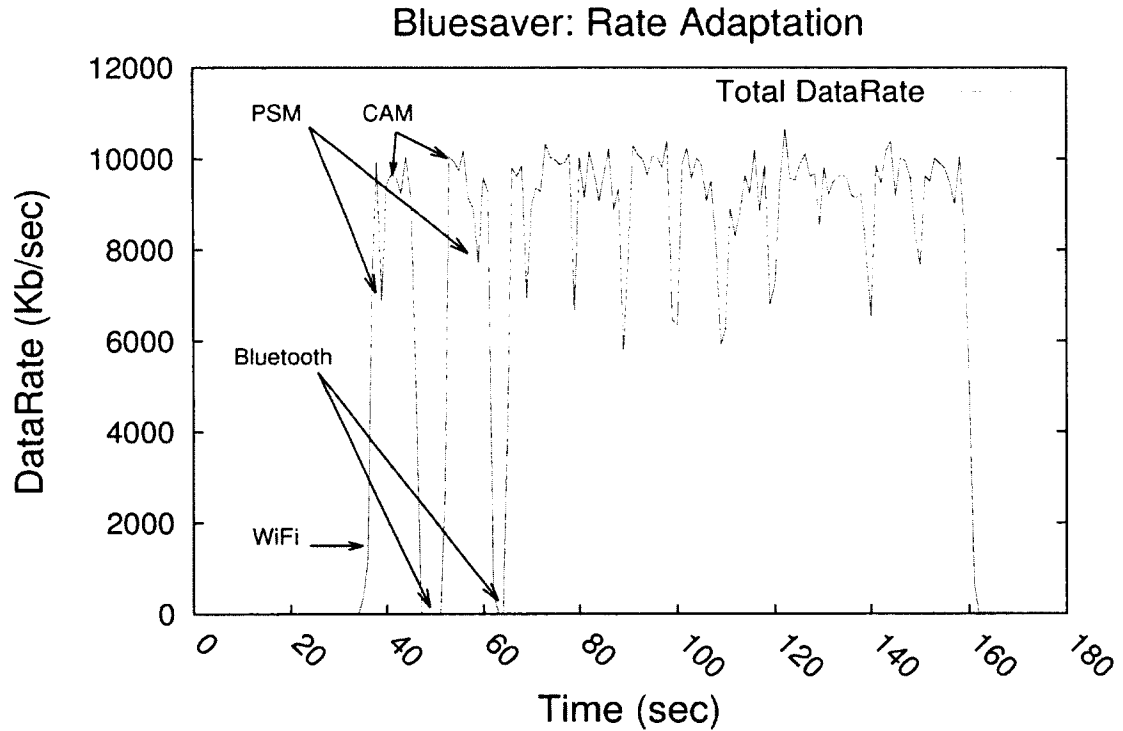


Figure 4.11: Rate Adaptation: Bluesaver switches from Bluetooth to WiFi while sequentially downloading a 10MB file followed by a 100MB file without interrupting the download.

rate exceeds the threshold, the AP will switch from Bluetooth to WiFi. When the phone detects that packets are received on the WiFi interface, it will disable Bluetooth and transmit packets over WiFi. When the data rate again drops below the threshold, it again switches back to Bluetooth.

We place a 10MB and 100MB file on a web server running on the same subnet as the AP. We then proceed to download the file using an http client on the phone. This is the worst case energy-wise for Bluetooth because the high WLAN speeds available over WiFi make Bluetooth inefficient. Figure 4.11 shows the results of this test. We first download a 10MB file, then wait a few seconds and download the same 10MB file again. A few seconds later, we download the 100MB file. As soon as the first download starts, Bluesaver rate adaptation detects that the download speed exceeds what Bluetooth can

handle. At that point, it will switch to WiFi. The packets are received over WiFi to quickly download the file. When the file is done downloading, the connection will fall back to Bluetooth to save energy.

We note that during the test, the WiFi driver did switch between CAM and PSM during the download as noted in the figure. It is not clear why the WiFi driver switched to PSM during the middle of the transfer. However, this explains the dips shown in the download of the 10MB files and shown especially clearly in the 100MB file. Additionally, due to limitations (Or a configuration error) of the AP software, we were unable to exceed speeds of 10Mb/sec.

Connection Adaptation:

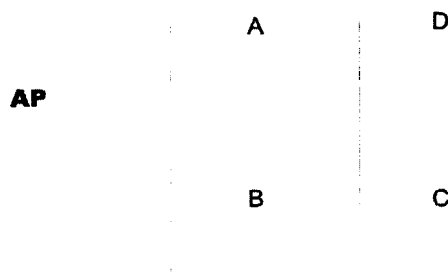


Figure 4.12: Room Layout showing lab environment for connection adaptation evaluation.

We measure how quickly Bluesaver can adapt to changes in the network environment. To perform this test, we setup a Bluesaver enabled AP in a building described in Figure 4.12. We initially setup WiFi only and walked from within one meter of the AP to points A, B, C, D, then A, B, C, D again. After which we proceeded back to the AP. The total distance from the AP to point D is about 10 meters. During the duration of the test we setup an ICMP ping running on the AP that will record the RTT time.

The results of this test are shown in Figure 4.13. The WiFi results show that around time 50 and 100s the phone experienced extreme packet loss and was unable to respond

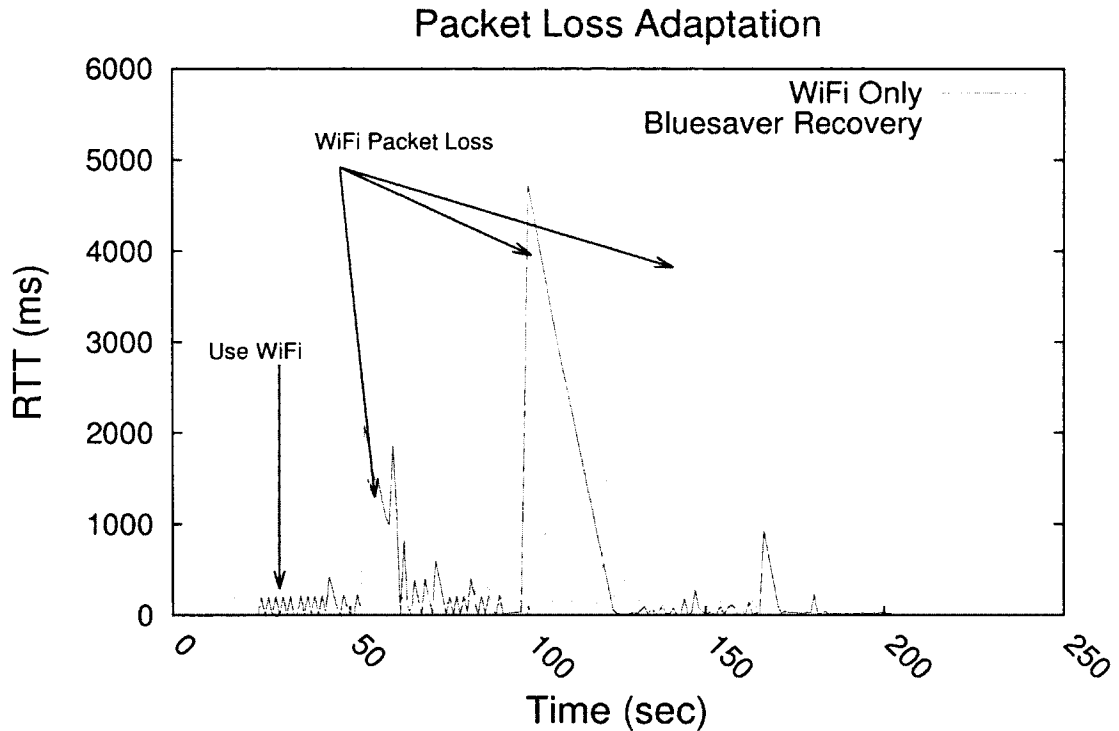


Figure 4.13: Connection Adaptation: Bluesaver switches from Bluetooth to WiFi when Bluetooth connection issues are detected. A constant ping from the Server to the phone was done for the duration of the test. When the switch to WiFi occurs at around 30 seconds, the ping test is unaffected.

for several seconds. Tracking down the root cause of packet loss is challenging, but most likely this is the result of a combination of the fact that the phone was rapidly moving and the multiple obstructions between the phone and the AP.

The same test was then performed with Bluesaver enabled. In this case, the Bluetooth Availability Manager running on the AP sent a steady I2ping to the phone during the duration of the test. At 30 seconds into the test, the BAM was unable to connect to the phone over Bluetooth. Immediately, the HCM was notified over the netlink socket and it started sending packets over WiFi for the duration of the test until the phone got back into Bluetooth range of the AP again. The phone then detected that packets were retrieved over WiFi and in turn started sending packets over WiFi as well.

The result is that at 30 seconds into the test, there is no perceptible difference in the RTT measurement. This shows that Bluesaver can switch between radios without adding any additional delay. After 30 seconds into the test, we see severe WiFi related packet loss similar to the WiFi only test which in this case is unavoidable.

We have shown that Bluesaver can adapt to fluctuating data rates and adverse network connection issues. Furthermore, Bluesaver is able to save energy when using low data rate applications by as much as 25% over existing solutions.

4.6 Conclusions

Smartphones suffer from the dilemma that network traffic to and from the device either has excessive latency and low power consumption or low latency with high power consumption. To address this problem, we have presented Bluesaver, a novel approach that combines the low latency, low power consumption characteristics of Bluetooth with high speed, higher power consumption characteristics of WiFi implemented at the MAC level. We have demonstrated that Bluesaver is able to adapt to changing network conditions by routing network traffic between different PHY interfaces. Finally, we have demonstrated that we can save up to 25% energy over existing solutions for certain types of network traffic.

In future work, we plan to extend the Bluesaver platform to better support multiple clients. As we described in the Design section, our current solution is to pair a single client with a dedicated Bluetooth adaptor. In future, we plan to extend this approach to use a load balancing algorithm to more efficiently support multiple clients.

Chapter 5

Conclusions & Future Work

We have demonstrated that network traffic traversing through Smartphones can be very power intensive. Through the use of a Network Traffic Aware Approach to Smartphone Energy Savings, we have demonstrated that not only can network traffic be made more efficient, we can also ensure that QoS requirements are met. We have made the following contributions:

- *Real-Time Traffic:* We have demonstrated that significant energy savings can be obtained by identifying idle periods where the WiFi radio can be put to sleep. At the same time, we honor the unique QoS constraints inherent within real-time network traffic.
- *Application Priority:* Existing behavior of WiFi power management in Smartphones simply considers combined data rate of all network traffic in which to modify the current WiFi power state. By prioritizing network traffic by application, we demonstrate significant energy savings over existing solutions. QoS requirements of high priority applications are honored by permitting only high-priority traffic to modify the current WiFi state.

- *Mixed Radio Data Driven Energy Savings:* By exploiting the fact that Smartphones come equipped with multiple Wireless radios, we combine the attributes of Bluetooth and WiFi for significant energy savings using a MAC based high-availability scheme. We have demonstrated that certain traffic that is very inefficient over WiFi due to its low data rate can be effectively streamed over Bluetooth without impacting QoS requirements.

Our goal of saving energy on Smartphones through effective network traffic driven wireless energy management policy is realized through our focus on three scenarios. Our work in SiFi demonstrates energy can be saved while in the midst of delay sensitive network traffic, utilizing silence periods to place the WiFi transceiver into a low power state. We have demonstrated through SAPSM that different application priorities can be exploited to save energy. By using machine learning techniques, the priority of applications can be determined; low-priority application traffic can then be optimized for energy efficiency. Finally, through our work Bluesaver we have shown that by using Bluetooth to augment WiFi PSM behavior through a WiFi MAC layer implementation, significant energy savings can result without impacting the end-user. Traffic with low data rates can be routed through Bluetooth, while high volume, high priority traffic can be routed over WiFi.

Further possibilities exist to extend this work as follows. One promising option is to combine all aspects of this work into a single software suite for Smartphones and investigate how all aspects of this work can be utilized together for further energy efficiency. Other options include extending this work to other battery constrained devices such as military applications.

Bibliography

- [1] M. Anand, E. B. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *ACM MobiCom*, 2003.
- [2] G. Ananthanarayanan and I. Stoica. Blue-Fi: Enhancing Wi-Fi Performance using Bluetooth Signals. In *ACM MobiSys*, 2009.
- [3] Android Native Development Kit, 2010. <http://developer.android.com/sdk/ndk/index.html>.
- [4] Android Technical Information, 2011. <http://source.android.com/tech/index.html>.
- [5] Android TrafficStats Information, 2011. <http://developer.android.com/reference/android/net/TrafficStats.html>.
- [6] Android Market Hits 500,000 Successfully Published Apps, 2011. <http://readwriteweb.com/mobile/2011/10/android-market-hits-500000-suc.php>.
- [7] Android Services Reference Documentation, 2012. <http://developer.android.com/reference/android/app/Service.html>.
- [8] T. Armstrong, O. Trescases, C. Amza, and E. de Lara. Efficient and Transparent Dynamic Content Updates for Mobile Clients. In *ACM MobiSys*, 2006.
- [9] M. Azizyan, I. Constandache, and R. R. Choudhury. SurroundSense: Mobile Phone Localization via Ambience Fingerprinting. In *ACM MobiCom*, 2009.
- [10] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. Neil Levine, and J. Zahorjan. Interactive wifi connectivity for moving vehicles. In *ACM SIGCOMM*, 2008.
- [11] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *ACM IMC*, 2009.
- [12] N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong. Users and Batteries: Interactions and Adaptive Energy Management in Mobile Systems. In *ACM UbiComp*, 2007.
- [13] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

- [14] Bluetooth Core Version 2.0 + EDR, 2009. Bluetooth SIG Std. Core Version 2.0 + EDR specification.
- [15] Bluetooth Core Version 3.0 + HS, 2009. Bluetooth SIG Std. Core Version 3.0 + HS specification.
- [16] Android Cloud to Device Messaging Framework, 2012. <http://code.google.com/android/c2dm/>.
- [17] C. Chatfield. *The Analysis of Time Series: An Introduction, Sixth Edition*. Chapman and Hall/CRC, 2003.
- [18] H. Choi and J. Lee. Hybrid Power Saving Mechanism for VoIP Services with Silence Suppression in IEEE 802.16e Systems. In *IEEE Communications Letters*, 2007.
- [19] R.G. Cole and J.H. Rosenbluth. Voice over IP performance monitoring. In *ACM SIGCOMM*, 2001.
- [20] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273--297, 1995.
- [21] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
- [22] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. In *ACM MobiSys*, 2010.
- [23] P. Drago, A. Molinari, and F. Vagliani. Digital Dynamic Speech Detectors. In *IEEE TON*, 1978.
- [24] J. Erman, A. Gerber, K. Ramakrishnan, S. Sen, and O. Spatscheck. Over The Top Video: The Gorilla in Cellular Networks. In *ACM IMC*, 2011.
- [25] A. Estepa, R. Estepa, and J. Vozmediano. A new approach for VoIP traffic characterization. In *IEEE Communications Letters*, 2004.
- [26] iPhone FaceTime bandwidth gets measured. <http://www.digitalsociety.org/2010/08/iphone-facetime-bandwidth-gets-measured/>, 2010.
- [27] Freeswitch, 2010. <http://www.freeswitch.org>.
- [28] M. Fujimoto, K. K. Ishizuka, and T. Nakatani. A Voice Activity Detection based on the Adaptive Integration of Multiple Speech Features and a Signal Decision Scheme. In *IEEE ICASSP*, 2008.
- [29] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157--1182, March 2003.
- [30] Y. Liu; E. I. and H. Qi. An energy-efficient QoS-aware media access control protocol for wireless sensor networks. In *IEEE MASS*, 2005.

- [31] IEEE 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, 1999. ANSI/IEEE Std. 802.11dcf.
- [32] Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements, 2005. ANSI/IEEE Std. 802.11e.
- [33] Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *In Advances in Neural Information Processing Systems 11*, pages 487–493. MIT Press, 1998.
- [34] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, pages 137–142, 1998.
- [35] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *ACM MobiCom*, 2002.
- [36] S. Kullback and R. A. Leibler. On Information and Sufficiency. In *The Annals of Mathematical Statistics*, 1951.
- [37] L. Lamel, L. R. Rabiner, A. Rosenberg, and J. Wilpon. An Improved Endpoint Detector for Isolated Word Recognition. In *IEEE TOASSP*, 2003.
- [38] J. Liu and L. Zhong. Micro Power Management of Active 802.11 Interfaces. In *ACM MobiSys*, 2008.
- [39] Y. Liu, F. Li, L. Guo, Y. Guo, and S. Chen. Bluestreaming: towards power-efficient internet P2P streaming on mobile devices. In *ACM MM*, 2011.
- [40] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *ACM SenSys*, 2010.
- [41] J. Manweiler and R. Choudhury. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation. In *ACM MobiSys*, 2011.
- [42] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. In *ACM SenSys*, 2008.
- [43] E. Miluzzoy, C. T. Corneliusy, A. Ramaswamy, T. Choudhury, Z. Liux, and A. T. Campbell. Darwin Phones: the Evolution of Sensing and Inference on Mobile Phones. In *ACM MobiSys*, 2010.
- [44] Monsoon Solutions, 2011. <http://www.msoon.com/LabEquipment/PowerMonitor>.
- [45] S.B. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *IEEE INFOCOM*, 1999.
- [46] Motorola RAZR, 2012. www.motorola.com.

- [47] V. Namboodiri and L. Gao. Towards Energy Efficient VoIP over Wireless LANs. In *ACM MobiHoc*, 2008.
- [48] NetFilter packet filtering framework, 2011. <http://www.netfilter.org>.
- [49] Netperf Networking Benchmark, 2011. <http://www.netperf.org>.
- [50] A. Pathak, Y. Hu, and M. Zhang. Fine Grained Energy Accounting on Smartphones with Eprof. In *ACM Eurosys*, 2012.
- [51] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. In *ACM MobiSys*, 2006.
- [52] Massimiliano Pontil and Alessandro Verri. Support vector machines for 3-d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:637--646, 1998.
- [53] A. Pyles, X. Qi, G. Zhou, M. Keally, and X. Liu. SAPSM: Smart Adaptive 802.11 PSM for Smartphones. In *ACM UbiComp*, 2012.
- [54] A. Pyles, Z. Ren, G. Zhou, and X. Liu. SiFi: exploiting VoIP silence for WiFi energy savings in smart phones. In *ACM UbiComp*, 2011.
- [55] Qualcomm-Atheros WB225 Reference Design, 2012. http://www.qca.qualcomm.com/media/product/product_106_file1.pdf.
- [56] A. Rahmati and L. Zhong. Context for Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer. In *ACM MobiSys*, 2007.
- [57] A. Rahmati and L. Zhong. Human-Battery Interaction on Mobile Phones. In *Pervasive and Mobile Computing*, 2009.
- [58] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. In *RFC 3261*, 2002.
- [59] E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. NAPman: Network-Assisted Power Management for WiFi Devices. In *ACM MobiSys*, 2010.
- [60] R. C. Shah, L. Nachman, and C-Y. Wan. On the performance of Bluetooth and IEEE 802.15.4 radios in a body area network. Third International Conference on Body Area Networks (BodyNets'08).
- [61] Sipdroid, 2010. <http://www.sipdroid.org>.
- [62] How much bandwidth does Skype need? <https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need>, 2012.
- [63] Sprint HTC HERO, 2010. www.htc.com/us/support/hero-sprint.

- [64] E. Tan, L. Guo, S. Chen, and X. Zhang. PSM-throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs. In *IEEE ICNP*, 2007.
- [65] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, and B. Krishnamachari. A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition. In *ACM MobiSys*, 2009.
- [66] X. Zhang and K. G. Shin. E-MiLi: Energy Minimizing Idle Listening in Wireless Networks. In *ACM MobiCom*, 2011.
- [67] R. Zhou, Y. Xiong, G. Xing, M. L. Sun, and J. Ma. Zifi: wireless Lan discovery via ZigBee interference signatures. In *ACM MobiCom*, 2010.

VITA

Andrew Joseph Pyles

Andrew is currently a cyber security researcher at MITRE corporation. Prior to that he has worked for several start-up companies in Silicon Valley before deciding to return to school. He has been at William and Mary full-time from 2009 to 2012 where he has studied under the direction of Professor Gang Zhou focusing on Smartphone energy research. He has published two papers in ACM Ubicomp.