SAPSM: Smart Adaptive 802.11 PSM for Smartphones

Andrew J. Pyles, Xin Qi, Gang Zhou, Matthew Keally and Xue Liu†
College of William and Mary, †McGill University
{ajpyles, xqi, gzhou, makeal}@cs.wm.edu, †xueliu@cs.mcgill.ca

ABSTRACT

Effective WiFi power management can strongly impact the energy consumption on Smartphones. Through controlled experiments, we find that WiFi power management on a wide variety of Smartphones is a largely autonomous process that is processed completely at the driver level. Driver level implementations suffer from the limitation that important power management decisions can be made only by observing packets at the MAC layer. This approach has the unfortunate side effect that each application has equal opportunity to impact WiFi power management to consume more energy, since distinguishing between applications is not feasible at the MAC layer. The power cost difference between WiFi power modes is high (a factor of 20 times when idle), therefore determining which applications are permitted to impact WiFi power management is an important and relevant problem. In this paper we propose SAPSM: Smart Adaptive Power Save Mode. SAPSM labels each application with a priority with the assistance of a machine learning classifier. Only high priority applications affect the client's behavior to switch to CAM or Active mode, 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.

ACM Classification Keywords

C.2.1 Computer-Communication Networks: Network Architecture and Design—*Wireless communication*; C.4 Performance of Systems: Design studies

General Terms

Design, Performance, Measurement, Experimentation

Author Keywords

Energy efficiency, WiFi, Adaptive Power Save Mode, Smartphones

INTRODUCTION

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 [1]. 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 [2] and real-time VoIP applications [3]. CAM consumes high power but delivers high performance and low latency.

There are a number of existing PSM and adaptive PSM mechanisms [4] [2] [5] that utilize PSM to save energy (described in the background section). Adaptive PSM alternates between PSM and CAM based soley 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 [6] 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, [7] 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 paper we present SAPSM, a Smart Adaptive PSM solution that prioritizes network traffic based on application priority, which we now define. Each application is tagged

UbiComp '12, Sep 5-Sep 8, 2012, Pittsburgh, USA.

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 PSM[1]. 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 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.

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.

[1]Low 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.

Background

First standardized in 1999 [4], 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 [2]. Because of this issue, Adaptive PSM [2] [8] 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 [9] differs between implementations. Our results show a 1.5 second idle timeout period in the HTC Hero, while [8] shows a 20-25ms timeout period for an iPhone 3GS. In the following subsections, we show the results from our survey of various devices.

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 1. During a one second interval, which is not configurable, both ingress and egress

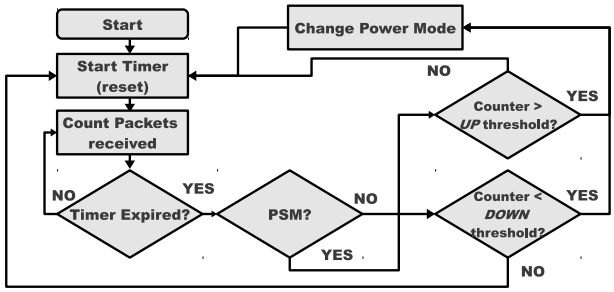

Figure 1. Sprint HTC Hero Adaptive PSM implementation; initial state is PSM, timer is set to one second.

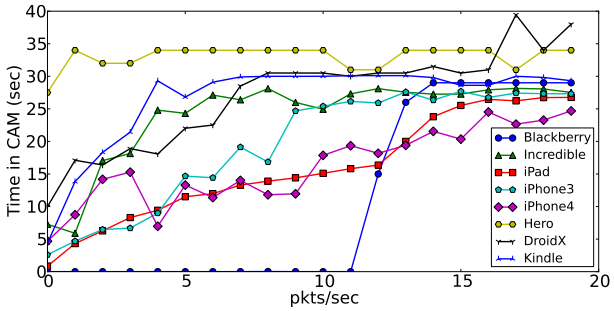

Figure 2. Adaptive PSM implementation response to UDP network traffic.

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 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.

Adaptive PSM Behavior of Different Smartphones (and other Handheld 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 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 [8]. 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 TCP[2] which is 60 bytes.

Smart devices			Traffic type reaction			
Model	Version	UP $threshold$	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 1. Smart devices tested

The results are shown in Table 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 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.

[2]Since 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.

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

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.

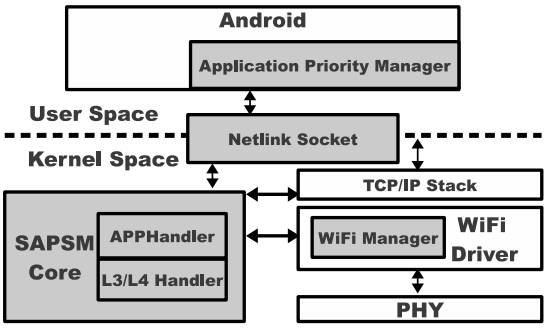

Figure 3. SAPSM Architecture; the packet flow is diverted through the SAPSM core kernel module.

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 [10]. Android provides multiple ways of running data in the background: Background threads and Android services [11] are just two such methods. Android services or background threads, however, are not necessarily an indication of low priority intent. For instance [12] uses a background service for receiving delay-sensitive RTP packets. Therefore, we cannot rely upon this factor alone to indicate low priority intent.

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. 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.

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 [13] 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.

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.

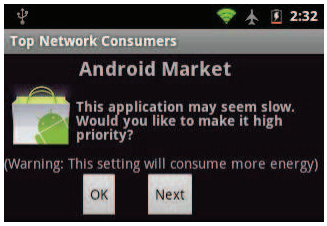

Figure 4. The pop-up window to assist user decision.

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 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 [14], image recognition [15], hand-written digit recognition [16], and bioinformatics [17]. In general, SVM has four advantages over other classifiers [18] [19]: (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.

USER STUDY

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.

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 2. Majors.

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 2.

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 occuring; video games for example are very interactive with respect to the user and the screen, but typically non-interactive with respect to to the network.

The accuracy of an SVM classifier depends on the input features. We use the *Sequential Forward Search* based feature selection algorithm [20] to select the best features from these 11 features. The algorithm returns two optimal features - the maximum and mean of RXRate.

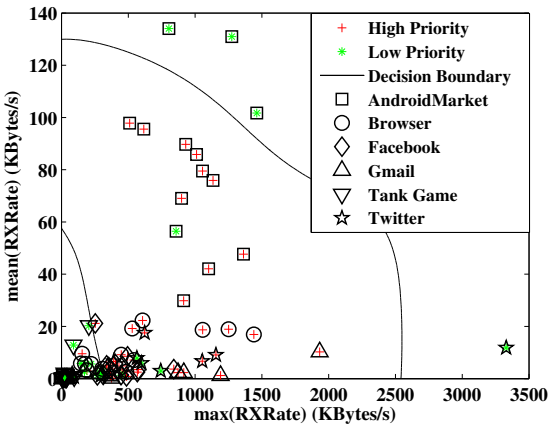

Figure 5. Application classification result.

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 [18]. 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 5 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.

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 5. 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 noticable. 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.

IMPLEMENTATION

We implement SAPSM on the Sprint HTC Hero [21]. 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 [22] 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 [23].

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.

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.

We use the Monsoon Solutions Power Monitor [24] 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.

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.

Traffic with no listening socket

In this section we evaluate the behavior of SAPSM, Static PSM and Adaptive PSM when subjected to traffic not as-

sociated 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.

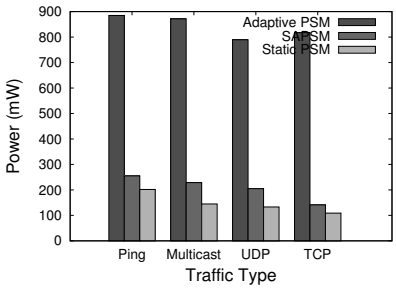

Figure 6. Comparison of power consumed from ingress traffic with no listening socket.

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 6. 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.

Low Priority Energy Inversion

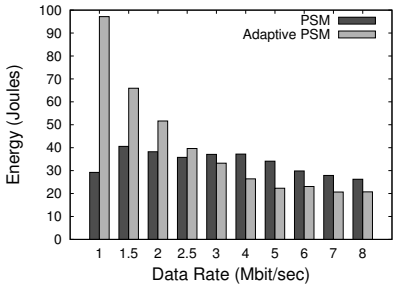

Figure 7. Energy inversion: 10MB file downloaded w/varying data rates.

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 [2], 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 7, 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 [25], 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.

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 8 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 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 [6]. 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

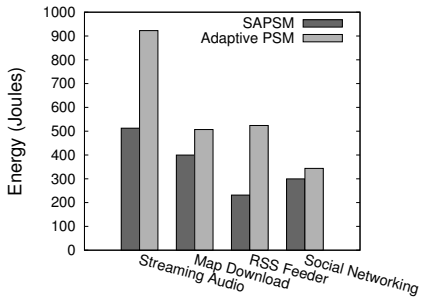

Figure 8. **Application energy comparison.**

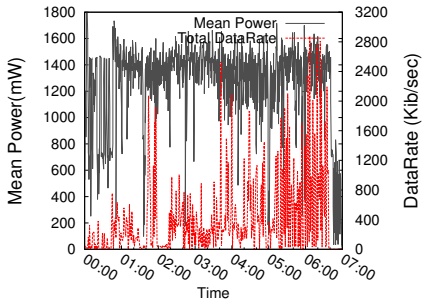

Figure 9. **RSS Reader with Adaptive PSM**

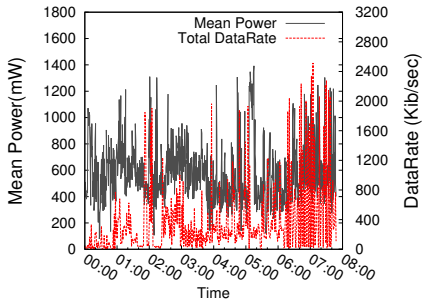

Figure 10. **RSS Reader w/ low priority SAPSM**

downloaded. For this test we download a map of a US 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 (9 and 10) 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.

SAPSM High Priority Networking Performance

While saving energy is important, having solid networking performance for high priority traffic is equally important. We use Netperf [26] 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. 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. 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.

	Throughput		Time Delay	
Type	(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. SAPSM high priority TCP Performance results with associated standard deviation.

RELATED WORK

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 [27] uses traffic shaping on the client to add burstiness to TCP traffic which contributes to more efficient PSM behavior. Sleep durations are modified [2] in reaction to current traffic levels maintaining a bounded RTT for more efficient adaptive PSM behavior. μPower management [28]

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 [29]. Most closely related to SAPSM, STPM [6] 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 [25] the difference in bandwidth between Wireless and WAN connectivity is exploited. Napman[8] reduces contention by staggering beacon intervals per client and the use of a fair scheduling algorithm. In [30] network contention reduction is explored: beacon periods are staggered by eavesdropping on APs 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 [1] silence periods are exploited in VoIP calls to save additional energy. Coolspots [31] uses multiple radios to only enable the radio with highest energy cost when required.

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 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.

REFERENCES

1. A. Pyles, Z. Ren, G. Zhou, and X. Liu. SiFi: exploiting VoIP silence for WiFi energy savings in smart phones. In *ACM UbiComp*, 2011.
2. R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *ACM MobiCom*, 2002.
3. V. Namboodiri and L. Gao. Towards Energy Efficient VoIP over Wireless LANs. In *ACM MobiHoc*, 2008.
4. IEEE 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, 1999. ANSI/IEEE Std. 802.11dcf.
5. 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.
6. M. Anand, E. B. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *ACM MobiCom*, 2003.
7. A. Pathak, Y. Hu, and M. Zhang. Fine Grained Energy Accounting on Smartphones with Eprof. In *ACM Eurosys*, 2012.
8. E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. NAPman: Network-Assisted Power Management for WiFi Devices. In *ACM MobiSys*, 2010.
9. A. Balasubramanian, R. Mahajan, A. Venkataramani, B. Neil Levine, and J. Zahorjan. Interactive wifi connectivity for moving vehicles. In *ACM SIGCOMM*, 2008.
10. Android Cloud to Device Messaging Framework, 2012. http://code.google.com/android/c2dm/.
11. Android Services Reference Documentation, 2012. http://developer.android.com/reference/android/ app/Service.html.
12. Sipdroid, 2010. http://www.sipdroid.org.
13. Android Technical Information, 2011. http://source.android.com/tech/index.html.
14. Thorsten Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *ECML*, pages 137–142, 1998.
15. 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.
16. Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.
17. 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.
18. Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
19. Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
20. Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.
21. Sprint HTC HERO, 2010. www.htc.com/us/support/hero-sprint.
22. NetFilter packet filtering framework, 2011. http://www.netfilter.org.
23. Android TrafficStats Information, 2011. http://developer.android.com/reference/android/ net/TrafficStats.html.
24. Monsoon Solutions, 2011. http://www.msoon.com/LabEquipment/ PowerMonitor.
25. F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. In *ACM MobiSys*, 2010.
26. Netperf Networking Benchmark, 2011. http://www.netperf.org.
27. E. Tan, L. Guo, S. Chen, and X. Zhang. PSM-throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs. In *IEEE ICNP*, 2007.
28. J. Liu and L. Zhong. Micro Power Management of Active 802.11 Interfaces. In *ACM MobiSys*, 2008.
29. X. Zhang and K. G. Shin. E-MiLi: Energy Minimizing Idle Listening in Wireless Networks. In *ACM MobiCom*, 2011.
30. J. Manweiler and R. Choudhury. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation. In *ACM MobiSys*, 2011.
31. 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.