SiFi: Exploiting VoIP Silence for WiFi Energy Savings in Smart Phones

Andrew J. Pyles
College of William & Mary
ajpyles@cs.wm.edu

Zhen Ren
College of William & Mary
renzh@cs.wm.edu

Gang Zhou
College of William & Mary
gzhou@cs.wm.edu

Xue Liu
Univ. of Nebraska Lincoln
xueliu@cse.unl.edu

ABSTRACT
Since one-third of a smart phone'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 paper 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 Smart phone and acheive 40% energy savings while maintaining high voice fidelity.

Author Keywords
Silence prediction, VoIP, energy efficiency, WiFi, smart phone.

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, Algorithm, Performance, Measurement, Experimentation.

INTRODUCTION
As demonstrated in [25], 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 [26] [24] [5], many newly developed smart phone related applications choose to involve the WiFi radio communication [25] [21] [3] [19] [32].

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 [30] 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, [15] [9] [1] propose to save WiFi energy by exploiting periods of idleness, while [28] improves scheduling on the WiFi Access Point (AP). [14] shows a general purpose 802.11 protocol changes to save energy for general real-time applications, and [23] analyzes the VoIP network traffic playout deadline to determine when to put the radio to sleep. Also, the Adaptive PSM [28] 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% [10] 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 [20], 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 paper 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.

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 paper, and interested readers are referred to [27]. 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.

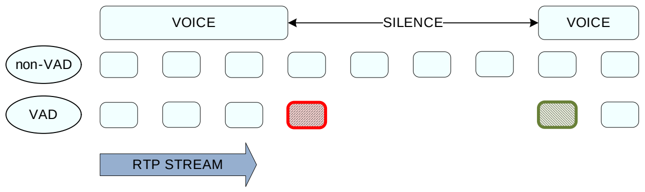

Figure 1. VAD and RTP

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 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.

RELATED WORK

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 [15], 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 [18] exploits small time periods (μ seconds) between MAC frames to save energy. Micro power sleep periods are effective since the speed of WLANs 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 [9] 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 [23] 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 [28] 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 [31] 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 [14] 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 downlink frames will not be triggered.

In [7] an exponential backoff 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 oversleeping.

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.

Lightweight Silence Detection

It has been shown that approximately 60% of a typical conversation is made up of silence [10]. 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 [11], 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 [13] [17], 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.

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.

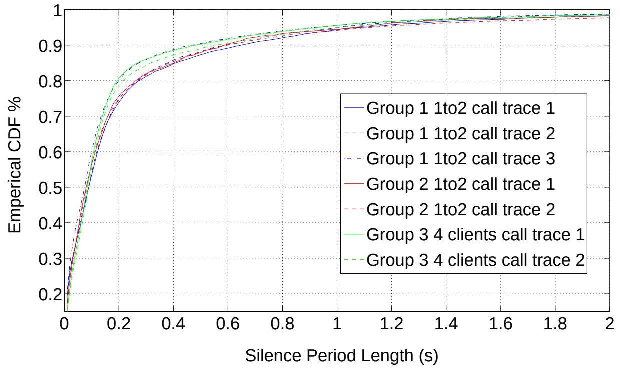

Figure 2. ECDF of Silence Length

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:

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

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.

Algorithm 1 ECDF Based Silence Prediction

```
Input: confidence interval $\beta$, observed silence length $\alpha$, $ECDF$
Output: predicted silence length $\Delta$
on event that a silence period has lasted for time $\alpha$, find the maximum $\Delta$ that satisfies $P(X > \alpha+\Delta | X > \alpha) \geq \beta$

if no $\Delta$ can be found then
    $\Delta = 0$
else
    $\alpha = \alpha + \Delta$
end if
return($\Delta$)
```

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. With R^2 value closer to 1, the predictions are more accurate.

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

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.

Determine the Runtime Training Length

To determine the proper length of the training period, we compute the Kullback-Leibler divergence [16] 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 3.

After the training period, we also check whether the ECDF

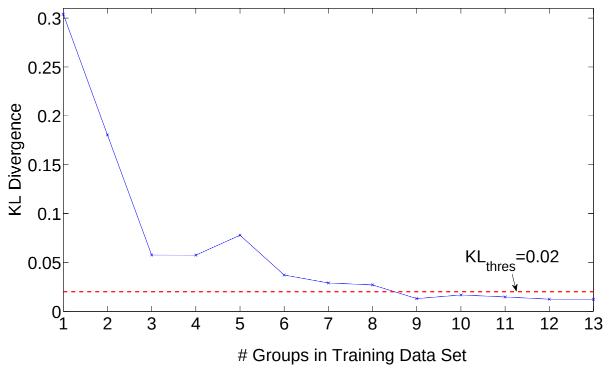

Figure 3. KL Divergence vs. Training Length

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.

The Observed Silence Length α

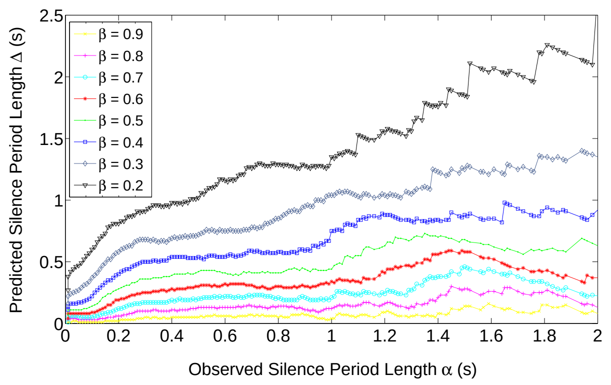

Figure 4. Observed Silence Period Length α vs. Predicted Silence Period Length Δ

Figure 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.

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 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 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 4 defines the expected error E of prediction which is also the expected delay.

$$\begin{aligned} d_i &= \begin{cases} \alpha + \Delta - x_i, & \text{if } \Delta \leq I \\ I - (x_i \bmod I), & \text{if } \Delta > I \end{cases} \qquad (3) \\ E &= \sum_{x_i \in (\alpha, \alpha+\Delta]} \frac{P(X > x_i) - P(X > x_{i-1})}{P(X > \alpha)} * d_i. \end{aligned}$$

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. 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) [6] 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.

SIFI: SILENCE PREDICTION BASED WIFI ENERGY ADAPTATION

We propose SiFi: silence prediction based WiFi energy adaptation. As shown in figure 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.

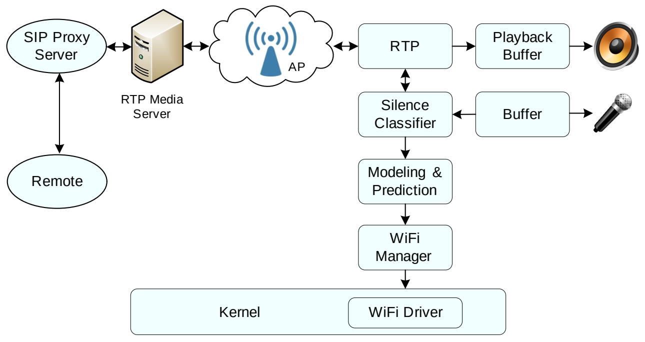

Figure 5. SiFi Architecture

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 [12] 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.

ANDROID PHONE IMPLEMENTATION

We implemented our design on the Sprint HTC Hero [30] 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.

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 [29], 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) [2] 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.

System Level Modifications

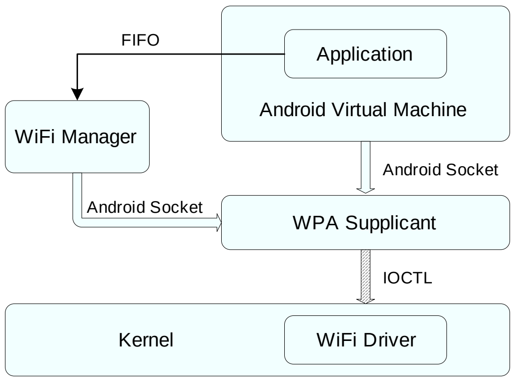

Figure 6. Android WiFi Architecture

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 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.

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. Baseline Average Power

Figure 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_supplicant 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.

PERFORMANCE EVALUATION

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.

Evaluation Setup

Our evaluation setup consists of a Linux server that runs a media server [12] and runs hostapd to serve as a WiFi AP. The media server is configured to host audio recordings.

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

Table 3. Adaptive PSM Settings in Sprint HTC Hero

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.

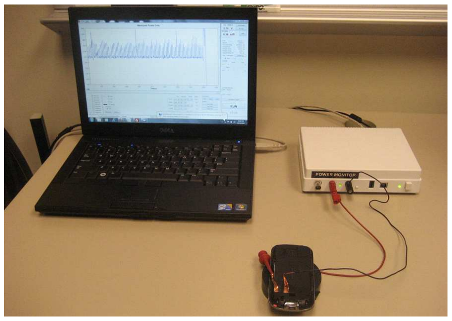

Figure 7. Energy Measurement Setup

We measure the energy use of the phone in real-time using a power meter from Monsoon technologies. Figure 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 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.

Evaluation Method

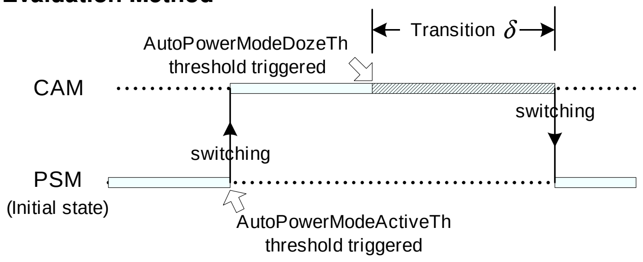
Transition δ
AutoPowerModeDozeTh threshold triggered
CAM
switching
switching
PSM
(Initial state)
AutoPowerModeActiveTh threshold triggered

Figure 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 [15]. Adaptive PSM is complectly controlled by the WiFi 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.

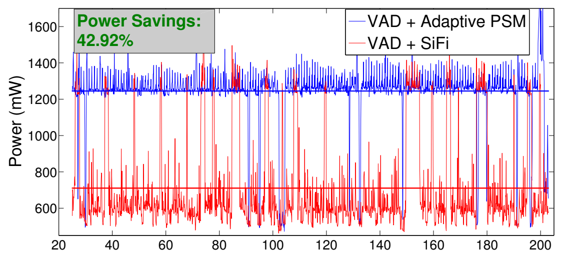

Figure 9. VAD + Adaptive PSM vs. VAD + SiFi

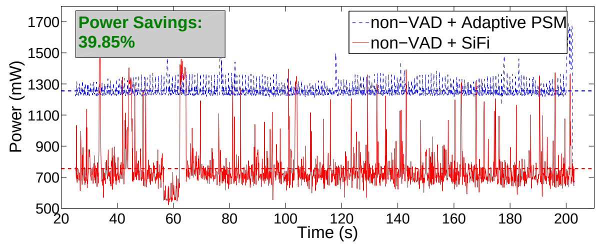

Figure 10. non-VAD + Adaptive PSM vs. non-VAD + SiFi

In Table 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. [28] cites several variations in implementation among common Smartphones. Agressive 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 agressive and default adaptive PSM. As noted in [28], Agressive 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 takest 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.

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 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 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).

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 [8] 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: $1 + 0.035R + 7 * 10^{-6}R(R-60)(100-R)$.

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

Table 4. SiFi Fidelity

Call	SiFi	#People	Duration (min:sec)	μPower (mW)	μOWD (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 5. Multiple Longer Call Tests: VAD + SiFi

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 expectaton factor. The latter is a subjective measurement that is higher when users expect higher call quality. We set A to zero as in [8, 4] since it is not easily quantified. Since we are using the g.711 codec, the R-Factor can then be simplified to [8]: $R \sim 94.2 - 0.024d - 0.11(d - 177.3)H(d - 177.3) - 30ln(1 + 15e)$. 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 [4]. We assume that the delay is identical in both directions and remove the clock skew according to [22].

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

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 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 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, 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.

CONCLUSIONS

In this paper, 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.

REFERENCES

1. M. Anand, E. B. Nightingale, and J. Flinn. Self-Tuning Wireless Network Power Management. In *ACM MobiCom*, 2003.
2. Android Native Development Kit, 2010. http://developer.android.com/sdk/ndk/index.html.
3. M. Azizyan, I. Constandache, and R. R. Choudhury. SurroundSense: Mobile Phone Localization via Ambience Fingerprinting. In *ACM MobiCom*, 2009.
4. A. Balasubramanian, R. Mahajan, A. Venkataramani, B. N. Levine, and J. Zahorjan. Interactive wifi connectivity for moving vehicles. In *ACM SIGCOMM*, 2008.
5. 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.
6. C. Chatfield. *The Analysis of Time Series: An Introduction, Sixth Edition*. Chapman and Hall/CRC, 2003.
7. 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.
8. R. Cole and J. Rosenbluth. Voice over IP performance monitoring. In *ACM SIGCOMM*, 2001.
9. F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. In *ACM MobiSys*, 2010.
10. P. Drago, A. Molinari, and F. Vagliani. Digital Dynamic Speech Detectors. In *IEEE TON*, 1978.
11. A. Estepa, R. Estepa, and J. Vozmediano. A new approach for VoIP traffic characterization. In *IEEE Communications Letters*, 2004.
12. Freeswitch, 2010. http://www.freeswitch.org.
13. 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.
14. 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.
15. R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *ACM MobiCom*, 2002.
16. S. Kullback and R. A. Leibler. On Information and Sufficiency. In *The Annals of Mathematical Statistics*, 1951.
17. L. Lamel, L. R. Rabiner, A. Rosenberg, and J. Wilpon. An Improved Endpoint Detector for Isolated Word Recognition. In *IEEE TOASSP*, 2003.
18. J. Liu and L. Zhong. Micro Power Management of Active 802.11 Interfaces. In *ACM MobiSys*, 2008.
19. 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.
20. 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.
21. E. Miluzzoy, C. T. Corneliusy, A. Ramaswamyy, T. Choudhuryy, Z. Liux, and A. T. Campbelly. Darwin Phones: the Evolution of Sensing and Inference on Mobile Phones. In *ACM MobiSys*, 2010.
22. S. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In *IEEE INFOCOM*, 1999.
23. V. Namboodiri and L. Gao. Towards Energy Efficient VoIP over Wireless LANs. In *ACM MobiHoc*, 2008.
24. 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.
25. A. Rahmati and L. Zhong. Context for Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer. In *ACM MobiSys*, 2007.
26. A. Rahmati and L. Zhong. Human-Battery Interaction on Mobile Phones. In *Pervasive and Mobile Computing*, 2009.
27. 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.
28. E. Rozner, V. Navda, R. Ramjee, and S. Rayanchu. NAPman: Network-Assisted Power Management for WiFi Devices. In *ACM MobiSys*, 2010.
29. Sipdroid, 2010. http://www.sipdroid.org.
30. Sprint HTC Hero, 2010. www.htc.com/us/support/hero-sprint.
31. E. Tan, L. Guo, S. Chen, and X. Zhang. PSM-throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs. In *IEEE ICNP*, 2007.
32. 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.