Towards Energy Efficient VoIP over Wireless LANs

Vinod Namboodiri, Lixin Gao
Dept. of Electrical and Computer Engineering, University of Massachusetts
Amherst, MA, USA
vnambood@ecs.umass.edu, lgao@ecs.umass.edu

ABSTRACT

Wireless LAN (WLAN) radios conserve energy by staying in sleep mode. With real-time applications like VoIP, it is not clear how much energy can be saved by this approach since packets delayed above a threshold are lost. In this work we propose the GreenCall algorithm to derive sleep/wakeup schedules for the WLAN radio to save energy during VoIP calls. The schedules provided by GreenCall consider the energy versus loss-rate tradeoff to ensure application quality is preserved. We implement GreenCall on commodity hardware and study its performance and point out possible limitations. We further extensively evaluate the effect of diverse network paths and different application parameters on possible energy savings through trace-based simulations. We show that, in spite of the interactive, real-time nature of voice, a significant amount of energy can be conserved through GreenCall.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Measurement, Performance

Keywords

VoIP, Wireless LANs, Energy Efficiency

1. INTRODUCTION

The ability to make Voice over IP (VoIP) calls over Wireless LANs (WLANs) is a common option provided by emerging dual mode phones/devices like the Apple iPhone and RIM Blackberry, among others [1]. The main driving factors behind behind this shift from a single cellular mode to dual cellular/WLAN mode in these devices are (i) The cost-effectiveness of making VoIP calls over the Internet through WLANs compared to cellular networks, and (ii) The lack of coverage of the cellular network in some outdoor areas, and indoor areas like the office or home where users typically spend greater than 30% of their time according to [2].

MobiHoc'08, May 26–30, 2008, Hong Kong SAR, China.

The caveat, however, with VoIP over WLANs is that now energy consumption of the WLAN interface can be significant. An active or even idle wireless network interface (WNIC) is a significant drain on the typically energy constrained devices that will be used for VoIP calls over Wireless LANs. For example, the specifications of Apple's iPhone [3] lists a talk time of 14 hours over the cellular network with WLAN interface off. When both the cellular and WLAN interface are on, the talk time is reduced to 8 hours. With very light web browsing and access of email over the WLAN once every hour, the talktime is reduced to 6 hours. Subjecting the iPhone to VoIP calls would further significantly reduce the talktime due to the much heavier workload. Even for higher end devices like laptops, at least 15-20% of the total energy capacity is consumed by an active WLAN interface [22]. Reducing the energy consumed by the interface during VoIP calls is thus a critical step towards extending the operating lifetime of these mobile devices.

For applications with long periods of inactivity like NFS, or those with large tolerable latencies of the order of seconds like web browsing, there is great potential to save energy by letting the WLAN radio sleep periodically [7,21]. The benefit of this approach with real time applications, however, is not clear. VoIP has tolerable latencies of the order of only hundreds of milliseconds and any greater delay degrades the quality of the user's call beyond perceptible limits. Packet generation intervals of the order tens of milli-seconds exacerbate the situation by making it difficult for the radio to spend any significant amount of time in the lower power sleep mode.

In this paper we address the issue of how to reduce energy consumption of the WLAN interface *during* a VoIP call while preserving the quality within acceptable levels. Such a software solution would work with legacy hardware and complement any advances through hardware solutions. Our contributions are the following:

1. **We propose the GreenCall algorithm that conserves energy during VoIP calls over WLANs.** The algorithm allows a controlled trade off between the quality of the call and energy conserved. Based on the maximum delay users can tolerate in their conversations, GreenCall adjusts how aggressively the interface tries to stay in the sleep state. This enables our algorithm to maximize energy consumption while targeting a specified level of application quality. Moreover, the algorithm requires a software upgrade only on the mobile device that desires to save energy. A corresponding software upgrade at the peer device on the other end of a VoIP call is beneficial, but not required. No modifications are necessary at any point of the WLAN infrastructure or the Internet.
2. **We present extensive evaluations of our algorithm through experiments on commodity hardware/software as well as through trace-driven simulations.** We implement our algorithm on commodity hardware and quantify the energy saved

and describe our experiences of the challenges faced in the process. Through our trace-based simulations, we subject our algorithm to diverse end-to-end network characteristics of paths to widespread geographic points of the Internet. We demonstrate that with our algorithm, 50-70% energy savings can be easily achieved on most paths *during* a call. This allows us to conclude that significant minimization of the energy consumed due to the WLAN interface can be achieved even for real-time interactive applications like VoIP.

The organization of this paper is as follows. Section 2 describes our approach to saving energy consumed due to the WLAN interface and formulates the problem we address in this paper. In Section 3 we develop the solution to our problem and present our GreenCall algorithm. In Section 4 we describe experimental results obtained by implementing GreenCall on commodity WNIC hardware/software. Section 5 extends our experimental results through trace-driven simulations and presents a more detailed evaluation of the GreenCall algorithm incorporating diverse network paths. Section 6 surveys related work in the area and concluding remarks are made in Section 7.

2. BACKGROUND AND PROBLEM DEFINITION

In this section we describe the issues faced in saving energy due to the WLAN interface during a VoIP session. Based on this background, we state the problem that needs to be solved.

2.1 Saving energy consumed by WLAN interface

The key idea of saving energy of the wireless interface is to allow it to sleep as much as possible reducing the time spent in idle state. There is typically about one order of magnitude difference between the power consumption in the idle and sleep states [25]. This is a difficult problem since the radio may not know when exactly it has to wake up to receive incoming packets and will lose them if it stays in the sleep state. Other researchers have thus proposed schemes that use multi-radio solutions. The data and control channels are separated, with the control channel generally using a lower power, always active radio to wake up the higher powered Wireless LAN radio (e.g. [25], [26]).

A standardized solution to this issue is the Power Save Mode (PSM) which was introduced in the IEEE 802.11 standard for infrastructure WLANs [4]. PSM allows a node to transition to the lower power sleep state when it is not actively sending or receiving packets by notifying the AP with a specific bit set called PS-Bit. Subsequently, the AP buffers any packets it receives destined for this node. Periodically a beacon is sent out from the AP that informs all associated nodes if they have any packets buffered through a traffic indication map (TIM). Each node on receiving notification of buffered packet(s), can leave sleep mode and request one packet at a time sending a special PS-Poll frame to the AP. Within these retrieved packets, the AP sets a 'more data' bit as long as there are pending packets. The client goes to sleep immediately after it finds no more packets buffered to it. When the client does not want to use PSM anymore, it sets the PS-Bit off and notifies the AP. The client's network card consumes much less power while sleeping by shutting off power to all components except for a timing circuit. Because PSM has been part of the standard for many years now, all current deployments are expected to have PSM built in. Using PSM obviates the need for any supplementary devices to reduce the energy consumption of the WNIC.

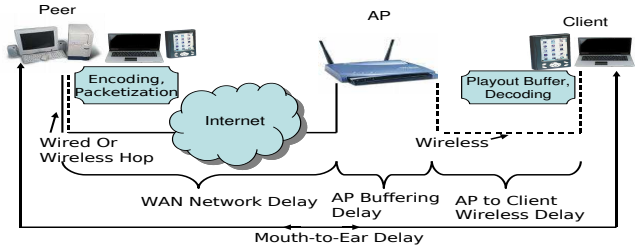

Figure 1: Illustration of end-to-end latency components of a VoIP call between a peer on a wired/wireless network and the client on a wireless network

2.2 Using PSM to save energy during a VoIP call

The two ends of a VoIP call are peers of each other. To simplify our description, we term the device on which energy savings is sought and runs our energy saving algorithm as the *client*. Our descriptions will be based on the perspective of the client. The device on the other end of the client will be referred to as the *peer*. If both ends are running an energy saving algorithm, they are symmetric for the purposes of our descriptions and either of these can be referred to as the client and other as the peer.

Looking at Figure 1, if the client uses PSM to go to sleep, any packets arriving from the peer will be buffered at the AP. If the arriving packets were delay-tolerant, the client could sleep for long durations (order of seconds) before collecting packets from the AP. VoIP traffic, however, has small tolerable latencies and each packet must reach the client by its playout deadline. Thus, the client sleep schedules must be precise enough to ensure no packets are lost due to missed playout deadlines.

To calculate such a strict sleep/wakeup schedule, we need to consider the latency (mouth-to-ear delay) of a packet from the peer to the client. It can be broken into different components as illustrated in Figure 1. The latency from the peer to client's AP is mainly the network delay for the packet once it is sent out from the application layer of the peer station. The peer incurs an encoding and packetization delay before it hands the packet to the network layer. Once a packet reaches the AP, it is buffered there until the client comes out of PSM and is ready to receive the packet. Finally, once the packet reaches the client, it is kept in a playout buffer to reduce jitter on playback. The minimum latency induced by the playout buffer is only the time to decode and playout the packet. When the mouth-to-ear delay exceeds a specific tolerable value (thus termed tolerable delay), the packet is dropped.

2.3 Problem Statement

We will now define the problem we consider in more detail. Figure 2 illustrates the possible packet arrival patterns for two cases at the client - when it does not go to sleep at all, and when it periodically goes to sleep. For simplicity, this illustration and associated description assumes that only the client is trying to save energy (we look at the case when both ends try to save energy in Section 3.2). The packet receive times when client is not using PSM is not a straight line because the network latencies from peer to client are variable. On the other hand, the line for playout deadlines is straight because it is the sum of a constant tolerable latency over packets generated at fixed intervals. The difference between the line representing receive times without PSM and line for playout deadlines is the time that the client has to play with to save energy. For the case when client periodically transitions to sleep, packets arrive at the client in bursts. The first packet delivered to the client after a sleep period is likely to be closest to its playout deadline

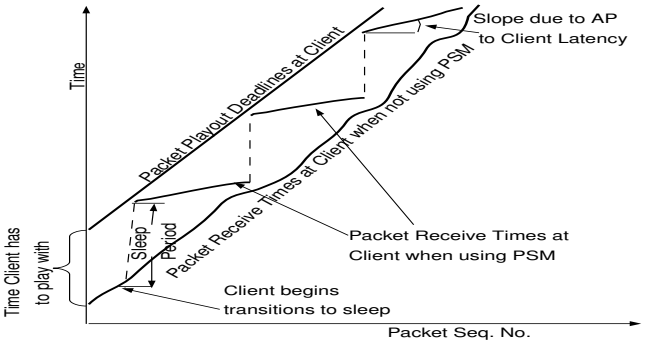

Figure 2: Timing of packet arrival at the client with and without using PSM with respect to playout deadlines.

compared to the other packets sent by the AP for the same sleep period. The lines showing receive times at client on wakeup also have a slight slope (over a horizontal line) incorporating the latency between the AP and client, and the transition time for the client to notify the AP about its transitions to and from the sleep state.

The client needs to calculate[1] a sleep/wakeup schedule that allows it to sleep as much as possible, while ensuring it receives all packets before their playout deadline. Larger sleep periods are preferable to smaller sleep periods as they minimize the total overhead incurred in transitioning to and from the sleep state by notifying the AP each time. In the context of Figure 2, it implies that sleep periods chosen must ensure the line representing packet receive times at client always stays below the line for playout deadlines. Another issue posed to the client during a VoIP call is that apart from receiving packets from the peer, it is also sending packets to the peer at fixed intervals. Thus, assuming it buffers any generated packets when it is in sleep state, it also needs to ensure that its sleep/wakeup schedule is such that these packets reach the peer by their playout deadlines

If the client knew the network latency for each packet throughout the call (i.e. it had knowledge of the line representing packet receive times without PSM) the problem can be easily solved. Unfortunately, network latencies[2] of packets arriving in the future are unknown to the client when it has to calculate its schedule. If the estimate of network latency used by the client is larger than the underlying network latency, it is missing out on opportunities to save energy by sleeping more. On the other hand, packet loss is likely if the underlying network latency increases above any estimate used by the client. To achieve energy savings under the circumstances, we trade-off the possibility of some packet loss due to transitions to sleep mode by introducing a parameter LR that specifies the tolerable loss rate of the application.

Let $\Gamma = \{\gamma_1 \cdots \gamma_n\}$ be the set of sleep periods used by the client during the call, with n being the number of times it transitions to sleep mode during the call. The energy consumed for this sleep/wakeup schedule can be modeled as

$$E_\Gamma = P_{tx}T_{tx} + P_{rx}T_{rx} + P_{idle}T_{idle} + P_{sleep}T_{sleep}, \qquad (1)$$

where P's are the known power consumption values and T's are the total time spent during the entire call in transmit, receive, idle and sleep states respectively. We have $T_{sleep} = \sum_{i=1}^{n} \gamma_i$ and $T_{idle} = T_{call\_duration} - T_{tx} - T_{rx} - T_{sleep}$. T_{tx} and T_{rx} can be treated as constants based on known time to send or receive each packet and known number of packets communicated during the call. Thus, E_Γ is completely characterized by the sleep schedule Γ. The energy savings achieved for some sleep/wakeup schedule Γ is the difference between energy consumption when client does not sleep at all, and energy consumption using Γ.

For the whole call, we seek Γ that maximizes energy savings while targeting a loss rate no greater than LR at both ends of the session. The specified loss rate LR is the sum of packet losses due to missed playout deadlines induced by running an energy saving algorithm, L_{es}, and underlying network losses (including delayed packets missing playout deadlines), L_{nw}; i.e. $LR = L_{nw} + L_{es}$. Even though L_{nw} cannot be controlled by the energy saving algorithm, L_{es} should be controlled in an attempt to maintain the total loss rate below LR at the end of the call. The degree of energy savings should degrade gracefully with increase in loss rate in contrast to an abrupt 'all or nothing' policy. Larger sleep durations are desirable, if possible, to minimize the overheads involved (more specifically, the AP notifications) in each transition to the sleep mode.

3. DERIVATION OF SLEEP/WAKEUP SCHEDULES

In this section we will describe our approach to derive sleep/ wakeup schedules and present our GreenCall algorithm to solve the problem as defined in Section 2.3. We will bifurcate our presentation of deriving sleep schedules into two cases: one where only the client is trying to save energy through PSM and the peer radio always stays in active mode, and another where both the client and peer desire to save energy. The first case happens in scenarios where the peer user does not care about energy. For example, such a peer device could be powered through a wall socket. We will conclude the section by presenting a variant of GreenCall which does not require feedback between the client and peer, thus enabling independent operation.

3.1 Derivation of schedules when PSM used only by client

Assume that our energy saving algorithm is running at the client. To calculate sleep periods, the client needs to perform the following three steps: (i) Determine playout deadlines of each arriving packet, (ii) Estimate times at which packets would have been received at the client if it never used PSM, and (iii) Calculate sleep period for future packets based on difference between the playout deadline and theoretical receive time at client without PSM of previously received packets. We begin by describing the calculation of playout deadlines.

Let t_s^i be the time at which a packet i is sent from the peer to client. Since voice needs to be encoded and packetized before it is sent, the time at which the voice content of packet i was generated is $t_s^i - T_{pktz}$, where T_{pktz} is the constant encoding and packetization delay. Let t_a^i be the time at which packet i arrives at the client and t_p^i be the time by which it has to be played out (also called playout deadline). Let l_{pc} be the estimated network latency from peer to the client, and let l_{ac} be the estimate of network latency from client's AP to client. We will describe our estimation methodology for these latencies later in this section when we present our algorithm.

The playout deadline for a packet i can be calculated as the sum of tolerable latency from the time the packet's voice content was

[1]For now assume that the client has control of when and how long it can sleep and is not restricted to the AP beacon interval schedule as in PSM. The client notifies the AP before sleeping and on wakeup. We will address the practical implications in Sections 4 and 5.

[2]By each packet's network latency we mean the latency the packet will suffer in traversing the network path from one end to the other.

Table 1: Notation used in calculation of sleep periods

t_s^i,t_a^i and t_p^i	Time when packet i is sent, arrives or has to be played out
T_{pktz},T_{pb}	Encoding and decoding delays
T_L,T_I	Tolerable latency and Packet generation interval
l_{pc}, l_{cp}	Estimated latency between peer and client and vice-versa
l_{ac}	Estimated latency between AP and client in both directions
$\hat{t}_{spare}^i$	Pseudo spare time of i observed at client
t_{spare}^i	Estimate of actual spare time of i
t_{spareH}^i	Estimate of spare time of packets after i based on last H packets

generated, or

$$t_p^i = t_s^i - T_{pktz} + T_L, \tag{2}$$

where T_L is the constant tolerable latency of all packets. Based on the arrival time of first packet at client and network latency estimate, it can calculate t_s^i for all subsequent packets simply as

$$t_s^i = t_a^1 - l_{pc} + (m-1)T_I, \tag{3}$$

where T_I is the constant packet generation interval between successive sequence numbered packets, and m is the sequence number[3] of a packet.

Now that the client can calculate the playout deadline of each packet, it requires an estimate of packet network latencies to calculate the receive times for each packet if it were never using PSM. For this we use the concept of *spare time* of a packet which is the difference between its playout time and arrival time at client. The spare time of any received packet i can be directly calculated by the client based on the difference between its arrival time and its playout deadline as

$$\hat{t}_{spare}^{\,i} = t_p^i - T_{pb} - t_a^i, \tag{4}$$

where T_{pb} is the time required to decode and play out a packet and each packet must reach the client in time to allow this operation[4].

Once the client has begun transitions to sleep, however, network latencies of subsequent packets it observes include the delays incurred at the AP's buffer. Thus, $\hat{t}_{spare}^i$ can be termed as the *pseudo-spare* time of packet i. The actual spare times (difference between playout deadline and packet receive time without PSM) of packets previously received, on which it can base its future sleep period calculations, is unknown. Our approach to solve this relies on the knowledge of arrival times of packets at the client and its last used sleep period. By adding the possible buffering delay incurred by a packet at AP to its observed spare time at client, we hope to reconstruct the actual spare time that would have been observed for the packet if it were never buffered at the AP.

A critical observation required at this point is that the first packet buffered during a sleep period possibly incurs the maximum delay among all packets in the AP buffer. Because a packet is generated by the peer every T_I ms, the first packet in buffer will likely come in no later than T_I after AP starts buffering for the client for that sleep period. Due to notification delay between client and AP, the AP starts buffering l_{ac} before client goes to sleep and stops buffering only l_{ac} after client wakes up. Thus, the arriving packet will have a minimum buffering delay of $\gamma^k + 2l_{ac} - T_I$, where γ^k is the last used sleep period, $k \in 1, \cdots, n$. If the first packet comes earlier, our re-constructed spare time will be an under-estimate (i.e. a conservative estimate) by up to T_I. If the first packet comes in later, but within the same sleep period, it will be buffered for lesser time at the AP, but will still be received at the client at the same time as it would have done if it was not late. Thus, the *buffering at AP during a client's sleep period is able to absorb some additional latency of late-arriving packets*.

We can then use the following equation to get an estimate of the actual spare time for each packet received at client.

$$t_{spare}^i = \hat{t}_{spare}^{\,i} + max(0, \gamma^k + 2l_{ac} - T_I), \tag{5}$$

where γ^k is the last used sleep period before packet i was received. The above equation gives an upper bound on the actual spare times of all packets as they arrive, and not just the first one. This bound is tightest for the first packet arriving in the AP buffer for a sleep period and provides the most accurate picture of the network latency incurred by any packet of that period.

Next, we will describe our technique to use the upper bound on actual spare times of previously received packets to estimate the actual spare times of packets arriving in the future; once the client has an estimate of actual spare times for future packets, calculating sleep periods is simple. We use the minimum of the last H spare times as an estimator of the spare time of future packets. That is, future network latencies are estimated as the largest network latency of the last H packets. Using the 'minimum' of spare times of the last H packets helps discard all the loose upper bounds we had of packets that did not arrive first in an AP buffering period, and provides an estimate closest to the minimum of actual spare times of previous packets. Formally, we calculate the spare time for a received packet i over a window of H packets as

$$t_{spareH}^i = min\ \ t_{spare}^j,\ \forall j = i - H + 1 \text{ to } i \tag{6}$$

where H is chosen at least large enough to ensure that the spare time of the packet that arrived first during the last sleep period is taken into account.

Now that the client knows how to calculate the spare time of packets arriving in the future, only a straightforward calculation of sleep period remains. The $(k+1)^{st}$ sleep period, γ^{k+1}, the client can use after receiving packet i and still ensure subsequent packets meet their playout deadlines can be calculated as

$$\gamma^{k+1} = t_{spareH}^i - 2l_{ac}. \tag{7}$$

How the calculated sleep periods after receiving each packet is utilized with PSM is explained in more detail in Section 3.3. Note that the sleep period calculation above relies heavily on estimates of many parameters; when these estimates are incorrect, there are either missed opportunities to save energy, or packets are lost. We rely on an adaptation of the sliding window H of previous packets to control the loss rate in the latter scenario and is the main feature of our GreenCall algorithm presented later in this section.

The above calculated sleep periods are based on packets received at the client from peer. To ensure that these sleep periods also allow packets sent by client to reach peer before their playout deadlines, periodic feedback is used between the two ends of the session. More details about the feedback mechanism follow later in this section.

3.2 Derivation of schedules when PSM is used by both client and peer

If the peer also desires to save energy and calculates sleep periods to use with PSM, both the client and peer must ensure that their calculations take into account the fact that each packet might be delayed at both ends. If the network latencies suffered by packets are symmetric in both directions, the sleep periods calculated

[3] i represents the order in which packets are received, while m represents the order in which packets are generated. Due to packet loss, the client may not receive all generated packets and thus we need to make this distinction explicit.

[4] For simplicity, we had ignored this constant time in our descriptions in Section 2.3

```
/*Phase 0: Initialization*/
Get values of timing constants, application parameters and algorithm
parameters related to history window H. Set counter k = 0
Based on feedback from peer about its status use C_share = 0 or 1
Obtain estimates l_pc, l_cp, and l_ac
WHILE For each packet i received as call continues
    /*Phase 1: Spare Time Calculation and PSM Execution*/
    Calculate spare time t^i_spare
    Transmit packets in send buffer when awake
    IF AP has no more packets buffered for client
        Increment k and calculate t^i_spareH
        Calculate possible new sleep period γ^k = t^i_spareH - 2.l_ac
        Sleep period to use γ^k = γ^k.C_share
        If γ^k > 0, execute PSM with sleep period γ^k
    ELSE
        Stay awake
    /*Phase 2: Adaptation and Possible Feedback*/
    If packet i misses playout deadline, increase packet loss count
    IF i > C_min and i mod C_interval = 0
        IF current packet loss rate > LR - τ_1
            Increase H
            IF peer not using GreenCall
                Send feedback to increase its l_cp estimate by C_fb
            ELSE
                Wait till H >= H_max before sending feedback to
                increase its l_cp estimate by C_fb
        ELSE IF current packet loss rate < LR - τ_2
                Decrease H
                Send feedback to decrease its l_cp estimate by C_fb
```

Figure 3: GreenCall Algorithm

at both ends should be the same. Under the circumstances, both ends using half the calculated sleep period is a fair way to share the possible sleep times between both ends and ensure packet meet their playout deadlines. In other words, if one end calculates its sleep period as γ and knows the other end to be running an energy saving algorithm, it uses a sleep period of $C_{share} \cdot \gamma$ where C_{share} is a constant for which the value 0.5 provides fairness in energy savings at both ends. As the call progresses, the client calculates sleep periods using Equation 5 and 6 and uses a sleep period equal to half the calculated value. To account for asymmetric network latencies between the two ends, the client relies on a combination of feedback to the peer and adjustments of its own sleep durations. Further details are explained with the presentation of our GreenCall algorithm.

3.3 GreenCall Algorithm

GreenCall handles unknown network latencies by keeping track of latencies suffered by previous packets received at client through a sliding window. This information is then used to calculate future sleep periods. The *size of this sliding window is chosen in relation to the current loss rate*. Consequently, at higher loss rates, more history information is used resulting in conservative sleep periods. This enables a smooth tradeoff between loss rate and energy savings and is the main feature of the algorithm.

The GreenCall pseudocode is presented as Algorithm 3. The algorithm can be divided into three phases: an initialization phase followed by two phases as each packet arrives.

The initialization phase, **Phase 0**, deals with the collection of parameters defined by the application as well as tunable parameters of the algorithm. The final step of this phase is to estimate the one way network latencies l_{pc} and l_{cp} between the client and the peer and vice-versa, and the one way latency between client and AP, l_{ac}. This is done by sending special control packets (ICMP echo packets) from the client to each of these points to get the round trip time (RTT). This RTT is then divided by two to be used as a one way estimate. To account for variability, these estimations are collected over a series of 10 packets with the maximum of these chosen in an attempt to avoid an under-estimate of network delays. Since, at this point the client has not begun transitions to sleep mode, these measurements give the true picture of latencies between these points without introducing any delays due to buffering at AP. **Phase 1** begins with the calculation of spare time for packets as the algorithm loops for each packet received until the call continues. Subsequently, once the AP has no more packets buffered for it, the client goes to sleep for duration γ^k. This duration γ^k considers whether peer is running GreenCall as well through the use of a constant C_{share} as described in Section 3. When the sleep period is not greater than zero, the client just stays in the constantly awake mode (CAM). To ensure that the client does not interrupt its sleep period to send packets, the client buffers any generated packets until it wakes up. On wakeup, the client contends for the medium with downlink packets from AP to send all its packets. **Phase 2** deals with the adaptation of H. By considering the spare times of last H packets, we ensure that the sleep period set is the minimum among all those packets. Large values of H will result in more conservative sleep periods (minimizing packet losses) due to the higher likelihood of including packets that have suffered a larger latency which might have occurred over time. On the other hand, if more network losses are tolerable or if the network latency is estimated to vary only slightly over time, we can save more energy by being more aggressive in selecting a sleep period with smaller values of H. To stay within a target loss rate LR, and achieve maximum possible energy savings, the algorithm constantly monitors the current loss rate and adapts the value of H. The monitoring begins after a minimum number of packets, C_{min} have been received, and is done every $C_{interval}$ packets thereafter. τ_1 and τ_2 are thresholds used to control loss rate below LR and avoid hysteresis. If the peer is running an energy saving algorithm, the client tries to control its loss rate through adaptation of H. Once the maximum H has been reached, it sends a feedback to the peer for it to increase its estimate of l_{cp} so that future sleep periods take that into account. A increased l_{cp} would decrease sleep times at peer and thus improve loss rate situation at client. Once loss rate is back below $LR - \tau_2$, the client sends feedback again to let peer decrease its l_{cp} to original levels. The adaptation of H is done through two constant factors: C_{incf} to increase it and C_{decf} to decrease it.

3.4 GreenCall operation without feedback

In some scenarios, it is desirable that the client run independently from the peer without any mutual feedback. For example, the peer might be running on a device on which adding new software (to run GreenCall) is not feasible or desirable. For such scenarios *we consider a separate variant* of GreenCall where only one of the two ends is sending packets at a time by using *silence suppression*[5] and the client tries to save energy only when it is not sending packets. The client now simply sends a packet when it generates one and goes to PSM only during times when the voice on the peer's side is packetized and sent. The sleep period calculations are done similarly as listed in GreenCall Algorithm shown in Figure 3. Because the client does not transition to sleep at all times through the call, this approach will reduce the amount of energy saved, but will

[5]During a call, for intelligible conversation, only one party is speaking at a time. Thus, only the packetization and transmission of the voice of this party is necessary. The packetization and transmission of silence of the other party is unnecessary and can be suppressed.

allow operation without any feedback. With only one end sleeping at a time (because only one side is talking at a time) each end uses $C_{share} = 1$. We compare the possible energy savings with this variant of GreenCall in our evaluations in Section 5.

4. EVALUATION OF GREENCALL WITH COMMODITY HARDWARE/SOFTWARE

We now present the results we obtain by running our algorithm on a laptop running Linux with the Intel PRO2200 network adapter connected to an IEEE 802.11g based AP.

4.1 Experimental setup

In our experiments we chose to emulate traffic with parameters derived from typical VoIP calls rather than use real VoIP sessions to provide more flexibility and ensure repeatability across tests. Packet level emulation also allows us to focus more on the networking issues and bypass application level operations involved that do not affect the results of this work. In our emulated VoIP session, UDP packets of 160 bytes were continuously exchanged between the mobile client and a peer for the duration of the call. The traffic generating interval was set to 30ms or 60ms which are typical voice frame packetization rates[6]. When the radio was in sleep mode, the generated packets were buffered and sent after the sleep period elapsed. *Silence-suppression* is sometimes used with VoIP traffic to reduce the load on the network by suppressing packets during silent periods. For example, VoIP application from Skype does not use it while the one from MSN does [9]. We studied the amount of energy consumed for cases with and without silence suppression. When silence suppression was used, we generated a trace of conversation between two people as per the recommendations of ITU-T [16] for generating artificial conversations. This generated trace had periods of single talk, double talk and mutual silence. The trace used had the client and peer each actively speaking for only about 40-50 % of the time. During the on-time of a node, packets are generated every fixed interval as mentioned above. For the following experiments, a 12 minute conversation is used which is the typical average length of a VoIP session [13]. The tolerable latency for our experiments was set at 250ms based on studies described in recommendations G.109 and G.114 of ITU-T [18, 19] which suggest 300ms as a good latency to aim for in terms of user satisfaction. Our 250ms tolerable latency is thus a conservative estimate.

We used the built-in battery instrumentation in laptops to measure energy. For example, in Fedora Core Linux, battery state (including remaining capacity and discharge rate) is updated periodically (order of a few seconds) in */proc/acpi/battery* due to the Advanced Configuration and Energy Interface Specification (ACPI) daemon. We kept the other energy consuming components of our measurement laptop constant across all our tests. We ran a script that reads the battery state before the client begins the VoIP session and read it once more after the session. We found the estimated energy consumption just for the wireless interface by subtracting the energy consumption of all other components measured through selective activation of groups of components[7].

After comparisons with two other wireless network cards, we found the Intel PRO2200 based on the *ipw2200* driver to be the most suitable in terms of PSM support. The IEEE 802.11g standard (implemented on the client adapter and APs used in our experiments) states that the sleep periods be specified only as multiples of AP's beacon interval (typically 100ms) [4]. Thus, the only sleep period we could use was 100ms. This also included a 25 ms timeout value[8] specific to our card, which resulted in sleep period durations of no more than 75 ms.

We specified the tolerable packet loss LR as 2% for all experiments. Total packet loss rates of up to 5% are known to provide a fair to good call experience based on previous studies [10, 20]. The parameters used for the dynamic H adaptation are: Initial $H = 100$, $H_{min} = 100$, $H_{max} = 1000$, $C_{interval} = 500$, increment factor $C_{incf} = 1.25$, and decrement factor $C_{decf} = 0.80$. We describe in more detail our rationale for selecting these values and the tradeoffs involved in selecting parameters related to H in Section 5.

Table 2: Energy Savings through GreenCall

	Silence Suppression		No Silence Suppression	
	30ms	60ms	30ms	60ms
Savings(%)	43	53	52	63

4.2 Results

We show the results we obtain for VoIP sessions with the mobile client in UC Berkeley, U.S.A and the peer in UMass Amherst, U.S.A[9] in Table 2. The peer did not run GreenCall for this experiment; we do a more detailed trace-based simulation study in the following section.

The energy saved for our experiments is shown in Table 2. It is interesting to note that energy savings is higher when silence suppression is not used. This is because, our algorithm allows more energy to be saved when more packets are actually being sent through the duration of the call. In terms of absolute values of energy, we actually found that applications with silence suppression use less energy compared to those without it due to the fewer packets communicated. Further, the inflexibility of waking up every 100ms, with a timeout value used as well, does not allow the potential of energy savings with silence suppression to be fully realized. Thus, from the perspective of energy, applications are highly recommended to use silence suppression.

The use of GreenCall did not introduce any packet losses for these experiments and the other losses (within network and delayed packets) were less than 1%. The primary reason for no packet losses in this section was the static sleep periods we were forced to use with PSM that still left plenty of spare time for the specific end-to-end path under consideration. We look at the issue of packet losses again in the next section where we study the GreenCall algorithm through trace-driven simulations over additional end points with the PSM parameter fully configurable.

5. GREENCALL PERFORMANCE WITH CONFIGURABLE SLEEP PERIODS

In this section we study the possible energy savings achievable if the sleep period in PSM were fully configurable and not limited to sleeping only for multiples of AP beacon interval. The detrimental effect of the static nature of PSM was first pointed out in [21].

[6]We confirmed the ease of setting parameters like codecs on the open source VoIP application Ekiga.

[7]For the interested reader, the energy savings for the laptop as a whole through GreenCall can be roughly estimated based on our experiments as one-fifth of that of the wireless interface alone.

[8]Many wireless interface drivers take this timeout value parameter which specifies the amount of time a radio should wait for inactivity of the radio before going into the sleep mode. The IEEE 802.11 standard does not specify the use of this parameter, and hence we treat the use of this parameter in this section as implementation specific only.

[9]These two end points of the session are geographically separated by a large distance, and 18 network hops according to *traceroute*.

Label	Location	Avg. RTT from S0 (ms)	StdDev RTT (ms)
S0	UMass Amherst, USA	0	0
S1	UC Berkeley, USA	99	5.5
S2	Wroclaw University, Poland	127	2.8
S3	CLARA Sao Paulo, Brazil	179	2.2
S4	UST, Hong Kong	255	13.4
S5	CDAC, India	320	13.9

Figure 4: Different sites used for the evaluation of GreenCall

Changing the beacon interval is not a good option as it affects other nodes in the network as well. The authors of [15] propose modifications to PSM in the IEEE 802.11 standard to decouple power management decision points from the AP beacon interval. The recently ratified power save for multimedia (WMM-PS) mechanism added to the legacy PSM would allow the client to retrieve packets from AP without waiting for a beacon [29]. This allows the client to control its sleep periods. At the time of this writing, we did not have access to any APs based on the newer IEEE 802.11e standard that supposedly implement this feature. So we experiment with a configurable sleep period version of legacy PSM through simulations in this section when exploring possible energy savings. In this version, once a sleep period is configured, the client notifies the AP each time it goes to sleep and wakes up (this should be implemented in hardware). To change sleep periods, the client would have to execute a system call.

We ran our GreenCall algorithm over multiple actual traces of network latencies using a custom built simulator. The aim was to see how our algorithm adapts as network latencies fluctuate over a period of time. We used similar values for parameters as used in Section 4, but allow the client to specify an exact sleep period. In the simulator, the model specified in Equation 1 was used with $T_{tx} = T_{rx} = 1$ms based on measured values in [11]. The power consumption values of the WNIC in various states were obtained from published specifications of our network card. This methodology was further calibrated to give comparable results to that obtained in our implementation in Section 4 when using the same sleep periods, thus allowing projections of energy savings when different sleep periods are used.

For our trace-based simulations we collected multiple wide-area network (WAN) latencies and wireless LAN (WLAN) latencies to an AP. The sum of these gave us the trace of end-to-end latency between the two points to be used in our experiments. Decoupling the WAN and WLAN traces provides re-configurability and allows studying the effect of each individually. The WAN traces were collected from University of Massachusetts (UMASS), Amherst, MA, USA (termed site S0) to different node locations within and outside the U.S with both endpoints having ethernet access to the Internet. The chosen end points were PlanetLab nodes to represent possible network latencies to different geographic points [5]. The names of these sites along with measured network characteristics from S0 are listed in Figure 4. We collected actual one way traces in both directions between site S0 and sites S1, S2, and S3. Due to inadequate NTP clock synchronization on S4 and S5, we used the approximation of RTT/2 to these sites. WLAN latency traces were taken from a public AP at UMASS. Experiments done with traces from another AP at a coffee shop gave similar results, and hence, are not shown in this paper. Later in this section, we will evaluate the effect of more highly loaded AP's.

5.1 Selection of size of sliding window H

We ran GreenCall to consider the tradeoff between energy savings and loss rate for the trace between S0 and S1. As expected, a

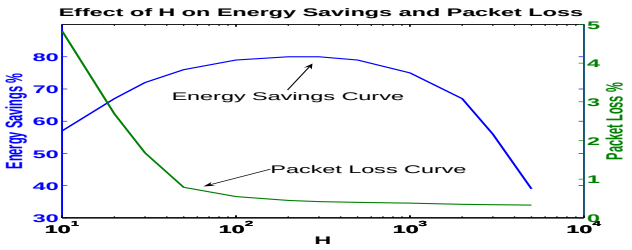

Figure 5: Energy Savings and Packet Loss rate as functions of H.

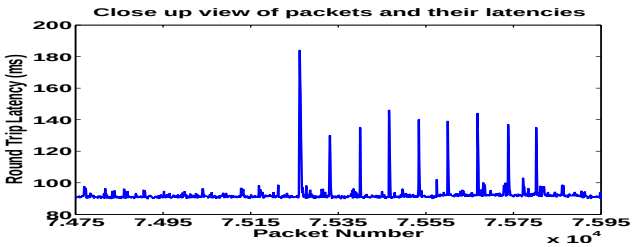

Figure 6: Close up view of the network trace from S0 to S1.

larger value of history parameter H reduces the loss rate as shown in the packet loss curve in Figure 5. A closer look at the trace under consideration in Figure 6 provides the answer to this behavior. It can be seen that the network latencies are stable most of the time with a 5-10 ms spread around the mean with a few sporadic spikes separated in time by around 100 packets. This explains why a value of H above 100 produces low packet loss rates. A more extensive measurement study by the authors of [23] shows similar network delay characteristics. It can also be seen that a value of H above 1000 provides little benefit, while requiring more usage of system memory, and thus can be set as the upper limit.

A small value of H, however, does not necessarily result in larger energy savings as shown in the energy savings curve of Figure 5. This is due to the large transition delay in executing a system call[10] for configuring a new, *different* sleep period. This transition delay incurred can significantly increase the time spent by the radio in idle mode, consuming more energy. A larger value of H forces the client to change sleep periods more infrequently and helps reduce the penalty due to switching delay. The energy savings curve shows that as H increases initially from small values, the energy savings increase with the reduction of the effects of switching delay. After a value of about $H = 100$ there are no further benefits; in fact, a further increase in H soon decreases the possible energy savings due to smaller sleep periods. Based on similar observations with other traces, we used a value of $H_{min} = 100$ in all our subsequent experiments.

Finally, we compared different approaches to adapt H (and also whether to adapt at all) based on loss rate. We compared various strategies like multiplicative increase multiplicative decrease (MIMD), additive increase additive decrease (AIAD), and multiplicative increase additive decrease (MIAD). These strategies were put to test for the first four traces (not the fifth due to its high inherent network loss rate). We specified an additional loss rate of

[10]We calculated on our wireless card a transition delay of 75 ms for the system call to take effect on the radio. The delay was measured by initiating the PSM parameter change and immediately performing a single packet ping to the AP.

0.1% due to GreenCall (over each trace's inherent network loss rate found separately) for all cases which is stringent enough to ensure adaptation is required. We found the best results, in terms of energy savings and satisfying loss rate targets, with the MIMD (1.25, 0.8) strategy with 72% average savings. To achieve the same loss rate target with a fixed H, we needed $H \geq 500$. Using $H = 500$ provided a slightly greater average energy savings of 74%. We still chose the adaptive strategy for GreenCall because it was capable of adapting for any target loss rate whereas the fixed strategy requires prior knowledge of amount of losses GreenCall is allowed to incur in addition to underlying network loss rate.

5.2 Basic experiments and results

Here we look at energy savings obtained under different settings. This provides insight on the expected energy savings with GreenCall for different scenarios. We will initially look at results for the scenario which we term as 'typical' where silence suppression was not used, packet generation interval was set to 30 ms, and peer was not running GreenCall[11]. Subsequently, we will look at results for scenarios that used silence suppression, or used a packet generation interval of 60 ms, or had the peer also running GreenCall. The results are shown in Figure 7.

The result for the typical scenario shows that greater than two-thirds of the energy consumed by the WLAN interface can be generally saved to different geographic points while keeping the packet loss rates down to tolerable levels as specified. The only exception was S5 to which the network latency was too high (introducing losses by missing playout deadline even without GreenCall) to have useful energy savings. We also observed a good balance between the important statistics of loss rate and energy savings at the client and peer. Consequently, in Figure 7 we have shown only the *smaller of the savings and larger of the loss rate at both ends* where applicable. The energy savings across the first four traces decreases very slowly in spite of large increases in average network latencies. This is because the greater latency only alters the duration of calculated sleep periods and thus number of transitions to sleep, and results in differences only due to the overhead of transitions. The total amount of time spent in sleep state is very similar for the first four traces. For the final trace, the higher latency (resulting in smaller sleep periods) coupled with greater network losses results in very conservative sleep periods by GreenCall. This was verified by the fact that the average value of H for the final trace at the end of the call was over 400, while for the first four traces it stayed at the initial value of 100. This large value of H takes into account the worst network latencies for this trace (which leave little or no spare times) over a longer period of time, possibly resulting in instances where the client does not transition to sleep at all.

The general trend is for higher energy savings when silence suppression is used. This is due to only one of the two ends sending packets, as well as periods of mutual silence where no packets are exchanged. Fewer packets minimize the time to send or collect packets to/from AP and allows transition to sleep faster. Note that this result is in contrast to the results in Section 4.2. The flexibility of sleep periods coupled with the removal of timeout parameter requirement brings forth the full advantages of silence suppression.

The performance of GreenCall with a large packet generation interval is less clear-cut. There are benefits due to fewer packets being sent and possibly smaller switching delay due to the resulting greater impact of the minimum value of H. For other traces, the larger interval reduced energy savings with the decrease of spare time as calculated in Equation 5 due to the larger T_I.

[11]These typical settings are used for all experiments in later subsections as well, with any changes specifically mentioned.

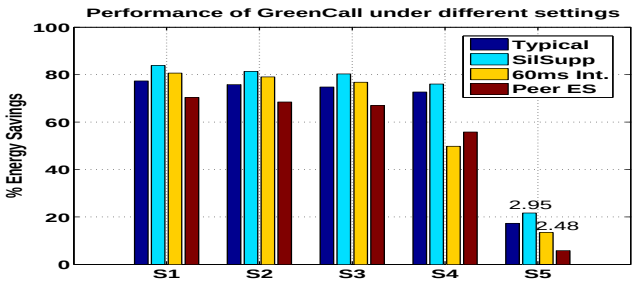

Figure 7: Energy savings with GreenCall under the typical scenario, with silence suppression, with a 60 ms packet generation interval, and peer also running GreenCall. Loss rates greater than target loss rate of 2% are shown explicitly.

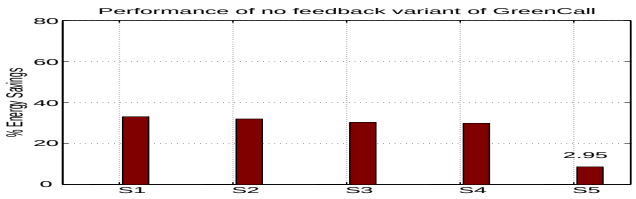

Figure 8: Performance of no feedback version of GreenCall with silence suppression. Loss rates greater than target loss rate of 2% are shown explicitly.

When the peer also ran GreenCall, the sleep periods were calculated with $C_{share} = 0.5$ instead of $C_{share} = 1$, which resulted in sleep periods of half the size. The greater number of transitions to sleep mode increased the penalty due to switching delays for changing sleep periods with PSM. Further, additional overhead is incurred in entering and coming out of sleep mode due to increase in transitions.

Discussion: The key result from the above experiments is that higher network latencies do not necessarily imply significantly lower energy savings. Energy can be saved as long as the underlying network latency is not high enough to induce losses by missing playout deadlines. Thus, energy can be saved for calls over a wide variety of paths on the Internet, not restricted to points in geographical proximity of each other.

5.3 Performance of no feedback variant

In Section 3.3 we had presented a variant of GreenCall when silence suppression is used that does not rely on feedback between the client and peer. Here we show the energy savings we can expect with this variant. The aim is to judge if any reasonable energy savings can be expected with the client and peer functioning independent of each other. The results in Figure 8 show that the no feedback version saves only about one-third of the energy of the WLAN radio. This is primarily because the client sleeps only during talk spurts of the peer, and each side talks for less than half the time for the conversation trace used (described in Section 4). These savings are impressive considering absolutely no coordination is required between the two end points, which increases the applicability of GreenCall.

5.4 Effect of asymmetric latencies

The network traces we used for experiments so far had more or less symmetric latencies from client to peer and vice-versa. To test the effectiveness of initial GreenCall latency estimate based

Trace	% Savings	% Error	Avg. Error (ms)	Max. Error (ms)	% Savings with Δ-Synch
S0→S1+0	77	0	0	0	77 (0-error)
S0→S1+5	77	0.55	-1.39	-3.44	77 (0-error)
S0→S1+10	77	1.15	-2.27	-5.98	74 (0-error)
S0→S1+20	76	4.75	-4.04	-10.91	72 (0-error)

Figure 9: % Energy savings and % errors due to asymmetric network latencies

on RTT's for calculating playout deadlines, we doctored the network trace from S0 to S1 to be asymmetric to various levels by adding a constant time to all packets of the trace in one direction. This preserves the underlying variability of latencies in the trace while introducing asymmetry. Due to asymmetric latencies, playout deadlines calculated by GreenCall are expected to be incorrect as it uses RTT/2 as its network latency estimate in Equation 3, and has no way of knowing the degree of asymmetry (without time synchronized clocks, of course, which we do not assume). We show in Figure 9 the % of packets that miss the 'actual' playout deadline of packets based on their tolerable latencies (or % error), the average error, and the maximum error among all received packets at the client (which is the side that under-estimates actual latency). The notation $S0 \rightarrow S1 + c$ denotes that a constant time c is added to to original trace from S0 to S1. GreenCall is unaffected in terms of energy savings by the asymmetry because both ends are unaware of this error in estimation of latency. As expected, there is an increase in error % and is a limitation without synchronized clocks. Interestingly, the error magnitudes are smaller than expected, and probably tolerable, mainly because the AP buffer is able to absorb much of the asymmetry. We also ran these same experiments with a known maximum time synchronization offset of $\Delta = 5$ms. As a result, the client can upper bound the actual first packet network latency allowing it to calculate playout deadlines correctly (with zero errors) in spite of the asymmetry in network latencies (which affects the RTT/2 method of estimation). This ensures that there are no errors. The down-side is that the energy-savings at the client decreases as it incorporates the greater asymmetric latency into its calculation of sleep periods.

5.5 Effect of WLAN contention

We have mainly focused on the effect of WAN latencies in the above experiments. Here we demonstrate the significant impact WLAN contention can have on GreenCall, and VoIP calls in general. Heavy WLAN contention increases both the average network latency as well as the variability of the links used in our experiments.

We collected traces to an AP with varying levels and types of surrounding traffic. The first type, termed `voip-1`, was a continuous VoIP call generated from an adjacent node to study effect of such traffic on the call in progress at the client. A 'heavier' form of contention was produced by adjacent nodes through single FTP sessions for the duration of the call. Depending on how many adjacent nodes (one or two) generated such FTP traffic, the scenarios were termed `ftp-1` and `ftp-2`. Hybrid traffic with a VoIP call from one adjacent node and a FTP session from another adjacent node was termed `voip-ftp`. Traffic without these other traffic types is also shown for reference, termed `typical`. Note that all four types of surrounding traffic levels considered are in addition to the underlying typical traffic. The level of traffic was monitored at all times through another node to ensure comparability across different scenarios. The WAN traces between S0 and S1 were used for these experiments.

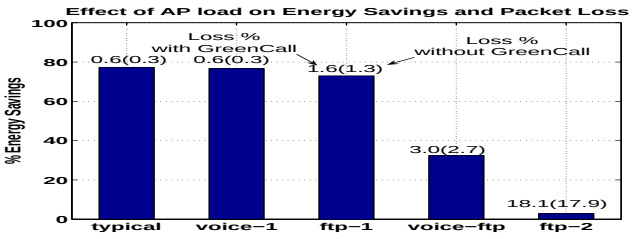

Figure 10: Performance of GreenCall under various background traffic scenarios. The number within parenthesis shows the loss rate without GreenCall running.

The results are shown in Figure 10 with loss rates with and without GreenCall shown. The performance of GreenCall was not affected much with surrounding traffic types `voip-1` and `ftp-1` due to only a small increase in average network latencies (1-10ms). For `voip-ftp` and `ftp-2` the average latency increased to 16 ms and 64 ms respectively with many packets to the AP encountering latencies in the range 100 to 1000 ms. This naturally led to those packets being dropped by GreenCall. Such high contention and resulting latencies would pose a problem for any VoIP application (looking at the high loss rates even without GreenCall). QoS guarantees at the MAC layer for VoIP traffic as provisioned in IEEE 802.11e standard is thus an approach in the right direction to handle the problem of contention for VoIP over WLAN.

6. RELATED WORK

The related work on saving energy due to the WLAN interface can be classified based on the type of traffic under consideration.

Non-VoIP Traffic: The authors of [21] presented the Bounded Slowdown Protocol (BSD) which bounds the delay to a user specified level, while maximizing energy savings. It is aimed at situations where there are long periods of user inactivity as in Web based traffic. The authors of [7] have advocated enabling knowledge of the application at the OS level; i.e. to save energy by tuning the parameters based on the intent and access patterns of applications. They specifically consider non-interactive applications like NFS, audio streaming and remote display which are more delay-tolerant than VoIP. Some other researchers propose approaches that rely on additional hardware to allow the client radio to save energy [24–26]. The work by [27] considers traffic based on audio/video streaming where the received throughput is important. Their client-side approach focuses on how the traffic can be forced to be sent in bursts from the server to allow maximum savings with PSM by adjusting the advertised TCP receive window in ACK packets. Another approach to send streaming multimedia packets in bursts from the server is presented in [6], where packets are buffered at the server and sent periodically. These approaches are suited for delay-tolerant multimedia traffic, and will not be suitable for VoIP traffic which is generated at a fixed rate and has an interactive nature. Further, with server-side buffering approaches, packets are already delayed when they arrive at the client, giving it very little leverage over amount of energy savings.

VoIP Traffic: The work in [20] presents an implementation of an energy saving algorithm for VoIP over wireless ad-hoc networks. They rely on turning the radio off and back on between VoIP packets to save energy. They observe that the transition time to and from the off mode is too large for all practical purposes and use simulations with small transition times for their results. In our work, we rely on using PSM of the IEEE 802.11 standard to switch the radio

between the low power sleep state and high power idle (on) state, without actually having to turn the radio off at any time. Moreover, our focus is specifically on infrastructure WLANs, with the peer of a VoIP session possibly any number of hops away on the Internet. A measurement based study on power consumption of a WiFi based phone was presented in [14]. The authors measure power consumption incurred for tasks like scanning, roaming, receiving beacons and draw inferences for VoIP applications. This work looks at energy from a micro-level perspective focusing only on the one hop network with the AP; similar micro-level work on the topic has been done by the authors of [12, 28]. Our work on the other hand looks at energy from a macro-level perspective focusing on end-to-end (multiple hops over the Internet) characteristics of a VoIP session. Micro-level studies would be complementary to our work. The work by the authors of [8] present the UPSD scheme where a node can collect any packets buffered at the AP when it wakes up and can go back to sleep without having to explicitly announce (frame exchange) to the AP its intentions to sleep. The UPSD scheme has been incorporated in the WMM-PS [29] scheme that was recently added to legacy PSM in newer standards like 802.11e. While this approach reduces the latency to receive packets by waking up more frequently than when using GreenCall, this scheme would have only limited utility when VoIP traffic uses silence suppression where there are no uplink data packets to trigger downlink frames. Though there are no AP notification overheads to transition in and out of sleep state, the smaller sleep periods could incur additional energy for the frequent transitions between idle and sleep mode. This energy for state transitions was not considered by our simple energy model in this work.

7. CONCLUSIONS

We have addressed the important problem of saving energy for mobile clients due to the wireless interface during VoIP calls. We presented the GreenCall algorithm that leverages the IEEE 802.11 PSM mode to save energy consumed by the wireless radio while at the same time ensuring that application quality is preserved. Through extensive evaluations, both through experiments and trace-based simulations over diverse Internet paths, we showed the utility of GreenCall and underscored the great potential of saving energy even with real-time applications like VoIP.

Acknowledgment

This work is supported in part by NSF grant CCF-0329794. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

8. REFERENCES

[1] CNET News. Wi-Fi phones make a splash. Can be found at http://news.com/Wi-Fi+phones+make+a+splash/2100-7351_3-5296745.html

[2] San Francisco Chronicle. New T-Mobile plan cuts costs by using Internet. Can be found at http://sfgate.com/cgi-bin/article.cgi?f=/c/a/2007/06/27/BUGG6QM5DR1.DTL

[3] Apple iPhone Technical Specifications. Can be found at http://www.apple.com/iphone/specs.html

[4] IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std 802.ll-1999, 1999.

[5] PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services. Can be found at http://www.planet-lab.org/.

[6] J. Adams and G.-M. Muntean. Adaptive-buffer power save mechanism for mobile multimedia streaming. ICC 2007.

[7] M. Anand, E.B. Nightingale and J. Flinn. Self-Tuning Wireless Network Power Management. MobiCom 2003.

[8] Y. Chen, N. Smavatkul and S. Emeott. Power Management for VoIP over IEEE 802.11 WLAN. WCNC 2004.

[9] W.-H Chiang, W.-C Xiao and C.-F Chou. A Performance Study of VoIP Applications: MSN vs. Skype. MULTICOMM 2006.

[10] L. Ding, R.A. Goubran. Assessment of effects of packet loss on speech quality in VoIP. HAVE 2003.

[11] L. M. Feeney, M. Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad-Hoc Networking Environment. INFOCOM 2001.

[12] B. Gleeson et. al. Exploring Power Saving in 802.11 VoIP Wireless Links. ICCMC 2006.

[13] S. Guha, N. Daswani and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. IPTPS 2006.

[14] A. Gupta, P. Mohapatra. Energy Consumption and Conservation in WiFi Based Phones: A Measurement-Based Study. SECON 2007.

[15] C. Hu, R. Zheng, J. Hou and L. Sha. A Microscopic Study of Power Management in IEEE 802.11 Wireless Networks. IJWMC,Vol. 1, Nos. 3-4, 22 February 2007 , pp. 165-178(14).

[16] ITU-T Recommendation P.59. Artificial Conversational Speech, March 1993.

[17] ITU-T Recommendation G. 107. The E-model, a computational model for use in transmission planning, 2003.

[18] ITU-T Recommendation G.109. Definition of categories of speech transmission quality, 1999.

[19] ITU-T Recommendation G.114. One Way Transmission Time, 2003.

[20] J. Kotwicki. An Analysis of Energy-Efficient Voice over IP Communication in Wireless Networks, Master's Thesis, Case Western Reserve University, March 2004.

[21] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. MOBICOM 2002.

[22] A. Mahesri and V. Vardhan. Power Consumption Breakdown on a Modern Laptop. PACS 2004.

[23] A. Markopoulou, F. Tobagi, and M. Karam. Loss and Delay Measurements of Internet Backbones. Computer Communications. Vol. 29, Issue 10, 19 June 2006, Pages 1590-1604.

[24] P. Shenoy and P. Radkov. Proxy-Assisted Power-Friendly Streaming to Mobile Devices. MMCN 2003.

[25] E. Shih, P. Bahl and M. J. Sinclair. Wake on Wireless: An Event-Driven Energy Saving Strategy for Battery Operated Devices. MOBICOM 2002.

[26] Suresh Singh and C.S. Raghavendra. PAMAS: Power Aware Multi-Access protocol with Sig- nalling for Ad Hoc Networks. ACM CCR, Vol. 28, No. 3, July 1998, pp. 5 - 26.

[27] E. Tan, L. Guo, S. Chen, and X. Zhang, PSM-Throttling: Minimizing Energy Consumption for Bulk Data Communications in WLANs. IEEE ICNP 2007.

[28] S-L. Tsao, C-H. Huang and T-M. Lin. Energy Conserving Packet Transmission Schemes for Video and Voice over WLAN. CCNC 2006.

[29] WMM Power Save for Mobile and Portable WiFi Certified Devices. WiFi Alliance, December 2005.