Energy-Efficient VoIP over Wireless LANs*

Vinod Namboodiri†, Member, IEEE, Lixin Gao‡, Senior Member, IEEE

Abstract—Emerging dual-mode phones incorporate a Wireless LAN (WLAN) interface along with the traditional cellular interface. The additional benefits of the WLAN interface are, however, likely to be outweighed by its greater rate of energy consumption. This is especially of concern when real-time applications, that result in continuous traffic, are involved. WLAN radios typically conserve energy by staying in sleep mode. With real-time applications like Voice over Internet Protocol (VoIP), this can be challenging since packets delayed above a threshold are lost. Moreover, the continuous nature of traffic makes it difficult for the radio to stay in the lower power sleep mode enough to reduce energy consumption significantly. In this work we propose the GreenCall algorithm to derive sleep/wakeup schedules for the WLAN radio to save energy during VoIP calls while ensuring that application quality is preserved within acceptable levels of users. We evaluate GreenCall on commodity hardware and study its performance over diverse network paths and describe our experiences in the process. We further extensively investigate the effect of different application parameters on possible energy savings through trace-based simulations. We show that, in spite of the interactive, real-time nature of voice, energy consumption during calls can be reduced by close to 80% in most instances.

Index Terms—Voice over IP (VoIP), Wireless LANs, Energy Consumption, Portable Communication Devices, Internet

I. INTRODUCTION

Dual mode phones like the Apple iPhone and RIM Blackberry are an emerging trend with both a cellular and a Wireless LAN (WLAN) interface 1. Apart from data access, the WLAN interface can also be leveraged for making Voice over Internet Protocol (VoIP) calls. This offers two advantages over traditional calling over the cellular interface: (i) Calls over the Internet through WLANs are more cost-effective, and (ii) These calls are not affected by lack of coverage of the cellular network in some indoor areas like the office or home or in certain outdoor areas.

The caveat, however, with using the WLAN interface is that now energy consumption is of greater concern. An active or even idle wireless network interface (WNIC) is a significant drain on the relatively limited capacity batteries found in smart phones. For example, the specifications of Apple’s iPhone lists a talk time of 14 hours with the cellular interface, but no more than 6 hours of operating lifetime with the WLAN interface for very light web browsing and email access. The D-Link V-Click dual mode smart phone lists a talk time of 5 hours with the cellular interface, and only 2 hours of operating lifetime with the WLAN interface 2. Subjecting these devices to real-time applications like VoIP would further significantly reduce the talktime due to the much heavier workload. Previous studies suggest that for high end devices like laptops, at least 15-20% of the total energy capacity is consumed by an active WLAN interface 3, while for low end devices like a PDA, this number increases to about 65% of the total energy consumption 4. Reducing the energy consumed by the WLAN interface for VoIP calls is thus a critical step towards extending the operating lifetime of these mobile devices when utilized for such applications.

For applications with long periods of inactivity like Network File Systems (NFS), or those with large tolerable latencies of the order of seconds like web browsing, energy can be saved by letting the WLAN radio stay in the low power sleep mode frequently and for long periods of time 5, 6. Adopting this approach with real-time applications, however, is more challenging. VoIP has tolerable latencies of the order of only hundreds of milliseconds, and any greater delay induced by periodic transitions to the sleep mode would degrade the quality of the call beyond the user’s tolerable limits. Packet generation intervals of the order of tens of milliseconds exacerbate the situation by making it difficult for the radio to spend any significant amount of time in the sleep mode.

In this paper we address the issue of reducing energy consumption of the WLAN interface during a VoIP call while preserving the quality within acceptable levels. Our approach is based on using an algorithm that can be implemented as a software solution to saving energy. This solution would work with legacy interfaces whose radios may not be as power-efficient as those of emerging interfaces. Further, when improved hardware solutions emerge that are more power-efficient, our approach would complement such advances. Our contributions can be summarized as the following:

1) We propose the GreenCall algorithm that conserves energy during VoIP calls over WLANs. Our algorithm saves energy by computing sleep/wakeup schedules that allow the radio to stay in the low power sleep mode for significant periods of time during a call. These schedules are computed keeping in mind the maximum delay users can tolerate in their conversations. 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 trace-driven simulations as well as experiments on commodity hardware/software. We evaluate our algorithm on commodity hardware and quantify the energy saved between widespread geographic points of the Internet and describe our experiences of the process. Through trace-based simulations, we further evaluate the effectiveness of our algorithm for various configurable parameters of VoIP applications. We demonstrate that, with our algorithm, close to 80% energy savings can be achieved on most paths during a call. More importantly, from a fundamental perspective, our evaluations show that significant reduction of the energy consumed due to the WLAN interface can be achieved even for real-time interactive applications like VoIP.

† Affiliated to Department of Electrical Engineering and Computer Science, Wichita State University; can be contacted at vinod.namboodiri@wichita.edu
‡ Affiliated to Department of Electrical and Computer Engineering, University of Massachusetts Amherst; can be contacted at lgao@ecs.umass.edu
 Users typically spend greater than 30% of their time at their homes or offices.
The organization of this paper is as follows. In Section II we provide some background on saving energy that is wasted in the idle mode of the WLAN interface and describe related work. Section III considers the end-to-end perspective of a VoIP call in saving energy consumed due to the WLAN interface and formulates the problem we address in this paper. In Sections IV and V we develop the solution to our problem and describe how our algorithm computes energy-efficient sleep/wakeup schedules. Details of our GreenCall algorithm and its pseudocode is then presented in Section VI. In Section VII we describe our experiences of evaluations of GreenCall on commodity hardware/software. Section VIII presents the evaluation results of trace-driven simulations that analyze the performance of GreenCall over diverse network paths in detail for various VoIP application configurations. Section IX discusses some practical aspects of the integration of GreenCall with existing VoIP clients and the expected utility of GreenCall for a typical user. Concluding remarks are made in Section X.

II. BACKGROUND AND RELATED WORK

We begin this section by providing some background on how energy wastage by time spent in the idle mode of the WLAN interface can be minimized. Subsequently, we will look at related work that saves energy in the idle mode classified based on the type of traffic under consideration: Non-VoIP traffic or VoIP traffic.

A. 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 by reducing the time spent in idle mode. There is typically about one order of magnitude difference between the power consumption in the idle and sleep states. 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., [9], [10], [11]).

A standardized solution to this issue is the Power Save Mode (PSM) which was introduced in the IEEE 802.11 standard for infrastructure WLANs [12]. 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. 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). This beacon is sent every beacon interval (BI) and is a pre-configured value at the AP.

Each node on receiving notification of buffered packet(s) within a beacon, can leave sleep mode and request buffered packets from 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 notifies the AP and the latter does not buffer packets destined to the client anymore. 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 support PSM. Using PSM obviates the need for any supplementary devices to reduce the energy consumption of the WNIC. As explained next, there have been important additions to PSM in recent years; thus, throughout this paper we refer to the scheme outlined above as legacy PSM.

An important change in PSM with the ratification of the IEEE 802.11e standard has been the introduction of the Unscheduled Power Saved Delivery (UPSD) mode. In the UPSD mode an uplink packet from a node to its AP triggers the downlink of any buffered packets at AP to the node. This mode takes advantage of the fact that a node’s radio wakes up from sleep to send its packets, and thus can receive packets as well at those times without having to wait for notification in the next beacon sent from the AP. Thus, the UPSD mode works in the same fashion as legacy PSM explained above except for the triggered packet delivery from AP. This allows a node to receive any buffered packets at the AP directly without having to first wait for notification in a beacon. Due to only minor differences in the legacy PSM and UPSD schemes, any reference to PSM in this paper implies the basic mechanism of going to sleep mode and waking up every beacon interval (BI) to check for buffered packets at the AP which is common for both schemes. We will make the reference explicit when necessary during our discussion of implementation on commodity hardware in Section VII.

On a related note, power management is well supported in other technologies like GSM through time division schemes, and IEEE 802.16e (WiMax) through similar power saving modes. The support provided in legacy IEEE 802.11 was not fine-grained enough, but this situation has improved with the more recent version of the standard with the introduction of the UPSD mode and related modifications.

B. Related Work

We bifurcate related work based on the type of traffic under consideration - those that have dealt with saving energy in the idle mode with non-VoIP traffic, and those that have specifically worked with VoIP traffic. We also briefly discuss related work done to gather Wide Area Network (WAN) latencies between different points. Network latency characterization is a critical aspect of our algorithm as it is designed to work over a large number of hops between the two ends of a VoIP session over the Internet.

We begin by providing an overview of work done with non-VoIP traffic. The authors of [8] 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. A similar approach of saving energy during periods of inactivity of a cellphone was presented in [13], [9]. In their work, a lower power radio is used to wakeup a higher power radio when calls arrive, while the higher power radio stays in sleep mode between calls. Thus, the focus of saving energy is between calls and not during calls like our work. 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 [14], [9], [10]. The work by [15] 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 [16], 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. The work by [17] is a scheme designed to allow the radio interface to transition to lower power sleep modes for idle periods much smaller than those supported by the legacy PSM standard (Section II-A), and is designed to work with any data traffic. This work relies on some modifications to the IEEE 802.11 standard and runs on custom-built hardware, and does not incorporate the recent changes to the standard like the UPSD mode. Our work on the other hand is targeted specifically to VoIP and its characteristics, works with commodity interfaces, including the most recent ones and compares and contrasts our contribution with the UPSD mode designed for real-time traffic like VoIP.

Now, we describe more general work done previously for saving energy due to WLAN interface specifically with VoIP traffic. The work in [18] 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 [19]. 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 [20], [21]. 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 [22] present the UPSD mode 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 mode has been incorporated in the WMM-PS [23] scheme that was added to legacy PSM in recent standards like 802.11e as explained before. While this approach reduces the latency to receive packets by waking up more frequently than when using GreenCall, the energy savings with this scheme is still significantly smaller than that of our algorithm. We compare this approach with our proposed GreenCall algorithm later in this paper.

Latency characteristics between different locations on the Internet plays an important role on VoIP call quality. There have been many studies on capturing latency characteristics, with some directly focusing on the impact on VoIP, and thus proposing algorithms to enhance the VoIP experience. A recent work by the authors of [24] provides an in-depth measurement study of delays on the Internet backbones and how expected behavior can be modeled. The work in [25] provides classical techniques for estimating network delay when used with VoIP applications. The work in [26], [27] focus specifically on the impact of delay characteristics of the Internet on VoIP traffic. Corresponding adjustments required during audio playout are discussed in [28].

### III. PROBLEM DEFINITION

Based on the background provided in the earlier section, we state the problem that needs to be solved in this section. We begin by studying the latency components of a VoIP call from an end-to-end perspective and then provide a more formal problem statement.

#### A. 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.

#### B. 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

Fig. 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**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAN Network Delay</td>
<td>Latency from the peer to the AP.</td>
</tr>
<tr>
<td>AP Buffering Delay</td>
<td>Latency from the AP to the client.</td>
</tr>
<tr>
<td>AP to Client Wireless Delay</td>
<td>Latency from the client to the peer.</td>
</tr>
<tr>
<td>Mouth-to-Ear Delay</td>
<td>Total latency experienced by the client.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Using PSM to save energy during a VoIP call</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

#### B. 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

<table>
<thead>
<tr>
<th>Problem Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. Using PSM to save energy during a VoIP call</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

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

For the whole call, we seek \( \Gamma \) 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.

The challenge involved in the design of GreenCall can be better understood by looking at some typical numbers of the parameters involved. Network latency between sites vary from 0-1000ms based on geographic distance and/or characteristic of the path. Tolerable latency for VoIP is widely used as 100-300ms. Latency to communicate between the client to its AP varies from 0-20ms based on contention on the medium. These numbers vary due to a number of factors which will be discussed at various places throughout the rest of the paper.

IV. 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 III-B. 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.

A. 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

2For 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 legacy PSM. The client notifies the AP before sleeping and on wakeup. We will address the practical implications in Section III-B.

3By each packet’s network latency we mean the latency the packet will suffer in traversing the network path from one end to the other.
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.

1) Playout deadlines: Let \( t^i_p \) 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 \( t^i_s - T^i_{pktz} \), where \( T^i_{pktz} \) is the constant encoding and packetization delay. Let \( t^i_a \) be the time at which packet \( i \) arrives at the client and \( t^i_p \) be the time by which it has to be played out (also called playout deadline). Let \( t^i_{pe} \) be the estimated network latency from peer to the client of the first generated packet, and let \( l_{ac} \) be the estimate of network latency of first generated packet 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 generated, or

\[
t^i_p = t^i_s - T^i_{pktz} + T_L,
\]

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

\[
t^j_s = t^i_s - t^i_{pe} + (m - 1)T_I,
\]

where \( T_I \) is the constant packet generation interval between successive sequence numbered packets, and \( m \) is the sequence number of a packet.

2) Network Latency: 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

\[
l^i_ac = t^i_p - T_{pb} - t^i_a,
\]

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.

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, \( l^i_ac \) 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 \( \gamma^k \) is the last used sleep period, \( k \in 1, \ldots, n \). If the first packet comes earlier, our reconstructed 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 the first packet received at client after a sleep period (also referred to as a significant packet henceforth).

\[
l^k_ac = l^k_ac + \max(0, \gamma^k + 2l_{ac} - T_I),
\]

where \( f(k) \) is a mapping from the first packet received after \( k \)-th sleep period \( \gamma^k \) to received packet number \( i \). The above equation gives an upper bound on the actual spare time of the first packet in the AP buffer. Thus, to summarize, we calculate the spare times of only selected significant packets through a call and will use these for the calculation of sleep periods. Knowing the playout deadline and spare times of packets, we can calculate the network latency (after accounting for encoding and packetization delay at the sender and decoding and playout delay at the receiver) of each significant packet as

\[
l_k = l^k_ac - T_{pktz} - T_{pb},
\]

3) Sleep Periods: Finally, we will use the playout deadlines and spare times calculated for the significant packets to derive sleep periods at various decision points through the call. A decision point can be defined as the time when the client has to calculate the next sleep period to use after receiving packets from the AP buffered during previous sleep period. If the client knew the network latency \( l_{k+1} \) of the next arriving packet \( k+1 \)th sleep period to use as

\[
\gamma^{k+1} = l^{k+1}_ac - 2l_{ac} - f^{(k+1)}_p - l_{k+1} - T_{pktz} - T_{pb} - 2l_{ac}.
\]

However, network latency of arriving packets is unknown - thus we need to predict a bound for the value \( l_{k+1} \) from above for the packet arriving after the decision point using the estimated network latencies of previously received packets calculated by Equation 4. We will describe our approach to predict a bound on network latency in Section 4. Another point of note is that 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 are described in the next section.

B. 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 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 $\gamma$ and knows the other end to be running an energy saving algorithm, it uses a sleep period of $C_{\text{share}} \cdot \gamma$ where $C_{\text{share}}$ 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 as in Section IV-A 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 implementation details are explained with the presentation of our GreenCall algorithm in Section VIII-C. The implications of our choice in terms of results and subsequent discussion are presented in Section VIII-C.

V. Bounding Network Latency and Adaptation with Loss Rate

In the previous section we showed how sleep periods can be computed through the call if we have a bound on future network latencies. In this section we look at how we can bound future network latencies through a systematic study of actual network latency characteristics. We will also present an adaptive approach to deal with cases where this bound does not hold and ensure that ensuing packet losses are controlled. We begin by giving details of the collection and description of the dataset we use to study how we can predict bounds on network latencies.

A. Network Latency Dataset

We collected multiple wide-area network (WAN) latencies and wireless LAN (WLAN) latencies to an AP. The sum of these gave us traces of end-to-end latency between any two end points. 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. The names of these sites along with measured network characteristics from S0 are listed in Figure 3. We collected actual one way traces in both directions between site S0 and sites S1, S2, and S3. Due to inadequate Network Time Protocol (NTP) clock synchronization on S4 and S5, we used the approximation of half the round trip time, or RTT/2 to these sites. WLAN latency traces were taken from a public AP at UMASS during the daytime. The number of users on the network varied from 10-15 users at the times the traces were collected with mainly HTTP traffic, and a few TCP/IP control messages. We term the collected dataset as UMASS5. The latency traces were collected at intervals of 30ms and 60ms as these are typical packet generation interval for VoIP calls and will reflect how network latencies evolve during a call. In this section we mainly study the latencies gathered for the 30ms interval. VoIP calls last for about 12 minutes which translate to greater than 20,000 packets for a 30ms packet generation interval. Thus, for our studies in this section we look at traces of 20,000 packets.

B. Network Latency Characteristics

As the call proceeds, estimates of previous network latencies are available for making predictions about bounds on future network latencies. Thus, the first step is to study how accurately previous latencies reflect future network latencies. We studied how often the next network latency exceeds the $P$-th percentile of previous network latencies. Thus, for a given $P$, we have statistical information on how many times the next latency will exceed the $P$-th percentile value (also called error rate or loss rate because packets exceeding this bound will be dropped as sleep periods are calculated based on the bound). Instead of considering all previously seen latencies while calculating the $P$-th percentile, we consider only the last $H$ network latencies of packets. This sliding window of $H$ latencies allows incorporation of recent events affecting latencies while discounting other prior events. This sliding window approach is storage and computation efficient a well.

We study network latencies of all traces in our UMASS5 dataset with the results shown in Figure 4. Each subplot is shown for a different value of $H$ and considers the effect of value of $P$ on loss rate for 20,000 packets of each trace. As expected, increasing values
of $P$ decrease the loss rate. When $P = 100$, i.e. the maximum latency of last $H$ packets is used as the latency bound for next packet, the loss rate is smallest. For all five traces, the variation in loss rate with $P$ is very similar. Increasing value of $H$ results in smaller loss rates as more historical information is used in calculation of bounds. However, looking at the evolution of loss rates with increasing $H$ across sub-plots, it can be seen that benefits of larger $H$ decreases fast, and a value of $H = 500$ provides barely any reduction in loss rate for the traces compared to $H = 375$. Thus, we can fix the value of $H = 500$ for calculation of network bounds. A look at Figure 5 showing network latencies from a 84 minute trace between nodes at UMASS Amherst and UC Berkeley provides the answer as to why this value of $H$ suffices to reduce loss rates to a small value. The latencies are fairly constant around a base value with small variations, with spikes every 100-200 packets. This can be more clearly observed from the corresponding zoomed snapshot in Figure 6. Thus, values of $H$ greater than 200 packets incorporate enough ‘important’ events to reduce loss rates.

It would be natural to think that using $P = 100$ would be the best strategy to bound network latency values with small loss rate. However, $P = 100$ also results in conservative latency bounds which reduces the size of sleep periods calculated based on these bounds, and hence reduced energy savings. Latency bounds for $P = 98$ or $P = 99$ offer less conservative latency bounds at the expense of additional loss rate (Figure 4). The comparison of values of error from predicted bounds for values of $P = 98$, 99, and 100 are shown in Figure 7 for $H = 500$. After a latency spike that exceeds the bound for $P = 100$ (shown as a negative spike representing under-estimation), the value of the error spikes immediately for the next $H$ packets as it uses the maximum latency seen as the bound. The bounds for $P = 98$ are the least conservative as it can discount some of these spikes when they fall above the 98-th percentile. Thus, given a target loss rate $LR$ by the user, the value of $P$ chosen should be the smallest that has a loss rate less than $LR$. Or put more formally, $P_{chosen} = min \forall P | L_P \leq LR$, where $P_{chosen}$ is the selected value of $P$ and $L_P$ is the associated loss rate for the value. Considering the UMASS5 dataset as an example (with only integral values of $P$), with $LR = 2\%$, we require $P \geq 99$ to satisfy this loss rate for all traces. As $P = 99$ is less conservative, that should be the choice for providing latency bounds compared to $P = 100$. In general, we expect most network latencies between any two sites to have similar variation in loss rate with latency bounds as those shown here. This observation is supported by other studies of Internet delay characteristics (e.g. [24]). For cases where the loss rate variation differs significantly, we introduce an adaptive property to the calculation of latency bounds as explained next.

C. Adaptation of Latency Bound with respect to Loss Rate

Note that the sleep period calculation in Section IV relies heavily on estimates of many parameters; when these estimates are incorrect, there are either missed opportunities to save energy, or packets are lost. Further, the bounds on latency for a certain value of $P$ may have a larger than expected loss rate for some trace, and it is important to adjust to such scenarios. We thus rely on adding or subtracting a shift value $S$ from the current latency bound to control how conservative or aggressive the predicted bound is during the call. If the current loss rate is greater than the user specified loss rate $LR$, the value of $S$ can be increased. Conversely, when the current loss rate is smaller than user specified loss rate, $S$ can be decreased. This adaptation helps balance the tradeoff between energy savings and loss rate.

For adaptation of $S$ we considered two alternatives to adjust its value - additive increase additive decrease (AIAD) and multiplicative
parameters defined by the application as well as tunable parameters of the algorithm. The final step of this phase is to estimate the one

This provides a balance between adapting too frequently to observe the effect on loss rate versus adapting too rarely to not have enough control on loss rate. We also considered scaling the latency bounds by a constant value instead of adding a constant value as above but found that it did not perform as well in terms of the two criteria mentioned earlier to judge the effectiveness of the adaptive scheme. We explain some more details of the adaptation in the next section where we introduce the GreenCall algorithm.

VI. GREENCALL ALGORITHM

Having described our approach to calculate sleep periods, in this section we present the complete GreenCall algorithm to derive sleep schedules during the call. GreenCall handles unknown network latencies by keeping track of latencies suffered by previous packets received at client and predicts latency bounds for future network latencies, based on which subsequent sleep periods are calculated. The magnitude of shift value $S$ used to for network latency bound depends on the current loss rate. Consequently, at higher loss rates, more conservative sleep periods are used. This enables a smooth tradeoff between loss rate and energy savings and is the main feature of the algorithm. 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.

The GreenCall pseudocode is presented as Algorithm 8. 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 first packet 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 latency estimate. To account for variability, these estimations are collected over a series of 10 packets with the second to largest 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. Note that this calculation is done only for the first packet after a sleep period as this signifies a decision point of the algorithm from where the next sleep period is derived. At this decision point, only the estimated spare time (as described in Section 14) of this packet is utilized along with values of known application constants. Subsequently, once the AP has no more packets buffered for it, the client goes to sleep for duration $\gamma_k$, the duration of $k$-th sleep period. This duration $\gamma_k$ considers whether the peer is running GreenCall as well through the use of a constant $C_{share}$ as described in Section 14. 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 $S$. Large values of $S$ will result in more conservative sleep periods (minimizing packet losses).
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 $S$. 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 $S$ as explained in Section VII. The monitoring begins after a minimum number of packets, $C_{\min}$, have been received, and is done every $C_{\text{interval}}$ packets thereafter. $\tau_1$ and $\tau_2$ 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 $S$. Once the maximum $S$ 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 $S$ is done through two constant factors: $C_{\text{incf}}$ to increase it and $C_{\text{decf}}$ to decrease it whose values we discussed in Section VII along with the value of $C_{\text{interval}}$. The values for other constants we used in the evaluations that follow are $C_{\min} = 100$, $f_b = 5$, $S_{\text{max}} = 100$, $\tau_1 = 0.05$ and $\tau_2 = 0.1$.

**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 and the client tries to save energy only when it is not sending packets. Thus, when it is the turn for the client’s user to talk, all packets generated are sent with no attempts made to transition to the sleep mode until it is the turn of the peer user to talk. The sleep period calculations are done similarly as listed in GreenCall algorithm shown in Figure 8. Because the client attempts to transition to sleep only at the time when the peer user is talking (as opposed to the typical case where it attempts to sleep at all times through the call), this approach will reduce the amount of energy saved, but will 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 $C_{\text{share}} = 1$. We compare the possible energy savings with this variant of GreenCall in our evaluations in Section VII.C.

**VII. Evaluation through Emulation**

In this section we discuss our experiences of evaluating GreenCall with commodity hardware/software through emulation. We study the specific case of no silence suppression with a packet generation interval of 30ms. We discuss how sleep periods in GreenCall can be set by the client. Specifically, we will describe why legacy PSM is not suited for this purpose and how the 802.11e standard’s UPSD mode can be utilized. We also motivate why GreenCall is a better alternative to save energy during VoIP traffic than the UPSD mode.

---

1During 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.

2Only the VoIP traffic is emulated. We have implemented the rest of the system on commodity hardware. The reason for emulating VoIP traffic is explained in Section VII.C. Practical aspects of integrating with VoIP clients are discussed in Section IX.

3We confirmed the ease of setting parameters like codecs on the open source VoIP application Ekiga. More details are provided later in Section IX.

We will do a more detailed study of how much energy can be saved with other packet generation intervals and scenarios in the following section through trace-based simulations.

**A. Experimental Setup**

We begin by describing our VoIP emulation setup and the challenges faced in implementing flexible sleep periods with GreenCall in commodity hardware, and how we overcome it. Next, we will describe the power measurement setup that is used to quantify energy savings of GreenCall. This will be followed by describing the user quality specifications for a call.

1) VoIP Emulation: The client was a Linux based laptop with a Belkin IEEE 802.11n wireless card operated through the RT2860 linux driver from Ralink Tech. Almost all 802.11n based drivers allow both the legacy PSM mode as well as 802.11e UPSD mode. We chose this card for its ease of configurability in Linux. For the following experiments, a 12 minute conversation is used which is the average length of a VoIP session [30]. 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 $T_1$ was set to 30ms or 60ms which are typical voice frame packetization rates. When the radio was in sleep mode, the generated packets were buffered and sent after the sleep period elapsed. The packetization delay $T_{pkdtz}$, and decoding and playout delay $T_{pd}$ were both set to 20ms. The client was kept at UMass Amherst while the peers were PlanetLab nodes that were used to collect our UMASS5 dataset shown in Figure 3. These pairs of nodes were expected to provide good insights on how much energy savings can be expected over diverse paths on the Internet owing to differences in proximity and geographic locations.

2) Legacy PSM Limitation and Leveraging 802.11e: The sleep periods calculated by the GreenCall algorithm can vary over a wide range. However, legacy PSM dictates that sleep periods specified by nodes be multiples of the AP beacon interval (BI), which is typically 100ms. This also means that GreenCall cannot set sleep periods less than BI which are quite common for paths between sites separated by large geographic distances like those between the U.S.A and India or China which have large network latencies as seen in the UMASS5 dataset in Figure 3. Thus, unless the 802.11 standards are modified to allow clients to set flexible sleep periods, the complete benefits of GreenCall cannot be realized.

We overcome this limitation for our evaluation by leveraging the Unscheduled Power Save Delivery (UPSD) mode in the 802.11e standard to allow for clients to indirectly control their sleep periods. As briefly explained in Section II in the UPSD mode, every uplink packet from client to AP triggers downlink packets buffered at the AP to be delivered to the client. Thus, the client can control when it receives packets from the AP by adjusting how long it buffers uplink packets. Note that this requires packets to be exchanged in both the uplink and downlink directions - this implementation of GreenCall will not work when silence suppression is used. The same limitation is also pointed out by in [23]. We will rely on our evaluations through trace based simulations to study how GreenCall works with silence suppression.
3) Power Measurement Setup: The power measurement setup (Figure 9) used was similar to that in [31], where a PCMCIA Extender (Sycard PCCEXtend 140A) was used to expose the pins of the wireless card. Voltage drop across a 1 ohm resistor in series was used to determine the current drawn, and hence the power consumption. A Tektronix TDS2002B oscilloscope was used to measure the average voltage consumption and project energy consumption for the call.

4) User Quality Specifications: The tolerable latency $T_L$ for our experiments was set at 250ms based on studies described in recommendations G.109 and G.114 of ITU-T [32], [33] 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 specified the tolerable packet loss $L/R$ 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 [34], [35]. From the perspective of Mean Opinion Score (MOS), the above parameters translate to a MOS score 3.82 for the G.711 codec, where the impairment due to codec is 0.59, while the impairment due to delay, loss and jitter amount to 0.63. The MOS score has a scale of 1-5 with 1 the worst quality and 5 the best quality. A MOS of 4 is considered good, while a MOS of 3 is considered only fair. The parameters specified for our experiments are closer to cellular quality as opposed to PSTN quality which typically has latency no more than 150ms and loss rate no more than 1%. We consider our specification reasonable for the user to allow for energy savings on the mobile device. We discuss the implications on energy savings of using more stringent specifications when we present our evaluation results in the following sub-section.

B. Energy Savings

Figure 10 shows the corresponding energy savings for all five destinations from UMass Amherst in our UMASS5 dataset for 30ms packet generation interval with no silence suppression. We took these measurements at five different times of the day and present the mean and standard deviation in the figure.

The energy savings to the first four sites are within the standard deviation of each other, giving the interesting result that smaller sized sleep periods do not necessarily result in lesser energy savings. On the contrary, we found that smaller sleep periods, like those used between S0 and S4 of ~75ms, can be just as effective. This is because the greater latency between sites 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 along with any additional overhead due to prevailing network and wireless channel conditions. The total amount of time spent in sleep state is very similar for the first four traces. As the sleep periods reduce further, like those between S0 and S5, the penalty due to greater number of wakeups starts reducing energy savings. Interestingly, we observe that the loss rate to S5 is close to 5%. The reason for this is that GreenCall was implemented using the 802.11e UPSD mode which keeps the client radio in sleep mode for at least the packet generation interval (30 ms in this case) each time before transmitting buffered packets. For a UPSD based implementation, sleep periods less than packet generation interval cannot be executed. On average, GreenCall calculates sleep periods less than this for the trace to S5 which should result in much smaller energy savings (as will be seen in our simulations in the next section). The forcible use of larger sleep periods for the trace to S5 resulted in higher energy savings, but the loss rate was much higher than the specified loss rate. Once calculated sleep periods drop below the packet generation interval, it would be better to come out of PSM altogether to reduce the loss rate. With the wireless card we performed the experiments with this, it was difficult to do without restarting the driver. This is however possible on many other cards (e.g. Intel PRO3945 ABG card).

From our measurements we found that the UPSD mode by itself saved about 40% energy as shown by the dotted line in Figure 10. The energy consumption for this mode stays almost constant for calls between any two points as the communication and sleep pattern remains the same, with differences only due to network and channel conditions (the small variations are not shown in plot for simplicity). As the AP sends all buffered packets to the client when the latter transmits every packet generation interval, the AP buffer delay is bounded by this interval. Thus, there are obvious advantages to the UPSD mode in terms of latency (as explained in more detail in [36]). However, by waking up every packet generation interval, the overhead of wakeups is much higher with UPSD than GreenCall. In GreenCall, the client buffers all generated packets for a sleep period and bursts them out at the end of the period. As the sleep periods used decrease, the difference in power consumption between GreenCall and UPSD will decrease. The difference in power consumption will become zero as the size of the calculated sleep periods decrease below the packet generation interval, which is the minimum sleep

---

10 These can be calculated from ITU G.107 and G.113 Appendix I which define the E-Model, an algorithm for calculating MOS from measured factors. A simple calculator based on these documents can be found at http://davidwall.com/MOSCalc.htm at the time of this writing. We also specified a jitter value of 10ms for these calculations.
interval of the radio in UPSD mode.

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 playback 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. This also means that more stringent specifications of tolerable delay would also save significant amount of energy. The only downside of a more stringent specification of tolerable delay is that no energy savings would be possible between end points that have latencies large enough to leave no time to sleep for the radio. For example, specifying $T_L = 150\text{ms}$ would result in no energy savings for calls between UMASS Amherst and China as the network latency of 127ms along with encoding and decoding delays (40ms) exceeds $T_L$. More insight of this key result will be provided in the following section.

VIII. Evaluations Through Trace-Driven Simulations

Our experiments on commodity hardware had limitations due to the use of IEEE 802.11e UPSD mode for implementing GreenCall. Utilizing the UPSD mode did not allow evaluations for the case of using silence suppression as uplink packets had to be sent to collect downlink packets from the AP. Also, the use of sleep periods smaller than the packet generation interval could not be implemented as the radio spends at least this much time in sleep mode for each transition when using UPSD. Thus, in this section 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, and how the duration of sleep periods corresponds to energy savings over different paths. To quantify the energy savings through the simulator we now rely on an energy model as opposed to actual measurements as in the previous section. We begin by describing our energy model followed by the results of our evaluations. The UMASS5 dataset traces were used for these experiments with the same user quality specifications as before.

![Image](image-url)

TABLE II

<table>
<thead>
<tr>
<th>Constant</th>
<th>$P_{tx}$</th>
<th>$P_{rx}$</th>
<th>$P_{idle}$</th>
<th>$P_{sleep}$</th>
<th>$t_{tx}$</th>
<th>$t_{rx}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>787 mW</td>
<td>787 mW</td>
<td>503 mW</td>
<td>44 mW</td>
<td>1 ms</td>
<td>0.5 ms</td>
</tr>
</tbody>
</table>

A. Energy Model

We allowed the client to specify an exact sleep period in the simulator. At the end of the call, with $\Gamma$ being the set of sleep periods used during the call, the energy consumption was found based on the energy model described in Equation 1 in Section III

$$E_T = P_{tx}T_{tx} + P_{rx}T_{rx} + P_{idle}T_{idle} + P_{sleep}T_{sleep}.$$  

As mentioned before in Section III the total duration of all sleep periods in $\Gamma$ uniquely determines the energy consumption of a call. This requires knowledge of power consumption in various modes as well as the time to transmit and receive a packet. We obtain these values by measuring these values on our Belkin wireless card conforming to the 802.11n draft specifications using the setup described in Section VII. The specific measured values are provided for reference in Table II where $T_{tx} = \sum_{i=1}^{U} t_{tx}$ and $T_{rx} = \sum_{i=1}^{V} t_{rx}$, where $t_{tx}$ and $t_{rx}$ are the time to transmit and receive each packet, with $U$ packets transmitted in total by the client and $V$ packets received in total by the client. For a VoIP call, given its duration, it is easy to compute $U$ and $V$ due to known packet generation intervals. The model does not take into account re-transmissions of packets which varies depending on channel conditions and traffic load of neighboring stations. Those effects can be more effectively understood from our implementation results in the previous section where energy consumed due to re-transmissions are also accounted for. We will discuss the effects of re-transmissions later in this section when we present our results.

B. Traffic Model

For our simulations, the traffic is based on constant packets generated with intervals of 30 or 60 ms. The network latency of these packets is based on gathered delay traces in the UMASS5 dataset in Figure 2. When no silence suppression is used, packets are exchanged between the client and peer constantly until the end of the call. When silence-suppression is used, we generated a trace of conversation between two people as per the recommendations of ITU-T [36] 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.

C. Results

In this section we present the results obtained by running GreenCall under different settings based on the experimental setup described in the previous section. In this section, we also examine the effect of asymmetric network paths on the playback deadline estimation accuracy of GreenCall. We conclude this section by studying the performance of GreenCall with varying levels of contention in the wireless link between the client and its AP.

1) 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. 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 3.

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. It can be seen that the energy savings are comparable to that found through our implementation, with a slight increase that can be attributed to not accounting for re-transmissions on the channel. The only exception was S5 to which the network latency was too high (introducing losses by missing playback deadline even without GreenCall) to have useful energy savings. Interestingly, it can be observed that the energy savings to S5 resulted in much higher savings in our implementations in the previous section than through the results of trace-driven

11Silence suppression is sometimes used with VoIP traffic to reduce the load on the network by suppressing packets during silent periods. This is arbitrarily employed by different application developers. For example, VoIP application from Skype does not use it while the one from MSN does.[5]

12These typical settings are used for all experiments in later subsections as well, with any changes specifically mentioned.
simulations here. The reason for this difference is that GreenCall was implemented using the 802.11e UDPSD mode which sleeps for at least the packet generation interval (30 ms in this case) even though the actual calculated sleep periods may be smaller. For all experiments we also observed a good balance between the important statistics of loss rate and energy savings at the client and peer. Consequently, in Figure 11 we have shown only the smaller of the savings at both ends, and larger of the loss rate at both ends. The energy savings across the first four traces decreases very slowly in spite of large increases in average network latencies as seen in the results of the previous section as well. For the final trace, the higher latency (resulting in smaller sleep periods) coupled with greater inherent network losses (those not due to GreenCall) results in very conservative sleep periods by GreenCall. This was verified by looking up the value of $S$ at the end of the experiment. Large values of $S$ mean that the algorithm had become increasingly conservative through the call.

To gain further insight we plotted the possible results for various possible network latency bounds used by GreenCall making use of our analysis of impact of radio transition overhead in Section II of [37] in Figure 12. These analytical results give the energy savings possible if a specific network latency value is estimated by GreenCall through a call when compared to the case when no energy saving scheme is used. The values of constants used were the same for other results in this section. This plot shows why, for network latencies less than 150-160ms, the energy savings changes very slightly with change in latency. When network latencies are greater, the number of transitions increases exponentially, thus decreasing energy savings rapidly. The implication of this behavior is that the impact of transition overhead becomes significant only when the network latencies become large enough to result in a small sleep duration to transition overhead ratio.

When silence suppression is used, with only one of the two ends sending packets or periods of mutual silence where no packets are exchanged, we surprisingly found that energy savings actually decreased slightly. We attribute this small difference to fewer packets being communicated overall. Note that the size of sleep periods does not change much with silence suppression because at least one end is talking most of the time and thus necessitates both ends to either send or receive. When network latency is very high, for example similar to that to S5, the transition overhead is reduced when silence suppression is used. This results in greater energy savings as compared to the case of no silence suppression for such sites.

The performance of GreenCall with a large packet generation interval typically results in smaller energy savings. This happens because a greater value of $T_f$ decreases the spare time calculated in Equation 3 due to greater uncertainty about the arrival time of a packet in the AP’s buffer. Fortunately, for VoIP calls, $T_f$ is typically no greater than 60ms. Due to the reasons explained above, sleep periods of half the size in this case, however, do not result in halving the energy savings.

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 overhead incurred due to notifications required between client and AP for each transition to sleep.

2) Performance of no feedback variant: In Section VI 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 13 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 VIII). These savings are impressive considering absolutely no coordination is required between the two end points, which increases the applicability of GreenCall. From a practical perspective, if silence suppression is not used, devices compliant with 802.11e standard can provide similar savings at each end without any coordination required.

3) 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 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 14 the % of packets that miss the
Effect of AP load on Energy Savings and Packet Loss

Fig. 15

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 15 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. Additional latencies on the WLAN link, for example, due to periodic handoffs cannot be solved with just QoS guarantees. However, such latencies are incurred very infrequently (order of seconds to minutes at least) compared to the communication of packets (order of milli-seconds) and would not have as much of an impact as contention with heavy bandwidth users.

We repeated the same experiments in a Coffee Shop scenario as well with 15-20 active users with similar traffic profiles and achieved similar results.

TABLE III

<table>
<thead>
<tr>
<th>Energy savings in a day with and without GreenCall during calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy (mWh)</td>
</tr>
<tr>
<td>GreenCall</td>
</tr>
<tr>
<td>No GreenCall</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Trace</th>
<th>% Savings</th>
<th>% Error</th>
<th>Avg. Error (ms)</th>
<th>Max. Error (ms)</th>
<th>% Savings with Δ-Synch</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0 → S1+0</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80 (0-error)</td>
</tr>
<tr>
<td>S0 → S1+5</td>
<td>80</td>
<td>0.45</td>
<td>-1.64</td>
<td>-3.41</td>
<td>80 (0-error)</td>
</tr>
<tr>
<td>S0 → S1+10</td>
<td>80</td>
<td>1.26</td>
<td>-2.68</td>
<td>-5.94</td>
<td>78 (0-error)</td>
</tr>
<tr>
<td>S0 → S1+20</td>
<td>80</td>
<td>4.68</td>
<td>-5.00</td>
<td>-10.91</td>
<td>75 (0-error)</td>
</tr>
</tbody>
</table>

Fig. 14

% Energy savings and % errors due to asymmetric network latencies

‘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 → 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 (up to 10ms asymmetry), 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 Δ = 5ms. 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.

4) 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 the same AP we used in the UMASS5 dataset mentioned in Section with varying levels and types of surrounding traffic over the underlying traffic of 10-15 users with mainly HTTP traffic. The first type, termed voip-1, was a continuous VoIP call generated from an adjacent node to study effect
values from our card and shows the possible energy savings by using phone. Table III summarizes our calculations based on measured
the user spends an average of 80 minutes per day talking on the
GreenCall. The first row shows the energy savings when legacy PSM
due to the wireless interface by running GreenCall with a typical
of-the-envelope calculations as to how much energy can be saved
by the wireless radio while
is a significant gain considering that saving power of the wireless
radio in the idle mode has been a well researched problem over the
years, and additional savings can only be obtained by saving power
during active communication as during VoIP calls.

In this section we discuss some practical aspects of GreenCall
implementation and its utility for an average user.

Integration of GreenCall with VoIP Clients
Here briefly discuss how the GreenCall algorithm can be work with
VoIP application software - our evaluations had only used emulated
VoIP sessions without looking at the integration aspect. For
the computation of sleep periods, GreenCall needs to get the packet
generation interval, packetization interval, and time to encode and
decode packets. These values can be obtained by knowledge of the audio
codec used through its publicly available specifications. To
test how easy it is to find information about the audio codec used for
a call, we studied the open source VoIP client software Ekiga
(formerly known as GnomeMeeting). As shown in Figure Ekiga
can be configured by the user to use any specific audio codec or
multiple possible codecs. When the two ends of the call negotiate
and decide on a codec, this information can be written into a log file
which can be read by the GreenCall algorithm. By mapping the codec
to its specified packet generation interval, packetization interval,
and encoding and decoding delays, GreenCall can compute sleep periods
for the call.

Another aspect of this integration is the ability to buffer packets
that are generated and sent at the end of sleep periods. In our Linux
based implementation, we used libipq which provides an API for
userspace packet queuing. This allowed packets to be buffered and
released at specified intervals as the algorithm runs during the call.
Similarly, libipq can be used to hand received packets over to the
application before their playout deadline.

Energy saving projections over a typical day
The main focus of our work was to save energy during VoIP
calls. The amount of energy consumed for a dual-mode or smart
phone would depend on the time spent on calls and time spent in
standby/idle mode in between calls. Here we present simple back-
of-the-envelope calculations as to how much energy can be saved
due to the wireless interface by running GreenCall with a typical
cellphone talktime profile. We use the profile calculated in[9] where
the user spends an average of 80 minutes per day talking on the
phone. Table III summarizes our calculations based on measured
values from our card and shows the possible energy savings by using
GreenCall. The first row shows the energy savings when legacy PSM
(card waking up every BI) is used when the phone is on standby and
GreenCall is used during calls. The second row shows the energy
savings when legacy PSM is used on the phone’s standby mode with
no power saving algorithm used during the call. It can be seen that
about 20% of energy savings due to the Wireless LAN radio can be
achieved during a typical day using GreenCall. Considering that
the WLAN interface is responsible for greater than 50% of the total
energy consumption of such devices (for e.g. see[38]), GreenCall


can be expected to save about 10% of the overall energy of a low-end
device like a smart or dual-mode phone, or a PDA. We believe this
is a significant gain considering that saving power of the wireless
radio in the idle mode has been a well researched problem over the
years, and additional savings can only be obtained by saving power
during active communication as during VoIP calls.

X. 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. This paves the way for
saving energy during active communication with more delay tolerant
traffic like audio and video as well.

REFERENCES
http://sfgate.com/cgi-bin/article.cgi?f=/c/a/2007/06/27/BUGG6QM5DR1.DTL
http://www.slashphone.com/115/5464.html

Vinod Namboodiri

Vinod Namboodiri is currently an Assistant Professor at the Department of Electrical Engineering and Computer Science at Wichita State University, USA. He graduated with a Ph.D. in Electrical and Computer engineering from the University of Massachusetts Amherst in 2008. His research interests include designing MAC and routing protocols, and developing energy-efficient protocols and algorithms for different wireless technologies like Wireless LANs, RFID Systems, Wireless Sensor Networks, and Wireless Mesh Networks. He has served on the technical program committees of IEEE GLOBECOM and IEEE ICC and is an active reviewer for numerous journals and conferences in the wireless networking and mobile computing areas.

Lixin Gao

Lixin Gao (S’93, M’96) is a Professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst. She received her Ph.D. degree in computer science from University of Massachusetts at Amherst in 1996. Her research interests include Internet routing, network security, network virtualization, multimedia networking, and energy efficient wireless communication. Between May 1999 and January 2000, she was a visiting researcher at AT&T Research Labs and DIMACS. She received an Alfred P. Sloan Fellow, and an NSF CAREER Award. She has served on numerous technical program committees/organizing committees of prestigious conferences such as ACM SIGCOMM, ACM SIGMETRICS, and IEEE INFOCOM, and on the Editorial Board of IEEE Transactions on Networking.