Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones

Yuvraj Agarwal[‡], Ranveer Chandra[†], Alec Wolman[†], Paramvir Bahl[†], Kevin Chin[¶], Rajesh Gupta[‡]
[†] Microsoft Research, [¶] Microsoft Corporation, [‡] University of California San Diego

ABSTRACT

IP based telephony is rapidly gaining acceptance over traditional means of voice communication. Wireless LANs are also becoming ubiquitous due to their inherent ease of deployment and decreasing costs. In enterprise Wi-Fi environments, VoIP is a compelling application for devices such as smartphones with multiple wireless interfaces. However, the high energy consumption of Wi-Fi interfaces, especially when a device is idle, presents a significant barrier to the widespread adoption of VoIP over Wi-Fi. To address this issue, we present Cell2Notify, a practical and deployable energy management architecture that leverages the cellular radio on a smartphone to implement wakeup for the high-energy consumption Wi-Fi radio. We present detailed measurements of energy consumption on smartphone devices, and we show that Cell2Notify can extend the battery lifetime of VoIP over Wi-Fi enabled smartphones by a factor of 1.7 to 6.4.

Categories and Subject Descriptors

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

General Terms

Algorithms, Management, Performance

Keywords

VoIP, cellular networks, Wi-Fi, smartphones, power management

1. INTRODUCTION

Voice-over-IP (VoIP) services are rapidly gaining acceptance over traditional circuit-switched voice communication networks such as the public switched telephone network (PSTN). Although there are many reasons behind this transformation, the two most compelling reasons are lower costs, and new functionality that is difficult to achieve with traditional voice networks. In homes, providers such as Vonage and SunRocket provide very low cost long-distance and international calling services. Skype provides free calling to other Skype users and only charges for calls to users outside the Skype network. In enterprises, VoIP also offers new functionality, especially when integrated with Wi-Fi networks: VoIP over Wi-Fi allows incoming phone calls to be automatically routed to a user's VoIP phone, regardless of where that user connects to the network. Other functionality benefits include integration with network services such as address books, file exchange in parallel with voice conversations, presence notification, video conversations, and call logging.

Simultaneously, a new class of mobile devices called *Smartphones* are gaining popularity. Smartphones integrate the functionality of PDAs and mobile phones into one device. They typically run a full-featured operating system, such as Windows CE or Linux, and most recent smartphones are equipped with multiple wireless network interfaces, such as Wi-Fi and cellular interfaces (GSM or CDMA). As smartphones become ubiquitous, users will demand the ability to use a single device for all their telephony needs. They will use their smartphone as a cellular phone primarily when on the road, and they will use it primarily as a VoIP phone when at work or at home. Therefore, VoIP over Wi-Fi has emerged as a critical application for smartphones. Vendors such as T-Mobile have recognized this trend and are in the process of rolling out new functionality that enables the handoff of calls between their GSM networks and their Wi-Fi networks [29].

One critical issue that presents a barrier to the widespread adoption of VoIP on smartphones is that of high energy consumption. In order for smartphones to receive VoIP calls over the Wi-Fi network interface, that interface needs to be on continuously. Unfortunately, the energy consumption of Wi-Fi interfaces when there is no data transfer taking place is comparable to that of when the interface is active [20, 23]. Furthermore, as we demonstrate in Section 3, the energy consumption of the idle Wi-Fi network interface, even with 802.11 power save mode enabled, vastly exceeds the energy consumption of the smartphone's GSM radio in its idle state. The better energy consumption of the GSM interface is achieved by rapid duty cycling of the GSM radio with predictable timing due to the TDMA MAC protocol, in addition to tight integration with cellular base stations. In contrast, Wi-Fi uses a distributed MAC (CSMA/CA) where devices always contend for access to the wireless medium thus leading to increased energy consumption due to excessive listening for traffic from other nodes.

In this paper, we present *Cell2Notify*, an energy management architecture that leverages the presence of multiple radios on the smartphone to reduce the idle energy consumption of the Wi-Fi radio. Cell2Notify attempts to minimize energy consumption by powering off the Wi-Fi interface when there is no VoIP call in progress, and powering it on only on the reception of an incoming VoIP call. To provide the wakeup mechanism for the Wi-Fi interface, we utilize the voice services of the GSM radio. An incom-

MobiSys'07, June 11-14, 2007, San Juan, Puerto Rico, USA.

ing ring over the GSM channel, combined with a unique caller-ID of that incoming call, serves as a unique identifier such that the smartphone can distinguish between a *wakeup* ring and a regular incoming phone call over the GSM interface. Upon reception of a wakeup ring, the smartphone powers on the Wi-Fi interface and then receives the actual incoming VoIP call.

Previous research efforts [20,23] on energy management for multi-radio devices have also investigated the idea of wireless wakeups: switching between radios on multi-radio devices to reduce the overall energy consumption. Since different radios usually have different performance and energy characteristics, these systems select the best radio for the current workload and keep other radios powered off. Cell2Notify is a continuation along this line of research, with two important distinctions. Previous approaches have faced significant barriers to deployment due to the substantial infrastructure modifications needed, whereas Cell2Notify simply requires software changes on the smartphone devices and on the VoIP proxy. There are no changes needed to the VoIP protocol (in our case SIP [21]), and no additional hardware infrastructure to deploy. Moreover, Cell2Notify is targeted at a specific compelling application of VoIP over Wi-Fi.

We present the design and implementation of Cell2Notify. We have implemented Cell2Notify on Asterisk, a commonly available open-source SIP proxy, and on Windows XP clients. Our measurements show that the additional latency imposed by our wakeup mechanism is less than two rings. Based on call logs from cell phones and office phones, we estimate that Cell2Notify can extend battery lifetime of a typical smartphone device by a factor of 1.7 to 6.4. We show the ease of Cell2Notify deployment by demonstrating a working prototype using the Cingular cellular network and Microsoft's corporate Wi-Fi network – we did not require any infrastructure changes to these networks, nor any cooperation from network administrators.

2. OVERVIEW OF A VOIP DEPLOYMENT

VoIP enables voice communication over IP-based networks, such as enterprise LANs or WANs as well as the Internet. VoIP protocols digitize voice into packets, and then send them using standard IP routing. Since VoIP does not require a dedicated and complex switching infrastructure as the PSTN does, it is much cheaper. It can also provide enhanced data services, such as video conferencing and fax at a much lower cost. In the rest of this paper, we mainly consider VoIP in enterprise LANs, although our protocols can be easily extended to work over the Internet.

We illustrate a typical enterprise VoIP deployment in Figure 1. The primary components of any VoIP deployment are a VoIP proxy server, VoIP enabled soft phones, and a VoIP gateway. The soft phones are PCs, PDAs or smartphones that are running software codecs and digitize voice packets. The VoIP proxy server acts as a rendezvous point for VoIP connections. It uses standardized signaling protocols, such as SIP [21] or H.323 [22], to establish a VoIP call between the calling parties. Once the call is connected, it is completed in a peer-to-peer fashion between the calling parties, without routing via the VoIP proxy. A typical VoIP deployment also integrates with the PSTN using a VoIP gateway. The gateway usually has an Analog Telephony Adapter (ATA) that bridges the calls between the IP-based LAN and the PSTN. In scenarios where one calling party is on the PSTN, the VoIP gateway server also plays the role of a VoIP endpoint. The VoIP proxy and the VoIP gateway services are often implemented by the same machine.

One of the most popular standards used in VoIP deployments is the Session Initiation Protocol (SIP). SIP [21] is a transport independent application-layer protocol that provides a framework for

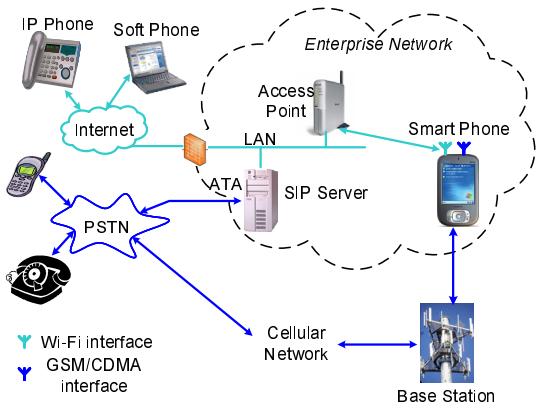


Figure 1: *A typical enterprise VoIP deployment. Outside callers can either make VoIP calls over the Internet or over the PSTN line. The SIP server uses an Analog Telephony Adapter (ATA) to translate the call from PSTN to IP and vice-versa.*

inviting end-hosts into a conversation. Similar to HTTP, SIP is a text-based protocol which makes it extremely simple, efficient and extensible. Soft phones use SIP to register with the VoIP proxy server. When the proxy receives a call for the soft phone, it sends a SIP `invite` message to the soft phone. In response, the soft phone may send a `ringing` message back to the server. When the user picks up the phone, it sends a SIP `200 OK` message that indicates that call setup is complete.

The widespread deployment of enterprise Wi-Fi networks adds an interesting dimension to VoIP in terms of support for mobility. An employee with a Wi-Fi VoIP phone can receive calls when working in a conference room or a colleague's office without relying on explicit call forwarding.

3. WIRELESS INTERFACE CHARACTERISTICS

In this section, we look at energy consumption and data transfer characteristics of different wireless interfaces. We investigate how these characteristics impact the selection of the best wireless interface to use for VoIP. In particular, we study the characteristics of two cellular data networks (GPRS/EDGE and 1xEVDO), as well as the Wi-Fi interface. We then profile the energy consumption of the entire smartphone device while performing various tasks to motivate the need for our Cell2Notify system.

3.1 Cellular Data vs. Wi-Fi

Since cellular radios are typically highly optimized to save energy, one possibility for making VoIP calls can be to use a smartphone's cellular data connection. We performed a set of measurements to investigate this alternative, and found that the cellular radio consumes significantly more power when used for data transmissions, even more so than the Wi-Fi interface. In this section, we present experimental results to show the energy consumption of two popular cellular data connections: GPRS/EDGE and 1xEVDO, and compare these numbers with the energy consumed over Wi-Fi. To the best of our knowledge, ours is the first paper that compares energy consumption of these wireless interfaces when used for VoIP communication.

We measured the energy consumption when accessing two dif-

ferent cellular data network technologies prevalent in the US, namely GSM and CDMA. The GPRS/EDGE data service is based on the GSM technology, and is offered by providers such as Cingular and T-Mobile. The 1xEVDO data service is based on CDMA and is offered by Verizon and Sprint. Since it is difficult to obtain accurate power measurements from a smartphone as we demonstrate in the next section, we used PC cards from Verizon and Cingular inserted in a laptop to obtain power measurements. For Cingular, we used the Sony Ericsson GC83 card to access their GPRS/EDGE network, and for Verizon we used the Verizon V620 card to access the 1xEVDO network. For both networks, we obtained good signal strength in the lab where we performed the experiments. For our Wi-Fi measurements, we used the commonly available Netgear WAG511 802.11a/b/g cardbus adapter.

To measure the power consumption of our PC cards, we plugged them into our laptop using a PC card extender device. The extender exposes various pins that help us in measuring the power used by the the PC card. Our setup is similar to the system used in [20, 23]. We attached a 20 m-ohm sense resistance in series with the wireless card, and measured the current through the resistor using a data acquisition system. The current multiplied by the supply voltage yields the power consumed by the PC card. We performed power measurements for three different states of each wireless card. The first is the "not connected" state, in which the cards were not connected to the data network. This corresponds to the "not associated" state for a Wi-Fi card. The second is the "connected and idle" state, in which the cards are connected to the data network but not sending any traffic. The third state is the "connected and active" state, where the card is connected to the network and is sending and receiving VoIP traffic over UDP. In our experiments, we used the popular g729 VoIP codec, which generates 50 byte VoIP packets at a data rate of 31.2 Kbps. We report the power measurements for various states of the cards in Figure 2.

As shown in the Figure, the power consumption of the V620 (1xEVDO) card is quite substantial in both the "not connected" and the "connected and idle" states. The SE-GC83 (GPRS/EDGE) interface consumes much less power in those states. The V620 utility actively tries to search for the data network, and shows the signal strength of the network even in the "not connected" state. Furthermore, it sends periodic keep-alive messages in the "connected and idle" state, and consumes significant power. On the other hand, the SE-GC83 utility does not connect unless asked to do so, and stays in a low power state when "connected and idle". Another interesting fact that is unique to the 1xEVDO radio is that the energy consumption is not as much dependent on the number of packets sent on the network as it is on the fact that the interface is switched on. This can be seen from the similar power consumed in the "not connected" and the "connected and idle" states for the 1xEVDO interface. Further, the 1xEVDO interface incurs a significant overhead in power, latency and network resources when the radio is woken up from sleep mode. Consequently, the 1xEVDO interface uses conservative policy to decide when to enter a deep sleep mode. Note that the Wi-Fi card consumes the most energy when it is not connected, as it keeps scanning for available wireless networks. The energy consumption reduces significantly when the card is connected (associated) as it enters IEEE 802.11 Power Save Mode (PSM) [12].

Of all three interfaces, the Wi-Fi interface is the *most power efficient radio during an active VoIP call*. It consumes less than half the energy of the V620, and less than 75% of the energy consumed by the GPRS/EDGE radio. This can be explained by the high transmit power used by the cellular radios to send data over much longer distances (sometimes even miles) compared to Wi-Fi, where the

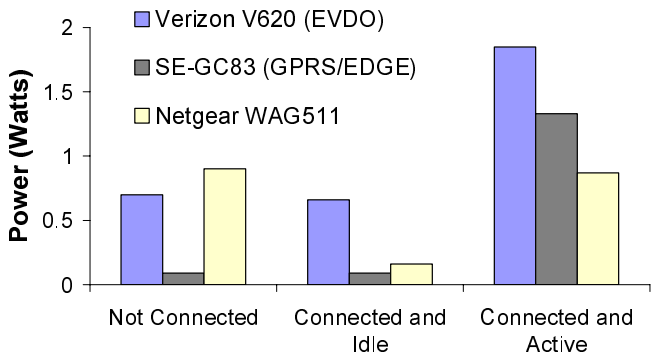


Figure 2: *Power measurements of 1xEVDO, GPRS/EDGE and Wi-Fi interfaces for different scenarios. The "Connected and Active" measurements show the power when transmitting 32 Kbps of VoIP traffic over UDP. Note that when active, VoIP over Wi-Fi consumes the least amount of battery power.*

Interfaces	Jitter (ms)	Packet Loss (%)
Verizon V620	25.25	7.6
Cingular SE-GC83	17.24	18.935
Netgear WAG511	0.9745	0

Table 1: *VoIP Quality over different network interfaces.*

AP is usually within a 100 meter distance. This is exacerbated by the strict real time requirements for VoIP and a short inter packet generation time, as a result of which the cellular radios have no opportunities to sleep and conserve energy.

Most of the power numbers we present in Figure 2 are consistent with a recent paper by Mahmud et. al. [18], which compares the power consumption of Wi-Fi and GPRS interfaces. However, in our measurements, we found that the Wi-Fi interface in the "Connected and Idle" state consumes significantly more power than what was reported in [18]. We believe our measurements are accurate as it is consistent with numbers presented in a number of related papers [2, 23]. Although it might be possible to further reduce the power consumption of the Wi-Fi interface, we note that our Cell2Notify scheme would still be beneficial as it completely disables the Wi-Fi interface when it is not in use in an active VoIP call.

In addition to high power consumption, the performance of cellular data interfaces is also not well suited for real-time applications, such as VoIP. We measured two metrics, jitter and loss rate, which are usually associated with the quality of a VoIP connection, and we present those results in Table 1. All three interfaces had a reasonably good connection to their respective networks. The results show that the quality of VoIP calls is much better over the Wi-Fi connection than over the cellular data networks. In fact, the high latency over the cellular data interface makes voice traffic intolerable.

There are several other reasons why the cellular data network is not ideal for VoIP traffic in an enterprise. The costs are greater, because all employees (or the enterprise) needs to purchase a cellular data plan, and these tend to be expensive. In most cases, this needs to be an unlimited data connection since VoIP calling generates a significant amount of traffic. The enterprise also has no control over calls using this approach, since the first hop from the smartphone is the cellphone carrier. Consequently, it is extremely

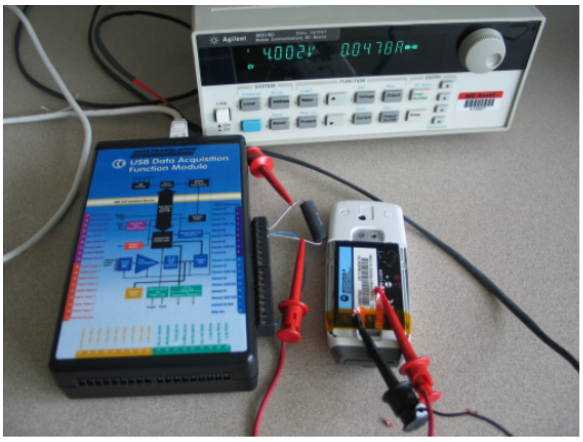


Figure 3: *Our experimental setup to measure the battery power consumption of a smartphone when different network interfaces are turned on and used.*

difficult to implement and manage any call handling system. Given the above factors, we conclude that it is preferable to use Wi-Fi for VoIP instead than a cellular data network.

3.2 Smartphone Power Measurements

We now measure the power consumption of a popular smartphone, the HTC Tornado (Cingular 2125). This device has an ARM TI 195 MHz processor, runs Windows Mobile 5.0 and has a TI-1100 802.11g Wi-Fi chipset. We subscribed to the Cingular voice plan for our experiments. We measured the power consumption of the smartphone for various states of its network interfaces, i.e. GSM and Wi-Fi, and we show that Wi-Fi is a major power drain if it is in the ON state at all times. We also use these numbers later to evaluate our Cell2Notify protocol.

Our experimental setup to measure the energy consumption of a smartphone is based on the technique described in [10]. We fully charged the battery of the smartphone and then removed the battery from the device for an hour. We then connected a 0.5 ohm sense resistor in series with the battery of the device, and measured the instantaneous current across the resistor at 50,000 samples per second using a data acquisition system. We illustrate our setup in Figure 3. We repeated this procedure for each of our experiments. All our experiments lasted five minutes each. To compute the power consumed by the smartphone, we multiplied the current with the average supply voltage of 3.7 Volts. The talk time for the Cingular 2125 is rated at 4 hours. With its 1150 mAH battery, this corresponds to a power consumption in an active cellular voice call of 1150*3.7/4 = 1063.75 mW.

We present the measured results in Table 2. In each of our experiments, we measure the total power consumption of the smartphone, not just the power consumption of the interface. We set beaming to off, the backlight timeout to five seconds which is the minimum possible, the display timeout to 1 minute (also the minimum possible), the light sensor to off, and the earpiece volume to the minimum value.

As we see from this table, the smartphone expends very little battery power to keep its GSM interface on when it is connected. However, it consumes much more battery power when its Wi-Fi interface is on. Note that the Wi-Fi card was using IEEE 802.11 power save mode. Even when the Wi-Fi radio is idle, the device consumes more than 15 times the battery power than in GSM idle mode. These numbers indicate that the total lifetime of a smartphone can be significantly increased if the Wi-Fi radio is turned off most of the time. This forms the primary motivation for our work on Cell2Notify, where we turn on the Wi-Fi device only when it is needed.

Scenario	Power
All Radios off (Flight Mode)	15.688 mW
GSM Idle	27.38 mW
Wi-Fi (searching)	1042.44 mW
Wi-Fi (connected)	441.82 mW
Wi-Fi (send/recv)	1113.811 mW

Table 2: *Power consumption of the Cingular 2125 smartphone for different states of its network interfaces.*

4. CELL2NOTIFY ARCHITECTURE

Cell2Notify increases the battery lifetime of smartphones by disabling the Wi-Fi radio when the user is not making a VoIP call. It enables the Wi-Fi interface only when either the user wants to initiate a VoIP call, or when the user is receiving an incoming VoIP call. In the latter case, Cell2Notify sends a wake up signal to the smartphone as a *ring* on the cellular interface (either GSM or CDMA). As noted in Section 3.2, the cellular interface consumes significantly less energy than the Wi-Fi interface when not in use, and users rarely disable it. Consequently, Cell2Notify results in significant energy savings when using smartphones for VoIP over Wi-Fi.

The design of Cell2Notify poses two primary challenges. First, the system needs to be easily deployable. Therefore, it should not require changes to the standardized protocols used by VoIP phones. Furthermore, Cell2Notify cannot require wholesale changes to network infrastructures it relies upon – neither the Wi-Fi infrastructure nor the cellular infrastructure. Second, disabling the Wi-Fi interface should not result in dropped calls nor significant delays. Cell2Notify must enable the Wi-Fi interface and complete the VoIP call within a reasonable amount of time. Finally it must handle scenarios where the user is an area that lacks either Wi-Fi or GSM coverage.

The Cell2Notify architecture addresses these challenges by requiring minimal modifications to the VoIP architecture illustrated in Figure 1. Cell2Notify only requires software changes at the VoIP proxy server and on the smartphone devices. Furthermore, all the software changes are implemented at user-level, and hence are easily deployable. Our prototype system works with the Session Initiation Protocol (SIP) [21], which is the most commonly used protocol to set up VoIP sessions. All our changes at the proxy server are to the SIP proxy's configuration files, which allows Cell2Notify to be deployed incrementally. Our system is also backwards compatible in that it supports users with phones that do not have a cellular interface, though those users will not obtain any of the energy saving benefits. Our system incurs acceptable call setup latencies, and we devise simple protocols to handle scenarios where the user is out of range of either the cellular or Wi-Fi network.

As shown in Figure 4, our system introduces two new components to an existing VoIP system. We enhance the VoIP proxy server of a traditional deployment with additional call handling rules, and call it the *Cell2Notify Server*. The Cell2Notify Server also maintains a table that contains the mapping of users (VoIP extensions) to their corresponding cell phone numbers. The other new component in the Cell2Notify system is the *Cell2Notify Client*, which is a traditional smartphone running our user-level service. Our service handles notifications sent by the Cell2Notify server. We describe our architecture in detail in the rest of this section.

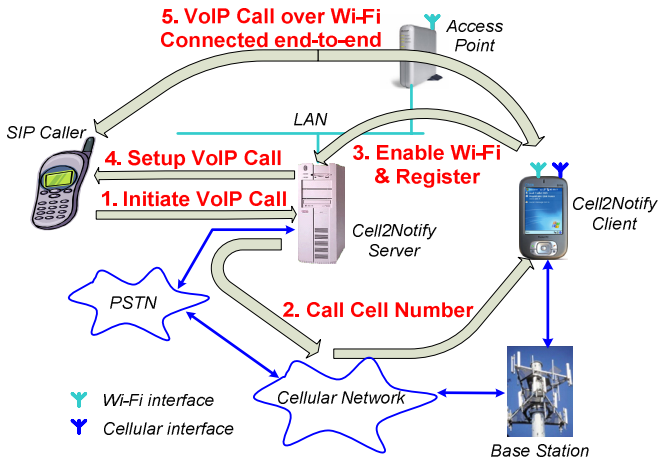


Figure 4: *Steps of the Cell2Notify protocol.*

4.1 Cell2Notify Protocol

The main steps of the Cell2Notify protocol are illustrated in Figure 4. *Registration*, which is not shown in the figure, is required before a device can utilize this architecture. In the Registration step the network administrator adds a new smartphone to use the VoIP system. During registration, the Cell2Notify server adds a mapping of the smartphone's VoIP extension to its cell phone number. The server also generates a unique Caller-ID (UID) that it will use as the Caller-ID when calling the smartphone to initiate a wakeup. The UID is 10 digits long, and its first digit is set to 0 to prevent collisions with existing phone numbers. This scheme provides basic security against Caller-ID spoofing. Since this UID is randomly generated and is different for different extensions, it is not trivial for attackers to send spurious wakeup calls. We also present a security enhancement to this basic scheme in Section 7.2. Finally, the smartphone is updated to set the VoIP extension and to store the UID that will be used by the server to contact it.

The Cell2Notify client disables its Wi-Fi interface whenever it receives a good signal from a cellular base station. When an incoming VoIP call arrives at the Cell2Notify server (Step 1 of Figure 4), the server looks up the client's extension in its table and retrieves the corresponding cell phone entry. The server then initiates a call to the client's cell phone number over the PSTN using an ATA (Step 2). When the Cell2Notify client receives this call, our user-level service traps the Caller-ID, and checks to see if the Caller-ID matches the Cell2Notify server's UID. If the Caller-ID does not match the service allows the call to ring on the device as a regular call. However, if the Caller-ID does match the server's UID then the service enables the Wi-Fi interface (Step 3). The client associates with a Wi-Fi Access Point (AP) and registers its IP address with the Cell2Notify server. The server can subsequently set up the VoIP call (Step 4), by sending the Cell2Notify client's credentials to the caller. The call is finally carried out end-to-end between the two devices without going through the server (Step 5). After the VoIP call ends, the Cell2Notify client disables the Wi-Fi interface.

We note that after Step 1, if the Cell2Notify server does not find a cell phone number corresponding to the client's extension, it simply proceeds to handle it as a regular SIP server. In other words, it attempts to set up the call if the client has previously registered, and otherwise it will send back a busy tone. Similarly, if after Step 1 the Cell2Notify server finds that the client has already registered, it attempts to setup the call as a regular SIP server, i.e. it directly calls the client's VoIP number.

4.2 Connectivity Scenarios

Cell2Notify needs to robustly handle situations where either the cellular network or the Wi-Fi network becomes unavailable. In these situations, our goal is to perform at least as well as a legacy VoIP deployment that does not use Cell2Notify. In this section, we enumerate the connectivity possibilities and describe the system behavior in each of those situations.

4.2.1 Registered Client, in Wi-Fi, Cellular Range

This is the ideal case for our protocol. The smartphone is in range of a known Wi-Fi network and has good cellular coverage. It has also previously registered with the Cell2Notify server, and its DHCP lease has not expired. Moreover, it has not moved recently, so it has cached state of the nearby APs. When someone calls the client, the Cell2Notify server sends a wake-up call on the cellular interface. The smartphone then enables its Wi-Fi interface, connects to the AP whose information it has cached, and sends a SIP register message to the Cell2Notify server. The server then connects the VoIP call over the smartphone's Wi-Fi interface.

4.2.2 Unregistered Client, in Wi-Fi, Cellular Range

In this scenario the client is in a Wi-Fi zone but has not yet connected and registered. In comparison to the previously described case, there is an extra step involved. Upon receiving the wakeup call over the cellular interface from the Cell2Notify server, the device enables its Wi-Fi interface and performs a scan to look for available APs. The rest of the steps are similar to the previous scenario. To address this case, the Cell2Notify server attempts calls to the client's SIP extension multiple times to allow enough time for the mobile device to look for available Wi-Fi APs.

4.2.3 Client in Cellular Range, out of Wi-Fi Range

We now consider the case where a client is not in a Wi-Fi zone. When the Cell2Notify server sends a wake-up call over the cellular interface, the device enables the Wi-Fi interface and scans for wireless networks. Since there is no wireless network available in this case, the Cell2Notify client never sends a SIP register back to the Cell2Notify server and eventually turns its Wi-Fi interface off to save power. To handle this scenario, we use a relatively long timeout value at the proxy. If the proxy cannot connect the call to the mobile device it has several options. Based on user preference, it can either forward the call on the regular cellular line after resetting the Caller-ID to the correct Caller-ID (not the UID), or it can request that the caller leave a voicemail. The first option will complete the call, although the call setup will incur extra latency equal to the timeout value of the SIP server. These options can be configured as part of the call handling rules (described in Section 4.3) for the VoIP extension of the smartphone, and can be customized based on user preference.

4.2.4 Client out of Cellular Range

Cell2Notify is based on two key properties of the cellular networks: low power consumption of the cellular radio and near ubiquitous connectivity. However in the rare case that there is no cellular coverage, our user-level service on the smartphone automatically enables the Wi-Fi interface and registers with SIP on the Cell2Notify server. At this point, the Wi-Fi interface only uses IEEE 802.11 power-save mode [12] to save energy. As soon as the Cell2Notify client detects cellular coverage, it sends a SIP deregister message and turns off its Wi-Fi interface. At this point

it reverts to using Cell2Notify wakeups on its cellular interface to enable its Wi-Fi interface.

4.2.5 Client Mobility

Mobility can cause a client to move in or out of cellular or Wi-Fi coverage. This can lead to a window of vulnerability where the state of the client may be different from what is known at the SIP server. For example, when a client moves into cellular coverage, it disables its Wi-Fi interface, although the SIP server might have initiated the signaling of an incoming call on the client's Wi-Fi interface. To handle these mobile scenarios, Cell2Notify requires the SIP server to simultaneously ring the cellular interface of the device while sending a SIP invitation on the client's Wi-Fi interface. So, even in the above scenario, when a client moves into cellular coverage, and disables its Wi-Fi interface, the call setup is successful. In the other scenario where a client moves out of cellular coverage, it immediately enables its Wi-Fi interface, and sends a SIP register message to the SIP server. Therefore, in this case, the latency is better than if the device was in cellular coverage. Finally, we note that the problem of handoff across Wi-Fi APs when a VoIP call is in progress, is out of scope for Cell2Notify, which is a signaling protocol for VoIP call setup.

4.3 Modifications to the VoIP Server

The above steps can be implemented over SIP, without significant modifications to a standard VoIP proxy server. To implement Cell2Notify, we only need to add *call handling rules* for each VoIP extension or user name that is registered with the Cell2Notify server, and no source code modifications to the VoIP proxy are needed. This rule-based call handling is implemented by many commercial SIP/VoIP proxies [31]. The set of SIP rules at the Cell2Notify server are as follows:

1. Send ring tone to caller.
2. Make call to callee's registered cell phone.
3. Dial the VoIP extension of callee. Retry after timeout.
4. Wait a few seconds for callee's response.
5. Send invalid tone to the caller if no response from callee.
6. Hang up if no response from callee is forthcoming.

In Section 5, we present the specific call rules we used in our prototype for the Asterisk SIP server. Step 1 informs the caller that the call is being handled. Step 2 tells the callee to enable its Wi-Fi interface and complete the call. Step 3 attempts to connect to the caller. The server retries this step a few times to account for variation in the time taken by the callee to associate and authenticate with the AP, and obtain an IP address using DHCP. Step 4 waits a little longer for a response. If there is no response from the callee, the server sends back an invalid tone to the caller (or voice mailbox of the callee) in Step 5 and hangs up the call in Step 6.

Since these changes are just rules added to the configuration file of the SIP server, Cell2Notify can be easily added to an existing VoIP deployment without adding any new servers or changing the infrastructure. Furthermore, Cell2Notify works within deployments that have VoIP phones without a cellular interface, or where some users prefer not to use Cell2Notify. Therefore, our system is incrementally deployable as well as backwards compatible.

4.4 Modifications to the Smartphone

We require a few changes to the smartphone devices, yet all these changes can be implemented relatively easily. We need the following additional features: (i) The ability to distinguish a *wake-up* call from a regular call over the cellular interface. (ii) The ability to power on the Wi-Fi interface. (iii) The ability to control association and authentication with a Wi-Fi network. (iv) The ability to monitor traffic over the Wi-Fi interface to power it off automatically at the end of a VoIP call.

As described in Section 4.1, the Cell2Notify server sends a unique ID (UID) to the mobile device as part of the registration process. The component of the smartphone that handles incoming calls needs to be modified to check the Caller-ID of all incoming calls against this UID. In case of a Windows Mobile based smartphone this can be done by modifying the connection manager. When the incoming Caller-ID does not match the UID, the incoming call is treated as a regular call. When the incoming Caller-ID does match the UID, the connection manager takes the following steps:

1. Do not send the call notification to the user.
2. Power on the Wi-Fi interface.
3. Authenticate and associate to the Wi-Fi network and request an IP address from the DHCP server.
4. Start up the SIP softphone user interface.
5. Send a SIP register message to SIP proxy with the destination address as the IP address acquired from the Wi-Fi network.

When the Cell2Notify server receives the SIP register message from the smartphone device, it can complete the SIP call. An important point to note is that the Cell2Notify server is not required to keep any state, since the SIP call is completed over the Wi-Fi interface of the mobile device and the voice session (using RTP) is established end-to-end. This makes our system highly scalable.

Once the VoIP call ends, the smartphone must detect this event and turn off the Wi-Fi interface to save energy. This may be complicated given the presence of other traffic on the Wi-Fi interface, in which case it may not be clear that the call has ended. To detect the end of a VoIP call, we have implemented an activity detector that monitors the wireless interface for data sent and received. Although VoIP sessions generate an almost constant quantity of data traffic during the lifetime of a session, the actual quantity of traffic is dependent on the codec used. Therefore, automatically distinguishing VoIP from other traffic is very difficult. Instead, our detector simply uses a conservative approach, powering off the Wi-Fi interface after a full ten seconds of network inactivity (although the interval length is configurable).

4.5 Other Applications

Until now we have focused on using Cell2Notify solely for VoIP calls. However, this architecture can be used to enable a number of other services for smartphones. For example, the Cell2Notify server can be configured to send e-mail notifications by using a different Caller-ID. The Cell2Notify client can use the Caller-ID to differentiate between VoIP and e-mail notifications. The smartphone can then connect to the mail server over Wi-Fi to download the e-mail message contents. Because many people receive a much larger number of incoming e-mails than phone calls, our notification system may impose a much larger load on the cellular network. To avoid this overload, we can tune the Cell2Notify server to only send these notifications for high priority e-mails, or for e-mails from a pre-specified group of people.

A similar application that can benefit from Cell2Notify is *Fax over Wi-Fi*. Any existing scheme for sending Fax over IP, such as T.38 [19], requires the Wi-Fi client to be enabled and hence drains battery power. With Cell2Notify, the Wi-Fi client can be disabled most of the time, and enabled only to receive the fax transmission. Cell2Notify also has applications outside the enterprise setting. For example, any VoIP provider, such as VoIP-User or Skype, can use Cell2Notify to notify their users of incoming calls at home. They would only additionally need the cell phone numbers of smartphones that would be used as receivers of the VoIP calls. In a similar vein, cell phone providers such as T-Mobile, who are moving towards UMA [29] could benefit from Cell2Notify. UMA allows a cell phone to use a Wi-Fi connection if available. However, the Wi-Fi device always needs to be enabled to receive incoming calls. Using Cell2Notify, they can disable the client's Wi-Fi device unless the client is either receiving a call or making one.

4.6 Alternatives to Cell2Notify

There are several alternatives to Cell2Notify. In this subsection we use three metrics to argue that notifications using a call over the cellular network is a better approach. The three metrics are: cost, deployability, and performance.

One alternative to Cell2Notify is Wake-On-Wireless [23]. This scheme requires a custom low power radio to be added to each smartphone, as well as to the enterprise wireless infrastructure. When a user receives a call, Wake-On-Wireless(WoW) sends a signal to the smartphone using the low power radio to enable the Wi-Fi interface. This scheme is more costly as this requires the deployment of other low power radios, and is also less deployable since it requires hardware changes on all the smartphone devices. On-demand paging [1] has the same goal. It requires Bluetooth hardware to be added to each AP. On receiving a call, the AP sends a signal via Bluetooth to the smartphone to enable the Wi-Fi device. Since smartphones mostly have a Bluetooth interface, this scheme is more deployable than WoW. However, it too requires changes to the infrastructure and is therefore costly. Furthermore, both Wake-On-Wireless and On-Demand Paging suffer from the range mismatch problem: the different wireless interfaces have different coverage ranges, and the low-power wireless interface typically covers a smaller region than the Wi-Fi interface. Therefore, the additional wireless infrastructure must be deployed at a higher density than the existing Wi-Fi deployment of access points.

Another approach to Cell2Notify would be to use an SMS (Short Messaging System) based notification system. This scheme is similar to ours except that it would send an SMS message to the smartphone over the cellular network. Although this scheme is as cheap and deployable as Cell2Notify, it suffers from poor performance. SMS usually incurs higher latency and is more unreliable than phone calls. This reduces the usability of this system.

5. PROTOTYPE IMPLEMENTATION

We are currently implementing the Cell2Notify system on Windows CE, a commonly used operating system on smartphones. In the meantime, for evaluation purposes, we have built a prototype of Cell2Notify using commonly available off-the-shelf components. The components of our prototype are illustrated in Figure 5. We implement the Cell2Notify server using a combination of the open-source Asterisk SIP Server [3] and the VoIP gateway provided by Junction Networks [13]. We emulate a smartphone using a combination of a cell phone and a laptop running Windows XP. We use a Sony Ericsson W810i cell phone with a built-in Bluetooth interface. The laptop also has built-in Bluetooth, and we use a Netgear WAG511 Cardbus card as the Wi-Fi interface. Finally, we use a

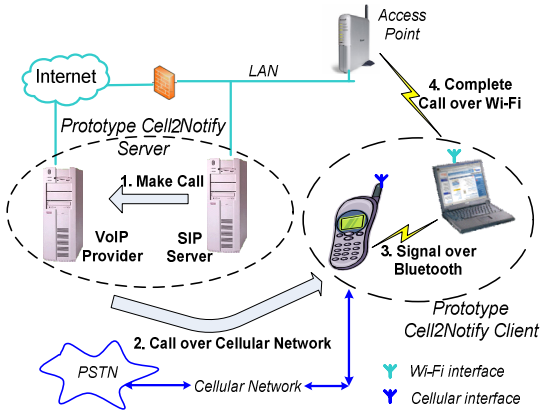


Figure 5: *Our prototype implementation of Cell2Notify. We implement the Cell2Notify server as a combination of a commonly available SIP Proxy and an Internet- based VoIP gateway. We emulate a smartphone using a combination of a cellphone that communicates with a Wi-Fi equipped laptop using Bluetooth.*

popular SIP client for Windows XP called X-Lite [7] as the VoIP softphone.

Our prototype requires minimal modifications to the above components. We made changes to the call handling configuration files of the SIP server, and we built a user-level *call-manager* service that runs on the Windows XP laptop. Our prototype demonstrates the ease with which Cell2Notify can be incrementally deployed in an existing VoIP system. Although our prototype is not the ideal implementation of the Cell2Notify architecture, it serves to demonstrate a working system and it is useful for evaluation of our architecture.

The steps of the Cell2Notify protocol for our prototype are shown in Figure 5. When someone makes a incoming call to a Cell2Notify client, the Asterisk SIP Proxy looks up the corresponding cellular number for the client, and makes a call to the client over PSTN using the Junction Networks gateway. When our cellphone receives the call, it notifies the laptop of the incoming call via Bluetooth. The call-manager service on the laptop then turns on the Wi-Fi interface and uses it to connect the call. When the call is complete, the call-manager turns off the Wi-Fi interface. In the rest of this section, we describe the implementation details of the Cell2Notify server and client components.

5.1 Prototype Cell2Notify Server

The Cell2Notify server only requires minimal modifications to the Asterisk SIP Server. We have added a mapping from SIP extensions to the corresponding cell phone number, and a set of call handling rules for each registered Cell2Notify client. Asterisk supports integration with a back-end database, thus allowing the cell phone mapping table and call handling rules to be implemented as separate tables in the database, and be linked to the Asterisk server. Presently we have manually added these mappings for each Cell2Notify client to the Asterisk configuration files. However, this task can be easily automated using the supported database functionality.

We implement the steps described in Section 4.3 as call handling rules in the Asterisk server. We define these rules for every registered extension or user name. To define the call handling rules, we use generic functions that are supported by most SIP proxies, such

as Ringing, Playback, Dial, Wait and *Set(CALLERID)*. The *Ringing* function sends back a ring notification to the caller. *Playback* plays a default welcome message and *Dial* dials a SIP extension. The *Wait* function waits for a specified duration before executing the next rule. *Set(CALLERID)* is interesting as it allows the Caller-ID of the outbound call to be set to an *arbitrary* number. In the following example, we present the call handling rules for a particular extension, say extension `7676`:

```
1. exten => 7676,1,Ringing
2. exten => 7676,2,Set(CALLERID(number)= UID)
3. exten => 7676,3,Dial(SIP/Cell-Number
   @jnctn,5)
4. exten => 7575,4,Wait(2)
5. exten => 7676,5,RetryDial(waiting|1|8|
   SIP/7676|30|Ttm)
6. exten => 7676,6,Playback(Invalid)
7. exten => 7676,7,Hangup
```

These rules define the steps executed by the Cell2Notify server when there is an incoming call for extension 7676. The first argument denotes the destination extension (7676) for the incoming call, the second argument is the rule order(1,2,..,7), and the third denotes the function (Dial, Ringing, etc.). Rule 1 executes the *Ringing* function and sends back a ring tone to the caller. The server then looks at Rule 2 and executes the *Set(CALLERID)* function with the UID as a parameter, essentially setting the Caller-ID to the UID for the next outbound call. As explained earlier the UID is different for each smartphone client using Cell2Notify and is negotiated during registration. Rule 3 places a call to the particular cellular number associated with extension 7676 using the Junction Networks gateway. Rule 3 is essentially needed to send a signal to the Cell2Notify client to turn on its Wi-Fi interface. In rule 4 the server executes *Wait* for 2 seconds to insert some delay before trying to contact the extension. On encountering rule 5 the server executes *Dial* to contact the SIP extension 7676 repeatedly 8 times with a 1 second interval between subsequent retries. These retries are needed because of the latency to turn on the Wi-Fi interface on the laptop device, and the latency to associate and authenticate over the Wi-Fi network. In the case where call is not connected or remains unanswered the server executes the *Playback* function as specified in rule 6, to send the caller an invalid extension or unreachable message. According to Rule 7, the server executes a *Hangup* to end the call. Rule 6 could be modified to playback another message, record a voice mail, forward the call to another extension, or even forward the call to the cellular number of the user.

When a call handling rule (such as Rule 3) requires the server to place a call on the regular telephone network, it uses an ATA or an external third party VoIP provider to bridge the IP based network with the PSTN. We have implemented both these options. In the first option, we used the Sipura ATA [24] and a privately leased PSTN line. For the second option, we used the Junction Networks VoIP gateway. Using an ATA may be preferable for an enterprise, because the call leaves the IP network within the enterprise itself. However, using a third party VoIP provider may be cheaper.

An architectural requirement for the Cell2Notify server is the ability to place a call over the PSTN using an arbitrary Caller-ID. We implement this using the *Set(CALLERID)* function of Asterisk in conjunction with the VoIP gateway of Junction Networks. The SIP server sets the desired Caller-ID as a parameter to the *Set(CALLERID)* function. Junction Networks allows users to provide their own Caller-IDs for outgoing calls, as long as it is *any* 10 digit number, and then places a call to the destination PSTN number with this Caller-ID using an ATA located in the Junction Networks data center. We are currently working on implementing this functionality on the Sipura ATAs. We discuss the implications of Caller-ID spoofing in Section 7.

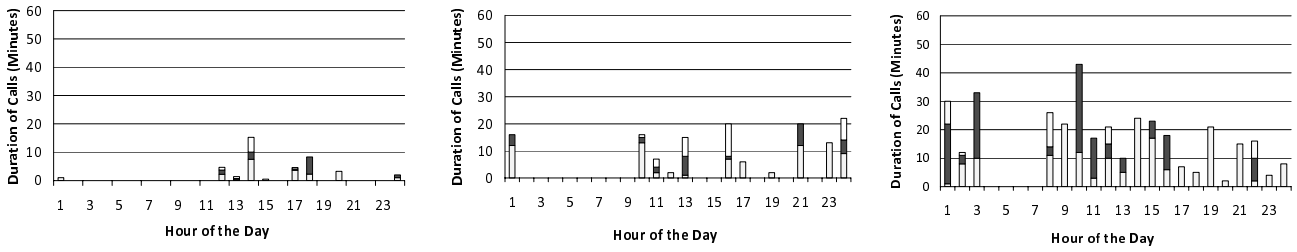
5.2 Prototype Cell2Notify Client

We now describe the implementation of the Cell2Notify client, focusing on three main challenges. First, we need a way to signal an incoming call on the Sony Ericsson cell phone to the call manager service on the Windows XP laptop, and we need to send the Caller-ID of the incoming call to the call manager. Second, we need minimize the delay in completing the call by reducing the delay imposed by the Wi-Fi authentication and association process. Finally, the call manager service needs to determine when the call ends and disable the Wi-Fi interface.

We address the first challenge without requiring modifications to the Sony Ericsson handset by configuring the Bluetooth interface on the laptop to appear as a Bluetooth headset to the cell phone. Consequently, an incoming call on the cellphone notifies the Bluetooth headset, which is in fact our laptop. We use Float Mobile Agent (FMA) [9] to configure the laptop Bluetooth interface to appear as a headset device. FMA is powerful phone editing software which has extensive support for Sony Ericsson handsets, including a rich set of APIs to control the handset. One feature of these APIs handles a *Call-Notify* event which our call manager service uses to trap an incoming call. We built a separate call handler on the FMA framework that checks the Caller-ID of each incoming call to see if it is from the Cell2Notify server, based on the unique ID that was exchanged as part of the registration process. FMA also provides a way to disconnect a call. If the Caller-ID matches that of the Cell2Notify server, the call handler disconnects the call and wakes up the Wi-Fi interface. If the Caller-ID does not match, the call handler lets the call through and ring on the handset.

To address the second challenge, our service uses caching to quickly associate with an Access Point and complete the call over Wi-Fi. When a wireless card is enabled, it usually goes through a series of steps before it obtains a valid IP address. For example, it scans the network looking for the best available AP, after which it performs the entire association procedure. Associating with an AP using the standard Windows XP Zero Configuration Service takes multiple seconds [6]. We optimize this step by caching the frequency channels of the most commonly used APs. We also turn off the Zero Configuration Service and implement tools to control the wireless interface from our own Cell2Notify service. When the Wi-Fi interface is turned on, we instruct the card to go to specific channels and attempt association to the wireless network. We have measured the total time to associate on a given channel to be less than 20 ms for the Netgear WAG511. Using this optimization, we are able to complete the association within a few hundred milliseconds, as shown in Section 6. Once the Wi-Fi card is enabled and has an IP address, we start the X-Lite SIP client. The SIP client sends a register message to the Cell2Notify server with its acquired IP address and completes the call over Wi-Fi.

Finally, we need a way to automatically detect the end of a VoIP call and turn off the Wi-Fi interface. After the Wi-Fi interface is enabled, our call manager service enters an activity monitoring mode. In this mode, it checks the number of packets sent and received on the Wi-Fi interface. It does not immediately disable the Wi-Fi interface when the number of packets is zero, as this might disconnect the call during a period of silence. Instead, the service uses some



(a) James's Office Phone (b) John's Cell Phone (c) Beth's Cell Phone

Figure 6: *Call logs of three users. James is an employee in an enterprise, and the heaviest office phone user among five employees we studied. John is a moderate cell phone user, and Beth is an extremely heavy cell phone user. She makes calls at 3 AM, and receives a call at 8 AM.*

hysteresis and only disables the Wi-Fi interface if there are no packets sent over it for a certain number of seconds. We experimented with various values and found that a delay of ten seconds was adequate. To avoid modifications to the SIP client application code, we terminate the SIP client process at the end of a call and restart it when a new call is initiated or received.

5.3 Is the Prototype Real?

We have built and demonstrated a working Cell2Notify prototype. One obvious concern is the practicality of our system, given that we have emulated the Cell2Notify client rather than implementing it on a real smartphone. We argue that all our changes can be easily migrated to a smartphone. The Bluetooth notification from the cell phone to the laptop will not be required when the GSM and Wi-Fi interface are on the same device. Our call handling routines would also be simpler on a smartphone, and we would not need the APIs provided by FMA. For example, on a smartphone running Windows CE the only modification required is to the Connection Manager on the smartphone device to implement the call handling functionality. Changes to enable and disable the Wi-Fi interface can also be easily migrated to the smartphone. In fact, we expect even better performance on smartphones since the SIP client will always be running on it, as compared to our prototype where we have to terminate and restart the X-Lite SIP client process.

6. SYSTEM EVALUATION

The utility of a mobile device is directly related to the useful operating lifetime before its battery needs to be recharged. Thus, the primary metric we use to evaluate our Cell2Notify system is the reduction in energy consumption, which directly translates to increased battery lifetime. We also evaluate the increase in end-to-end latency that a caller experiences when making a call to a Cell2Notify client. Our results show that using Cell2Notify, users can greatly increase the total usage lifetime of their Wi-Fi enabled smartphones when using VoIP, while experiencing only a nominal increase in initial call-setup latency.

6.1 Reduction in Energy Consumption

To quantify the energy savings enabled by Cell2Notify, we first measured the power consumption of various commonly used wireless cards. The 802.11 standard [12] specifies various modes of operation for the interface but not the specific implementation details. Table 3 below illustrates the power consumption of several Wi-Fi interfaces in the normal mode of operation, Awake Mode(AM), and the low power mode called Power Save Mode (PSM), achieved by duty cycling the wireless interface. The Cisco PCM-350 is sometimes referred to in research literature for the sake of comparison, although it is known to be quite power inefficient. The Netgear MA701 and Linksys WCF12 cards are the most power efficient among the cards that we have measured and thus we use the Linksys WCF12 as a baseline for comparison. Once enabled, a Wi-Fi interface usually takes some time to stabilize, before reaching a state where it can perform active data transfer. Similarly, when disabling a Wi-Fi interface it takes some time before the power drawn by it becomes negligible. In addition to measuring the power consumption of these wireless cards, we have also measured the power consumption of a Windows Mobile based smartphone. The power consumption for the Cingular 2125 was reported in Section 3.2 earlier.

Vendor	*Average Power*		
	Idle(AM)	Idle (PSM)	Active
Cisco PCM 350	1300mW	390mW	1600mW
Linksys WCF12	690mW	256mW	890mW
Netgear MA701	780mW	264mW	990mW

Table 3: *Measured power consumption for 802.11b cards*

The effective energy savings for a particular user are somewhat dependent on their usage patterns. As stated earlier, our Cell2Notify scheme keeps the Wi-Fi interface of a smartphone switched off at all times, except during an active VoIP call. Thus, a user who uses their phone for sporadic conversations will end up saving more energy, in contrast to a heavy user who communicates more frequently. Energy saved by our low power architecture is thus directly dependent on the amount of idle time experienced by a mobile device.

In order to study typical usage patterns, we gathered detailed cellular phone call-logs of different users. Using these call logs we construct a similar trace of periods of communication activity and inactivity, that would be experienced if the users were using VoIP over Wi-Fi instead. Using these call traces we accurately estimate the level of energy savings enabled by the Cell2Notify architecture. We then compare this to the energy consumption of these devices, if they were using the standard 802.11 operating modes, AM and PSM respectively. This technique of using call-logs is similar to the one used in Wake-on-Wireless [23].

Figures 6(a), 6(b) and 6(c) show the calling patterns of users James, John and Beth respectively. James is a real employee in an enterprise and is the heaviest user among five of his colleagues in our study group. John is a light user with an average talk time of about 5 minutes per hour. Beth on the other end is a hypothetical

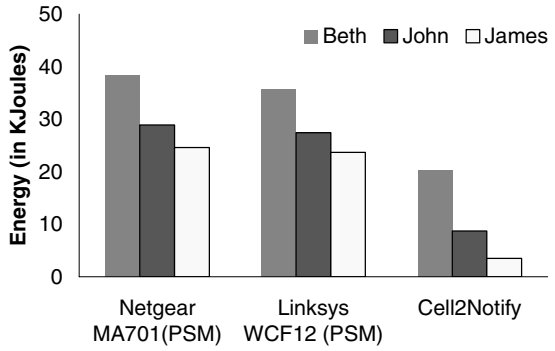


Figure 7: *Energy consumption using two cards, with and without Cell2Notify for three different users. Cell2Notify saves more energy for lighter usage patterns.*

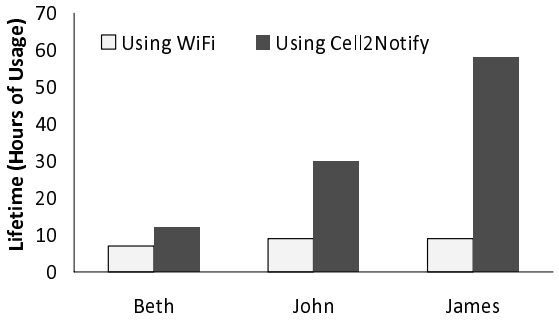


Figure 8: *Energy consumption of a Cingular 2125 with and without Cell2Notify for three users. We assume that the user does not use the smartphone for any other purpose, but only for making and receiving VoIP calls.*

person with a relatively heavy usage pattern, with an average talk time of 15 minutes per hour. On the horizontal axis, the hour of the day is shown ranging from 0 hours to 23 hours. The total number of minutes that a user was actively communicating over the phone are marked on the vertical axis. The different shaded subsections for each vertical column depict the number of calls made in that hour and the duration of each call. These call logs are illustrative traces that help evaluate the *estimated* energy savings for these three usage patterns.

Figure 7 plots the total *communication* energy consumption for the various users calling patterns in a 24 hour period. The graphs shows the energy consumed in the wireless interface when two low power Wi-Fi cards (MA701, WCF12) are used, compared to the energy consumption when utilizing the Cell2Notify architecture. As can be seen even Beth, with a heavy usage pattern, can save up to 47% of the energy consumption compared to using the Wi-Fi cards in the Power Save Mode (PSM). John and James, who have lighter usage patterns end up saving 70% and 87% respectively of the energy consumed compared to using the Netgear MA701 in PSM mode.

In essence, lowering the energy consumption leads to longer battery lifetime of a smartphone. To quantify the effects of our scheme in terms of increased lifetime we measured the power consumption

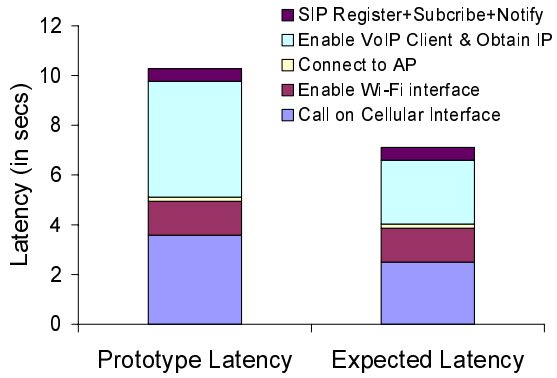


Figure 9: *Breakdown of various steps of the Cell2Notify protocol in call-setup latency. The right bar shows the expected latency with our proposed optimizations. Even without optimizations, the extra delay is around ten seconds, which is* less than two rings.

of a Wi-Fi enabled Smartphone (Cingular 2125) in various modes of operation. Using our detailed power measurements reported in Section 3.2 and the rated capacity of the phone battery (1150 mAH), we determine battery lifetime. Figure 8 shows the increase in battery lifetime for the three usage scenarios. Our base comparison is using the Wi-Fi in always on mode for the smartphone. As can be seen Beth experiences a 70% increase in battery lifetime by using Cell2Notify. John and James on the other hand experience a 230% and 540% increase in lifetime, primarily because of their light usage patterns.

6.2 End-to-End Latency

The reduction in energy consumption when using the Cell2Notify architecture has an associated tradeoff with respect to the added latency in connecting a VoIP call. Since the mobile device that is the end recipient of the VoIP call has its wireless interface switched off, there are multiple steps that have to be taken before the device can actually accept the call over Wi-Fi. Each of these steps has an associated latency overhead. In this section we evaluate these latencies for our prototype implementation. We also provide detailed measurements of these latencies for other platforms. Some of the latencies are fixed costs which are beyond our control, for example the time taken to connect a call over the cellular network, while some of the other latency components can be optimized. Using these measurements we can provide a reasonably accurate estimate of the lower bound on the total end-to-end latency that a device using our architecture experiences.

Cell2Notify Protocol Step	*Latency (in seconds)*	
	Standard Dev.	Max Value
Call on GSM	0.098	3.7
Enable Wi-Fi	0.265	1.7
Connect to AP	0.073	0.36
Enable VoIP Client	0.105	4.8
Obtain IP Address	1.08	4.44
SIP Operations	0.025	0.488

Table 4: *Standard deviation and maximum values for various steps of the Cell2Notify protocol. Note that the steps "Enable VoIP Client" and "Obtain IP Address" occur in parallel.*

Figure 9 shows the breakdown of the call-setup latency introduced by various steps of the Cell2Notify system. The bar on the left in the figure shows the latencies measured on our prototype implementation using the combination of a Windows XP laptop and an SE-810i cellular phone. The column on the right shows the expected latency for the case of a final product implementation on a smartphone. Each latency value presented in the Figure is an average over a minimum of ten runs. We present the standard deviation and maximum values for each of these steps in Table 4.

Our measurements show that the average added latency for our prototype implementation is around ten seconds. This extra wait is equivalent to two rings received by the caller. We believe this overhead is minimal and acceptable in most scenarios. Furthermore, we expect a real smartphone and enterprise deployment of Cell2Notify to incur an overhead of around seven seconds, which will provide a more seamless experience to users of Cell2Notify.

A big chunk of the overhead is the time taken by the SIP server to call the GSM interface of the Sony Ericsson cell phone. It is difficult to accurately quantify this overhead, since the caller (server) and callee (Sony Ericsson handset) are on two different machines. We used a stopwatch to measure this time for over 20 runs, but we are aware of possible inaccuracies due to human reaction times. However, we note that our reaction times will likely result in an overestimate of the latency. As we see in Figure 9, the time taken for the Cell2Notify server to call the GSM interface of the Sony Ericsson phone is around 3.7 seconds in our prototype. A large portion of this overhead seems to be the time taken to call Junction Networks, and for Junction Networks to make a long distance call to our cell phone. To estimate the time it would take in a real prototype, we tested calling a local cell phone number using the Sipura [24] ATA that we have set up in our lab. We note that this time was only 2.5 seconds. Since most enterprises will have their private VoIP gateway, this seems to be a reasonable estimate in a real prototype.

We further explored the lower bound of this delay when the VoIP gateway is on the enterprise LAN. We placed calls from the SIP server to the client phone through the Junction Networks VoIP gateway, and for different types of client phone connections. We placed 10 calls for each connection type, and present the distribution of the time taken to place these calls in Table 5. We note that it takes an extra second to place a call to the GSM phone, and the connection quality of the phone does not add a significant latency. Furthermore, it takes much lesser time to place a call on the CDMA phone. This latency is comparable to the time taken for placing a call to the land line phone, which is around 2.4 seconds.

Client Phone (Signal)	*Latency (in seconds)*		
	Avg.	Std Dev.	Max.
GSM Cingular (Excellent)	3.6	0.074	3.6
GSM Cingular (Poor)	3.78	0.092	3.9
CDMA Verizon (Fair)	2.44	0.117	2.6
Landline Phone	2.41	0.074	2.5

Table 5: *Distribution of time taken by the SIP server to "ring" a phone for various connection types. We present the latency to ring a land line number as a reference.*

Our optimization of using cached Access Point BSSIDs gives good results. Our Cell2Notify client is able to associate with the AP in less than 200 ms. We used three different APs on three different frequency channels in our experiments. We disabled the card and randomly picked an AP to associate with in each run. We also measured the default time to connect to an AP without our optimization of caching the AP information resulting in a much higher overhead, between 3 and 4 seconds in each run. This latency is expected since without our optimization, the wireless card goes into scan mode. It stays for over 100 ms in each channel (all 802.11 a and g channels), and only then associate with the best AP.

Another significant latency in Cell2Notify is the time to obtain an IP address and bring up the softphone. As shown in Figure 9 this overhead is around 5 seconds in our prototype. Although it only takes about 2.5 seconds to obtain the DHCP address, the total time to start the X-Lite SIP client process takes around 5 seconds. As mentioned earlier, we had to restart the SIP client process to avoid modifications to the SIP client code. In an actual implementation over smartphone, we do not expect this artificial overhead of restarting the SIP client to be present. Instead, the only overhead should be the time required to obtain a valid IP address.

After the softphone has initialized and obtained a valid IP address it sends a SIP Register message, and a Subscribe message to the Cell2Notify server, which together take less than 0.5 seconds. Once the server receives the SIP register from the Windows XP SIP client, it connects the call. These steps have very low overhead.

Finally, we note that since most users are willing to tolerate up to five rings (25 seconds) after call connection to reach the voice mail, the less than ten seconds (2 rings) delay introduced by Cell2Notify is acceptable in most scenarios. Ideally a user study would be useful to estimate the actual impact of this increase in latency. We plan to investigate this as part of our future work.

7. DISCUSSION

We now discuss various issues in the design of Cell2Notify. We first discuss the legality of our approach, and show how our system can be secured against spoofed Caller-IDs. We then discuss the concerns that cellular operators may have to the deployment of Cell2Notify.

7.1 Is Caller-ID Spoofing Legal?

There is no law in the US against Caller-ID spoofing [30]. Caller-ID over PSTN is sent using the SS7 signaling protocol. Before the days of VoIP, expensive equipment was required to spoof a Caller-ID. With VoIP, one can introduce fake Caller-ID information when passing the call from IP to PSTN. There are a number of commercial services [26, 27] that allow users to make calls from a spoofed Caller-ID. This has led to a few abuse cases of "pretext calls", where people pretend to be someone else to extract private information [30]. Therefore, in a recent development, the FCC is investigating the use of Caller-ID spoofing for fraudulent purposes. However, since Cell2Notify does not attempt any fraudulent activity, we do not expect it to be affected in the near future.

7.2 Handling Spoofed Caller-IDs

Given that Caller-ID spoofing is legal in some countries such as the US, we need to protect against attackers who might spoof the Caller-ID of the Cell2Notify proxy causing the smartphone to enable the Wi-Fi card and waste battery power. We can thwart this attack by authenticating the Cell2Notify proxy at the client using standard cryptographic techniques. One way to achieve this is to use the S/KEY system [11], which originated from Lamport's scheme [17] as follows. The Cell2Notify proxy shares a different secret key with each VoIP user, which is set up during secure registration. The first Caller-ID used by the proxy is the last nine digits of a one-way hash applied n times over the secret key, where n is a large number. The first digit of the Caller-ID is set to 0 to avoid collisions with a PSTN phone number. The subsequent Caller-ID is an $n-1$ times one-way hash of the secret key, and so on. The

Cell2Notify client authenticates the proxy by applying the one-way hash on the Caller-ID to see if it matches the previous Caller-ID. Given a strong hash function, this scheme can provide reasonable protection against a spoofed Caller-ID attack.

7.3 Concerns of Cellular Operators

Cellular operators have a valid reason for blocking the Caller-ID of the Cell2Notify server. After all, Cell2Notify only uses their network as a signaling channel. Consequently, cellular operators do not stand to gain by allowing Cell2Notify. We have several reasons to believe that cellular operators might be willing to allow Cell2Notify to make signaling calls over their network. Cell2Notify imposes little load on their network as for every incoming call to the VoIP phone, we make one signaling call over the cellular network which does not last more than a few seconds. Even assuming that the VoIP phone has similar usage characteristics as the cellular phone (in Section 6 we show in fact that an enterprise phone is used quite infrequently), a ring for every incoming call imposes little extra overhead. Furthermore, users might be willing to pay an extra "connection charge" to achieve longer battery lifetime. In some cases, the enterprise may be willing to pay a flat fee to cellular operators to support this service. We also believe this work is extremely timely given the launch of T-Mobile's UMA service [29]. Cellular operator's supporting UMA [14] can provide Cell2Notify service as an additional selling point. Finally, we note that it might be technically infeasible for cellular operators to block calls from the Cell2Notify server, since the proxy uses a different Caller-ID each time it sends a signal using the mechanism described in Section 7.2.

8. RELATED WORK

Several projects have investigated techniques to optimize the energy consumption of the Wi-Fi interface in battery powered mobile devices. These techniques range from protocol optimizations in various layers of the networking protocol stack for single Wi-Fi radio based systems, to techniques that leverage multiple radios on the same device. In the case of systems based on a single Wi-Fi radio, researchers have explored various optimizations at the application layer [8, 16], transport layer [5] and MAC Layer [15, 32]. However, as we have shown earlier, the power consumption for Wi-Fi in the lowest power mode (PSM) is still quite substantial even when the device is idle. Cell2Notify, in contrast proposes the use of a long range cellular radio, which has an order of magnitude less power consumption than Wi-Fi PSM, to notify a Wi-Fi smartphone of an incoming call.

Taking into account the high idle power of Wi-Fi, the idea of using a second lower power radio to wake-up a higher power radio, has been proposed [1,23]. Wake-on-Wireless [23] proposes the use of a second special-purpose radio that serves as a wake-up channel for a Wi-Fi radio. The authors have proposed a PDA based phone usage scenario for their system, similar to Cell2Notify. However the choice of the short range custom radio necessitates multiple intermediate proxies and presence servers in order to notify the PDA-phone of an incoming call. On-Demand-Paging [1], builds on the idea of [23], to use a commodity Bluetooth radio present on mobile devices to serve as a low power paging channel for Wi-Fi. The primary difference between our scheme and both Wake-On-Wireless and On-Demand-Paging is our design choice to leverage the much *longer* range cellular radios compared to their choice of *short* range radios. This has two important advantages. First, since Cell2Notify uses cellular radios with almost ubiquitous coverage, the area of operation is much larger. Second, the infrastructure support needed for our scheme is minimal, with only minor software modifications needed at both the client device and an existing VoIP proxy in terms of call handling rules. Comparatively both the above schemes need substantial additional infrastructure support, while still limiting the area of operation to their region of deployment.

Another set of related work looks at using multiple radios for active data transfer, rather than just wake-up [4, 20, 25]. Turducken [25] investigates the application scheduling problem across heterogeneous subsystems to maximize the battery lifetime of a mobile device. Cell2Notify addresses a different problem of enabling the Wi-Fi interface only when required for a specific VoIP over Wi-Fi scenario within the context of a smartphone. Another related work, called CoolSpots [20], builds on the ideas first presented in [4], and presents algorithms to opportunistically use either the Wi-Fi or Bluetooth interface to increase the battery lifetime of a device. In areas where the device and the Wi-Fi Access Point are within Bluetooth coverage, CoolSpots uses flow characteristics to determine the best interface to use for the flow. In contrast, Cell2Notify is geared towards a specific VoIP over Wi-Fi application. It uses the second cellular radio purely for signaling. In fact, Cell2Notify is complimentary to CoolSpots, and if the smartphone also has a Bluetooth radio, we could use CoolSpots to determine the best radio (Wi-Fi or Bluetooth) to route the VoIP traffic, after Cell2Notify has signaled an incoming call over the cellular interface.

A very recent industry trend is the convergence of Wi-Fi and cellular services, using a technology called Universal Mobile Access (UMA). For example, chipset vendor Kineto [14] and mobile service provider T-Mobile [28] recently tested a service that allows a subscriber to make unlimited phone calls from the home hotspot or T-Mobile hotspots [29]. UMA increases coverage and reduces the cost for mobile operators. Our approach is complimentary to UMA. Devices using UMA could use our protocol to increase the battery lifetime of dual radio devices.

9. CONCLUSION

In this paper we present a new system, called *Cell2Notify*, which leverages the cellular interface on a smartphone to reduce energy consumption of VoIP over Wi-Fi enabled smartphones. We quantify the performance of cellular data networks when used for VoIP and compare these results with Wi-Fi. We conclude that Wi-Fi consumes less power and delivers better performance than cellular data networks. To the best of our knowledge, ours is the first research paper to present such measurements. We present the Cell2Notify architecture that turns the Wi-Fi interface off when it is not in use. The Cell2Notify Server places a call on the smartphone's cellular interface to notify the device of an incoming call. On receiving this notification, Cell2Notify turns on the smartphone's Wi-Fi interface and completes the call over Wi-Fi. Our system works with existing technologies and requires minimal changes to an enterprise's VoIP deployment. We have built a prototype of Cell2Notify and evaluated it in detail. We have shown that in most cases, Cell2Notify incurs less than two rings (10 seconds) of call setup latency while more than doubling the average battery lifetime of a smartphone.

Acknowledgements

We thank our shepherd Mark Corner and the MobiSys anonymous reviewers for their feedback on this paper. We also thank Jitu Padhye for insightful discussions during the design of Cell2Notify, and Patrick Verkaik and Sudipta Kundu for their feedback on the Cell2Notify prototype. We thank David Campbell and Eric Putnam for helping us with the smartphone power measurements.

10. REFERENCES

[1] Y. Agarwal, C. Schurgers, and R. Gupta. Dynamic Power Management Using On Demand Paging for Networked Embedded Systems. In *Proceedings of the 2005 Conference on Asia South Pacific Design Automation*, pages 755–759, New York, NY, USA, 2005. ACM Press.

[2] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning Wireless Network Power Management. In *Proceedings of the Annual ACM/IEEE International Conference on Mobile Computing (MobiCom)*, pages 176–189, New York, NY, USA, 2003. ACM Press.

[3] Asterisk. The Open Source PBX. http://www.asterisk.org/.

[4] P. Bahl, A. Adya, J. Padhye, and A. Wolman. Reconsidering Wireless Systems with Multiple Radios. *ACM Computer Communication Review*, Jul 2004.

[5] D. Bertozzi, A. Raghunathan, L. Benini, and S. Ravi. Transport Protocol Optimization for Energy Efficient Wireless Embedded Systems. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE'03)*, page 10706, Washington, DC, USA, 2003. IEEE Computer Society.

[6] R. Chandra, V. Padmanabhan, and M. Zhang. WiFiProfiler: Cooperative Fault Diagnosis in WLANs. In *Proceedings of the Annual ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2006.

[7] COUNTERPATH. X-Lite 3.0 telephony client. http://www.xten.com/.

[8] J. Flinn and M. Satyanarayanan. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Transactions on Computer Systems*, 22(2):137–179, 2004.

[9] FMA. floAt's Mobile Agent Online. http://fma.sourceforge.net/.

[10] GSM World. TW 09 Battery Life Measurement Technique. http://www.gsmworld.com/documentgs/index.shtml.

[11] N. Haller. The S/KEY One-Time Password System. RFC 1760, February 1995.

[12] IEEE802.11b/D3.0. Wireless LAN Medium Access Control(MAC) and Physical (PHY) Layer Specification: High Speed Physical Layer Extensions in the 2.4 GHz Band, 1999.

[13] Junction Networks. SIP, IAX, IAX2 and Asterisk VoIP Service for Business. http://www.junctionetworks.com/.

[14] Kineto Wireless. The UMA Company. http://www.kinetowireless.com/.

[15] R. Krashinsky and H. Balakrishnan. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *Proceedings of the Annual ACM/IEEE International Conference on Mobile Computing (MobiCom)*, pages 119–130, New York, NY, USA, 2002. ACM Press.

[16] R. Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. *Wireless Networks*, 6(4):263–277, 2000.

[17] L. Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, November 1981.

[18] K. Mahmud, M. Inoue, H. Murakami, and M. Hasegawa. Energy Consumption Measurement of Wireless Interfaces in Multi-Service User Terminals for Heterogeneous Networks. In *ICICE Transactions on Communications*, 2005.

[19] G. Parsons. Real-time Facsimile (T.38) - image/t38 MIME Sub-type Registration. RFC 3362, August 2002.

[20] T. Pering, Y. Agarwal, R. Gupta, and R. Want. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. In *Proceedings of the Annual ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2006.

[21] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[22] H. Schulzrinne and J. Rosenberg. A Comparison of SIP and H.323 for Internet Telephony. *Proceedings of Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1998.

[23] E. Shih, P. Bahl, and M. J. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *Proceedings of the Annual ACM/IEEE International Conference on Mobile Computing (MobiCom)*, 2002.

[24] Sipura. SPA-3000 Analog Telephony Adapter. http://www.sipura.com/products/spa3000.htm.

[25] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical Power Management for Mobile Devices. In *Proceedings of the Annual ACM/USENIX International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2005.

[26] SpoofCard. Be Who You Want To Be. http://www.spoofcard.com/.

[27] Spooftel. The Worlds Leader In Caller ID Spoofing. http://www.spooftel.com/.

[28] T-Mobile. Stick Together with T-Mobile. http://www.t-mobile.com/.

[29] The New York Times. T-Mobile Tests Dual Wi-Fi and Cell Service, October 2006. http://www.nytimes.com/.

[30] Thomas J. Navin, Chief, Wireline Competition Bureau, FCC. H.R. 5126, the Truth in Caller ID Act of 2006, May 2006. http://www.fcc.gov/ola/docs/navin051906.pdf.

[31] voip-info.org. The VoIP Wiki. http://www.voip-info.org/wiki/.

[32] H. Woesner, J.-P. Ebert, M. Schlager, and A. Wolisz. Power Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective. *IEEE Personal Communications*, 5(3):40–48, June 1998.