

# Augmenting Network Interfaces to Reduce PC Energy Usage

Yuvraj Agarwal<sup>†‡</sup>, Steve Hodges<sup>†</sup>, James Scott<sup>†</sup>, Ranveer Chandra<sup>†</sup>, Paramvir Bahl<sup>†</sup>, Rajesh Gupta<sup>‡</sup>

<sup>†</sup>Microsoft Research, <sup>‡</sup>University of California San Diego  
<sup>†</sup>{shodges, jws, ranveer, bahl}@microsoft.com, <sup>‡</sup>{yuvraj, gupta}@cs.ucsd.edu

## Abstract

Reducing the energy consumption of PCs is becoming increasingly important with rising energy costs and environmental concerns. Sleep states such as S3 (suspend to RAM) save energy but prevent the PC from responding to network traffic, for example remote desktop logins, file transfer requests, or content distribution protocols such as bittorrent. We conducted a background survey showing that having these applications running is the reason why many users leave PCs on even when idle.

We present Somniloquy, an architecture that augments network interfaces to allow PCs in S3 to be responsive to network traffic, transparently to remote applications and to the network itself. We show that many applications can be supported without application-specific code in the augmented network interface, including file sharing, remote desktop and VOIP. We show that a further class of applications can be supported with modest processing and memory resources in the network interface, including instant messaging presence and file sharing, for example downloading using bittorrent while the PC is asleep.

Our prototype implementation of Somniloquy using a USB peripheral showed significant energy saving potential in experiments, with 24x less power draw for desktops and 11x for laptops compared to an idle power-on state.

## 1 Introduction

Many personal computers (PCs) are left on for much or all of the time, even when a user is not present, despite the existence of low power modes such as suspend-to-RAM (S3) and hibernate (S4). The resulting electricity usage is having an increasing cost, both in monetary terms and at a cost to the environment. Both of these costs may be increased further by the use of air-conditioning in places where the PCs are located.

PCs are kept on in this way for a variety of reasons (see Section 2), including ensuring remote access to local files, remote access to the desktop user interface, providing reachability for users via incoming email, instant messaging (IM) or voice-over-IP (VoIP) clients, file sharing/content distribution, and so on. Unfortunately, this remote access is not compatible with current power-saving schemes such as S3/S4, in which the PC is not responsive to remote network events. Existing solutions for sleep-mode responsiveness such as Wake-On-LAN [15] and others (see Section 7) have not proven successful “in the wild” since they rely on infrastructure or application-level support or manual user action, presenting barriers to deployment and use.

We present an architecture for network interfaces, called Somniloquy<sup>1</sup>, which significantly reduces PC power consumption during the largely idle periods when there is no local user present, but without compromising the remote accessibility enjoyed by always-on systems. This is done by embedding functionality into the PC’s network interface so that it can adequately process network traffic autonomously. This allows other subsystems (CPU, disk, display, I/O buses etc.) to be powered down. The network interface is capable of signalling a wake-up event when network traffic are received matching configurable conditions.

The key benefit of Somniloquy over existing systems is that it is transparent to the user, to the remote applications and to the networking infrastructure. It relies on additional functionality in each client to accomplish this, but the added materials cost is minimal as many current network interfaces already have significant general purpose processors embedded. Depending on the application, new application-specific functionality may sometimes be required in the network interface; in Section 3 we present examples representing two classes of applications supported by Somniloquy. Since Somniloquy is

---

<sup>1</sup>somniloquy: the act or habit of talking in one’s sleep.

entirely client-based, it can be incrementally deployed in new PCs with a built-in Somniloquy-capable network interface, or legacy hardware can be upgraded by using a peripheral network interface.

This paper makes the following contributions:

- We present Somniloquy, a network interface architecture that enables connectivity to PCs even when they are asleep. In contrast to prior automatic wake-up systems (see Section 7), Somniloquy is transparent to the network. It does not require changes to either remote applications or to the infrastructure, and works with desktops as well as mobile/nomadic devices.
- We present a proof-of-concept prototype showing that Somniloquy can be realized today, using a USB based low power network interface that consumes little power (Section 4). We show how a number of applications, such as file sharing, remote desktop, VoIP, instant messaging and bittorrent can be implemented without requiring any changes to the infrastructure. Furthermore, our prototype also demonstrates the ease of providing legacy support to existing PCs, requiring just an external peripheral and software drivers.
- We evaluate our system in a testbed setting (Section 5), in which energy consumption is reduced 24x for desktops and 11x for laptops while incurring an extra latency of just 4 to 7 seconds for an application-layer operation such as retrieving a file on the sleeping PC.

## 2 Survey on use of low power modes

Anecdotal evidence suggests that users often leave computers powered on even when they are largely idle. For example, the authors are aware of many desktop PCs that are habitually left switched on overnight “just in case” the main user of that machine needs remote access to it. In order to validate our intuition that this is commonplace, and to get data on when and why people do or do not use low-power modes, we conducted an informal web-based survey. We distributed the survey by asking friends to answer it and to pass it on to their friends in turn. We gathered 107 usable responses, of which 65% are from the USA, 29% are from Europe, and 6% are from other countries (N=102). 74% of respondents are male (N=105), the median age is 28.5 (min 21, max 58), and 60% work in IT-related fields (N=98).

We do not claim that this survey provides a statistically significant view of all PC users because it was conducted using a very informal distribution method and we have a relatively small sample size. However, we do believe the

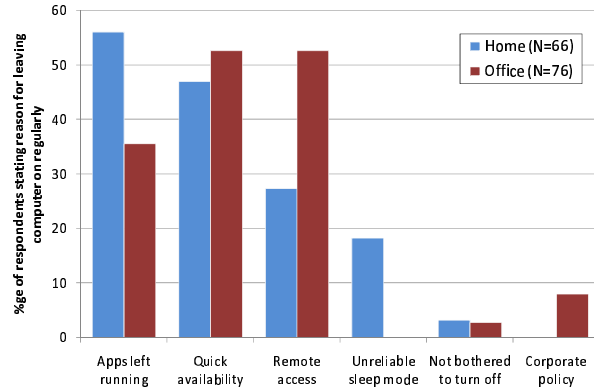


Figure 1: Survey respondents’ reasons for leaving their PCs on when nobody is using them.

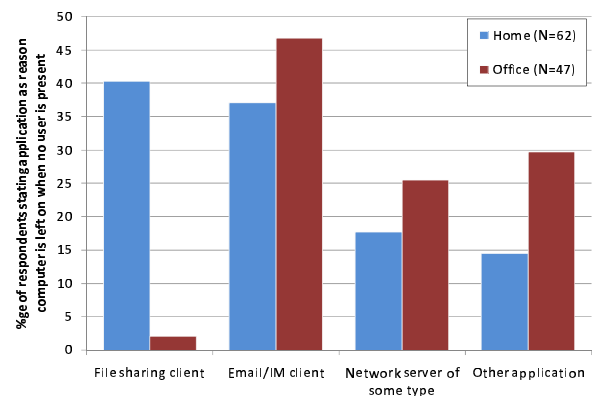


Figure 2: Applications cited by survey respondents as active when nobody is using the PC.

survey respondents come from user groups which could benefit significantly from power saving for their PCs, and we use the responses to motivate our work.

Nearly all our respondents (99%) are aware of PC sleep modes, and 74% regularly use some form of sleep on at least one PC (N=107). In comparison, just 45% of respondents are familiar with Wake-On-LAN (29% for respondents who do not work in the IT field), and just 3% make use of it (none for users not in the IT field) (N=105). This supports our intuition that the existing Wake-On-LAN solution is not widely used.

We asked respondents about both their home PC use and their PC use at work. Respondents have an average of 2.8 PCs in their homes, and 30% of these are regularly left on even when nobody is using them (N=99). At work respondents use an average of 1.6 PCs per person, 75% of which are regularly left on even when nobody is using them. This indicates that both the home and the office scenarios are important targets for Somniloquy, as both have significant proportions of always-on behavior.

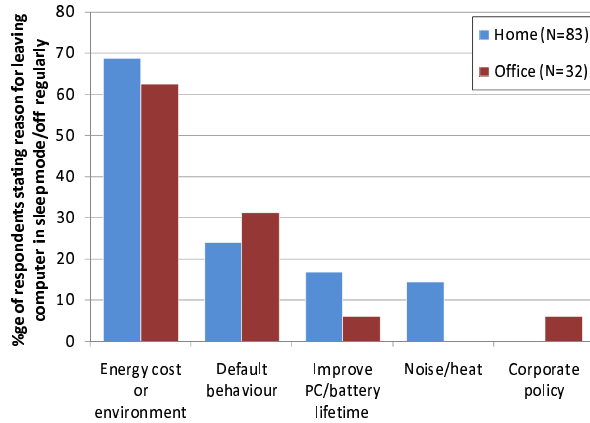


Figure 3: Survey respondents’ reasons for turning their PCs off when nobody is using them.

We further asked respondents why they kept their PCs on and what applications they left running when users were absent. The responses are shown in Figures 1 and 2. The majority of the respondents who leave home PCs on do so because they are running applications in the background, the most popular of which are file sharing/content distribution clients such as bittorrent. Receiving email and IM notifications are also popular reasons for leaving home PCs on followed by the need for access to a network server on the machine, most often as a media server in the case of the home. As we shall describe in later sections, we have designed Somniloquy to address all three of these scenarios. A variety of other applications which fall outside the three main categories of file sharing, email/IM and network server access were cited by a small proportion of respondents, with no obvious categorization.

Following “running applications in the background”, the next most popular reason given for leaving a PC on in the home was quick availability, and another related reason given was that sleep modes were unreliable. The issue of how quickly and reliably PCs can enter and leave low-power modes is not something that we directly address with Somniloquy. However, the desire for quick availability does motivate the continuing enhancement of sleep modes and this is something that Somniloquy intrinsically leverages.

Responses for office PCs differ from those for home PCs in a few major ways. File sharing clients are rarely used (almost never cited as the reason for leaving office PCs on), but the use of always-on email/IM clients and network servers are more prevalent. The range of other (miscellaneous) applications was also wider, including some applications such as batch servers which are not conducive to low-power execution. But overall, leaving

applications running was less of a motivation for leaving the PC on for office users. Instead, remote access was more often cited as a reason for leaving office PCs on than for home PCs, which tallies with the authors’ anecdotal experiences. Removing the need to leave the office PC on “just in case” remote access is required is one of our key motivations for developing Somniloquy.

Figure 3 shows the reasons that respondents gave for using low-power modes. The most frequently cited reason by far was the energy and environmental cost of leaving PCs on. For home PCs, respondents also reported that noise/heat and device lifetimes were important considerations, though these were less of a concern for offices. Interestingly, relatively few respondents said there was a corporate policy concerning low-power modes, and some respondents actually reported that the policy was to leave PCs *on* despite the energy cost, so that the PCs could be kept up-to-date with the latest software patches.

In summary, our survey strongly motivates the development of an architecture that can enable PCs to go into a low-power mode and yet still be responsive to network traffic due to applications such as email, IM, remote desktop, remote file access and file sharing. A system built in this way would significantly reduce the overall energy usage of PCs at home and in the office without compromising the needs of users.

### 3 The Somniloquy Architecture

Somniloquy is an architecture for network interfaces, which allows PCs to be put into a low power sleep state (namely S3) without compromising remote accessibility. This gives users a practical alternative to leaving their PCs switched on when not in use. Our primary aims during the development of Somniloquy were:

- to enable a computer in a sleep state (such as S3) to be reachable across the network and
- to do this transparently to network routers and servers, so that it requires no new hardware or software in the network.

We accomplish this by augmenting the PC’s network interface hardware with an always-on, low power embedded CPU. Despite consuming significantly less power than the host processor, this *secondary processor* is still capable of running an embedded operating system including an entire networking stack. When the host PC is powered on, the secondary processor does little; the PC runs its regular networking stack which communicates directly with the network interface hardware as normal. However, when the host processor is asleep, power to the

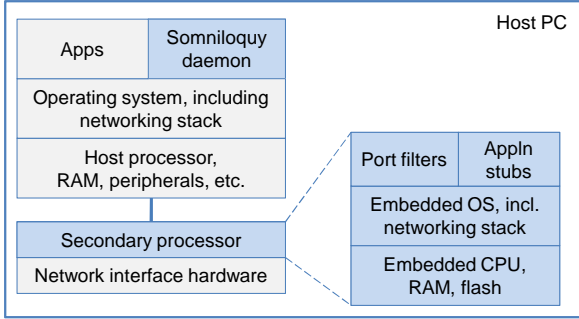


Figure 4: Somniloquy augments the PC network interface with a low power secondary processor which runs an embedded OS and networking stack, network port filters and lightweight versions of certain applications (stubs). Shading indicates elements introduced by Somniloquy.

network interface and the associated secondary processor is maintained, allowing it to perform many network operations autonomously. This forms the basis for Somniloquy’s ability to maintain the PC’s presence on the network, and hence preserve a level of reachability, even when the host processor is suspended. Figure 4 shows the basic architecture of Somniloquy.

Of course, the secondary processor in Somniloquy is not able to deal with all classes of network traffic autonomously, and it is sometimes necessary to wake up the host PC. However, we have designed Somniloquy in a way that minimizes such wake-ups for many application scenarios. In particular, there is enough intelligence in the network interface to support autonomous operation for most of the scenarios highlighted in Section 2.

In the rest of this section we describe in more detail what we mean by “autonomous operation” and in particular how we can implement “application stubs” to process network traffic on the secondary processor for a number of popular applications. We also consider the conditions under which the host processor must be woken up. But first we describe in more detail what happens when the host is put into sleep mode.

### 3.1 Putting the host to sleep

Ultimately, Somniloquy is intended to enable users to put their PCs into a low power state such as suspend to RAM (S3) more frequently than they currently do. To achieve this, we have not introduced any new approaches to initiating the state transition in PCs. We rely on existing approaches such as initiation manually, or automatically after a period of inactivity and/or based on time-of-day. Whatever mechanism is used to initiate sleep, a Somniloquy daemon running on the host processor first ensures that the secondary processor has a copy of the state per-

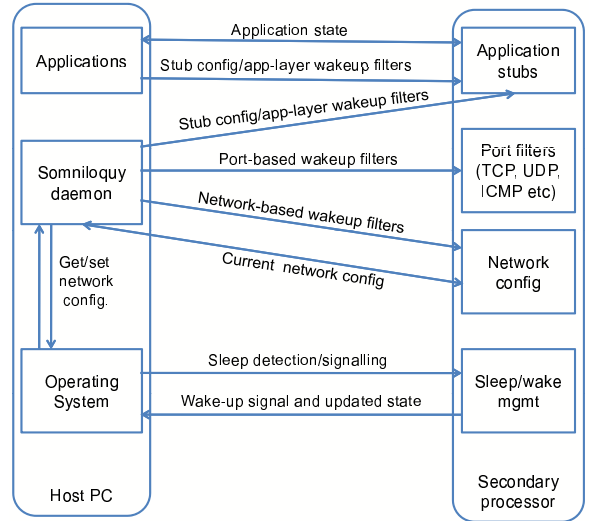


Figure 5: Somniloquy software components on the host PC and the secondary processor, and their interactions.

taining to the current network connection, such as the DHCP lease details and the current IP address. The daemon also specifies which events will trigger wake-up of the host (Section 3.2), and which application stubs need to be run (see Section 3.3).

Following the transfer of this information to the secondary processor, the host enters suspend as normal. Of course, any applications running on the host will be suspended at this point, and any active connections will consequently be dropped. However, new connection requests will now be received by the network stack running on the secondary processor since it is operating with the same configuration as the host. In this way the PC’s transition into suspend will be transparent to remote hosts on the network.

### 3.2 Triggering wake-up

To maintain reachability while the host processor is suspended, the secondary network interface processor clearly needs to support a level of autonomous operation. At the most basic level it should handle traffic at the link and network layers, such as ARP requests, pings, DHCP and so on — thereby maintaining basic presence on the network. However, in addition to this, the secondary processor is also able to run packet filters which are designed to compare the contents of incoming packets with a set of pre-configured trigger conditions. A match of one of these conditions will cause the host processor to be woken from suspend. Filters may be created at the link layer, network layer, transport layer or application layer, with progressively higher complexities and higher

capabilities.

At the link layer, the secondary processor can be told to wake the computer when it detects a particular MAC layer packet, such as the magic packets used by Wake on LAN. But more significantly, it is possible to create an analogous trigger condition at the network layer. For example, a special “ping of life” ICMP packet can be used to mimic the magic packet of WoL, but operating at an Internet scale through standard routers rather than being limited to a single subnet.

Trigger conditions at the transport layer may also be specified; incoming traffic directed at specific TCP and UDP ports can be detected by the secondary processor in order to wake up the host. Filtering only for specified ports has the obvious side effect that traffic on other ports will be rejected. Somniloquy also provides the ability to set up further trigger conditions for example only causing wake-up when traffic from certain IP addresses is received.

Although the host will wake up within a few seconds, it will not receive the packet(s) which triggered the wake-up. Somniloquy leverages the robustness of many protocols designed for Internet use, which are designed to cope with packet loss through automatic retransmission. For example, any protocol using TCP as the transport layer will retransmit automatically at exponentially increasing intervals, so a retransmitted packet will eventually be received by the computer even if resume from sleep takes several seconds. Of course, the longer the computer takes to resume, the larger the subsequent delay before another retransmission will be.

This simple packet filter based approach to triggering wake-up has the advantage that application-specific code does not need to be executed on the secondary processor. None-the-less, it is sufficient to support many scenarios such as remote file and desktop access requests. However, for some applications a more sophisticated approach is needed to maintain reachability without causing spurious wake-ups. This is described in the following section.

### 3.3 Autonomous operation at the application layer

To support the full range of applications highlighted by our informal user survey, we require the secondary processor to run what we refer to as “application stubs” — code that performs a subset of the functionality of specific applications running on the host. An application stub is more powerful than simply using port-level triggers for a number of reasons. First, many application protocols such as those for instant messaging (IM) and email may require periodic activity on the client side in order to maintain the connection, e.g. sending “heart-

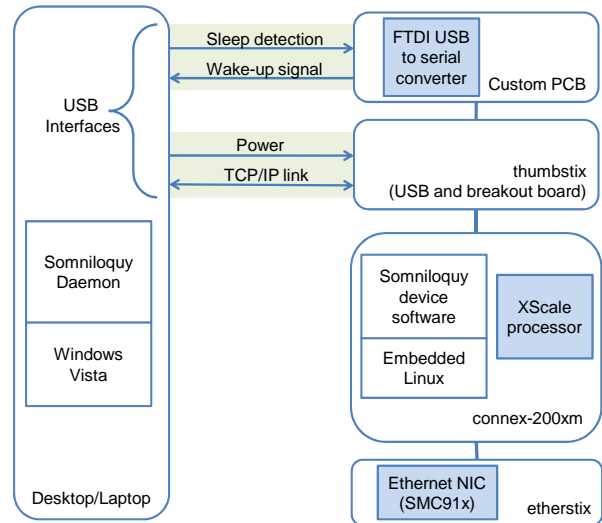


Figure 6: Block diagram of the Somniloquy prototype system - Wired-1NIC version. The figure shows various components of the gumstix and the USB interfaces to the host laptop.

beat” packets allowing the presence information in IM to remain current. These can be handled by the secondary processor without waking the main processor. Second, the use of application code means that event filters can specify events at this layer, e.g. wake on incoming VoIP calls only from my friends outside work hours and not my work colleagues, or wake only for software patch updates that are deemed critical.

The application stubs are written separately for each application. They are required to support four APIs. First, they need to support login for the particular application, including the credentials and the encryption mechanism used by the application. Second, every stub has to handle messages that do not require the host processor to be woken up. These include keep-alive messages, or messages that update the state of the application, such as a presence update for an IM application. Third, stubs need to implement wake-up on application events, such as a VoIP call from a specific contact. Finally, the stubs need to implement transfer of state when the main processor is woken up, such as the transfer of files for a bittorrent stub. We have implemented these stubs for an IM client supporting a number of IM protocols and for a bittorrent client, as described in Section 4.

## 4 Prototype Implementation

We have prototyped Somniloquy using *gumstix*, a low power modular embedded processor platform manufactured by Gumstix Inc. We chose a gumstix-based solu-

tion due to low power consumption, small form factor and the wide range of processors and peripherals supported.

## 4.1 Somniloquy Hardware and Software

The gumstix components we use for Somniloquy include a connex-200xm processor board, an etherstix network interface card (NIC) (for wired Ethernet), a wifistix NIC (for Wi-Fi), and a thumbstix combined USB interface/breakout board. The processor on the connex-200xm is a low power 200 MHz PXA255 XScale processor with 16 MB of non-volatile flash and 64 MB of RAM. The etherstix provides a 10/100BaseT wired ethernet interface plus an SD memory slot to which we have attached a 2GB SD card. The thumbstix provides a USB connector, serial connections and general purpose input and output (GPIO) connections from the XScale.

We added USB-to-serial functionality to the above components using a custom designed PCB that incorporates a single chip — the FT232RL from FTDI. This is attached to the computer via a second USB port and to the thumbstix module (and thence to the XScale processor) via a two-wire RS232 serial interface plus two GPIO lines. One GPIO line is connected to the FT232RL's ring indicator input to wake up the computer, and the second GPIO line is connected to the sleep output for detecting when the computer is in S3 state.

As mentioned above (and shown in Figure 6), the computer is connected to the secondary processor via two USB connections. One of these provides power and two-way communications between the two processors. It is configured to appear as a point-to-point network interface (“USBNet”), over which the gumstix and the host computer communicate using TCP/IP sockets. The second USB interface provides sleep and wake-up signaling, and a serial port for debugging. The use of two USB interfaces is not a fundamental requirement, it is simply for ease of prototyping.

Our hardware works on any desktop or laptop with USB, and support either a wired or wireless network connection (depending on the gumstix NIC used). We run an embedded distribution of Linux on the gumstix that supports a full TCP/IP stack, DHCP, configurable routing tables, a configurable firewall, SSH and serial port communication. This provides a flexible prototyping platform for Somniloquy with very low power operation.

We have implemented the Somniloquy host on Windows Vista. We use Vista's power management APIs to trap a suspend event, and invoke the Somniloquy daemon before the host goes into the S3 state. This transfers the network state required (MAC address, IP address, and in the case of the wireless prototype, the SSID of the AP) and other state needed to configure the wakeup triggers



Figure 7: Photograph of the gumstix-based Somniloquy prototype - Wired-1NIC version.

as discussed in Section 3.

## 4.2 Maintaining Network Reachability

We now describe how our prototype ensures network reachability when the computer is asleep. We have actually implemented three versions of Somniloquy which are appropriate for different scenarios. These are described below.

### 4.2.1 The augmented network interface

Our first Somniloquy prototype provides an external network interface with an embedded secondary processor. This is based on a gumstix with a wired Ethernet interface and a ThumbStix USB interface which is used to connect it to the host PC. We call this implementation the *Wired-1NIC* version. The architecture is shown in Figure 6, and a photograph of the prototype is shown in Figure 7.

When the host computer is awake, this is configured to act as a layer 2 bridge between the USBNet interface to the host PC and the Ethernet interface of the gumstix. We disable the host's internal Ethernet interface, thus all traffic passes through the gumstix. When the host computer goes to sleep, it transfers state to the gumstix, including its IP address. The gumstix then stops the bridging functionality, resets the MAC address of its network interface to be the same as that of the host computer's USBNet interface, and sends a DHCP renew packet with the host's IP address. It consequently appears to the rest of the network as the host processor with the same IP address, MAC address, and hostname. On detecting the host computer waking up (which may occur due to the

gumstix waking it up, or due to the host PC being woken by a user), the gumstix immediately resets its MAC address and enables bridging.

We have implemented this system with etherstix, which has the SMC91x 100 Mbps Ethernet interface. Although we have a 100 Mbps interface, the above prototype is limited by the speed of the USBNet implementation of the gumstix and the overhead of bridging traffic using the PXA processor. As we shall see, this limits bandwidth significantly. Ultimately, the ideal solution is to build a complete network interface from scratch, interfacing the host PC directly to the network interface hardware for high performance and also enabling an embedded secondary processor to do interface to it for low power. This would not suffer from the bandwidth bottlenecks of our wired NIC prototype. However, the Wired-1NIC version does provide us with a proof-of-concept of an augmented network interface which, as we will show, allows us to demonstrate significant power savings.

#### 4.2.2 Systems with pre-existing network interface

An alternative to engineering a single NIC with a fast data path is to make use of the fact that existing PCs already have Ethernet ports. We can use the existing port for networking while the computer is awake and the gumstix's Ethernet port while the computer is asleep. We have also prototyped this *Wired-2NIC* version, which requires minor changes in the gumstix software. That software never enables bridging, but it still copies the MAC address of the host PC when it is asleep (and uses a different one when it is awake). The MAC address copied is not that of the USBNet interface, but that of the host's own Ethernet interface. The gumstix again uses DHCP to acquire the same IP address as the sleeping host, and again wakes it up using the USB connection when suitable trigger conditions are met (described later).

#### 4.2.3 Systems using Wi-Fi

In addition to the two wired prototypes outlined above, we have also implemented a wireless version of Somnloquy. We were unable to implement a one NIC version since the Marvell 88W8385 802.11 b/g chipset present on the wifistix does not support layer 2 bridging. Instead we have implemented a *Wireless-2NIC* version. This receives a few more items of information from the host PC before going to sleep, namely the access point (AP) it is currently associated with, and other AP names and credentials which the gumstix is allowed to connect to (similar to the list kept inside the host's Wi-Fi configuration interface). This allows the Wireless-2NIC version to handle host mobility while still being asleep, since the gumstix can detect and connect to new APs that are seen.

### 4.3 Application stubs and filters

We have implemented support for two classes of applications and have tested both with a variety of applications on all three versions of the prototype. The first class of applications require transport layer based wakeup of the host computer, and do not need application specific code to be running on the gumstix. The second class of applications require application stubs on the gumstix.

#### 4.3.1 Transport-layer triggers

We have implemented a flexible packet filter on the gumstix using the BSD raw socket interface. Every application in this class is associated with a regular expression for packet matching that can trigger host wakeup. For example, handling incoming remote desktop requests might require the host to be woken up when the gumstix receives a TCP packet with destination port equal to 3389. The packet filter applies regular expression filters to the received packets, and wakes up the host computer on an expression match.

We note that waking the host computer is not enough; the incoming connection request must somehow be conveyed to it. We accomplish this by using the iptables firewall on the gumstix to filter any response to TCP packets that the gumstix does not handle itself. Thus, the packets are not acknowledged by the gumstix, and the TCP stack on the remote client will send retries. After the host has resumed, one of the retries will very likely reach it (since it is using the same IP and MAC addresses) and it can respond. Using port-based filtering, we have implemented wake-up triggers for four applications: remote desktop requests (RDP), remote secure shell (SSH), file access requests (SMB), and voice over IP calls (SIP/VOIP).

#### 4.3.2 Application stubs

We have implemented application stubs for two common applications as proofs-of-concept: instant messaging (IM) and bittorrent content distribution/downloading.

For the IM stub, we used a console-only based version of an IM client called *finch* that supports many IM protocols such as MSN, AOL, ICQ, etc. To ensure our goal of a low memory and CPU footprint we customized finch to include only the features salient to our aim of waking the host processor when an incoming chat message arrives. This only requires authentication, presence updates and notifications; we disabled other functionality. The host processor transfers over the authentication credentials for relevant IM accounts before going to S3. The gumstix then logs into the relevant IM servers, and if an incoming message arrives it triggers wakeup.

For the bittorrent stub we customized a console based client, *ctorrent*, to have a low processor and memory

footprint on the gumstix. Due to the robustness of the bittorrent protocol, this stub automatically resumes incomplete torrent downloads that were initiated on the host computer. Prior to suspending to S3, the host computer transfers the '.torrent' file and the portion of the file that has already been downloaded to the gumstix. The bittorrent stub on the gumstix then resumes download of the torrent file and stores it temporarily on the SD flash memory card which is part of the low power subsystem (see Figure 4). A download complete event can be used as a trigger to wake up the host. The downloaded file is then retrieved by the host when it resumes using a file transfer protocol.

Both stubs are still work-in-progress. The IM stub could be more configurable, for example waking the host up on more complex events such as a particular contact coming online, or a message with particular keywords. The bittorrent client could wake the host up when the SD card gets full, it could take into account any portions of files already downloaded. Our goal in developing the stubs to date has been to show that stubs can be resource frugal (in terms of processing and memory) such that the hardware requirements for Somniloquy remain modest resulting in significant net gain in power consumption.

## 5 System evaluation

In this section, we quantify the benefits of Somniloquy and study various tradeoffs in our design. We first microbenchmark the power consumption of standalone desktops, laptops and the gumstix hardware. We then present the power consumption benefits of Somniloquy compared with an "idle" host processor and the latency incurred in moving between these modes. Following that, we discuss the effects of Somniloquy in the context of the various applications that have been described. We conclude the section with a discussion of the energy usage reduction that Somniloquy makes possible.

### 5.1 Microbenchmarks for power consumption and latency

We used a commercially available mains power meter, *Watts-Up*<sup>2</sup> to measure the power consumption of laptop and desktop PCs under various operating conditions. To measure the power consumption of the gumstix alone, we built a USB extension cable with a 100 mΩ 0.1% sense resistor inserted in series with the +5 V supply line and used this cable to connect the gumstix to the computer. We calculate the power draw of the gumstix from the voltage drop across the sense resistor. All the numbers presented in this section are averaged across five

<sup>2</sup><http://www.wattsupmeters.com/>

Condition	Optiplex 745	Dimension 4600
Normal idle state	102.1W	74.6W
'Base power'	102.1W	74.6W
Suspend state (S3)	1.2W	2.0W
Time to enter S3	9.4s	7.5s
Time to resume from S3	4.4s	13.4s

Table 1: Power consumption and S3 suspend/resume time for two desktops under various operating conditions. In all cases the processor is idle and the hard disk is spun down. The power consumed by other peripherals such as displays is not included.

Condition	Lenovo X60	Toshiba M400	Lenovo T60
Normal idle state	16.0W	27.4W	29.7W
Backlight minimum	13.8W	22.4W	24.7W
Screen turned off	11W	18.3W	21.3W
'Base power'	11W	18.3W	21.3W
Suspend state (S3)	0.74W	1.15W	0.55W
Battery capacity	65Wh	50Wh	85Wh
Base lifetime	5.9h	2.7h	4.0h
Suspend lifetime	88h	43h	155h
Time to enter S3	8.7s	5.5s	4.9s
Time to resume from S3	3.0s	3.6s	4.8s

Table 2: Power consumption and battery lifetime of three laptops under various operating conditions, and the time to change power states. For all power measurements, the processor is set to the lowest speed and is idle, the hard disk is spun down and the wireless network interface is on.

runs.

**Desktops:** Table 1 presents the power consumption for two Dell desktop machines: a dual core OptiPlex 745 with 2 GB RAM running Windows Vista, and a 2.4 GHz Pentium 4 Dimension 4600 with 512 MB RAM running Ubuntu Linux. The power consumption of the desktop in S3 is two orders of magnitude less than when it is awake, and agrees with published data concerning power consumption of modern PCs [6]. We use the term 'base power' to indicate the lowest power mode that a PC can be in and still be responsive to network traffic (without using Somniloquy). The time to wake up the desktop from S3 and reconnect to the network is a few seconds. We measured these times using a stopwatch.

**Laptops:** Table 2 presents the power consumption of three popular laptops: a Lenovo X60 tablet PC with 2GB RAM running Windows Vista, a Toshiba laptop with 1GB RAM running Windows XP, and a Lenovo T60 laptop with 1GB RAM running Windows Vista. The base power is between 10 to 20 W, resulting in a battery life-

	Gumstix state	Power
Wired version		
1	gumstix only - no ethernet	210 mW
2	gumstix + Ethernet idle	1073 mW
3	gumstix + Ethernet bridging	1131 mW
4	gumstix + Ethernet broadcast storm	1695 mW
5	gumstix + Ethernet unicast storm	1162 mW
Wireless version		
6	gumstix only - no Wi-Fi	210 mW
7	gumstix + Wi-Fi associated (PSM)	290 mW
8	gumstix + Wi-Fi associated (CAM)	1300 mW
9	gumstix + Wi-Fi broadcast storm	1350 mW
10	gumstix + Wi-Fi unicast storm	1600 mW

Table 3: Power consumption for the gumstix platform in various states of operation.

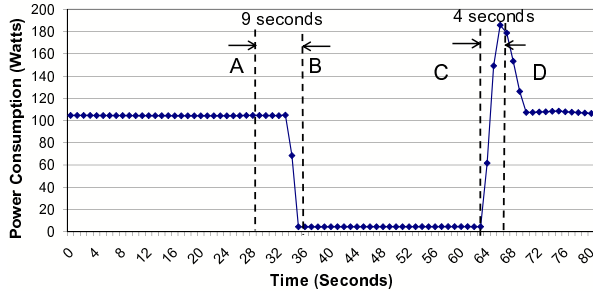


Figure 8: Power consumption of our desktop testbed. At point A a suspend request is made, which completes at point B. Then a wake-up event occurs at point C and the machine is fully resumed by point D.

time of around 4 to 5 hours. Using S3 can dramatically extend the battery lifetime, to between 90 and 150 hours for the laptops we tested, although the laptop is unreachable in this state.

**Gumstix:** Table 3 shows the power consumed by the gumstix (with both etherstix and wifistix) in various states of operation. The gumstix has a base power of approximately 210 mW when no network interface is present (row 1). A gumstix with an active network interface typically consumes approximately 1200 mW (rows 2 and 8), however with an associated Wi-Fi interface in power save mode it consumes only 290 mW (row 7). Broadcast and unicast ‘storms’ (continuous traffic) increase the power consumption by a few hundred milliwatts<sup>3</sup>. Importantly, the power consumption of the gumstix is approximately one tenth that of an awake laptop in the lowest power state, and approximately 50 times less than an active desktop.

<sup>3</sup>Wi-Fi broadcasts are sent at 6 Mbps while unicasts are sent at 54 Mbps in our setup. Consequently a unicast storm consumes more power than a broadcast storm.

## 5.2 Somniloquy in operation

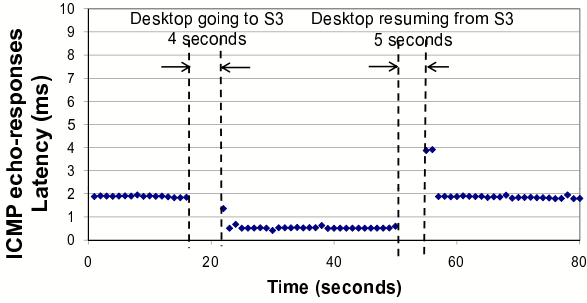
We now report the power consumption of Somniloquy in operation. For these measurements we use two testbed systems: a desktop (Dell OptiPlex 745 with 2 GB RAM running Windows Vista) with the Wired-1NIC prototype of Somniloquy, and a laptop (Lenovo X60 tablet PC running Windows Vista) with the Wireless-2NIC version of Somniloquy. Thus, our tests span both Ethernet and Wi-Fi networks, and both the integrated single network interface, and the higher performance versions which uses the existing internal network interface. The test traffic is generated using a standard desktop machine running on the same (wireless or wired) LAN subnet as the testbed machine.

Figure 8 shows the power consumption of our desktop testbed. Initially the desktop’s host processor is awake and using the gumstix for bridging, and the whole system draws 104 W of power. At time ‘A’ a state change to S3 is initiated by the user. This request completes at time ‘B’ after which the power draw of the system is approximately 4.4 W, i.e. 24x less. This power is split between the gumstix, the DRAM of the PC, and other power chain elements in the PC. Subsequently at time ‘C’ the gumstix, which has been actively monitoring the network interface, wakes up the host in response to a network event. This request completes at time ‘D’ when the host system has fully resumed. As the figures illustrate this resume takes about 4 seconds. We do not show the laptop figure for space reasons; the trace looks very similar with a starting power of 16 W with the screen on (which drops to 11 W if the screen is turned off), an S3 power of 1 W (11x less than the screen-off case) and a resume time of 3 seconds.

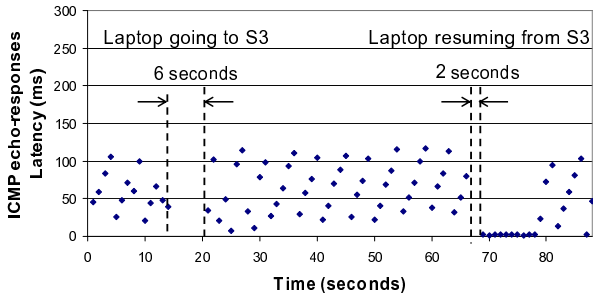
To quantify the network responsiveness of Somniloquy we continuously sent ICMP ECHO (ping) packets to the IP address of our testbeds, and measure the ping latencies when the host machine is put to S3 and woken up again. We plot the time series for the desktop and laptop testbeds in Figure 5.2. We see that the system reachability is maintained even when the host machine is in sleep state. Furthermore, the time for the system to wake up is small – a few seconds in both cases.

The variation in ping latency for the laptop prototype (Figure 9(b)) is because both the laptop and the gumstix use IEEE 802.11 PSM mode, i.e. they wake up only once every AP beacon interval (100 ms). For the wired prototype 9(a) the slight difference in ping times ( 1ms) is attributed to the additional delay in bridging by the gumstix.

These traces show that both the wired and wireless versions of Somniloquy, and both 1NIC and 2NIC configurations operate as desired. In contrast, note that a non-Somniloquy system would simply have dropped ev-



(a) Desktop



(b) Laptop

Figure 9: ICMP echo-response (ping) trace for going into S3 and subsequently resuming from S3 for desktop and laptop PCs.

ery ping between sleeping and waking, while the power consumption would not have been significantly lower.

Somniloquy results in power savings of 24x for our desktop testbed and 11x for our laptop testbed, while allowing them to remain responsive to network events in summary.

### 5.3 Performance for transport layer port triggered applications

The ping latencies above are measured at the network layer. We now quantify the application-layer latency (as perceived by users) incurred by the applications which make use of transport-layer triggers. For these experiments, we use the same two testbeds as above, with the addition of a third testbed based on the Wired-2NIC prototype (using same desktop machine as the Wired-1NIC case), providing a direct comparison between the 1NIC and 2NIC cases. In each case the latency reported is the mean over five test runs.

Figure 10 reports the time taken to satisfy an incoming application-layer request for four applications. For each application, we show the latency for “awake” operation (i.e. when the host is on and directly responds to the request) and when the host is in S3 and Somnilo-

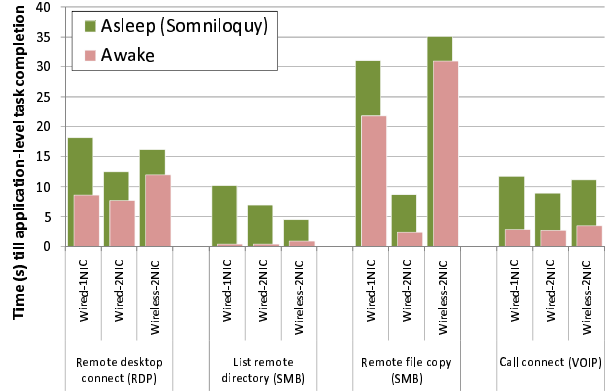


Figure 10: Application-layer latency for three Somniloquy testbeds and four application types.

quy prototype receives the incoming packet and triggers wake-up of the host. Note that in the 1NIC case the data in the awake test still goes through the gumstix (using layer 2 bridging), while in the 2NIC cases the gumstix is never used as the requests go to the host’s internal network interface.

The four applications we tested were:

**Remote desktop access (RDP):** Here we used a stopwatch to time the latency between requesting a remote desktop session and the remote desktop being displayed. A stopwatch was used to ensure that true user-perceived latency was measured. The gumstix was configured to wake the main processor on detecting TCP traffic on port 3389 (the RDP port).

**Remote directory listing (SMB):** A directory listing from the Somniloquy testbed was requested by the tester machine (via Windows file sharing, which is based on the SMB protocol). The time between the request being initiated and the listing being returned was measured using a simple script. The secondary processor was configured to initiate wake-up on detection of traffic on either of the TCP ports used by SMB, i.e. ports 137 and 445.

**Remote file copy (SMB):** The SMB protocol was used again, but this time to transfer a 17 MB file from the Somniloquy testbed to the tester machine.

**VOIP call (SIP):** A voice-over IP call was placed to a user who had been running a SIP client on the Somniloquy laptop before it had entered S3, and the SIP server was responded with a TCP connection to the testbed, causing the gumstix to trigger wakeup. A similar procedure was used in [1]. Once again, the latencies were measured using a stopwatch to measure true user-perceived delay.

As Figure 10 shows, Somniloquy adds between 4-10 s of latency in all cases. This is partly due to the latencies described in Section 5.2 for resuming from S3, i.e. 4-5 s

Accounts	Processor 95% percentile	Memory 95% percentile
None	0.0%	5.9 MB
MSN only	10.0%	6.5 MB
MSN+AOL	21.6%	6.7 MB
MSN+AOL+ICQ	26.0%	6.9 MB

Table 4: Average processor and memory utilization for the IM stub for various configurations. Total memory for the gumstix is 64 MB.

for the desktop and 2-3 s for the laptop. Further latency is due to the delay for TCP to retransmit the request, and to respond to the request (which may take longer since the PC has just resumed). Note that the Wired-1NIC prototype shows higher latency than the Wired-2NIC prototype, this is due to the overhead of MAC bridging and the slower speed of the USBNet IP link. The latter is particularly obvious in the file copy test, where the file copy time with the Wired-2NIC case is much faster than for Wired-1NIC (although the Wired-1NIC speed is still faster than Wireless-2NIC).

While Somniloquy does result in 4-10 s of additional application-layer latency, these delays are acceptable for real usage in exchange for the benefit of 20x-50x power saving. Furthermore, wake-ups will often be rare events followed by a session of use which may total minutes or hours, e.g. for remote desktop or VOIP calling, so the initiation latency shrinks in comparison.

## 5.4 Application stubs

We implemented application stubs for two applications — IM and bittorrent — as described in Section 4. Here, we evaluate the overhead of running those applications on the low power gumstix.

To study the overhead of IM clients, we run the corresponding application stub using up to three different IM protocols simultaneously – MSN Messenger, AOL Messenger and ICQ Chat. Table 4 shows the processor utilization and memory footprint of the Wired-1NIC prototype when running these IM clients.

To evaluate the overhead of the bittorrent stub on the gumstix, we initiated downloads from a remote website<sup>4</sup> into the 2 GB SD card of the Wired-1NIC gumstix. We varied the memory cache available to the stub while conducting single download, and then tested two simultaneous downloads. The results in Table 5 show that the memory footprint of the stub increases proportionally to the cache size while the processor power consumption remains constant. When there are two simultaneous downloads, each instance of the stub uses memory pro-

<sup>4</sup><http://www.legaltorrents.com/>

Configuration	Processor 95% percentile	Memory 95% percentile
<i>Single download</i>		
4MByte cache	16.0%	6.5 MB
8MByte cache	16.0%	10.6 MB
16MByte cache	16.1%	18.9 MB
<i>Two simultaneous downloads (4Mbyte cache)</i>		
1st download	16%	6.5 MB
2nd download	24%	7.0 MB

Table 5: Average processor and memory utilization for the bittorrent stub for various configurations. Total memory for the gumstix is 64 MB.

portional to its specified 4 MB cache. The power consumption of the gumstix did not exceed 2 W during these experiments.

In conclusion, we have shown that with application stubs, even fairly complex tasks such as offloading bittorrent downloading can feasibly be conducted using the gumstix platform while in S3 mode, saving 11-24x power in our testbeds. This validates the use of Somniloquy with application-layer triggers, and directly addresses two important application areas as motivated by our survey in Section 2.

## 5.5 Energy savings using Somniloquy

We conclude this section by extrapolating the amount of energy saved by Somniloquy for desktop computers, and the increase in battery lifetime for laptops.

### 5.5.1 Reducing desktop energy consumption

Our testbed desktop PC consumes 102 W in normal operation and <5 W in S3 with Somniloquy. Somniloquy therefore saves around 97 W. On this basis, if Somniloquy were to be deployed in an environment where a PC is actively used for an average of 45 hours each week (i.e. 27% of the time), this would result in 620 kWh of savings per computer in a year. Assuming 0.61 kg CO<sub>2</sub>/kWh<sup>5</sup> and US\$ 0.09/kWh<sup>6</sup>, this means an annual saving of 378 kg of CO<sub>2</sub> (which is around 10% of the average global individual’s yearly emissions<sup>7</sup>) and US\$ 56 per computer. We believe this is significantly higher than the bill of materials cost of the components required to implement a commoditized Somniloquy-enabled net-

<sup>5</sup>[http://www.eia.doe.gov/cneaf/electricity/page/co2\\_report/co2report.html](http://www.eia.doe.gov/cneaf/electricity/page/co2_report/co2report.html)

<sup>6</sup>[http://www.eia.doe.gov/cneaf/electricity/epa/epa\\_sum.html](http://www.eia.doe.gov/cneaf/electricity/epa/epa_sum.html)

<sup>7</sup><http://www.sciencedaily.com/releases/2008/04/080428120658.htm>

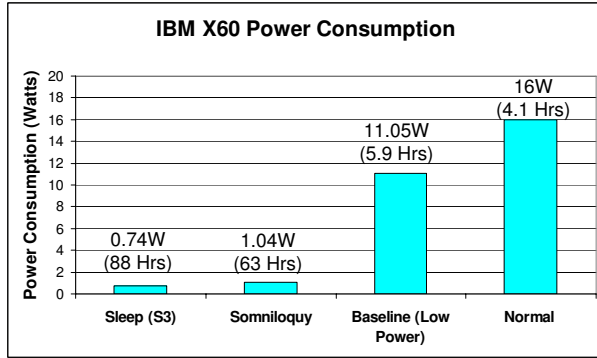


Figure 11: Power consumption and the resulting estimated battery lifetime of a Lenovo X60 using Somniloquy. The lifetime is calculated using the standard 65 Watt hour battery of the laptop.

work card. In this case, deployments of Somniloquy-enabled devices would pay for themselves within a year.

### 5.5.2 Increasing laptop battery lifetime

Figure 11 shows the average power consumption of the laptop testbed when operating normally (i.e. no power saving mechanisms), with standard power saving mechanisms in place (the base power), when Somniloquy (Wireless-2NIC) is operational, and standard S3 (without the gumstix attached). Somniloquy adds a relatively low overhead of 300 mW to S3 mode, resulting in a total power consumption which is close to just 1 W, as compared to the 11 W of the idle laptop. This means that when the laptop needs to be attached to the network and available for remote applications but is otherwise idle, it can be put into S3 to give an order of magnitude decrease in power consumption and a resulting increase in battery lifetime from 5.9 hours to 63 hours (using the standard 65 WH battery).

## 6 Future agenda

In this section we discuss future directions to be explored with the Somniloquy architecture.

### 6.1 Security

Somniloquy raises a number of security concerns that must be tackled before wide deployment can occur. We are actively working on a full analysis of the threat model and techniques to manage these threats. Here we briefly discuss some of the relevant issues.

The functionality that Somniloquy provides, of waking PCs due to remote events, can be misused to conduct attacks designed to spuriously wake up the PC and

waste energy. This is particularly important for mobile devices where this kind of denial-of-service attack might result in a drained battery. It might be regarded as unlikely since the attacker would not gain anything directly, it is likely to be a “nuisance” attack. Solving this issue fully might involve offloading application layer authentication/encryption to the Somniloquy-enabled network interface, so that only authenticated remote hosts are allowed to trigger wakeup.

The complexity of having network layer, transport layer and application layer functionality in the Somniloquy-enabled network interface does increase the risk of bugs causing security holes. A security hole on the Somniloquy device may cause it to reveal state that the host PC has passed to it, such as credentials. A security hole allowing remote “root” access and reprogramming of the Somniloquy-enabled network interface has even more severe implications. For example, that device can be used to sniff network traffic around the host, to launch subsequent attacks (e.g. Worm-style spreading), to electronically spy in locations which it is carried to by the unaware user, to deny network connectivity to the host, or to launch an attack on the host itself. It should be noted though that these security problems are not new ones, since Somniloquy is simply performing a set of actions that the host would perform anyway if it was awake; one way of viewing this is instead that the “attack surface” [12] is increased.

### 6.2 Application support

As discussed in Section 4, application support in Somniloquy is divided into two classes — those applications that simply need transport-layer triggers (e.g. remote desktop access), and those applications that need some element of application-layer functionality to execute in the augmented network interface. We realize that writing application stubs for the second class of applications is non-trivial. To simplify this step, we are looking at using tools such as the Generic Application-Level Protocol Analyser [5] to generalize this process across a range of protocols, by parsing the protocol automatically and delineating the protocol calls that need to be handled in the network interface and those which require waking the host.

### 6.3 Operating system and platform integration

We have shown how Somniloquy can be used to augment a network interface. This approach is relatively self-contained, since changes can be confined to the network interface hardware which is often manufactured separately from the rest of the PC platform. Furthermore,

the software footprint of Somniloquy on the host is small, with only some applications requiring modification, and a simple user-space daemon to pass network association details and credentials and a list of transport-layer port triggers to Somniloquy. Thus, Somniloquy functionality can be added to existing systems with a simple USB peripheral as we have shown.

Looking further ahead, there is the potential for much deeper integration of Somniloquy into both the hardware and software of the host. From the hardware perspective, Somniloquy’s secondary processor and resources can be integrated into the host platform rather than being associated with a single network interface, allowing functionality to span multiple network interfaces, and potentially being merged with other sleep-mode functionality such as auxiliary displays<sup>8</sup>. From the software perspective, one potential optimization that can be realized by closer integration is in reducing the application latency described in Section 5. This latency is in part due to the host configuring the network interface from scratch on boot, performing link-layer configuration, ARP, DHCP, etc to rejoin the network. However, this state should not be recreated from scratch since the Somniloquy system has up-to-date details of such state which can instead be passed directly to the operating system, reducing this latency.

## 7 Related Work

Power management for computers has been explored extensively, primarily for battery constrained mobile devices with desktops getting more attention of late. In the case of mobile devices most power management techniques focus on either specific subsystems, such as the processor [8], communication [3, 14], memory [13] or the storage subsystem [7]. On the other hand, systems such as Odyssey [9] aim to reduce the total system-level power consumption of a mobile device by trading off application fidelity with reduced the energy consumption. Dynamic power management (DPM) to reduce the power consumption across various subsystems has also been proposed [19, 4]. Utilizing multiple network interfaces on the same platform to save power by trading off the power characteristics of one interface with another has also been explored [18, 1]. A few of these techniques are generally applicable to desktops as well. All of these approaches are, however, unable to reduce the “base power” of a computing device, which as stated earlier is still significant in both laptops [19, 20] and desktops [10, 11].

Another approach to maximize the battery lifetime of mobile devices is proposed by Turducken [20]. It uses

multiple-tiers of different power consumption characteristics, such as a laptop and a PDA, where a low power tier (PDA) can perform tasks on behalf of the higher power tier (laptop) to save energy by duty-cycling the higher power tier. Turducken considers a mobile device scenario with a laptop, a PDA and a sensor device forming the tiers supporting applications that are synchronous in nature such as checking for email or synchronizing network time. Somniloquy uses a two-tiered model, similar in spirit to Turducken, with several key differences. First, Somniloquy requires only a single network interface as shown in Figure 4. In contrast, Turducken requires each tier to have its own network interface. Second, Somniloquy maintains “transparent” reachability to a sleeping PC by ensuring that the secondary processor has the same hostname and IP address as the main processor. Turducken does not enable reachability to a sleeping host. It’s design is based on a pull model, where a low power tier polls for information, and wakes up the higher tier when there is an interesting event. Consequently, Somniloquy can support a richer set of applications, including, remote desktop, IM, VoIP and bittorrent. Finally, we have demonstrated a more feasible prototype of Somniloquy using a small form-factor and low-power gumstix.

Reducing the power consumption for desktops computers [10, 6, 2] and the underlying internet routing infrastructure [11] have started to get more attention off late, primarily motivated by rising energy costs and impact on the environment. Researchers have proposed selectively turning off routers and switches, scaling link speeds [10] and re-routing internet traffic [11] to save energy.

To reduce power consumed by desktops, the use of proxies on their behalf while they are asleep [6, 10] has been proposed although not implemented. Very recently, researchers have proposed the idea of proxies on the network to handle processing on behalf of a sleeping PC. Allman et. al. [2] propose preliminary ideas for using “assistants” on the network to handle network operations for a desktop, while it sleeps. Sabhanatarajan et. al. [17] describe the design of a high speed packet filter for a network interface to enable desktops to go to sleep. We take these ideas further, and propose a complete system for offloading applications to the network interface to enable network reachability. We also show the feasibility of our system using a real prototype and quantify the amount of power savings that can be achieved using our approach.

Until now, research and industry had focused on offloading processing of packets from the host processor of a server to the network interface, primarily to improve performance. For example, TCP offload [16] and TCP Chimney<sup>9</sup> offload process TCP packets in the network in-

<sup>8</sup><http://www.sideshowdevices.com/laptops>

<sup>9</sup>[http://www.microsoft.com/whdc/device/network/TCP\\_Chimney.msp](http://www.microsoft.com/whdc/device/network/TCP_Chimney.msp)

terface itself and also send TCP acknowledgements without involving the host processor. Somniloquy uses a similar offloading paradigm, but for PCs, and to conserve energy instead of improving performance.

## 8 Conclusions

We have presented Somniloquy, an system augmenting network interfaces to provide PCs with a low-power sleep state which is capable of maintaining network connectivity while the host PC is asleep, transparently to the network and to remote applications. Our prototype implementation, based on a USB peripheral, includes support for waking up the PC on network events such as incoming file copy requests, VoIP calls, instant messages and remote desktop connections, and we have also demonstrated that file sharing/content distribution systems (e.g. Bittorrent) can run in the augmented network interface, allowing for file downloads to progress without the PC being awake.

Our work was motivated by a survey which showed that users often leave their home and particularly their office PCs on rather than using sleep modes, in order to keep applications running such as file sharing clients, email/IM clients, and network servers (remote desktop, remote file access, etc). Somniloquy has directly targeted this user-driven need, and tests show power savings of 24x are possible for desktop PCs left on when idle, or 11x for laptops. The electricity savings made are such that deploying a productized version of Somniloquy could pay for itself inside a year.

We have already discussed some of the wide range of future work in this area, from necessary steps before deployment such as addressing security issues to the wider possibilities for Somniloquy-like functionality to be more deeply integrated into PC hardware, operating systems and applications to make further energy savings.

## References

- [1] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless wakeups revisited: energy management for voip over wi-fi smartphones. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 179–191, New York, NY, USA, 2007. ACM.
- [2] M. Allman, K. Christensen, B. Nordman, and V. Paxson. Enabling an energy-efficient future internet through selectively connected end systems. In *6th ACM Workshop on Hot Topics in Networks (HotNets)*. ACM, November 2007.
- [3] M. Anand, E. B. Nightingale, and J. Flinn. Self-tuning wireless network power management. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189, New York, NY, USA, 2003. ACM Press.
- [4] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, 2000.
- [5] N. Borisov, D. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo. A generic application-level protocol analyzer and its language. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS)*, 2007.
- [6] K. Christensen, C. Gunaratne, and B. Nordman. The next frontier for communication networks: power management. *Computer Communications*, 27(18):1758–1770, 2004.
- [7] F. Douglass, P. Krishnan, and B. Marsh. Thwarting the power-hungry disk. In *USENIX Winter*, pages 292–306, 1994.
- [8] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic performance setting for dynamic voltage scaling. In *MobiCom '01: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 260–271, 2001.
- [9] J. Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, 2004.
- [10] C. Gunaratne, K. Christensen, and B. Nordman. Managing energy consumption costs in desktop pcs and lan switches with proxying, split tcp connections, and scaling of link speed. *Int. J. Netw. Manag.*, 15(5):297–310, 2005.
- [11] M. Gupta and S. Singh. Greening of the internet. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 19–26, New York, NY, USA, 2003. ACM.
- [12] M. Howard, J. Pincus, and J. M. Wing. Measuring relative attack surfaces. In *Proceedings of Workshop on Advanced Developments in Software and Systems Security*.
- [13] H. Huang, P. Pillai, and K. G. Shin. Design and implementation of power-aware virtual memory. In *Proceedings of the USENIX Annual Technical Conference 2003*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.
- [14] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. *Wireless Networks*, 6(4):263–277, 2000.
- [15] P. Lieberman. Wake-on-LAN technology. [http://www.liebssoft.com/index.cfm/whitepapers/Wake\\_On\\_LAN](http://www.liebssoft.com/index.cfm/whitepapers/Wake_On_LAN).
- [16] J. C. Mogul. Tcp offload is a dumb idea whose time has come. In *HotOS*, pages 25–30, 2003.
- [17] K. Sabhanatarajan, A. G.-R. M. Oden, M. Navada, and A. George. Smart-nics: Power proxying for reduced power consumption in network edge devices. In *ISVLSI '08*, 2008.
- [18] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 160–171, New York, NY, USA, 2002. ACM Press.
- [19] T. Simunic, L. Benini, P. Glynn, and G. D. Micheli. Dynamic power management for portable systems. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19, New York, NY, USA, 2000. ACM.
- [20] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: hierarchical power management for mobile devices. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, 2005.