SleepServer: A Software-Only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments

Yuvraj Agarwal *Stefan Savage* *Rajesh Gupta*

Computer Science and Engineering
University of California, San Diego
{*yuvraj,savage,gupta*}*@cs.ucsd.edu*

Abstract

Desktop computers are an attractive focus for energy savings as they are both a substantial component of enterprise energy consumption and are frequently unused or otherwise idle. Indeed, past studies have shown large power savings if such machines could simply be powered down when not in use. Unfortunately, while contemporary hardware supports low power "sleep" modes of operation, their use in desktop PCs has been curtailed by application expectations of "always on" network connectivity. In this paper, we describe the architecture and implementation of SleepServer, a system that enables hosts to transition to such low-power sleep states while still maintaining their application's expected network presence using an on-demand proxy server. Our approach is particularly informed by our focus on practical deployment and thus SleepServer is designed to be compatible with existing networking infrastructure, host hardware and operating systems. Using SleepServer does not require any hardware additions to the end hosts themselves, and can be supported purely by additional software running on the systems under management. We detail results from our experience in deploying SleepServer in a medium scale enterprise with a sample set of thirty machines instrumented to provide accurate real-time measurements of energy consumption. Our measurements show significant energy savings for PCs ranging from 60%-80%, depending on their use model.

1 Introduction

"Turn off lights and equipment when they are not in use." This simple exhortation heads the list of the Environmental Protection Agency's "tips" for making businesses energy efficient. The reasons are straightforward. In the U.S., commercial buildings consume over one third of all electrical power [9] and, of these, lighting and IT equipment are the largest contributors (roughly 25% and 20% respectively in office buildings according to one 2005 study [10]). However, while it has been relatively straightforward to address lighting use (either through education or occupancy sensors), IT equipment use has been far more resistant to change. Indeed, in a recent empirical study across a number of buildings on our campus, we measured that between 50% and 80% of all electrical power consumption in a modern building is attributable to IT equipment (primarily desktops) [4].

This finding can be unintuitive. First, the computer equipment industry is working hard to reduce power consumption at all levels. Thus, we expect desktop power consumption to be decreasing, not increasing. Second, modern hardware and operating systems possess mechanisms for entering low-power modes when not in use. However, the overall impact of both has been limited in practice. For example, while individual components are indeed much more energy efficient, the capability per desktop has also increased. Thus, while a typical desktop system from 2002 might consume roughly 60-75 watts when idle, the same is also true for today's desktops. Even machines designed and marketed as "low-power" desktops, such as Dell's Optiplex 960 SFF, routinely consume 45 Watts when they are unused.

Compounding this issue is the fact that while today's machines *can* enter a low-power sleep state, it is common that they do not – even when idle. Here the problem is more subtle. Today's low power mechanisms assume that – like lighting – the absence of a user is a sufficient condition for curtailing operation. While this is largely true for disconnected laptop computers (indeed, low-power suspend states are more frequently used for such computers), it is not compatible with how users and programs expect their connected desktops to function. The success of the Internet in providing global connectivity and hosting a broad array of services has implicitly engendered an "always on" mode of computation. Applications expect to be able to poll Internet services and download in the background, users expect stateful applications to act on their behalf in their absence, system administrators expect to be able to reach desktops remotely for maintenance and so on. Thus, there is implicitly a high "opportunity cost" associated with not being able to access and use computers on demand.

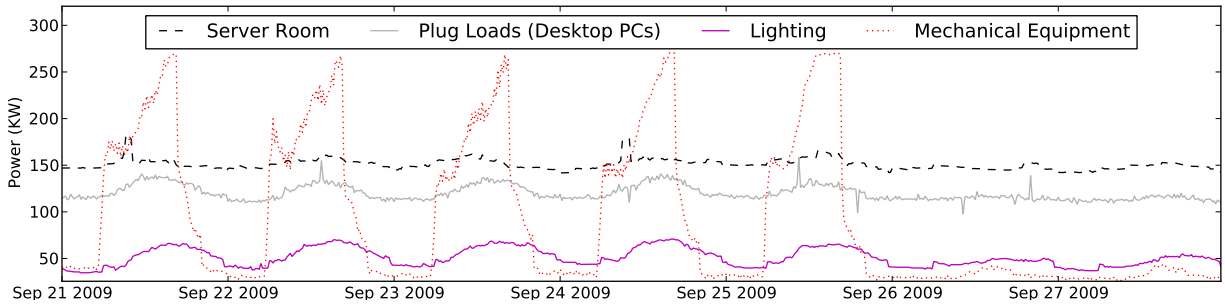

Figure 1: *Detailed breakdown of the various power consumers inside the CSE building at UC San Diego [4], for a week in September 2009. Desktop computing equipment, which make up majority of the plug loads, and the IT equipment in the server rooms account for almost 50% to 80% of the base load of this building.*

However, we, as well as others, have observed that this demand for "always on" *behavior* is distinct from truly requiring an "always on" system. Indeed, it can be sufficient to present the *illusion* that a desktop is always on — suspending it when idle, proxying minor requests on its behalf and dynamically waking it up if its power and state are truly needed [3, 5, 16, 20]. Unfortunately, all of these systems have imposed significant barriers to deployment in implementing this illusion — either requiring significant modifications to network interface hardware and in some cases the host OS software. Such requirements not only represent new expenses, but also require the active participation of third parties (especially network silicon merchants) over the full range of systems in broad use. Thus, in spite of the significant potential for energy savings, we are unaware of any such systems that have been fielded in practice or had practical impact on power consumption in the enterprise sector.

In this paper, we focus on this deployment challenge. Our goal is to provide the same power savings of prior research prototypes, such as our own Somniloquy system [3], yet do so within the confined rubric of existing commodity network equipment, host hardware, operating systems and applications. Indeed, at the *idea level* SleepServer is similar to Somniloquy, but from a practical standpoint SleepServer addresses a different set of challenges that arise directly from our experience in deploying it in our department. The remainder of this paper describes our two contributions: First, we motivate and explain the architecture and implementation of the SleepServer system, which transitions machines to a sleep state when idle, while transparently maintaining the "always-on" abstraction should their services be needed, using a combination of virtual machine proxies and VLANs. Second, we present the results of our pilot SleepServer deployment across a heterogeneous sample of thirty desktops in active use and monitored in real-time by dedicated power meters, including an empirical quantification of the (significant) power savings, an analysis of our system's scalability and cost, and a description of our qualitative experience concerning user feedback and behavior modification.

2 Background

Over the last several decades, the pervasive adoption of information technology has created a new demand for electrical power. Partly due to this reason, the share of U.S. electrical power consumed by commercial buildings has grown 75% since 1980 (it is now over a third of all electrical power consumed) [9]. In a modern office building, this impact can be particularly striking.

For example, Figure 1 shows the power consumption of the CSE building at UC San Diego, broken down by various functions: lighting, server computing, plug loads and HVAC. While the overall electrical usage varies from 320KW to 580KW over the course of a year [4] — generally due to increases in air-handling and cooling — the baseline load is highly stable. Indeed, computer servers and desktop machines connected as plug loads account for 50% (during peak hours on weekdays) to 80% (during nights and weekends) of the baseline load and vary by no more than 20KW over the course of the year.

Given their large consumption footprint, it is not surprising that increasing the energy efficiency of IT equipment has long been an active area of research. However, most of these efforts fall into three distinct categories. One approach focuses on reducing the active power consumption of individual computing devices by utilizing lower power components [11] or using them more efficiently [12, 21]. The second class of energy saving techniques, especially popular in data centers, look at migrating work between machines — either to consolidate onto a smaller number of servers [8] (e.g., using virtual machines [19]) or to arbitrage advantageous energy prices in different geographic zones [23]. Finally, the third class of energy management techniques consider opportunistically duty-cycling subsystems, such as wireless radios

[2, 22, 24], networking infrastructure [14, 21] or even entire platforms[3, 18, 25], during periods of idleness or low use. SleepServer falls into this third category of energy management approaches.

The duty-cycling technique exploits the capability of modern hardware to enter low-power states while maintaining transient state. For example, modern desktops support the ACPI S3 (Sleep/Standby) state, which can reduce power consumption by 95% or more [1]. One approach to using this capability, embodied in modern versions of most operating systems, is to simply place the system in a low-power state after it has been idle for some period of time. Unfortunately, as mentioned earlier, this conflicts with the behavior of users and software that implicitly assume an "always on" abstraction.

To manage this problem, today's network interfaces (NIC) implement features, such as "Wake-on-Lan" [17], that allow sleeping systems to be awakened upon receiving network traffic (frequently a special packet). While this mechanism is quite important, it does not address the key question of *when* a machine should be woken. If this mechanism is activated for every packet then energy savings quickly disappear. Conversely, if its use is too restricted then the "always on" abstraction is lost and users lose the ability to freely access their machines. Consequently, a gap exists between the abstraction levels over which WoL works and the level at which its operation is useful in real-life systems.

Some recent variants have attempted to address these concerns through proprietary hardware and software support. For example, Intel Remote Wake allows the "wake up" capability to be integrated into server software so, for example, a VoIP server could be enabled to wake one of its client machines [15]. Apple's Wake-on-Demand takes a similar approach, allowing client machines using Bonjour advertised services to be "woken" when accessed via Apple networking hardware (WiFi APs) [6]. While neither approach is general, they reflect precisely the need to encode some dynamic triggering policy to preserve application and user transparency.

To generalize this policy, several systems incorporate additional low-power processors into the network interface itself[3, 25]. Using this approach, requests from the network can be parsed and evaluated even when the rest of the system itself is in a low-power sleep state. Moreover, due to their generality these low-power processors can even process requests on behalf of the sleeping system instead of waking it, thus maximizing the amount of power saved. Unfortunately, such approaches face a significant deployment barrier as they require non-trivial changes to network interface hardware.

Finally, a set of projects [5, 16, 20] have explored the notion of implementing this "always on" functionality via network *proxies* that maintain a limited network presence on behalf of sleeping PCs. Nedevschi et al.[20] provide an in-depth look at network traffic to evaluate the design space of a network proxy, while the Network Connection Proxy (NCP) [16] proposes modifications to the socket layer for keeping TCP connection alive through sleep and resume transitions. SleepServer is most similar in spirit to these efforts, but is distinguished from prior work both in offering an actual implementation and not requiring changes to existing hardware, software or networking infrastructure. We argue that these are necessary requirements for any system to see practical use in the enterprise setting.

3 SleepServer: Architecture

We had several goals in mind when we started to design a network-proxy, especially for an enterprise setting. First, the proxy must be able to maintain the network presence of any host on the local network while maintaining complete transparency to other end hosts in the network and to network infrastructure elements such as switches and routers. Second, since the proxies themselves add to the total power consumption, they must be highly scalable and therefore be able to service hundreds of hosts at any given time for maximum energy savings. Third, the proxy should be able to provide isolation when it is servicing individual hosts while providing mechanisms to scale resource allocation based on the proxying demands of individual hosts. Fourth, the proxy must address management aspects, such as providing mechanisms to enable and disable the proxying functionality for hosts dynamically, viewing the status of supported hosts in the system, and maintaining security. Fifth, the proxy should be able to support a heterogeneous environment with different classes of machines running different operating systems. Lastly, we wanted to achieve all of the above goals purely in software without requiring any additional hardware to the individual end hosts or any changes to the networking infrastructure.

Based on these design goals, our SleepServer—network-proxy architecture is illustrated in Figure 2. In an enterprise LAN environment, one or more SleepServers (SSR) can be added in addition to the host computers (H) proxied by the SleepServer. These SleepServer machines have a presence on the same network segments or subnets as the proxied hosts, i.e. they are on the same Layer-2 broadcast domain. A SleepServer can proxy for machines on different subnets using existing Virtual LAN (VLAN) support that is common to commodity routers and switches. Of course, there can be multiple SleepServers, each servicing only a particular VLAN(s) for security isolation if required by enterprise policy.

The various components of a SleepServer are shown

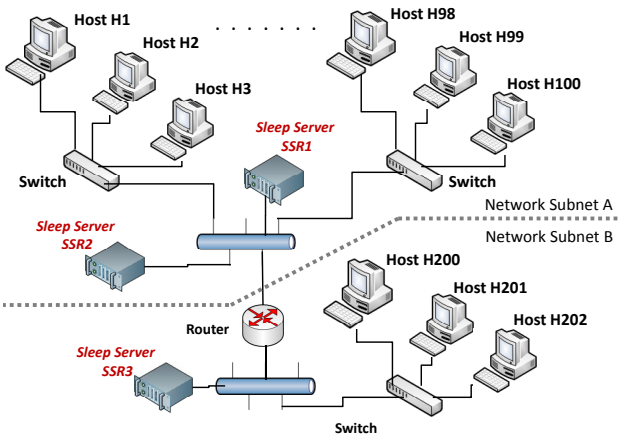

Figure 2: *An example deployment of Sleep Servers in an enterprise setting. Since there are many more hosts in Subnet A there may be more than one SleepServer (SS1 and SS2) to handle the load while there is only one SleepServer (SS3) needed in Subnet B with fewer hosts.*

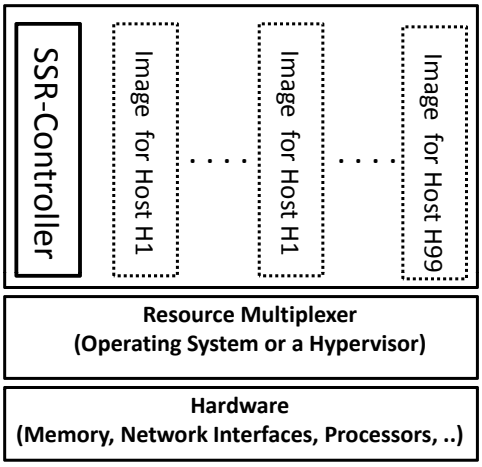

Figure 3: *A example SleepServer serving a collection of host PCs (H1, ...H99). All resource sharing and access to the hardware is mediated by the SleepServer controller software module running on the SleepServer.*

in Figure 3. As shown in the figure, access to the underlying hardware is by a resource multiplexer, which can be either an operating system or a hypervisor/Virtual Machine Monitor (VMM) such a XEN [7]. For each host H that the SleepServer is proxying for, there is a corresponding *Image* I that is instantiated. This image is responsible for maintaining the network presence of the host when it is in a sleep state. Although it is possible to build a host image as a stand alone process that can respond to the various network protocols, we chose a VMM-based architecture for simplicity and expediency. Since VMs are typically based on existing operating systems, all the standard protocols (e.g. ARP, ICMP) are already supported while support for others can be easily added. In contrast a process based approach would require adding support for the myriad of standard protocols. Furthermore, it is unclear how a process oriented approach would handle stateful applications that need application specific code (described in Section 3.2). VMMs also already have existing support for isolation between host images, for resource allocation and sharing, and for managing security and networking between images. While VMs may use more resources than a standalone process, our results show that the VM solution offers sufficient scalability for our purposes without requiring significant additional engineering. In addition to the host images, the SleepServer supports a privileged *controller* domain that is responsible for various SleepServer functions. This SSR-controller manages the creation and configuration of individual host images, communication between the SSR-Client software and the host images, and resource allocation and sharing among the host images.

Each host PC using SleepServer has a software component installed (SSR-Client) that communicates with the SSR-Controller. When a particular host is enabled for use with a SleepServer, the SSR-Client first connects to the SleepServer machine in its network subnet, and specifies its network parameters such as its MAC and IP address and its firewall configurations. In addition, the SSR-Client sends the state of the running applications on the host, and any open TCP or UDP ports to the SSR-Controller. The information sent by the host is received by the SSR-Controller which then creates an 'image' of that particular host using the specified network parameters. The network parameters of this image are configured to mimic those of the particular host. The base firewall configuration of this image can be identical to the one on the host, or can be made more restrictive. When the host is asleep, its image can respond to incoming packets on its behalf. In case an application request is received that requires the host itself to respond the SSR-Controller wakes up the host and disables its image on the SleepServer.

3.1 Handling State Transitions

The basic operation for a SleepServer is as follows. Before a host PC transitions to a low power mode such as sleep, the SSR-Client software running on it sends a message to the SSR-controller with the state transition information. The controller then enables the corresponding Image for that host. Additionally, in order to have packets re-routed to the SleepServer and the Image of the host, the controller needs to reconfigure the Layer-2 switches to re-learn the network topology. To do this in a seamless way, without requiring any special functional-

ity provided by only high-end switches, the SleepServer uses a combination of gratuitous ARPs and packets sent to the gateway in the subnet. Since these packets are sent by the Images of the host on the SleepServer, the Layer-2 switches learn the MAC address of the Image and subsequent packets for that host are sent to the switch ports that the SleepServer is connected to.

Similarly, when the host transitions out of a low power mode, the SSR-Client traps this event and sends a message to the SSR-Controller notifying it of the transition. When the SSR-Controller gets this message it disables the host image, thus stopping if from responding on the behalf of the host. The SSR-Client on the host also sends gratuitous ARP messages and packets to the subnet gateway which cause switches in the network infrastructure to learn the MAC address and forward any subsequent packets meant for the the host to the switch port that the host is connected to.

3.2 Host Images on the SleepServer

The *host-images* on the SleepServer are responsible for maintaining full network presence on behalf of the host when they are asleep. In principle, these host-images have their own TCP/IP stack, memory, processor resources and persistent storage. The host images do not need to run the same OS as the host computer. The image of a particular host is configured with the identical network configuration as the host itself (IP, MAC address) and it can essentially masquerade as the host and respond to network events when the host is asleep. However, processor and memory resources allocated to an image are generally much less than those available on the host machine itself, as these host images are configured to only maintain network presence and any application stubs that may be necessary to run (described later in this section). For example, a host image on the SleepServer may only have 64MB of memory allocated, while the actual host may have several Gigabytes of memory. Furthermore, shared resources such as the processor and network bandwidth allocation of the images are multiplexed between several other images on the same SleepServer, providing scalability and the ability to host hundreds of images on the same SleepServer.

Supporting Stateless Applications: Stateless applications do not maintain long running sessions or have a persistent connection open. To support these applications, the image responds appropriately to connection requests by doing one of two actions. First, the image can respond on behalf of the host for certain requests by sending an appropriate response, such as replying to ICMP requests or responding to ARP queries. Recall that since the network parameters of the image are the same as the host, they appear identical to the other hosts on the network. Second, for incoming requests that require the resources of the host itself, for example an incoming SSH connection to the host or an SMB request for data stored on the host computer, the image is disabled and the controller is notified. To ensure that the original connection request is handled appropriately, it is essential that the image of the host does not respond to it. Instead we rely on the fact that most applications are based on protocols, such as TCP, that normally retry connection requests in case of packet loss. Applications based on unreliable delivery protocols such as UDP usually handle packet loss at the application layer by retransmitting requests. Applications that are essentially stateless and connect to well defined ports, such as remote access requests using RDP (TCP Port 3389) or SSH (TCP Port 22), incoming SMB file sharing requests(TCP port 445), and requests to a web server (TCP Port 80), can be supported using this mechanism.

On receipt of the notification from a host image, the controller automatically generates a wakeup packet to wake up the host. This can be done using either Wake-on-LAN (WoL)[17], which can be found on most PCs, or by utilizing newer technologies like Intel AMT. WoL allows PCs to be woken up on receipt of several different kinds of packets. 'Wake on Directed Packets' and 'Wake-on-Any Packet' unfortunately cause too many wake ups since even broadcast traffic causes the PC to wake up. Instead, we use the "magic-packet" variant of Wake-on-LAN which can be sent by a SleepServer in the subnet to wake up the host from a sleep state.

Supporting Stateful Applications: Stateful applications maintain continuous state and send periodic keep alives or keep connections open. For these applications and protocols, capturing application semantics is essential in order to proxy for them by the image of the host on the SleepServer. To support these stateful application we require application specific code to be running on the images on the SleepServer. This is in contrast to the stateless applications mentioned in the previous section that do not require any application specific code on the images. A majority of these stateful applications run in the background, and can be active even when the user is not present in front of the system. Examples include maintaining presence on IM networks, long running and unattended web downloads, participating on P2P networks such as BitTorrent, and advertising available services and content using protocols such as Bonjour and uPNP.

To support these applications we take an approach similar to Somniloquy[3] where we run reduced functionality variants of the main applications, called 'application-stubs'. These stubs have significantly reduced processor and memory requirements as compared to the original applications running on the hosts. The key idea in developing a stub is to remove all the code com-

ponents of the application that are not needed on the host image, such as the user interface. Similar to Somniloquy, these stubs can be created by either writing them from scratch or by removing components of an existing application. In some cases console versions of the same applications are already available, such as the `pidgin` IM client and its console version called `finch`, which can be used as a starting point.

Although the process to build stubs is similar to that used in Somniloquy, there are several key differences that make it significantly simpler in the SleepServer architecture. First, because SleepServer can be run on any x86 based server computer, the host images themselves can also run on this industry standard architecture. Therefore, porting applications and building stubs for the SleepServer images is as simple as building an application for a regular computer with all of the standard libraries and packages available. In contrast, Somniloquy used an additional piece of hardware, with a different processor architecture, and required cross compiling applications. Second, the host images running on a SleepServer are based on software VMs, and as a result the amount of resources allocated to each host image can be dynamically changed. For example, the image of a particular host performing heavy downloads would have more memory and processor resources allocated to it, while the image of another host that is just replying to ICMP echo requests would have less. This is not possible with Somniloquy as each host PC has a dedicated piece of Somniloquy hardware physically attached to it, each with a fixed amount of resources.

In some cases it becomes necessary to transfer data between the host and its image running on the SleepServer. For example, consider a download stub that continues a long running download on behalf of the host when it is asleep. Once the host wakes up, the downloaded data needs to be transferred to the host. In the SleepServer architecture this state and data transfer can be handled by storing the data locally in the persistent storage provided to each image and then later sending it to the host over the network when the host is awake. Another option is to set up a network storage for each host, which can even be hosted by the SleepServer itself. The host and its image can then access the same unified storage to store data that is needed for SleepServer operation.

3.3 Scalability and Resource Sharing

Scalability, in terms of the number of hosts supported simultaneously on a single SleepServer, is an important design goal for both cost and energy savings. To keep the cost of the SleepServer low, we want to base the SleepServer on commodity components and have it support a large number of hosts. Therefore, we ensure that the individual images start with the smallest possible footprint, both in terms of disk space used and the number of processes they create, in order to minimize processor and memory usage. Furthermore, each image is further customizable such that only the application stubs or software modules that are needed by each host are loaded onto their respective images.

Beyond CPU usage, the potential scalability bottlenecks lie in the memory usage and network bandwidth requirements. Currently we allocate memory statically to the host images and only have as many images concurrently running as can fit in the main memory of the SleepServer. Given that our images start out with very modest memory allocations (64MB or less), a SleepServer with 32GB of memory can support over 500 simultaneously executing host images. Furthermore, since the host images are based on Virtual Machines (VM), we can employ techniques such as Difference Engine [13] which exploits memory compression techniques to significantly reduce the memory use of VMs. For multiplexing access to the processor and the network interfaces, we rely on the resource sharing provided by the underlying VMM.

3.4 Management in Enterprises

Security and manageability are important considerations in enterprises. System administrators are reluctant to add and support technology solutions that add administrative costs. We have implemented management modules that allow administrators to view in real time a 'heart-beat' of the systems supported on SleepServer. Since all state transitions such as hosts going to sleep and resuming are logged by the SleepServer controller, it can also provide users of those particular PCs feedback on their energy usage in real time and their estimated energy savings. SleepServer administrators can check the health of the host machines and see if they are transitioning in and out of sleep modes successfully. SleepServer also adds to the observability of the state of machines. For example, it is possible to tell the difference between a computer in a sleep state against one that has crashed. Through the centralized management interface, administrators can also set up host specific policies, such as waking up some hosts at designated times, and perhaps even staggering wake ups to minimize spikes in energy usage.

Similarly, failure detection and recovery are important, such as handling the case when a SleepServer goes down. Note that under all circumstances the hosts that are sleeping can still be woken up normally by a user action such as a key press on the keyboard. Hosts that are awake and were not being serviced by the SleepServer are not affected by a SleepServer failure, while hosts that were asleep will lose network connectivity. Fur-

thermore, if the SleepServer is unavailable any hosts that want to transition to sleep and maintain their network presence and availability can no longer do so. The SleepServer architecture handles these failure cases using several mechanisms. First, for a temporary failure or an intentional reboot after updates the SSR-controller re-creates the state of the various hosts from its logs and restarts all the host images to their original conditions. Second, multiple SleepServers can exist and proxy for a particular host. The different SSR-Controllers in this case communicate with each other to provide redundancy and load balancing. Finally, hosts can discover and check for the availability of their SleepServer and in case the SleepServer is not responding, they can look for alternatives. If no SleepServer is available the SSR-Client running on the host alerts the user about the lack of an appropriate SleepServer in the network and can let the user decide if they still want to transition to sleep.

Security and Isolation of the Host images: Addressing the security implications of SleepServer is important since multiple host images are hosted on the same SleepServer. We need to ensure that the host images do not increase the attack surface of the hosts within an enterprise while keeping them safe from outside attackers. Furthermore, the individual images should not be able to receive and intercept each others network traffic.

While we do not currently have a comprehensive security evaluation, there are several features and safeguards in our SleepServer architecture that address security. First, since the SleepServer is based on a VMM architecture the SSR-Controller domain runs at a higher privilege level than the individual images. The SSR-Controller therefore has the responsibility to add rules to route traffic to the appropriate image. As such, only the packets that are meant for a particular host image are send to it, in addition to broadcast and multicast traffic. Second, the host images are not accessible by users of the host PCs directly. Instead, all communication between the SSR-Client software and the host image goes through the SSR-Controller. Third, the firewalls on the host images is configured to be very restrictive and opened only to the ports for which the host and its application stubs require. Note that the firewall on the host images can be configured to be even more restrictive than that of the actual host. Fourth, the host images only communicate with the SleepServer controller directly to get configuration changes and can be patched by the controller. Furthermore, we enable only the essential services and programs in the host images, and as such the attack surface is relatively narrow. Finally, recall that on a valid incoming connection request the particular host is immediately woken up from sleep by the controller and its image stops responding. In this case, the security implications are identical to the case where the host remained awake.

4 Implementation

We have implemented SleepServer on a commodity server computer and are currently serving over thirty desktop users on it. In this section we outline our implementation of SleepServer, specifically highlighting how we support the design goals mentioned earlier in Section 3. There are three primary software components that are required. The first is the SSR-Client software that runs on the host computer. The second component is the SSR-Controller which runs on a SleepServer computer. The third component are the host images themselves, each supporting a host PC using SleepServer.

4.1 SS-Client Software for Hosts

SleepServer currently supports several common operating systems, such as Microsoft Windows (XP, Vista and 7) and Linux (tested on Ubuntu). Windows based operating systems have standardized power management interfaces but different distributions of Linux can have different interfaces to handle power management and therefore require different client software.

While SleepServer can support multiple low-power states, in our evaluation we use only the standard 'sleep' state or suspend-to-RAM (ACPI State S3) across all machines [1]. In some cases this requires changing the BIOS settings of the host to enable the S3 state. Since we are leveraging Wake-on-LAN functionality, and specifically 'magic packets', we require the appropriate options to be enabled in the BIOS, the device drivers and the operating system. Most PCs manufactured in the last decade support S3 and Wake-on-LAN, although these modes often need to be enabled explicitly.

The SSR-Client software on the host computers is responsible for providing mechanisms to detect power management events, such as transitions in and out of sleep modes, and transfer state information to the SSR-Controller such as firewall configuration, network information (IP, MAC addresses), applications and events that the host wants to be notified for.

Note that modern operating systems already have user-configurable power management idle timers that use events, such as keyboard, mouse, and CPU activity to determine when the host is inactive and able to sleep. SleepServer users can use the same interface to configure their idle preferences. It is important to note that almost all of the users in our deployment had these power management timers disabled before using SleepServer since they wanted to be able to access their PCs at all times. In our evaluation we compare the additional energy savings gained by using these automatic idle timeouts, as compared to having no automatic idle timeouts and instead asking users to manually put their machines to sleep.

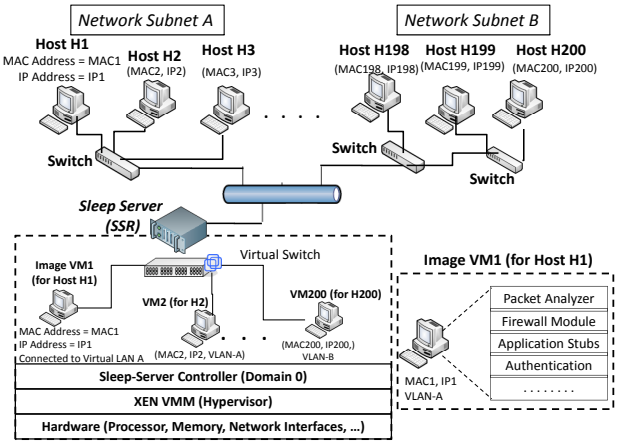

Figure 4: *SleepServer implementation based on XEN.*

Windows Platforms: The SSR-Client for Microsoft Windows is comprised of several programs and services. The first component is an initial setup program that is used to read the firewall configuration of the host PC as well as its network configurations (IP address, Host name, MAC address) and send this information to the SSR-Controller. Anytime these parameters change, this program sends an update to the SSR-Controller. The second component is a 'PowerNotifier' that is responsible for updating the SSR-Controller of any change in the power state of the host. Since there are various ways a user in Windows can transition to sleep modes, the PowerNotifier service installs hooks directly into the Windows Power Management Interface (WMI) so that it is notified on any power state changes. When a suspend or resume from sleep event occurs the PowerNotifier component sends a message to the SSR-Controller.

Linux Platforms: Similar to our implementation for Windows platforms, we have several components that run on the Linux host. The Ubuntu distribution allows access to the power management events through the ACPI subsystem and our PowerNotifier service for Linux is installed by placing appropriate hooks into this ACPI framework. PowerNotifier is called on both sleep and resume from sleep events and is responsible for communicating with the SSR-Controller. Additionally, we use standard Linux tools such as `iptables` and `ifconfig` to get network and firewall configurations.

4.2 SleepServer

We have implemented the SleepServer on a commodity Dell PowerEdge PE2950 server, which is configured with two quad-core Intel XEON 5550 processors, 32GB of RAM, a 1TB SATA disk drive and dual gigabit interfaces. Figure 4 illustrates the logical organization of our SleepServer prototype, and the host images running on it. The SleepServer runs a XEN 4.0 [7] hypervisor/VMM (using the 2.6.32 pvops kernel) and the SSR-Controller (domain0 in XEN) is based on Ubuntu 9.10. We have modified the XEN utilities to allow creation of customized SleepServer Virtual Machines (VM) (domU's in XEN) representing host images based on supplied network parameters such as Host name, IP address, MAC address, etc. We configured the SSR-Controller to have several virtual interfaces that allow it to be placed on all of the VLANs (eight different subnets) in the CSE department network at UCSD. We also configured the department managed switches so that traffic on all these VLANs is forwarded to the switch port that the SleepServer is connected to. The SSR-Controller then sets up software network bridges automatically for each configured VLAN. We initialize the VMs to only have access to those VLANs that the host they represent are originally on, with identical network parameters (IP, MAC address etc) as the hosts. For example, image VM1 for Host H1 can only access VLAN A and will have the same IP and MAC addresses as the host H1 (Figure 4). Additionally, the SSR-Controller and the host VMs communicate over a separate private network.

The SSR-Controller listens for messages from the hosts on several well defined UDP and TCP ports. Recall that our SSR-Client software running on the hosts automatically sends state transition messages to the SSR-Controller. On receipt of these messages the SSR-Controller enables (if the host is going to sleep) or disables (if the host is resuming from sleep) the VM for the appropriate host. The SSR-Controller is also responsible for reconfiguring the VM for a particular host when the SSR-Client sends an update, such as adding or deleting new applications. Additionally, the SSR-Controller maintains a log of all sleep/resume events received. These recorded events are used by a separate status module which allows users to view the status of their PCs. This log is also key in calculating the duty-cycle of all the hosts served on the SleepServer, and can provide estimates of energy savings given the average power draw of the machine in sleep and active modes. The SSR-Controller has a wake-up module that is used to generate wakeup packets using Wake-on-LAN to resume a sleeping host when needed. Note that since both the SSR-Controller and the wakeup-module have a presence on all the VLAN's, they can send Wake-on-LAN magic packets on any department subnet (same layer-2 domain) without any router configuration. Finally, the SSR-Controller has a performance monitor module that periodically measures statistics such as processor usage, network throughput, and free memory using hooks provided by XEN. Feedback from this module can be used by the SSR-Controller to wakeup some hosts and disable their corresponding VM images in case the load

on the SleepServer exceeds capacity. In case of multiple SleepServers on the same subnet, individual SSR-Controllers communicate with each other to provide load balancing and redundancy.

Although we have implemented SleepServer on a separate server machine, it is feasible to have a scenario where the SleepServer functionality can be supported on enterprise PCs themselves. A subset of enterprise PCs can run a hypervisor and the SSR-controller and can host the images of other PCs that are asleep thus proxying for them. However, there may be other implications of this approach that we have not fully evaluated such as maintaining security, resource allocation and isolation.

4.3 Host VM Image Image

The host images running on the SleepServer are XEN VMs based on the standard x86 architecture executing a stripped down version of Ubuntu Linux. After installation the VMs take up less than 300MB out of their initially allocated 1GB disk image. Given that only the essential services are run inside the VM, our initial memory allocation of 64MB is more that sufficient with most of the memory free (>40MB) after boot up.

Each VM is configured to have several software modules to support SleepServer operation, as illustrated in Figure 4. Each VM has a full TCP/IP stack and can therefore respond on behalf of the host to packets such as ICMP echo-requests or ARP queries. The VMs have a firewall based on the `iptables` package which is configured to be identical to the host firewall by the SSR-Controller. The 'Packet-Analyzer' (PA) module is used to support stateless applications such as incoming RDP or SSH requests. The PA module is based on the BSD raw socket interface and is used to parse incoming packets to look for matches on one or more fields of packet headers, such as incoming requests on particular TCP and UDP ports. In case the incoming packet matches one of the application ports, the firewall is configured to not send a response to the initial request. Instead the PA sends a message to the SSR-Controller and disables the network interface of the VM. Upon receipt of this message, the SSR-Controller uses the wakeup module to send a wakeup packet to the host as described earlier that relies on retries inherent in TCP. When the host resumes, it receives one of the retransmits and the session can be established. For stateful applications we have implemented application stubs similar to those proposed in Somniloquy [3]. We currently support several application stubs namely a multi-protocol instant messaging stub, a background web download stub and a BitTorrent stub that allows participation in P2P networks. Given the x86 compatible architecture of the VMs, and the availability of standard libraries and tools, implementing these stubs is significantly easier than on the specialized Somniloquy hardware device.

In a way, supporting a large number of stateful applications may be considered a barrier to deployment since each application requires its own corresponding stub. In our experience with deploying SleepServer, we did not observe this to be an issue, especially in enterprise settings, for several reasons. First, a significant portion of users can be supported without requiring stubs, as long as seamless connectivity (responding to ARPs, ICMP) and user selectable wakeup on incoming connections is handled (e.g. SSH, RDP, SMB, backups and updates). This observation is in fact similar to the findings of previous measurement based studies [5, 20] in this space. The rest of the users in our deployment requested support for a relatively small number of common services (and hence stubs). The "fall-back" position of waking up the PC provides a fail-safe default for those applications or protocols that we do not yet proxy. Indeed, additional stubs will only improve upon the energy results we report.

4.4 Discussion

An emerging use model in enterprising computing is based around Virtualized Desktops environments. The common scenario is when all desktops reside on a centralized server and users utilize thin clients to connect over the network to their desktops. In this setting we believe the lightweight proxying functionality that SleepServer offers can potentially increase the density of inactive desktops, thus improving scalability. Another scenario is based on the assumption that each Desktop PC runs a Hypervisor itself (e.g. XEN or VMware) and the actual users 'virtual desktop' runs as a VM on top of the VMM/hypervisor. When the user is in front of his actual PC, his virtual desktop runs on the local VMM and when the user steps away or logs out the entire VM can be migrated to a central server, or pool of servers. The advantage of this architecture over SleepServer is that no application stubs are needed since entire VMs are migrated. However, a potential drawback of this approach is its limited scalability since the amount of state that needs to be transferred for each virtual desktop can be quite large. For example, the memory footprint alone may be up to several gigabytes based on the hardware configuration of the host PC and the entire OS state, including all applications. is transferred and has to be kept running even if the user only wants a small subset of the functionality when they are away. Furthermore, the local persistent storage may also need to be migrated. Despite using techniques like memory ballooning and deduplicating memory, the scalability of a virtual desktop based infrastructure will most likely be significantly limited than that of SleepServer which uses very lightweight

	Machine Type	Year	OS	Average Power in S3	Average Power when on (idle)	Time to Resume from S3 (network)
1	Dimension 4500	2002	WinXP	2.5 Watts	75 Watts	29 (+/- 4.1) seconds
2	Dimension 4500	2002	WinXP	2.4 Watts	61 Watts	28 (+/- 1.6) seconds
4	Dimension 4600	2004	WinXP	4.4 Watts	76 Watts	29 (+/- 3.0) seconds
4b	(Same as above - Dual Boot)	2004	Ubuntu	4.6 Watts	74 Watts	12 (+/- 1.8) seconds
5	Dimension 4700	2005	WinXP	2.2 Watts	111 Watts	30 (+/- 10.0s) seconds
6	Optiplex SX260 Small Form Factor (Desktop + SSH/CVS Server)	2004	Ubuntu	5.1 Watts	67 Watts	10 (+/- 7.44) seconds
7	Optiplex GX280 Small Form Factor	2005	WinXP	3 Watts	86 Watts	25 (+/- 5.3) seconds
8	Optiplex 755 (Desktop, SSH + file server)	2007	Ubuntu	2.8 Watts	84 Watts	14 (+/- 2) seconds
9	Optiplex 745	2007	Vista	3.3 Watts	107 Watts	8 (+/- 1.4) seconds
10	DELL XPS 720 (Drives LCD Display + Webserver)	2008	Win XP	4.2 Watts	314 Watts	9 (+/- 7.7) seconds
11	Optiplex 960 Small Form Factor	2009	Win 7	2.3 Watts	45 Watts	12 (+/- 5) seconds

Table 1: *Power consumption for an example set of PCs in our deployment. Resume from S3 times are much better for newer machines and operating systems. Base power consumption of newer PCs still remains high and power consumed in S3 ranges from 1/20 to 1/75 of that in idle mode.*

VM images which can be as low as 32MB in footprint. Going forward, we do believe that Virtual Desktop based solutions and the significantly lightweight proxying approach offered by SleepServer are synergistic and both may be useful based on specific use cases. For example, a SleepServer can potentially host full virtual desktops when particular stubs may not be available. We also believe that any investments that application vendors make into building stubs for their applications will be useful for both a lightweight VM based approach taken by SleepServers, as well as potential hardware solutions that add proxying functionality to network interface hardware[3, 16].

5 Evaluation

We first present micro benchmarks highlighting our experience with deploying SleepServers to various hosts in our department. We then evaluate the SleepServer itself, benchmarking its power consumption under various loads and measuring the latencies for management tasks such as creating, starting and shutting down new host images. We also present experimental data about the scalability of SleepServers demonstrating that we can easily scale to serve several hundred hosts on a single SleepServer machine. Finally, we present data that shows the energy savings of the various host PCs in our deployment.

5.1 Micro Benchmarks

We have deployed SleepServer on a variety of host PCs in our department building. In total we have over thirty desktop PC users including a couple of laptop users participating in our SleepServer deployment. The users range from faculty and students to full time staff workers to give us a mix of use-scenarios. Also, the mix of machines range from PCs that are well over 7 years old to those that are fairly new. The operating systems running on these PC range from Linux (Ubuntu) to all versions of Windows, including numerous Windows XP machines.

Table 1 shows the distribution of some representative PCs that are part of our SleepServer deployment. Our goal in benchmarking these systems was to see whether we could observe any trends in the design of PCs and operating systems. We benchmarked these systems based on power consumption in various states of operation, and the latency of these systems when they resume from sleep. We only show the latency measurements to resume from sleep, since latency to go to sleep is less important from a usability standpoint. We have instrumented all the machines in our deployment to provide real time energy measurements using a commercial energy meter from WattsUP devices[1]. We have also made this energy data available to SleepServer users to view over the web using an 'Energy-Dashboard' interface that we have designed [4]. In addition to viewing their power usage in real time, users can also look at long term trends such as comparing their usage over different time periods.

Our instrumentation of the thirty desktop computers in our SleepServer deployment using energy meters gives us long term power use data, allowing us to measure and quantify the impact of using SleepServers under different usage scenarios. We observed that most users in our deployment did not put their machines to sleep before

[1] www.wattsupmeters.com

they started to use SleepServers, as measured by over five months of power usage data by these machines.

Table 1 reports the power consumption and latency values for an example set of SleepServer PCs. We do not include the power consumed by LCD displays connected to these PC, since most of them are configured to go into sleep modes on inactivity. Several interesting observations can be made from the table. First, the power consumption in sleep (S3) mode for most of the PCs is significantly less than when they are in idle mode. This is even true across operating systems (line 4 and 4a for the same PC in the table). Second, the power consumption of PCs has not come down significantly during the last 7-8 years, as idle power for desktops remains around 80 Watts for even new PCs. Third, the latency to resume from sleep varies significantly across platforms. We measure the latency to resume by measuring the time from a wakeup event, such as a key press on the keyboard, to the time it takes for the network stack on the host PC to respond to an incoming ICMP packet. Although the display and logon screens on the host may come up earlier, we believe measuring the latency for a network response is a better metric to use. Table 1 shows that in some cases resume latencies are up to 30-40s (line 5), with a large standard deviation in time to resume. We also notice that the resume time on different operating systems (line 4 and 4a) on the same hardware platform are significantly different. We believe this is mostly due to the different applications, devices and drivers that are installed on PCs over time and can cause delays in startup. Importantly, as we can see from the table, resume times are getting significantly better as we move to more recent hardware (2007 and newer) and modern operating systems.

5.2 Scalability of SleepServer

The hardware and software configuration of our SleepServer prototype was presented earlier in Section 4. We measured the power consumption of our prototype under various operating conditions using a WattsUP device. We also measured the latency to create a new SleepServer image for a particular host, and the time to start up an existing VM and shut it down. These latencies are important to consider for dynamically creating new VMs when new hosts are added to a SleepServer.

The latency and the power consumption values are shown in Table 2. The latency to create a new host image from scratch is on the order of two minutes. This includes creating the image, installing the SleepServer supporting software, configuring the SleepServer controller and updating all packages and security updates. To reduce this latency, the SleepServer allows creation of a pool of VMs, which can be updated with the network

	SleepServer function	Time (seconds)
1	Creating a new host image	120s (+/- 10)
2	Starting up a host image	11s (+/- 1)
3	Shutting down a new host image	12s (+/- 1)
	Sleep-Server - State	**Power (Watts)**
4	Idle State, no host images running	213 W
5	Hosting 200 *idle* host images	221 W
6	Download + Write to Disk	255 W
7	CPU benchmark, (100% CPU util.)	308 W

Table 2: *Benchmarking the Sleep-Server: Latency and Power Measurements*

configuration of a host. The time to start up an existing VM and shut it down is around ten seconds. To reduce the startup latency even more we have enabled only the essential services in the VMs. This latency is important, since it means that given the transition times presented earlier in Table 1, it is possible to *dynamically* start up VMs and have them activated by the time the host finishes its transition to sleep. Alternatively, the host VMs can be started up if memory and CPU on the SleepServer are not a constraint. Finally, the time taken for our prototype SleepServer machine to boot up from a powered off state, to recreate state information from its logs, and to start up the VM images is on the order of a few minutes.

Next, we tested the scalability of our SleepServer prototype by instantiating a large number of VMs on it and measuring the effect on the processor and the memory utilization and impact on I/O performance. Since we allocate 64MB of memory to each VM, that gives an upper bound of approximately 500 VMs executing simultaneously for the 32GB of main memory in our SleepServer prototype. Unfortunately due to some limitations in XEN and the Linux kernel, we were unable to scale beyond 200VMs. The limitations relate to the low number of statically defined software interrupts in the XEN kernel, as well as the number of block devices (disks) supported. We have reported these limitations and the fix should be released in an upcoming update.

Figure 5a shows how increasing the number of VMs impacts the overall CPU and the memory utilization of the SleepServer. The processor utilization increases linearly and remains low (20%) even at 200 VMs (idle), giving almost 80% idle time for the CPU. The low CPU utilization is as expected, since most of the idle VMs are in a blocked state waiting for I/O (e.g. network packets) requests. The memory utilization also increases linearly as we increase the number of VMs, since each VM uses an additional 64 MB. Next we benchmark the performance of these VMs under I/O load, by setting up an experiment where a number of VMs download data from a fast local webserver using a web download stub. As

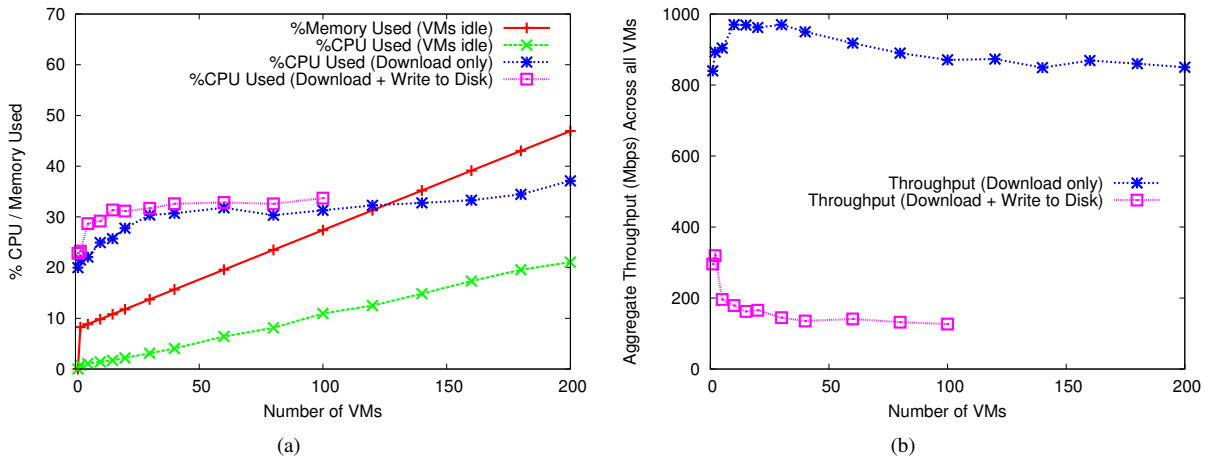

Figure 5: *Effect of scaling the number of VMs. The graph on the left (a) shows the memory and the CPU utilization as we increase the number of VMs. The amount of memory used under additional network traffic by the individual VMs does not change and is therefore not shown. The graph on the right (b) shows the total aggregate throughout observed by all the VMs as we increase the number of downloads.*

we increase the number of VMs simultaneously running this benchmark, we measure the CPU and memory usage and the aggregate throughput observed by the VMs. We report these figures for two cases: when the download is not saved to disk marked as 'Download Only', and when the VMs save the downloaded data to their local storage marked as 'Download + Write'. When the VMs are not saving the data to disk, the aggregate network throughput is shared evenly between all VMs, and the downloads almost saturate the 1Gbit link (>800Mbps for 200VMs). The CPU utilization increases to 40% even at 200 simultaneous downloads. However, when the VMs are writing to disk, CPU utilization rises to about 35%, while the download throughout reduces to about 136Mbps (Fig 5b) for 50 simultaneous downloads and to 126Mbps for 100 downloads. This can be explained by disk seek times, caused by each VM writing to its image, starting to dominate as the number of VMs increase, thus limiting performance. We did not measure 'Download Write' performance beyond 100VMs since we started to observe disk driver timeouts for some of the VMs. Using faster disk drives or striping the VMs across separate local hard drives on the SleepServer, or by using a network storage element should expectedly improve performance. As dicussed in Section 4 earlier, the SSR-Controller can detect this condition and choose to wake a few of the hosts to alleviate any I/O bottlenecks. We measured the average ICMP latency from a local machine to the VMs on the SleepServer, as a measure of network responsiveness of the VMs under load. The round-trip latency was under 5ms under all conditions.

The primary goal of SleepServer is to enable users to put their PCs to sleep to save energy, while maintaining their availability to both network and application level events. Using the power consumption logs captured by the WattsUP meters, we can calculate the energy consumed by the various PCs over different periods of time and use that to calculate the energy savings. The energy savings for users is also dependent on how often users actively put their machines to sleep. As an experiment, we first let users use SleepServer in a mode where they were responsible for putting their machines to sleep manually. For the next week we modified the standard power management settings for some users such that after one hour of idleness, as detected by the power management functions of the host OS, the PC would automatically go to sleep. Note that for both these cases, the users were aware that they would be able to use the SleepServer functionality to access their machine and maintain connectivity when their computers were in sleep mode.

Figure 6 shows the power consumption trace for a typical user of our system drawn from the group of thirty users. This figure compares the power consumption of the user's PC over a 2 week period, first without SleepServer (August 31st - September 13th) and then when the user started to utilize SleepServer (September 14th - September 27th). Additionally, for the first week of deployment (Sept 14th - Sept 20th) the users were asked to put the machine to sleep manually when it was not in use, while for the second week (Sept 21st - 27th) the one hour idle-timeout was instituted. For the first week the energy consumption of this user dropped by 30% as seen by the frequent transitions to sleep. There were however several cases during the first week when the users forgot to put their machines to sleep despite the fact that they were not actively using the PC (e.g. Sept

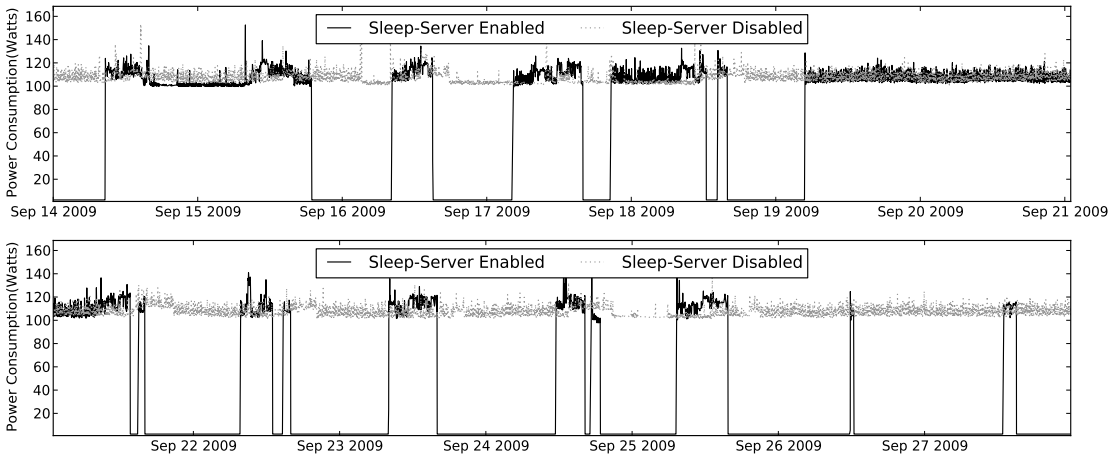

Figure 6: *Comparing the Power Consumption for a Desktop PC with and without Sleep-Servers. For the first two weeks from August 31st - Sept 13th the user was not using SleepServer, while from Sept 14th to September 28th, SleepServer operation was enabled. Additionally, from Sept 22nd onwards the PC was set to automatically go to sleep within one hour of idleness.*

15th, Sept 20th and 21st). In Week 2, when we instituted the one hour timeout policy, there were more transitions to sleep (Sept 22nd, Sept 24th), even during the day. The end result was that the user saved an additional 54% energy between Sept 21st - 27th over the previous week, giving a total enegy savings of 68% over the period where the PC was always on. We also notice that the user logged in to his PC remotely during the weekend (September 27th and 28th), and that the PC went back to sleep afterwards.

5.3 Energy Savings Using SleepServer

Of course, the energy savings for a particular user or a PC are based on its usage scenario. Graduate students in our department tend to stay longer, while most staff and faculty have relatively fixed hours. A significant fraction of people do however connect to their PCs remotely, and in some cases even run services like a web server or a CVS repository on their machines which would normally preclude them from putting their machines to sleep. Using SleepServer our entire deployed set of more than thirty users were able to put their machines to sleep. In Figure 7 we have plotted a representative set of eight host PCs for two weeks in September 2009. To simplify the chart, we have plotted a step function denoting the state of these PCs rather than absolute power consumption values. The times when the host is active (and its image on the SleepServer is disabled) is marked by an 'A', while 'S' marks the times when the PC is asleep (and its image on the SleepServer is enabled). The hosts are ordered from top to bottom in terms of energy savings, with PC1 seeing the most savings and PC8 seeing the least.

There are several important observations from Figure 7. First, we can clearly see the advantages of instituting the one hour idle timeout for certain users. Users of PC2 and PC3 forget to put their machines to sleep and as a result their PCs remained on through the weekend of Sept 12th/13th (marked by a '1' in the chart). When the automatic timeouts were instituted, most of the PCs remained asleep for longer periods of time including over the weekend of September 19th/20th (marked by a '3' in the chart). Second, while there were some trends in terms of machines being turned on in the morning when the users came in to work, the distribution of when the machines are on or sleeping using SleepServers is quite varied over the week. Users of PC4 and PC8 for example log in to their PCs to work over the weekend (marked by a '2' in the chart). This points to the fact that a simple scheduled policy of waking up PCs at pre-determined work times does not suffice. By mining the SleepServer controller logs we can also determine what caused particular PCs to wakeup. PC1 for example runs a Web server; any request to access the website therefore causes the SleepServer to wake up the machine. After a configured period of inactivity PC1 goes back to sleep causing frequent state changes. It is important to note that a major fraction of the users in our deployment were running one or more application that otherwise would have required the user to keep their machine powered on. During the course of our study, for example, our measurements show that 22 out of the total 30 machines needed to be woken up. Furthermore, despite the limited number of stubs we currently support, 6 out of 30 users utilized one or more stubs during our study. Of course the particular stubs that are required may depend on the enterprise environment, and we expect to gain more experience as

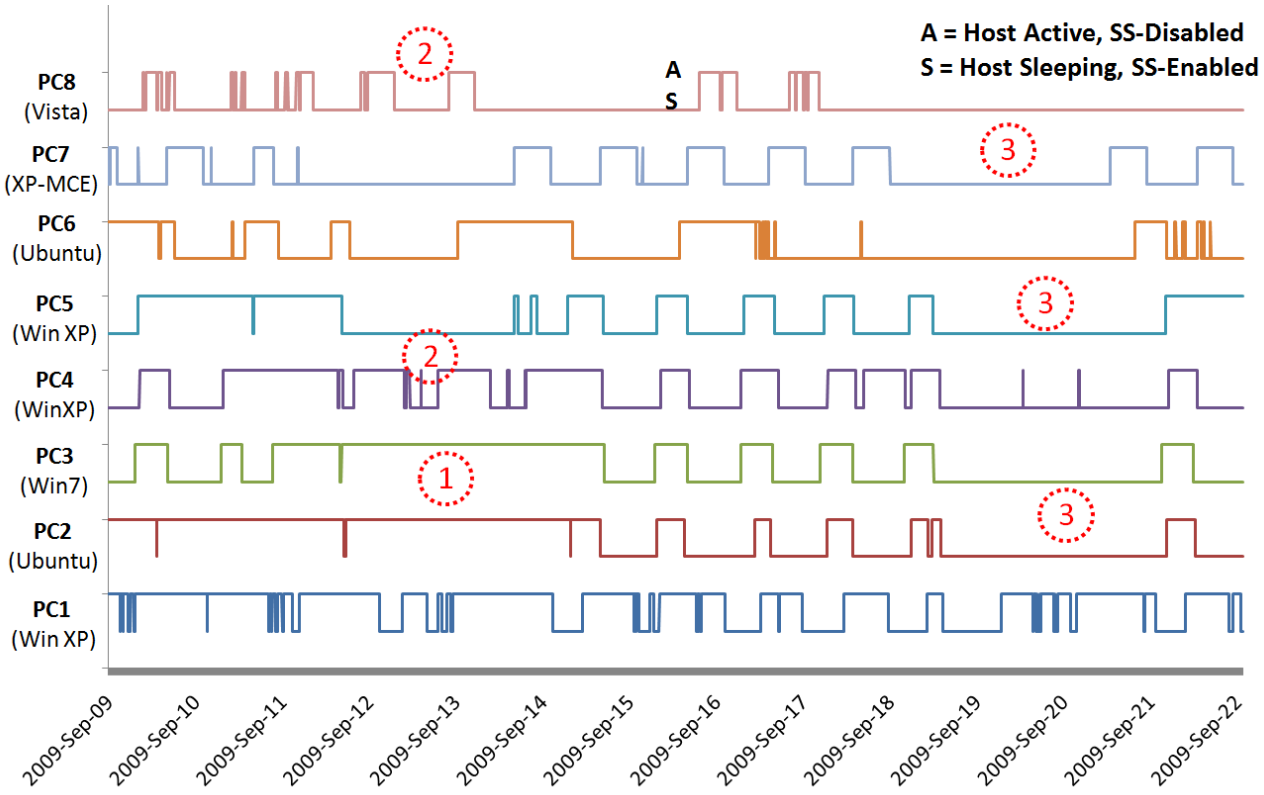

Figure 7: *Showing eight different hosts on using SleepServer over a two week period. For each host the graph shows the times when the host was on and its image on the SleepServer was disabled (denoted by A) and when the host as asleep and the SleepServer was proxying for it (denoted by S).*

we deploy SleepServers further. However, we do believe that it is important to support the capability of handling stateful applications in the SleepServer architecture for widespread adoption. The energy savings for the example set of 8 PCs shown in Figure 7 is significant, ranging from 27% (PC1) to 81% (PC8) for this two week period. The measured energy savings across all machines in our deployment for the month of September range from 27% to 86%, with an average savings of 60%.

6 Conclusion

In this paper we have presented SleepServer, a software-only implementation of proxy architecture that allows end hosts to utilize low power sleep modes frequently and opportunistically to save energy, without sacrificing network connectivity or availability. Within enterprise networks, a SleepServer machine can maintain network presence on behalf of a host while its sleeping by responding on behalf of the host seamlessly and waking it only when required. SleepServers are easily deployable since they require no changes to existing hardware, software or networking infrastructure and can be supported entirely using a simple software agent on the end hosts. We demonstrate that SleepServer is portable across a range of operating systems and hardware platforms and show how our prototype implementation can scale to support hundreds of client hosts on a single commodity server.

SleepServer is both practical, easy to deploy and very scalable. A large number of clients (and thus VMs) that are doing intensive disk activity might limit scalability due to heavy disk I/O. However, a case can be made to limit or avoid putting such machines to sleep. Instrumenting thirty heterogeneous desktop users, we show energy savings ranging from 27% to 86% with an average savings of 60%. Extrapolating from these results and assuming an average idle power consumption of 93Watts per desktop (from Table 1), and a use factor of 40% (machines are asleep for 60% of the time), we expect to reduce the baseline power use of the CSE building from 320KW to 245KW during nights and weekends. At current California energy prices of 9 cents per KW-Hr, this translates to over US$ 35,000 in annual cost savings alone, easily paying for the cost of a Sleep-Server within a few months.

7 Acknowledgements

We would like to acknowledge Thomas Weng for his feedback on the paper and his invaluable help with setting up the energy vizualization framework, the Energy Dashboard. We also wish to thank the anonymous reviewers of our paper for their detailed feedback. Finally, we are grateful to our shepherd, Orran Krieger for guiding us towards the final version of the paper. This work is in part supported by the NSF CNS-0932360 grant and Multiscale Systems Center (MuSyc) under the Focus Center Research Program (FCRP) supported by DARPA/MARCO.

References

[1] ACPI. Advanced Configuration and Power Interface Specification, Revision 3.0b, 2006.

[2] 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*, 2007.

[3] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage. In *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, 2009.

[4] Y. Agarwal, T. Weng, and R. Gupta. The Energy Dashboard: Improving the Visibility of Energy Consumption at a Campus-Wide Scale. In *in BuildSys '09*.

[5] M. Allman, K. Christensen, B. Nordman, and V. Paxon. Enabling an Energy-Efficient Future Internet Through Selectively Connected End Systems. In *6th ACM Workshop on Hot Topics in Networks (HotNets)*.

[6] Apple. Wake-on-Demand. `http://support.apple.com/kb/HT3774`.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[8] J. Chase, D. C. Anderson, P. Thakkar, A. Vahdat, and R. P. Doyle. Managing Eenergy and Server Resources in Hosting Centers. In *Proceedings of SOSP '01*, 2001.

[9] DOE. Buildings Energy Data Book, Department of Energy, March 2009. `http://buildingsdatabook.eren.doe.gov/`.

[10] DOE. CEC End-Use Survey, CEC-400-2006-005, March 2006. `http://www.energy.ca.gov/ceus/`.

[11] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic Performance Setting for Dynamic Voltage Scaling. In *Proceedings of MobiCom '01*, 2001.

[12] J. Flinn and M. Satyanarayanan. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Trans. Comput. Syst.*, 22(2):137–179, 2004.

[13] D. Gupta, S. Lee, M. Vrable, S. Savage, A. Snoeren, G. Varghese, G. Voelker, and A. Vahdat. Difference Engine: Harnessing Memory Redundancy in Virtual Machines. In *Proc. of 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, 2008.

[14] 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*, 2003.

[15] Intel. Intel Remote Wake Technology. `http://www.intel.com/support/chipsets/rwt/`.

[16] M. Jimeno, K. Christensen, and B. Nordman. A Network Connection Proxy to Enable Hosts to Sleep and Save Energy. In *IEEE IPCCC '08*.

[17] P. Lieberman. Wake-on-LAN technology. `http://www.liebsoft.com/index.cfm/whitepapers/Wake_On_LAN`.

[18] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of ASPLOS '09*. ACM New York, NY, USA, 2009.

[19] R. Nathuji and K. Schwan. Virtualpower: Coordinated Power Management in Virtualized Enterprise Systems. *ACM SIGOPS Operating Systems Review*, 41(6):265–278, 2007.

[20] S. Nedevschi, J. Chandrashekar, B. Nordman, S. Ratnasamy, and N. Taft. Skilled in the art of being idle: reducing energy waste in networked systems. In *Proceedings of USENIX NSDI '09*.

[21] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In *Proceedings of USENIX NSDI '08*.

[22] T. Pering, Y. Agarwal, R. Gupta, and R. Want. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. In *MobiSys*, 2006.

[23] A. Qureshi, H. Balakrishnan, J. Guttag, B. Maggs, and R. Weber. Cutting the Electric Bill for Internet-Scale Systems. In *SIGCOMM*, 2009.

[24] 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.

[25] 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.