Somniloquy: Maintaining Network Connectivity While Your Computer Sleeps

Yuvraj Agarwal[†‡], Steve Hodges[†], James Scott[†],
Ranveer Chandra[†], Paramvir Bahl[†], Rajesh Gupta[‡]

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

ABSTRACT

Reducing the energy consumption of computers is becoming increasingly important with rising energy costs and environmental concerns. It is especially important for mobile devices such as laptops where battery lifetime is always an issue. Sleep states such as S3 (suspend to RAM) save energy but prevent the device from responding to incoming network events, for example remote desktop logins and file transfer requests. Thus many people do not use S3 and instead leave their computers plugged in and active. Mobile users do not have this flexibility, and as a result their machines are unreachable during the periods when they are in S3.

Somniloquy allows computers in S3 to be woken based on incoming network traffic such as incoming VoIP calls or remote file access requests, or other events such as the presence of particular access points. This is done by adding a secondary embedded processor attached to a network interface, forming a low-power domain that remains active even when the rest of the computer is in S3. This secondary processor acts transparently on behalf of the computer, sharing the same MAC address, IP address, host name, etc. Remote servers and network hardware remain completely unaware of the low-power state.

We present a prototype implementation of Somniloquy using a USB peripheral, which is therefore easily retrofitted to existing computers. Our prototype achieves a ten-fold increase in battery lifetime compared to an idle computer not in S3, while only adding 4-7s of latency to respond to application-layer events. Our system allows computers to appear always-on when they are in fact talking in their sleep.

1. INTRODUCTION

As the use of networking has proliferated, computers need not only to respond to local users, but also to remote users. Remotely triggered actions include, for example, access to local files, access to the desktop GUI, or establishing two-way communication with a local user via a VoIP call (when the local user has indicated their availability). Unfortunately, this remote access is not compatible with current power-saving schemes such as suspend-to-RAM (S3), during which local users can easily initiate wakeup but the computer is not responsive to remote network events.

This situation causes inconvenience to mobile users, who have no choice but to place their laptops, tablet PCs and/or ultra-mobile PCs (UMPCs) into a sleep state when they are not in active use, since otherwise the battery would run down too quickly. It also causes significant waste of energy because mobile and desktop computers alike are often left plugged in and always-on rather than put in a sleep mode, in case there are incoming events for them to respond to.

This paper presents Somniloquy[1], an architecture that allows computers to respond to incoming network communication events in near real-time while simultaneously being in a low-power state. It operates by supplementing the computer with a secondary low-power processor that performs actions on behalf of the main processor whenever the computer is in a low power state such as S3. These actions can include connecting to and maintaining connectivity with local access routers such as wireless access points (APs), communicating with network-based services such as DNS servers, dynamic DNS servers and voice-over-IP (VoIP) proxies, and replying to incoming requests such as ARP IP address resolution requests or ICMP pings. The secondary processor is also able to signal the computer to come out of the low-power state in response to network events such as an incoming remote desktop request (use of the computer's GUI remotely) or incoming VoIP call.

Significant power savings are achieved with Somniloquy since the secondary processor requires much less power than the computer as a whole, with our prototype system exhibiting a ten-fold increase in battery lifetime. At the same time, the system responds to network events in near real-time, in our prototype a latency of only 4-7s was added to respond to incoming application requests such as VoIP calls and remote file transfers. A particular advantage of our architecture is that the computer's sleep status can be made transparent to both the local network and to remote servers, and even transparent to applications running on the computer.

Somniloquy therefore allows mobile users to leverage S3 for good battery lifetime without sacrificing reachability from the network. It also allows stationary computers such as laptops which are mains powered, desktops or even servers to dra-

[1]somniloquy: the act or habit of talking in one's sleep.

matically reduce their energy consumption while idle, again maintaining reachability. Furthermore, Somniloquy can be added to existing computers easily, as it can be implemented entirely as a USB peripheral and associated software drivers.

We make the following contributions.

- We present Somniloquy, a system that enables connectivity to computers even when they are asleep. In contrast to prior automatic wake-up systems (see Section 3), Somniloquy is transparent. It does not require changes to either the applications or the infrastructure, and works with mobile/nomadic devices.
- We present a practical way in which Somniloquy can be realized today, using a USB based low power device (gumstix) that consumes little power. We show how a number of applications, such as file sharing, remote desktop, VoIP and push e-mail can be implemented without requiring any changes to the application or the infrastructure. Furthermore, our USB-based prototype is easily applied to existing computers, requiring just an external peripheral and software drivers.
- We evaluate our system in a real setting, and show that energy consumption is reduced ten-fold (and hence battery lifetime is increased ten-fold) while incurring an extra latency of just 4 to 7 seconds for an application-layer operation such as retrieving a file on the sleeping device.

2. SCENARIOS

Before we continue, we present three simple scenarios that demonstrate the need for Somniloquy.

- The user is mobile, e.g. in a cafe or airport, and is waiting for an important voice call using VoIP on their laptop, but their battery is running slightly low, and having it wait in an active on state is wasting battery that could be used during the call itself.
- The user responds to some emails using their laptop on the train home, where there is no free wireless access. When they arrive home they are greeted by their spouse and young children, and would rather not have to wake the laptop up just to have it send the emails in the outbox.
- The user occasionally works from home using a personal notebook, and when doing so prefers to connect to a computer (desktop or laptop) in their office for remote desktop use and for access to files, since that machine has greater resources, work-related software and a backed-up and large disk. However, they are conscious of the monetary and environmental costs of leaving their work computer on 365 days a year just in case they were to want remote access.

For the first two scenarios, the user's laptop could benefit from Somniloquy for application reachability during sleep and for the ability to wake up when a suitable AP comes into range, send the emails, and go back to sleep. For the third scenario, if the work computer were to run Somniloquy, the mobile user could access it remotely at any time without incurring the energy cost of continuously keeping that computer on.

3. BACKGROUND

Computers employ various techniques to reduce power consumption. For example, the hard disk can be spun down, the display can be dimmed or shut off completely, system memory can be put into lower power states and the main application processor can be operated at a reduced clock frequency and supply voltage. Another way of reducing system power is to operate the network interfaces in low-power modes where possible. A common example is that of 802.11's power save mode (PSM) [12, 13]. When using PSM the 802.11 radio is duty-cycled to save power, periodically waking up to check for any packets destined for it which have been buffered at the 802.11 access point (AP). By varying the period of wake-ups, the radio can save power with an associated increase in packet delivery latency. Various modifications to the parameters of PSM have been proposed, such as a bounded delay addition which maintains the level of power savings of PSM while reducing latency [14]. PSM reduces the power consumption of the radio interface from typically almost 1W (without PSM) to around 250mW [1, 2, 21]. However, even with these measures in place, modern laptops still draw a minimum of around 10-20W if they are to remain responsive to incoming network traffic. We refer to this minimum as the 'base power.' The only way to reduce this power consumption further is to switch the computer off completely, or put it into a so-called 'sleep' mode such as the suspend to RAM state (S3) where the system state is maintained in RAM but most of the other system components are powered off. The advantage of maintaining system state in this way is that the laptop can be resumed to an active, usable state much more quickly than would be possible otherwise, typically within a few seconds. We present more detailed figures on laptop power consumption in various modes of operation including S3 in Section 7.

Of course, when a device is in the S3 state, its utility is very limited. It can respond to local user initiated 'resume' events such as pressing a key or opening the laptop lid. The universal serial bus (USB) also allows certain peripherals to generate resume events, although it mandates that the peripheral operates in a suitably low-power mode whilst the host is in S3 to prevent excessive system power drain. In this way, it is possible for a key press on an external USB keyboard to cause a laptop in S3 to resume, for example.

To improve the responsiveness of a laptop in S3, Wake-on-LAN [16] (WoLAN) and its wireless equivalent wake-on-WLAN (WoWLAN) provide another mechanism to wake up a device based on certain packets received over a network interface. In both cases power to the respective network interface is maintained when the main processor is asleep. However, both schemes have some major limitations which have to date limited their adoption. Since they operate at the link layer, the network interface may no longer have a valid IP address when the host is powered down, and is thus unable to respond to IP traffic. Instead, the interface only responds to suitably addressed link layer requests, i.e. those with the matching MAC address of the particular network interface. These include broadcast packets such as ARP requests, which are typically very common and result in very frequent wakeups. Since this is impractical, an alternative scheme known as the 'magic packet' is also supported. In this case, a specially-formed packet containing the MAC address of the network interface of the target machine is transmitted as a subnet directed broadcast and

only this magic packet triggers wake-up. However, the router support required to allow magic packets to be sent from hosts outside the local subnet is not widely enabled thus limiting the scope of this scheme. Special management stations acting as proxies can be added to the networking infrastructure to send magic packets, but these are unable to maintain application transparency or support mobility in the case of a device moving between different subnets or networks. Most importantly, there is only very coarse-grained control of wake-up in both WoLAN and WoWLAN. There is no mechanism for devices to specify under which conditions they should be woken up, even if different machines are interested in wake-up events from different sources or relating to different applications.

4. RELATED WORK

Several projects have investigated methods to reduce the power consumption of mobile devices to increase their battery lifetime. Most of these techniques attempt to reduce the power consumption of the various subsystems of a mobile device, such as the processor [8], communication [4, 15, 21, 22], the memory subsystem [10] or the storage device [18]. 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 [27, 7]. All of these approaches are, however, unable to reduce the "base power" of a mobile device such as a laptop, which as stated earlier is still significant [17, 28].

Various projects have proposed leveraging heterogeneous wireless network interfaces with different power consumption characteristics to save power [5, 21, 23, 26]. Cell2Notify [1], Wake-on-Wireless [26] and On-Demand-Paging [2] propose the use of separate low power radios to serve as a separate out of band wake-up channel for a higher power WiFi radio. However, the use of heterogeneous radios requires either additions to the infrastructure [2, 26] or modifications to intermediate proxies [1]. CoolSpots [21] and Context-for-Wireless [23] aim to reduce energy consumption of mobile devices by dynamically choosing between multiple available wireless interfaces for data transfer based on various factors. However, these projects are all based on the assumption that the wireless interface is the major power consumer, which is valid for smartphone class devices but not so applicable to laptops.

Wake-on-WLAN [19] is a research project based on the scenario of rural connectivity. In Wake-on-WLAN, wake-up signals propagate down a mesh network to establish network connectivity on-demand, building up the network path one hop at a time until end-to-end connectivity is achieved. Wake-on-WLAN is therefore specifically geared towards reducing the power consumption of custom built intermediate routers, rather than end mobile devices such as laptops.

The One Laptop Per Child project (OLPC) are working on including mesh networking as feature of their XO laptop [20]. This would allow a sleeping OLPC to act as a router in a mesh network. There are similarities in the approach using an augmented network interface, though their work is targeted at retaining network coverage despite a lack of infrastructure, while our work targets existing network access points and retaining application responsiveness.

Turducken [28] and Triage [6] take a different approach towards reducing the power consumption of mobile devices by proposing a multi-tiered architecture with each heterogeneous tier having a different power consumption profile. Triage is geared more towards energy efficient micro-servers in a sensor network scenario, while Turducken [28] considers mobile devices such as laptops. For applications such as NTP, IMAP email synchronisation and maintaining web caches, which can all be handled by periodic polling for updates, Turducken proposes executing the applications themselves on the lower power consuming tier, while keeping higher tiers in a lower power state. In this way, a tier can act as a proxy for tiers above it. In order to save energy the lower power tier is then further duty cycled to trade-off consistency of data with power consumption. Somniloquy uses a two-tiered model similar in spirit to Turducken, however we aim to support a different class of applications for which the laptop needs to present an 'always reachable' abstraction. In contrast to Turducken, while the laptop is in S3 our secondary processor is always on and connected to WiFi whenever it is available. It can therefore respond to the appropriate network events by waking up the laptop immediately, maintaining transparent reachability. Applications that are inherently asynchronous, such as remote access (RDP, SSH), incoming VoIP calls and push email, cannot be handled by a periodic wake-up and polling scheme can thus be supported by Somniloquy. Furthermore, the primary processor (i.e. the laptop) and the secondary processor share the same MAC and IP addresses and hostname, presenting a single device abstraction to the network. This is unlike Turducken where the multiple tiers act in concert but do not use a single-device abstraction.

Very recently, Allman et al. [3] have proposed the notion of 'selective connectivity' for hosts on the internet. The authors propose that such hosts may be put into a low power state to save energy, while generic software or hardware 'assistants' can take over some network functionality, although they do not mention any specific details on how these would be implemented. The paper also talks about application primitives that might need to be in place to support this new model of connectivity, targeting mainly target desktop computers rather than wireless mobile devices. We present the challenges we faced and the design decisions that we made in building and evaluating our architecture, which supports some of the ideas raised by the authors.

5. THE SOMNILOQUY ARCHITECTURE

In this architecture description, we focus on Somniloquy's applicability to mobile devices such as laptops, since that is the nature of our prototype. However, our architecture is also applicable to stationary PCs (desktops, servers), and we will revisit this possibility in Section 8.

The primary aims of Somniloquy are:

- To enable a laptop in a sleep state (such as S3) to continue to be responsive to network events.
- To do this transparently to network routers and servers, and require no new hardware/software in the network.

We accomplish this by adding hardware to the laptop, comprising a low-power secondary processor attached to a network interface. The overall concept, illustrated in Figure 1, is that the secondary processor remains active when the laptop is in

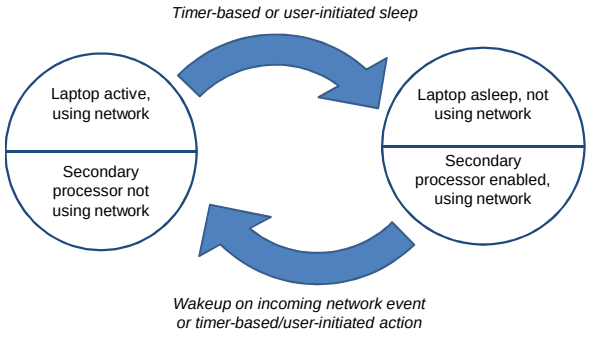


Figure 1: High-level operation of Somniloquy. The secondary processor actively uses the network when the laptop is asleep. At any time, only one of the laptop or the secondary processor is actively using the network.

S3, and performs networking operations on its behalf (i.e. using the same MAC address, IP address, etc as the laptop). The secondary processor hands off control to the laptop when the laptop wakes up from sleep state. We describe the design of Somniloquy in detail in the rest of this section.

5.1 Somniloquy hardware

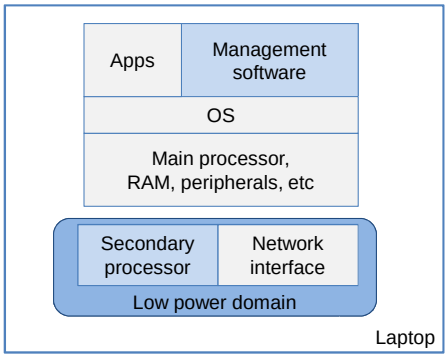
There are two main ways to implement Somniloquy: integrated into a laptop as shown in Figure 2(a), or as a separate peripheral as in Figure 2(b). The advantages of an integrated solution are reduced cost and a more compact form factor, which avoids the duplication of network interface hardware. Potentially this would also result in a more highly optimized system. The advantage of a peripheral solution is that it can be retro-fitted to existing systems and is also easier to prototype and experiment with. This is facilitated by the fact that many existing laptops already have support for powering peripheral devices while in S3, e.g. via USB.

Note that in Figure 2(b) the laptop potentially has an internal network interface as well as the one available through the Somniloquy peripheral — and by this we mean specifically an interface of the same technology (e.g. 802.11a/b/g). There are two ways to handle this. The laptop's built-in interface can be permanently disabled so that the peripheral interface is used when it is on and by the secondary processor when the laptop is in S3. Alternatively, the laptop can use its internal network interface and the secondary processor can use its separate network interface. However, in order to maintain our aim of transparency to the network, the two interfaces must share the same MAC address (using MAC address spoofing). This creates a few extra demands on the software in order to transfer state between the two network interfaces, and to make sure that they are never active simultaneously.

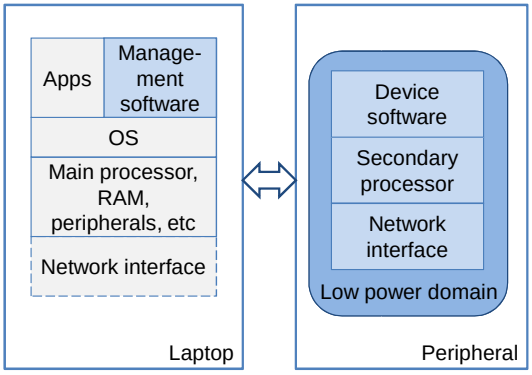
The rest of this section will discuss the hardware configuration where Somniloquy is implemented as a separate peripheral, but where the laptop has a network interface using the same technology as that peripheral that it uses when it is awake. Architecturally, we believe this is the most complex case.

5.2 Operation of Somniloquy

We describe the operation of Somniloquy by walking through the timeline of a laptop going into suspend, whereupon the



(a) Integrated approach



(b) Legacy approach

Figure 2: Somniloquy hardware for (a) future laptops that come with a secondary processor and network interface, and (b) existing laptops where the secondary processor and network interface can be attached via a peripheral connection, for example, via USB

secondary processor performs certain network-related operations on its behalf and at some point most likely wakes up the laptop again. Figure 3 shows some of the interactions described in this section. The dashed lines show features that have not been completely implemented in our first prototype of Somniloquy .

5.2.1 Going to sleep

Before going to sleep, the laptop sends information to the secondary processor on the wireless networks which it can associate to, a list that the operating system typically keeps to perform "wireless zeroconfig" association. This includes authentication credentials where appropriate. When going to sleep, it also transfers the *current* networking state to the secondary processor, i.e. the AP which is currently being used and the DHCP lease details if appropriate (including the currently assigned IP address). The transfer of this data is triggered by the event of the laptop going to sleep.

The secondary processor detects when the laptop has entered sleep by sensing a USB suspend. It then activates its network interface, and configures it with the current network configuration (including the MAC address) provided by the laptop. Ideally the secondary processor should silently move

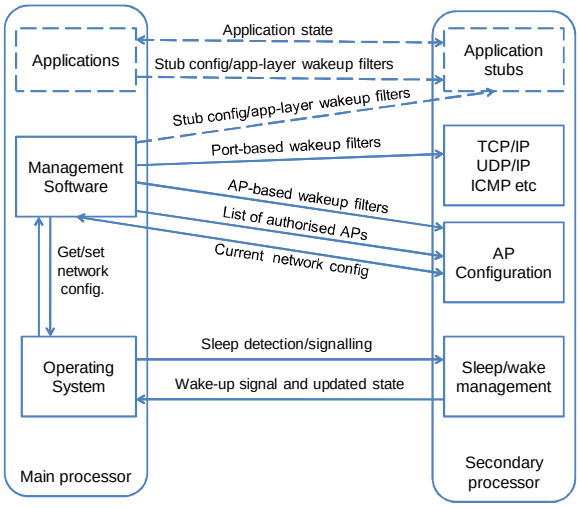


Figure 3: Somniloquy software components on the laptop and the secondary processor, and their interactions. The dotted lines represent components that have not been completely implemented, but are nice to have.

into an associated state with the AP and start using the IP address that the laptop was previously using, in actual fact it is acceptable for the secondary processor to reassociate as normal with the AP and make a new DHCP request, since both association and DHCP protocols are resilient to re-execution, and the secondary processor will reacquire the same network details. To be specific, we use a "DHCP renew" packet, in which the secondary processor specifies the preferred IP address (which was previously in use). Once complete, it appears to the network and to remote servers that the same system is still connected; any holes in firewalls will still be open (as long as the same AP is used — see Section 5.2.4 for further discussion).

5.2.2 *While asleep*

The secondary processor uses filters on the contents of incoming packets to determine when to wake the laptop. These filters can happen at the link layer, network layer, transport layer or application layer, with progressively higher complexities and higher capabilities. As application layer filters are more complex, they are described separately in Section 5.2.4.

At the **link layer**, the secondary processor can be told to wake the laptop when it detects a particular AP or set of APs. This is useful in situations where a laptop may have network operations to perform but no network connectivity, as it can then go to sleep and instruct Somniloquy to wake it when there is connectivity. By default, the detection of a suitable AP causes the secondary processor to associate, and this in turn enables higher layer functionality as discussed below.

At the **network layer**, the secondary processor can use a Dynamic DNS server such as the publicly available DynDNS (www.dyndns.org) to associate its current IP address with a long-lived hostname. Conceptually, this means that whilst asleep the laptop can still be reachable to incoming traffic no matter where it is in the network. This is true even if the laptop is mobile as long as it is within range of a suitable AP. In this way it is theoretically possible to create a "ping of life" (in contrast to the "ping of death[2]"), a special ICMP packet to mimic the magic packet of WoWLAN, but operating at an Internet scale through standard routers rather than limited to a single subnet. Although this would work in some situations, it may be that the AP or router behind the AP either (a) uses IP masquerading to share a single IP address, or (b) has a firewall preventing incoming traffic. In this case, application-layer techniques discussed in Section 5.2.4 must be used.

At the **transport layer**, incoming traffic directed at specific TCP and UDP ports can be detected by the secondary processor and can then generate a wakeup of the main processor. This simple approach has the advantage that application-specific code does not need to be executed on the secondary processor. Filtering only for relevant ports has the obvious side effect that traffic on other ports will be rejected. Somniloquy also provides the ability to further restrict the set of ports/applications which can cause wake-up, or to cause wake-up only when traffic from a certain subset of IP addresses is received.

5.2.3 *Waking up*

The secondary processor must support a mechanism to trigger the laptop to wake up. This could be accomplished using a hardware relay (i.e. mimicking a power button being pressed), or by using the wake-up functionality built into many buses such as USB. Of course, the laptop can also be woken by other causes such as a user pressing a resume button, or by an internal timer. It is important that the secondary processor detects this so that it stops performing operations on the laptop's behalf (since there may be duplicate identity issues e.g. with duplicate MAC addresses otherwise). In the case of a USB connected secondary processor, this can be done by detecting USB coming out of suspend.

When a laptop wakes from suspend, it typically resets its hardware and device drivers, and then performs wireless network association from scratch. While this is acceptable, one optimization that Somniloquy enables is the transfer of association details (channel, SSID, security credentials, DHCP lease details including IP address, etc) from the secondary processor to the main processor. This expedites the process of rejoining the network. Further optimizations are possible if the main and secondary processors share a common network interface (as in Figure 2(b), since the physical hardware of the card is already in the correct mode. However, these optimizations will likely require modifications to the operating system.

Although the laptop has been woken up and may have reassociated quickly, it has probably missed incoming application-layer packets, in particular the packets which triggered the wake-up. We can handle this in two ways. Firstly, many protocols designed for Internet use are designed to cope with loss and automatically retransmit packets. 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 laptop even if resume from sleep takes several minutes. Of course, the longer the laptop takes to resume, the larger the subsequent delay before another retransmission will be. The second way of handling this is for the secondary processor to proactively inject the packets which it received on behalf of the laptop (particu-

[2] `http://insecure.org/sploits/ping-o-death.html`

larly the one(s) which triggered wakeup) into the laptop's receive queue. Since the MAC and IP addresses are the same, the injected packet will be processed correctly. This may be simpler in incarnations of Somniloquy where the network interface hardware is shared between the two devices.

5.2.4 Application layer stubs

We now discuss operations by which the secondary processor can perform application-layer tasks on behalf of the laptop. Note that this functionality is not completely implemented in our prototype, and the ensuing discussions are included for completeness.

At the application layer, the secondary processor can run "stub" code which performs a subset of the functionality of specific applications on the laptop. This is a more powerful technique than simply using port-level triggers for a number of reasons. First, many application protocols such as those for IM and email may require periodic activity on the client side in order to maintain the connection, e.g. sending "heartbeat" packets allowing the presence information in IM to remain current, or sending periodic polls for new emails in the POP protocol. Second, in case of mobility when the laptop moves to a new AP, it may obtain a new IP address and would in that case have to re-log-in to application servers. Even if a given application server supported a "push" protocol, client-initiated connections may be necessary to ensure that holes are made in any firewalls as described previously. Third, the use of application code means that event filters can specify events at this layer, e.g. wake on an incoming VoIP calls only from my friends outside work hours and not my work colleagues, or wake only for software patch updates if the laptop is plugged in/the battery level is not low.

To enable application layer functionality, appropriate stubs for each of the desired applications may be loaded onto the secondary processor before the laptop goes to sleep. Furthermore, configuration details such as login credentials have to be transferred, and also the current state of that application on going to sleep (so the secondary processor can seamlessly pick up on that state). This state must also include transport-layer state such as TCP sequence/acknowledgement numbers.

One could imagine that suitably augmented application clients on the laptop would be Somniloquy-aware and support these operations, including the provision of stub code and its configuration. However, application layer functionality could also be achieved without modified client-side applications, if a Somniloquy configuration application were used to specify these details for transfer to the stubs.

On waking up, there is the potential for network/transport layer and application layer state on the laptop to be out of date due to operations performed by the secondary processor on its behalf. For example, if packets in a TCP stream were generated by the secondary processor, then the sequence and acknowledgement numbers would have moved on and when the main processor tried to continue using that stream there would be errors leading to the reset of the stream. This can be addressed in three ways. First, the secondary processor can avoid *sending* any packets in a particular stream, or more generally only send packets using protocols/streams which do not require an updated client-side state (e.g. using UDP to send heartbeats if possible). This means that no state inconsistency will occur on wakeup. Second, the secondary processor can transfer the updated state at the transport layer and/or application layer to the laptop on wakeup — this requires modification to the networking stack and/or to applications in order to accept this state update, but doing this widens the set of possible actions that Somniloquy can perform while asleep without causing errors on wakeup. The third option is to rely on the built-in error resilience of particular protocol(s) and not transfer the state, causing error conditions but recovering from them. An example would be if an incoming email from the user's spouse caused wakeup — to determine who the email was from might have updated the state of the connection to the server causing the laptop to drop that connection, but the connection would be restarted by the email application and the email would be received anyway.

5.3 Configuring Somniloquy

Control should remain with the user as to when Somniloquy is active, and we achieve this by using the current dialogs that specify how a laptop goes to sleep (e.g. whether the lid closing means do nothing, suspend, or hibernate) and adding Somniloquy as another option.

If Somniloquy is active, as we have stated, the laptop sends to the secondary processor a list of events on which to wake, including events at the link, network, transport or application layers. One question that remains is how the laptop knows what events are worth waking up for, i.e. what the user's preferences are. We accomplish this by detecting the network ports that applications have open, and then allowing the user to configure (periodically, not every time Somniloquy is activated) which applications should be allowed to wake the device. This configuration is similar in nature to current operating systems' firewall configuration in which the applications allowed to access the network are configured. In cases that applications are aware of Somniloquy, they can request wakeup functionality be enabled for them via similar dialog boxes as for enabling access through the firewall.

5.4 Mobility and naming

An important goal of Somniloquy is to enable reachability in the event of mobility. A laptop in sleep state should be reachable even when it handoffs from one AP to another, obtains a new IP address, or moves across domains, e.g. when a laptop in S3 moves from the user's workplace to their home.

Somniloquy increases reachability by (i) maximizing the amount of time the secondary processor is connected (by opportunistically connecting to WiFi networks), (ii) implementing naming services on the secondary processor (by responding to naming service requests, such as DNS, WINS, ARPs), (iii) supporting the use of application-layer stubs to maintain application-server connectivity despite mobility, and (iv) supporting a globally reachable name for the laptop wherever possible using DynDNS. To support DynDNS, Somniloquy requires client software to run on both the laptop and the secondary processor. When the IP address changes, the client software communicates with a centralized server hosted by www.dyndns.org. The machine is then accessible over the Internet using the name: $< machinename >$.dyndns.org. We note that the machine might still not be reachable if it is behind a NAT. Rather than taking a fresh approach to solving this problem, we plan to borrow ideas from the well-studied literature on NAT traversal, such as STUN [24], TURN [25], Teredo [11], and others.

5.5 Security

We now present a short analysis of the security threats raised by our architecture. While this is by no means rigorous or complete, it does give a high-level indication as to the issues raised.

The functionality that Somniloquy provides, of waking the laptop based on network events, can form the basis of a denial-of-energy attack. If an attacker were to learn how to trigger wakeup events, a Somniloquy-enabled device could be repeatedly woken by an attacker, wasting its energy in contrast to a device which was asleep and not wakeable. We combat these using two possible responses. One is to implement secure protocols at the various layers on Somniloquy and only wake on events that are received using such secure protocols, e.g. not waking when an AP is seen but only when it is securely associated with using (e.g.) WPA, and not waking when an incoming packet over TCP is received but only after an initial secure handshake such as SSL has been conducted so that the remote host is known to be legitimate. The problem with this solution is that the complexity of the secondary processor's tasks rises significantly, as does the amount of state that needs to be transferred to and from the secondary processor when going to sleep or resuming. This increases the materials and deployment cost, and will likely reduce the energy benefits. A less secure but much simpler solution is to detect false wake events and either inform the user then or automatically modify the filters to omit those events and inform the user offline. While the functionality of Somniloquy is impaired, the user can then take out-of-band action to determine the cause of the attack (which may also be accidental rather than malicious) and act upon it.

The complexity of having network-facing code running in both the main and secondary processors does increase the risk of bugs causing security holes. A security hole on the secondary processor causing it to reveal data kept on it has the potential to reveal network access or application credentials that have been transferred to it. A security hole allowing "root" access and reprogramming of the secondary processor has even more severe implications, for example, that device can be used to launch subsequent attacks, it can be used to electronically spy in locations which it is carried to by the unaware user, or it could use that processor's privileged communications path to the main processor to attack the rest of the device. For that reason, Somniloquy limits the code that runs on the secondary processor to programs that monitor, and "at most" wake up the laptop with very limited state transfer.

Despite the denial-of-energy issue and the added complexity of two devices to secure, Somniloquy does not in some sense intrinsically bring any new security problems, in that the secondary processor is simply performing a set of actions that the main processor would perform anyway if it was awake. As such it is not expected that deployment of Somniloquy will cause large security concerns in practice.

6. PROTOTYPE IMPLEMENTATION

We have implemented a prototype based on the Somniloquy architecture using a standard Lenovo X60 and a low power peripheral platform called the *gumstix* manufactured by gumstix Inc[3]. Both processors use an 802.11 b/g (WiFi) network interface. We now present the details of that implementation with reference to the more general architecture presented in Section 5.

[3]`http://www.gumstix.com`

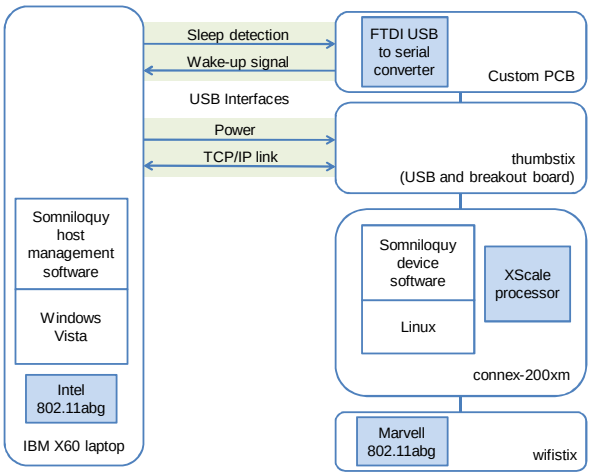


Figure 4: Block diagram of the Somniloquy prototype system. The figure shows various components of the gumstix and the USB interfaces to the host laptop.

6.1 Hardware

A block diagram of our prototype is shown in Figure 4 and a photograph of the laptop with the gumstix attached is shown in Figure 5.

The Lenovo X60 has an internal Intel 3945 802.11 a/b/g module and runs Windows Vista. It supports several sleep states, including S3 and S4 (suspend to disk), but we only use S3 since the resume time is much faster than S4. We explicitly configured Vista not to use a further sleep state "hybrid sleep" (S5) in which the RAM is written to disk but the computer is left in suspend (S3) mode (though able to turn off and resume from disk if power is low) since this also causes suspend latency to increase significantly.

We implemented the Somniloquy model presented in Figure 2(b), i.e. we use an external device with its own network interface (the gumstix) as the low power secondary processor but we switch to the X60's internal 3945 WiFi module when the main processor is operational. This allows us to simplify the implementation of the gumstix since it does not have to act as a WiFi interface when the computer is on, in other words it is only ever used by the secondary processor.

We chose a gumstix-based solution due to the wide range of peripherals supported, the convenient form factor and low power consumption. For our prototype we use a processor board (connex-200xm), a WiFi module (wifistix) and a combined USB interface/breakout board (thumbstix). The processor on the connex-200xm is a low power 200MHz PXA255 XScale processor with 16MB of non-volatile flash and 64MB of RAM. The wifistix incorporates a Marvell 88W8385 chip which is optimised for low power operation; previous measurements have shown that the Marvell chip consumes around half the power as some other WiFi chipsets when using WiFi's Power Save Mode (PSM) (100mW versus 250mW) [21]. The thumbstix provides a USB connector, serial connections and general purpose input and output (GPIO) connections from the XScale.

To this, we added a custom designed PCB incorporating a single chip, the FT232RL from FTDI, which provides USB-to-serial functionality. This is attached to the laptop 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. The latter are connected to the FT232RL's ring indicator input and sleep output and provide a means of waking the computer and a means of detecting when the computer is in S3 respectively.

The gumstix platform executes an embedded distribution of Linux, supporting a full TCP/IP stack, DHCP, configurable routing tables, wireless configuration tools, a configurable firewall, SSH, serial port communication, etc. This allows us to implement the various functionality needed to support Somniloquy.

6.2 USB connection

As mentioned above (and shown in Figure 4), the laptop is connected to the secondary processor via two USB connections. One of these provides power and two-way communications between the two processors, whilst the second provides sleep and wake-up signaling, and a serial port for debugging. We did this for ease of prototyping, but in a more refined implementation a single USB port could be used.

Modern laptops continue to power their USB ports in S3 mode. Although a USB peripheral is allowed to draw up to 2.5W from its host when enumerated (500mA at 5V), the USB specification technically requires peripherals to draw no more than 2.5mW when the bus is suspended. This is not enough power to operate the gumstix. However, in practice it turns out that many USB peripherals actually consume significantly more than this, and doing so does not cause any issues for the USB hosts we tested (three different laptops and one desktop machine).

The USB connection between the gumstix and the laptop is configured to appear as a point-to-point network interface. We use statically assigned IP addresses and normal TCP/IP sockets based communication over this link. The ring indicator signals to the laptop that it should resume from S3, while the sleep signal is active whenever the laptop is in S3 mode. Additionally, the serial port connected using the FTDI interface is used as a debug console.

6.3 Software on the laptop

We have added some software to the laptop to transfer state to the gumstix before the laptop goes to sleep. If the computer is currently associated, this transfers the current SSID of the associated wireless network, the IP address and gateway/DNS IP addresses, and whether these details were provided by a DHCP server or were statically configured. We also send a list of other wireless networks which the gumstix is authorised to attach to. This is achieved using a daemon on the gumstix listening on a specified port on the point-to-point connection with the laptop.

In addition to network details, the laptop provides the gumstix with a set of filters to determine when the laptop should be awoken. These are discussed further in section 6.5.

We initiate sleep on the laptop using our own script placed as a shortcut on the desktop. This ensures that the state on the gumstix is up-to-date before putting the laptop into S3 mode. We could have performed this transfer automatically by hooking into the operating system to trap the suspend event, but using an explicit action facilitated comparative testing with normal suspend.

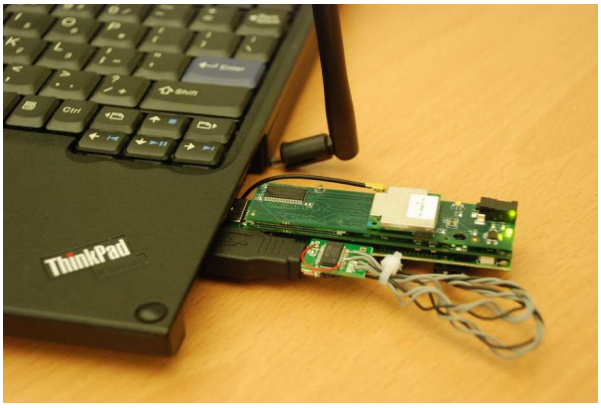


Figure 5: Photo of Somniloquy prototype. The gumstix can be seen connected to one USB port while the custom PCB with the FTDI chip is connected to the other USB port on the laptop.

6.4 Gumstix activity when the laptop enters S3

On the gumstix, the sleep signal from the laptop is used to turn on the Marvell WiFi interface when the laptop goes into S3 and turn it off again as soon as the laptop starts to resume. Since we configured the laptop and gumstix wireless interfaces to have the same MAC address, this was done to ensure that the two were never active at the same time as various protocols assume that devices attached to a network have unique MAC addresses. When active, the Marvell interface is put into PSM to maximise battery lifetime.

Once S3 has been detected and the Marvell interface enabled, the gumstix proceeds to configure its wireless interface with the parameters received from the laptop (e.g. SSID, DHCP or Static IP, and in the latter case what gateway and DNS servers to use). If the gumstix uses DHCP, the DHCP server is requested to re-lease the same IP address that was previously allocated by using a 'preferred address' field in the DHCP protocol. This was always successful during the operation of the prototype, which is not suprising since the MAC address does not change.

The gumstix also executes a dynamic DNS client (inadyn[4]) to update a central server with its domain name-to-IP address mapping. In case of mobility within a single WiFi network the gumstix can also seamlessly perform MAC layer (layer 2) hand off between various APs to maintain connectivity.

In this state, the gumstix is responsive to AP beacons, ARP packets and ICMP pings, and it can receive incoming TCP/IP packets. In short, the gumstix has transparently taken the place of the laptop's internal WiFi connection with respect to network reachability.

6.5 Triggering wakeup

In our current prototype we have not yet implemented application stubs on the gumstix (see Figure 3). We have instead implemented a simple and flexible packet filter using the BSD

[4] `http://inadyn.ina-tech.net/`

raw socket interface. Our packet handler applies regular expression (regex) filters to the packets received (from the IP layer and higher). When the packet handler detects a regex match on any field or combination of fields, it triggers a wake-up of the host laptop.

We use the linux 'iptables' firewall configuration utility to disable any outgoing responses to TCP requests. This stops the gumstix from sending TCP reset packets in response to a TCP connection that it knows nothing about (that the laptop had for example previously set up with a remote host). It also stops the gumstix from sending a TCP SYN-ACK packet in response to an incoming TCP SYN packet. In both cases, if those events warrant waking up the laptop, a suitable packet filter would be in place and so the laptop will wake and eventually handle the packet. It's important that outgoing responses from the gumstix are disabled since they would invalidate the state of those TCP connections (by resetting them or by sending a SYN ACK containing state that the laptop is not aware of when it wakes). In the current implementation we rely on TCP to retransmit those packets, we do not forward them from the gumstix to the laptop.

Currently, the packet filters we use simply search for incoming packets on TCP or UDP port numbers that the laptop software has specified before going to sleep. This is enough to realize a number of applications, described below. However, our prototype is not yet capable of performing application-layer keep-alives that may be required for certain applications.

Using port-based filtering, we have implemented wake-up triggers based on the incoming remote desktop requests (RDP), remote shell (SSH), file access requests (SMB), and voice over IP calls (SIP). As such, even with this simplest of implementations of Somniloquy, we can provide support for a number of applications. The advantage of such a simple implementation is the small modification required to the laptop and no modification to applications. However, we expect our prototype to be readily extensible to support more of the features described in Section 5 such as waking up on application-layer criteria, for example an IM from a particular person.

7. SYSTEM EVALUATION

The goal of the Somniloquy architecture is to allow a computer such as a laptop to continue to communicate whilst it is in S3, and to do so in a way that is transparent to the network and which does not significantly impact S3 battery lifetime. In this section we present our evaluation of the Somniloquy prototype presented in Section 6. To quantify the power consumption of Somniloquy and the impact this has on S3 battery lifetime, we start by evaluating the power consumption of a standard laptop in its various modes of operation. We then present detailed power consumption figures of the prototype Somniloquy hardware under a number of different test conditions. The power consumption and communication latency of the complete Somniloquy system is presented, and the corresponding effect on system battery lifetime is evaluated.

7.1 Base power consumption of laptops

As mentioned in Section 3, a number of techniques are frequently employed to reduce laptop power consumption. To understand the effect of these and provide a suitable baseline against which we can compare the power consumption of our proposed architecture, we evaluated three popular laptops. These machines, a Lenovo X60 tablet PC running Windows Vista (our prototype machine but without the gumstix attached), a Toshiba Portege M400 laptop running Windows XP, and a Lenovo T60 laptop running Windows Vista, were tested under a variety of conditions including the suspend to RAM state (S3). This data is presented in Table 1. Initially we used a commercially available mains power meter, *Watts-Up*[5], to collect power consumption data, having first removed the battery from the device to prevent any spurious battery charging current. Watts-Up supports a 100mW resolution for measuring average power, and a USB interface which we attached to a separate machine used for logging. However, it turns out that in S3 (and even when switched off) the power consumed by the laptop, as reported by Watts-Up, fluctuates significantly. Therefore, Watts-Up was not suitable to measure the laptop's power consumption in S3. For this reason we switched to a technique which uses the 'gas gauge' chip present in modern laptop batteries to measure the amount of energy drawn from the battery over a known period, and calculate the power consumption from this number [28]. We corroborate these numbers with the measured power consumption from Watts-Up (except when the laptop is in S3).

Condition	Lenovo X60	Toshiba M400	Lenovo T60
No power management	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

Table 1: Power consumption and battery lifetime of three laptops under various operating conditions. In all cases the processor is set to the lowest speed and is idle, the hard disk is spun down and the wireless network interface is on.

From the table it can be seen that laptop 'base power', the minimum power consumption where the device is operating and can be reached from the network, is on the order of 10-20W, resulting in a battery lifetime of around 4 to 5 hours. (Note that both of the Lenovo machines were fitted with extended life batteries.) Putting the device into S3 dramatically extends lifetime of course, to between 90 and 150 hours for the laptops we tested, and although in this state the laptop is unreachable, it can be resumed and reconnected to the network in the order of several seconds.

In order to get a feel for the typical time taken to enter S3 and subsequently to resume from S3, these times were measured (using a stopwatch) for the three evaluation laptops. Table 2 shows the mean times for each of these experiments (in each case averaged over five runs again). The suspend time was measured from the time the request was made to the laptop power LED going out, and similarly the resume time was measured between the resume event and the user interface being operational.

7.2 Gumstix power consumption

[5] `http://www.wattsupmeters.com/`

Condition	Lenovo X60	Toshiba M400	Lenovo T60
Time to enter S3	8.7s	5.5s	4.9s
Time to resume from S3	3.0s	3.6s	4.8s

Table 2: Time to resume from S3 and suspend into S3 for three laptops.

	gumstix + WiFi state	Power
1	gumstix only – no WiFi	210 mW
2	gumstix + WiFi associated (PSM)	290 mW
3	gumstix + WiFi associated (CAM)	1300 mW
4	gumstix + WiFi un-associated	1300 mW
5	gumstix + WiFi scanning	1350 mW
6	gumstix + WiFi broadcast storm	1350 mW
7	gumstix + WiFi unicast storm	1600 mW

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

To characterize the power consumption of the gumstix, we built a USB extension cable with a 100mΩ 0.1% sense resistor inserted in series with the +5V supply line and used this to connect the gumstix to the laptop. By measuring the voltage drop across the sense resistor we can calculate the power draw.

The power consumed by the gumstix platform in various states of operation are reported in Table 3. In creating the prototype, we tried to minimize power consumption, thus we chose the lowest power processor version (200MHz XScale) which has a base power of almost 210mW (WiFi interface disabled), as can be seen in row 1 of the table. Unfortunately the Linux build on the gumstix currently lacks the ability to use the dynamic voltage and frequency scaling controls supported by the XScale which we believe would lower the base power.

The Marvell 88W8285 WiFi module is a relatively low power WiFi chipset [21, 1] consuming only an additional 80mW of power (row 2) when it is associated to an AP using the 802.11 power save mode, which duty cycles the radio to save power [12, 13]. In comparison, when the WiFi interface is in the 802.11 continuous awake mode (CAM) it consumes almost 1100mW (row 3).

Unfortunately, there is no low power mode for an unassociated radio (row 4), which consumes as much power as CAM when there is no nearby AP that the gumstix can associate to. To save energy in these cases, the secondary processor can simply duty cycle the radio, periodically performing an active scan and powering down the radio in between. Since an active scan typically takes 150-200 milliseconds, with a 5 second duty cycle the average energy consumption is dominated by the 210mW drawn by the processor.

The final rows of Table 3 report the power consumption for the gumstix in case of continuous packet reception using either the broadcast address (row 6) or the unicast address (row 7) of the gumstix. The higher unicast power can be explained since broadcast packets are sent using a lower bitrate. In normal operation, we do not expect this power level to be sustained for any length of time. In any case, the total power consumption of our prototype even in the worst case of continuous network usage is under 2W, and the average use case is more likely to be around 0.3W, both of which are significantly lower than our X60's base power consumption of 11W (see Table 1).

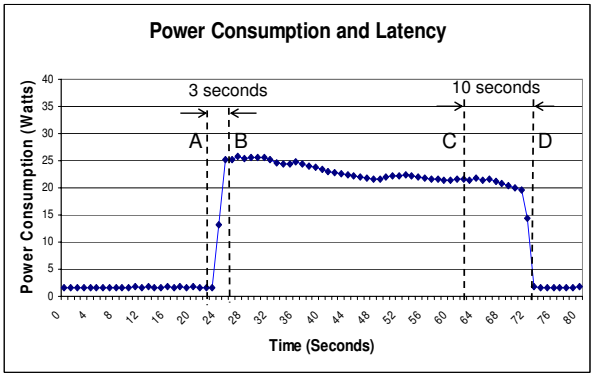


Figure 6: Power consumption of the entire system during operation. A laptop wakeup event occurs at point A and the machine is fully resumed by point B. At point C a suspend to RAM request is made, which completes at point D.

7.3 Latency and power consumption during state transition

Figure 6 shows the power consumption of our prototype system during operation. Initially the laptop is in S3 but at time=24 seconds (label A in the plot) the secondary processor wakes the laptop in response to a network event. Note that during S3 the gumstix is active and is in PSM. As the diagram shows, resuming takes around 3 seconds. Incoming events such as VOIP calls or remote file transfer requests which cause the secondary processor to trigger wakeup can be responded to by the laptop, without a latency that is too high either for humans or for TCP's retransmission timers to give up on the connection. In Figure 6 the laptop is returned to S3 at time=64 seconds (label C), and within 10 seconds the power draw has returned to the quiescent level.

The worst case delay for our system would arise if an incoming network event occurred just as the laptop had just been instructed to go into suspend mode, a process which takes 10 seconds for our prototype (and depends on the operating system and applications running at the time). In this case, the 10 second latency would be incurred, then a retransmission of the network event would have to take place and be detected by the gumstix, before a wake-up would be initiated. A further retransmission would then be required. Even in this unlikely case, the network event will eventually cause a response.

7.4 Network-layer reachability

Our test infrastructure for the following experiments consisted of a single AP which the prototype laptop was configured to communicate with (and which in turn instructed the gumstix to communicate with). For some experiments we additionally used a *test laptop* also associated with that AP.

Figure 7 shows the results of ICMP ECHO (ping) messages sent from the test laptop to the prototype laptop, which goes from active mode to S3, and later back to active. From the figure we can see the connectivity gap is minimal – 6 seconds for suspend and 2 seconds for resume, and that we have fulfilled our aim of transparent IP reachability of a device despite

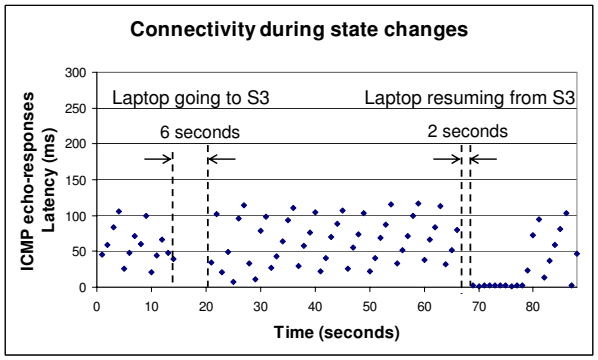


Figure 7: ICMP echo-response (ping) trace for prototype going into S3 and subsequently resuming from S3.

it being in suspend mode.

The missed pings occur between the time that the laptop WiFi interface is disabled and the gumstix interface is enabled and associated with the AP, and vice versa. As explained in Section 6, we conservatively disable the Marvell WiFi when the laptop signals that it is gone to sleep or is waking up based on the Sleep signal received from the USB interface. This signal is conservative in that it signals Sleep after the laptop has already turned off all its network interfaces, and signals resume at the very beginning of that process. (On our platform we observed the timing to be synchronized with the case LED indicator indicating S3 mode).

The latency of the pings can be explained since both the laptop and the gumstix use WiFi's PSM mode to avoid being active up more than once every 100ms and hence save power. The laptop delays around 10 seconds before going into PSM mode, while the gumstix enters PSM immediately.

We can however envisage an infrequently occurring race condition, where the laptop's WiFi card may have disassociated with the AP but the gumstix WiFi was not yet associated and an incoming network packet arrives at the AP. In this case the AP may return an ICMP error message. In this case, we currently rely on application- or human- resilience, in that an important request would be retried. With further prototyping and optimization, particularly with integrated Somniloquy functionality as shown in Figure 2, the laptop can be made to not explicitly disassociate, and the gap can be reduced by further optimization so that the system as a whole does not miss a single AP beacon.

7.5 Application-layer reachability

Table 4 shows the time taken to respond to an incoming network request for four different scenarios. In each case, the latency under 'normal operation', when the laptop is powered on and the primary processor is connected to the network, is shown by way of a baseline. The second row of the table shows the latency exhibited when Somniloquy is operating — in all cases this is more than under normal operation, because the secondary processor initiates wake-up of the main processor when the first communications attempt is made and only when the main processor is up and running, and connected to the network, can the request (which will have been re-transmitted in the mean time) be honored. The third and fourth rows show the overhead introduced by Somniloquy, both in terms of absolute delay in seconds and as a percentage of the total time taken to complete the task. In each case the latency reported is the mean of five tests.

The four scenarios which we tested were:

1. Remote desktop access (RDP): Here a stopwatch was used to time the latency between requesting a remote desktop session and that remote desktop being displayed. The secondary processor was configured to wake the main processor on detection of TCP traffic on port 3389 (the RDP port). Note that the overhead of Somniloquy in this case is much less than the table would indicate in some ways, because the ensuing desktop session may well be comparatively long-lived, such that the time taken to initiate it is not particularly significant.

2. Remote directory listing (SMB): A directory listing from the Somniloquy laptop was requested remotely (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 initiate wakeup on detection of traffic on either of the TCP ports used by SMB, namely ports 137 and 445.

3. Remote file copy (SMB): The SMB protocol was used again, but this time to transfer an 17MB file from the Somniloquy laptop to a remote machine.

4. 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. The SIP server was configured to respond to the connection request by establishing a TCP connection with the target machine in a similar way to that presented in [1], and this in turn was detected by the secondary processor in order to trigger wakeup. Once again, the latencies presented in the table were measured using a stopwatch.

To broadly summarize the results presented in Table 4, Somniloquy adds a constant overhead of around 4 seconds in most cases, which is a reasonable setup latency for most applications (since the ensuing sessions are usually much longer). This overhead is dominated by the resume time of the laptop (around 3 seconds) plus the time for TCP retransmission of the original request from the remote host. The exception to this was the VOIP application, in which it introduces a significantly longer delay. This is because the SIP client running on the resumed laptop has to re-register with the SIP sever before the incoming call (which triggered the wakeup) can connect. This takes an additional 4 seconds or so, on average.

7.6 Effect on battery lifetime

Figure 8 shows the average power consumption of the Lenovo X60 tablet used in the prototype when operating normally (i.e. no power saving mechanisms), with standard power saving mechanisms in place (the base power), when Somniloquy is operational (laptop in S3 with the gumstix secondary processor attached and connected to the network), and standard S3 (without the gumstix attached). Somniloquy adds a relatively low overhead of 300mW to S3 mode, resulting in a total power consumption which is close to just 1W. compared to the 12W of the idle laptop. This means that when the laptop needs to be attached to the network and available for remote applications

	Remote desktop access (RDP)	Remote directory listing (SMB)	Remote file copy (SMB)	VOIP call (SIP)
Normal operation	11.9s	0.9s	30.9s	3.4s
Using Somniloquy	16.1s	4.5s	35.0s	11.1s
Overhead of Somniloquy	4.2s 35%	3.6s 400%	4.1s 13%	7.7s 226%

Table 4: Mean application response latency when using Somniloquy.

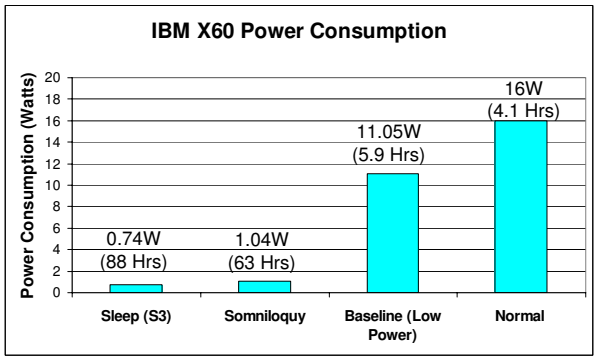


Figure 8: Power consumption and the resulting estimated battery lifetime of an Lenovo X60 using Somniloquy. The lifetime is calculated using the standard 65 Watt hour battery of the laptop.

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 a 65WH battery).

7.7 Summary of results

In summary, our prototype system has fulfilled our aims of having both transparent application-layer reachability and low power consumption. We have shown that the prototype can respond to application requests with an overhead of around 4 seconds, while exhibiting a tenfold increase in lifetime compared to an idle-but-powered on laptop, and consequently a ten-times decrease in energy requirements. On the other hand, the overhead of Somniloquy compared to standard S3 is low, with a 28% reduction of suspend-mode lifetime.

8. FUTURE DIRECTIONS

We feel that there are a number of interesting and potentially valuable directions for future extensions to the prototype system described in this paper. Some of these are discussed here.

8.1 Integrating Somniloquy into devices

Our prototype implementation is based on an external peripheral using a duplicate network interface. This is easily retro-fitted to existing devices. However, a more elegant solution is to build Somniloquy directly into the laptop, for example by modifying the built-in network interface to incorporate a secondary processor operating in a separate power domain. In fact since the network interface already includes an embedded processor for packet processing, a separate processor may not even be required. (The OLPC mesh networking effort [20] targets an integrated processor in the Marvell 88W8388 NIC in similar fashion.) This would prevent a duplication of hardware and hence reduce the bill of materials cost, would improve the form factor, and would likely reduce power consumption further.

Another improvement would involve integrating Somniloquy with the laptop operating system. Currently, when a computer resumes from sleep, it typically resets its network interface, and begins to search for available networks from an unassociated state. With Somniloquy, a network association may already exist, with authentication and DHCP complete. Therefore, the operating system need not apply a reset but instead learn the current networking parameters from the card, and hence start using networking functionality more quickly.

8.2 Hibernation

Our results show that the responsive lifetime of laptops can increase ten-fold by using S3 mode. However, S3 mode devices have an intrinsic power drain associated with maintaining system RAM. To further lengthen the responsive lifetime, let us briefly explore how Somniloquy might be used in hibernate (S4) mode.

If S4 is used, we encounter a few problems. Firstly, it is not possible (as far as we are aware) to get USB power in S4. This could be overcome by using another power tap, e.g. directly to the battery, or by tighter integration with the laptop's power subsystem. Alternatively, with a legacy peripheral, this could operate using a small external battery for the Somniloquy device; if this battery runs low, Somniloquy need only wake the laptop and place it in S3 for the time it takes to recharge the battery.

Secondly, S4 resume times are significantly longer than for S3, so we must consider whether incoming TCP connections may suffer unacceptably long delays before they are replied to. One way of stopping TCP timing-out would be for the secondary processor to reply to an incoming TCP packet with a TCP packet advertising a zero receive window. This should have the effect of stopping the remote TCP's timers; when the laptop has resumed and been passed the packet by the gumstix, it will reply properly with a non-zero receive window. We have not yet tried this idea but put it forward as an interesting possibility.

Using S4 and a further power-optimized version of Somniloquy, we can potentially achieve always-reachable lifetimes measured in weeks.

8.3 Green computing

In addition to lengthening battery lifetimes, Somniloquy's ability to reduce energy consumption while maintaining reachability has another very important application domain: that of environmentally-aware or "green" computing. Many millions of computers are habitually left powered on and active

in homes and offices. In many cases, this is deliberately done to ensure the computer remains reachable over the network, e.g. for remote file access, applying software patches, remote desktop access, etc.

Using Somniloquy, we can obtain the best of both worlds of a computer in low-power mode that is also reachable using standard application-layer protocols over the network, without special network or router support as is required for Wake-On-LAN. Since Somniloquy functionality can be added as a USB peripheral (as in our prototype), it can even be added to existing computers without internal modification.

9. CONCLUSIONS

We have presented the Somniloquy architecture, which aims to support transparent reachability for computers in a low-power state such as S3. Our prototype implementation based on a gumstix USB peripheral achieved this aim, increasing latency for various application by around 4-7s only, while decreasing the energy requirement ten-fold. This enabled a reachable lifetime of 63 hours, compared to 6 hours in idle mode and 88 hours in S3 when not able to respond to incoming events.

There is much future work possible in this area. The inclusion of application-layer stubs in the secondary processor extends the capabilities of the system but poses significant challenges in the transfer of state. While we have discussed ways of addressing these challenges we have not implemented these yet and there are likely to be unforeseen hurdles to overcome. The closer integration of Somniloquy in terms of both internal hardware and in operating system support for devices that remain active during S3 could provide significant further power savings and latency gains. However, arguably the greatest potential impact for Somniloquy lies in the area of green computing and in the use of Somniloquy to reduce the collective energy consumption of the large number of computers which are habitually left powered on but idle in case remote access is required.

10. 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] Y. Agarwal, C. Schurgers, and R. Gupta. Dynamic power management using on demand paging for networked embedded systems. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 755–759, New York, NY, USA, 2005. ACM Press.

[3] 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)*. ACM, November 2007.

[4] 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.

[5] P. Bahl, A. Adya, J. Padhye, and A. Wolman. Reconsidering Wireless Systems with Multiple Radios. *ACM CCR*, Jul 2004.

[6] N. Banerjee, J. Sorber, M. D. Corner, S. Rollins, and D. Ganesan. Triage: balancing energy and quality of service in a microserver. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 152–164, New York, NY, USA, 2007. ACM.

[7] 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.

[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] 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.

[11] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 3489, Februrary 2006.

[12] IEEE 802.11b/D3.0 Wireless LAN Medium Access Control(MAC) and Physical(PHY) Layer Specification. High Speed Physical Layer extension in the 2.4Ghz band, 1999.

[13] IEEE 802.1x-2001. IEEE standards for Local and Metropolitan Area Networks, 1999.

[14] R. Krashinsky and H. Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 119–130, New York, NY, USA, 2002. ACM Press.

[15] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. *Wireless Networks*, 6(4):263–277, 2000.

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

[17] J. Lorch. A complete picture of the energy consumption of a portable computer, 1995.

[18] Y.-H. Lu and G. de Micheli. Adaptive hard disk power management on personal computers. In *GLS '99: Proceedings of the Ninth Great Lakes Symposium on VLSI*, page 50, Washington, DC, USA, 1999. IEEE Computer Society.

[19] N. Mishra, K. Chebrolu, B. Raman, and A. Pathak. Wake-on-wlan. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 761–769, New York, NY, USA, 2006. ACM.

[20] OLPC. Mesh network details. One Laptop Per Child Project: http://wiki.laptop.org/go/Mesh_Network_Details.

[21] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 220–232, New York, NY, USA, 2006. ACM.

[22] T. Pering, V. Raghunathan, and R. Want. Exploiting radio hierarchies for power-efficient wireless device discovery and connection setup. In *VLSID '05: Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05)*, pages 774–779, Washington, DC, USA, 2005. IEEE Computer Society.

[23] A. Rahmati and L. Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 165–178, New York, NY, USA, 2007. ACM.

[24] J. Rosenberg and J. W. C. H. R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, March 2003.

[25] J. Rosenberg, R. Mahy, and C. Huitema. Traversal Using Relay NAT (TURN). (Internet draft, work in progress), March 2006. URL ftp://ds.intenic.net/internet-drafts/draft-ietf-intserv-guaranteed-svc-08.txt.

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

[27] 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.

[28] 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.