A Network Connection Proxy to Enable Hosts to Sleep and Save Energy

Miguel Jimeno and Ken Christensen
Dept. of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
{mjimeno, christen}@cse.usf.edu

Bruce Nordman
Environmental Energy Technologies Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
bnordman@lbl.gov

Abstract

Billions of dollars of electricity are being used to keep idle or unused network hosts fully powered-on only to maintain their network presence. We investigate how a Network Connectivity Proxy (NCP) could enable significant energy savings by allowing idle hosts to enter a low-power sleep state and still maintain full network presence. An NCP must handle ARP, ICMP, DHCP, and other low-level network presence tasks for a network host. An NCP must also be able to maintain TCP connections and UDP data flows and to respond to application messages. The focus of this paper is on how TCP connections can be kept alive during periods of host sleep by using a SOCKS-based approach called green SOCKS (gSOCKS) as part of an NCP. The gSOCKS includes awareness of the power state of a host. A prototype implementation of gSOCKS in a Linksys router shows that TCP connections can be preserved.

1. Introduction

One of the most urgent challenges of the 21st century is to engineer new technologies that can enable a transition towards a more sustainable society with a reduced CO_2 footprint. Network hosts consume a large and growing amount of energy. Data centers and servers consumed an estimated 1.5% of total US electricity consumption in 2006 for a total cost of about $4.5 billion [21]. However, even greater is the electricity consumed by network hosts not in data centers. The EPA estimates that PCs in the US consume 2% of all electricity consumed [10]. It is estimated that 9% of the electricity consumed by commercial buildings is from office equipment – much of this equipment being network connected PCs [22]. For a typical US household, the addition of a single 80 W PC powered-on 24/7 will add about 6.5% to the utility bill [9]. Beyond PCs are the growing numbers of commercial and consumer devices – such as set-top boxes and game consoles – that are connected to the Internet and have become network hosts.

Much of the electricity consumed by network hosts is wasted. Being connected to the Internet requires some active participation. When hosts fail to do this, they "fall off the network" and applications fail. Billions of dollars worth of electricity every year are used to keep network hosts fully powered on at all times only for the purpose of maintaining network connectivity or "presence" [18]. If not for the need for network connectivity most of these hosts could be asleep the majority of the time, with significant energy savings resulting. Surveys have found that about 60% of office desktop PCs are left on continuously [20], [27]. It is the need to maintain network connectivity that contributes to the disabling of existing power management features in many PCs. Saving this wasted energy can be done by 1) redesigning network protocols and applications, or 2) encapsulating the intelligence for maintaining network presence in an entity other than the core of the networked devices. The second option we call Network Connectivity Proxying (NCP) where an NCP is an entity that maintains full network presence for a sleeping network host [18].

The NCP concept has been previously defined by the authors [6], [7], [13], [18] and prototyped by Microsoft [1]. What has not been fully addressed is how to preserve existing TCP connections when a network host goes to sleep. Our motivation is that applications such as SSH and IM (which maintain permanent TCP connections) can be proxied. The contributions of this paper are:

- A formal definition of the requirements for a Network Connectivity Proxy (NCP).
- A design, implementation, and evaluation of a SOCKS-based approach to preserve TCP connections and UDP flows for sleeping hosts.

The remainder of this paper is organized as follows. Section 2 describes the network connectivity problem. Section 3 defines the requirements for the NCP. Section 4 describes the architecture of the NCP focusing specifically on a "green SOCKS" (gSOCKS) component. Section 5 describes the prototyping of gSOCKS. Section 6 describes the evaluation of the prototype. Section 7 describes related work. Finally, Section 8 contains a summary and describes future work.

978-1-4244-3367-4/08/$25.00 ©2008 IEEE

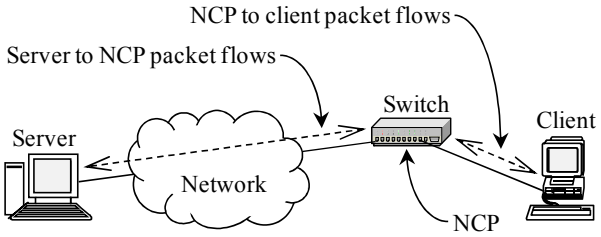
2. The Network Connectivity Problem

Why are the majority of office and home PCs left powered-on even when not in active use, such as over nights and during weekends? There are several major reasons for this with two most prominent: 1) the annoyance to a user of having to wait for a PC to wake-up out of sleep state, and 2) the need to maintain network connectivity at all times to allow for remote access and/or for network-centric applications to maintain their state. The first reason is becoming less significant as PCs become able to wake-up faster, and the second reason is becoming more significant as more applications and protocols rely on persistent Internet connectivity.

To maintain network connectivity a host must be able to support a number of application and protocol primitives including:

- Maintain host-level reachability by responding to periodic ARP requests in order to not age-out of the last-hop router ARP cache.
- Maintain its IP address by generating periodic DHCP lease requests in order to not lose its IP address (if using DHCP to obtain an IP address).
- Maintain its manageability by responding to ICMP packets, such as ping.
- Support NetBIOS name resolution by responding to NetBIOS name queries as appropriate (if running NetBIOS protocol and applications).
- Maintain application-level reachability by responding to TCP SYN packets sent to open (listening) ports.
- Maintain or preserve application state (for example, current user workspace and data) for any applications with open long-term TCP connections.
- Maintain or preserve application state by responding to any number of application-level messages including heartbeat messages and specific requests for service.

In addition, it is desirable in many cases that any packets sent to a host during the time it is sleeping not be lost. Since the above list is not complete, it is future work to better understand what packets are received by a sleeping host and which of the received packets can be ignored, which require an immediate reply, and which can be buffered for later processing. Very important is the ability to respond to ARP packets even when sleeping. Ethernet NICs that support DMTF's Alert Standard Format (ASF) Specification 2.0 [2] can respond to ARP packets for a sleeping host. This is done by keeping the NIC fully powered-up when the host is in a sleep state (with its CPU, display, memory, and storage powered-down). The powered-up NIC can also wake-up the host when specific packets pre-determined to trigger a wake-up are received.



Note: The NCP within the switch covers for the client host when it is sleeping to maintain full network presence.

Figure 1. System view of the NCP

If a host can remain reachable even when sleeping and keep its NIC powered-up, it can then be woken-up from the network by "trigger" packets that the NIC recognizes. The Magic Packet was invented for this purpose in the mid-1990s. Most NICs today support both Magic Packet and the ability to pattern match for specific packets (such as a TCP SYN packet) to trigger a wake-up. Wake-up does not solve the problem of network connectivity. Wake-up triggered on pattern matching can cause both missed and unnecessary host wake-ups resulting in applications that fail and/or reduced energy savings.

Estimating the exact cost of the network connectivity problem is difficult. There are about 170 million desktop PCs in the US (adapted from the data summarized in [23]). Assuming an 80 W power consumption of an idle PC, 120 hours of idle overnight and weekend hours per week, and 60% of PCs not powered-down during these hours results in about 50 TWh/yr of electricity wasted per year in the US (this is $5 billion at $0.10 per kWh). Further waste occurs from idle notebook PCs, set-top boxes, etc. left on. In [20] a similar calculation is made resulting in an estimate of $1.72 billion wasted per year for office PCs alone. One reported study suggests that half of all electricity consumed by PCs is wasted [17]. *Clearly, solving the network connectivity problem should be of great interest to the research community.*

3. Enabling Host Sleep with an NCP

Hosts connected to a network maintain their presence to other hosts by correctly generating and responding to messages for both network protocols and applications. A Network Connectivity Proxy (NCP) as proposed by the authors in [18] is an entity that implements the key network presence capabilities of a network host in order to allow the host to sleep, yet it allows the host to appear to other devices to be fully operational and connected to the network. Thus, an NCP can enable a network host to sleep and save energy where without the NCP it could not sleep even if inactive or idle.

Figure 1 shows a network with two network hosts labeled as a client and server and an NCP located within a switch to support the client host. The NCP could also be located in another host on the network, within a wireless access point, or within the NIC in the sleeping host.

The following assumptions must hold for the network host covered by an NCP:

- Have a sleep mode that can be entered and exited on application and/or operating system command.
- Be able to fully exit sleep mode in the time span of a few seconds or (preferably) less.
- Have a sleep mode that preserves all local protocol and application state.
- Support a remote packet-based wake-up method such as Magic Packet and/or pattern matching (if the NCP is in a remote location; not in the NIC).
- Support the ability of applications in hosts to be able to block the host from entering a sleep state if and when the application is actively using CPU, network, or other resources.

The following are the system-wide goals that a system with a client host covered by an NCP should achieve:

1. A host must be allowed to sleep and not lose its network presence.
   a. A host must maintain its IP address and be reachable by edge routers and switches.
   b. Valid incoming requests for TCP connections to a host must be honored.
   c. Existing TCP connections to a host must not be dropped and data must not be lost.
   d. UDP packets sent to a host must not be lost.
2. During the time a client host is sleeping, remote state (such as application state in a server host) must be maintained in all cases.
   a. Application keep-alive messages and/or other application messages must be responded to as required by the application.
3. Changes to network applications and protocols must not be required in the client or server host.

3.1 Requirements for the NCP

The NCP has specific assumptions and requirements as follows. The assumptions expected to hold for the NCP are:

- Is always fully powered-on and connected to the network.
- Is within the same MAC-level domain (and thus same IP subnet as well) as the host it is covering.
- Has equivalent security measures in place as the host for which it is covering.

The requirements for the NCP are organized into four categories: IP connectivity, TCP connections, UDP data flows, and network applications and higher-layer protocols. The minimum requirements for supporting IP connectivity are:

1. Have a much smaller incremental power consumption than a network host (for example, 10x less power use).
2. Know the power state – minimally, off, sleep or on (awake) – of the network host.
3. Use the IP address(es) of the host it is covering for.
4. Be able to support ARP, DHCP, and ICMP protocols for a sleeping host to maintain its host reachability, address, and manageability.
5. Be able to operate behind a typical NAT service.

Additional requirements for supporting TCP connections are:

6. Be able to listen for valid TCP connection requests and other requests coming from other network hosts to a sleeping host.
   a. Be able to wake-up a host for a valid incoming TCP connection request and enable the connection to be established.
7. Be able to maintain permanent TCP connections for a sleeping host and buffer incoming data.
   a. Be able to immediately re-start TCP connection data flow to a host when it wakes up.
   b. Be able to immediately deliver buffered TCP data to a host when it wakes-up.
   c. Be able to close TCP connections when it can be determined that a host has been removed and is no longer present.
8. Be able to wake-up a sleeping network host when NCP buffers are nearly full to prevent buffers from filling-up and potentially losing packets or blocking the server from sending data.

Additional requirements for supporting UDP data flows are:

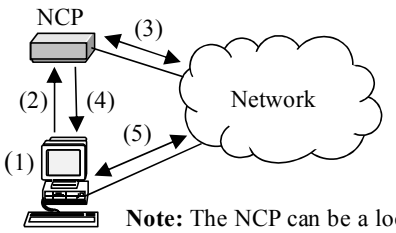
9. Be able to buffer incoming UDP packets for sleeping hosts.
   a. Be able to immediately deliver buffered UDP packets to a host when it wakes-up.

Additional requirements for supporting network applications and higher-layer protocols are:

10. Be able to keep network applications executing as if the host was not sleeping
    a. Be able to respond to routine application messages as required by the application.
    b. Be able to generate routine messages as required by the application.

4. Architecture of the NCP

In this paper we specifically address NCP requirements (1), (2), (6), and (7) (that is, power state



Note: The NCP can be a located external to, or internal within, the host. If within the host (e.g., in the NIC) then there is only one network connection.

Figure 2. Functional view of the NCP

signaling and support for TCP connections). Requirements (5), (8), (9), and (10) are beyond the scope of this paper and are future work. Requirements (3), (4), and (5) have been largely addressed in previous work (see the Related Work section of this paper).

4.1 Overview of SOCKS version 5

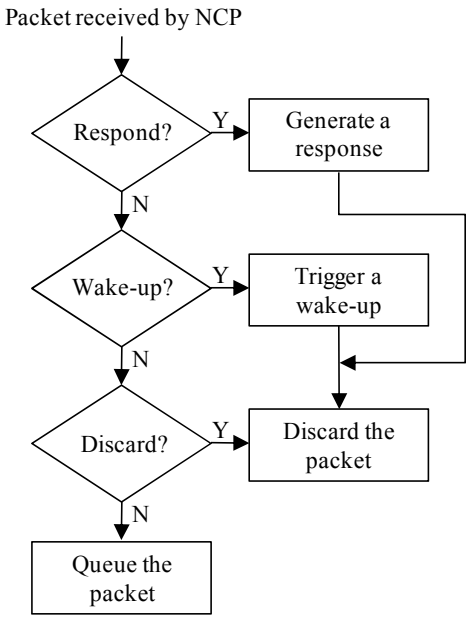
The SOCKS standard [16] describes how TCP connections and UDP packet flows can be relayed through an intermediate host that executes a SOCKS daemon. We use the SOCKS service as a key component in the NCP to meet the requirements for supporting TCP connections and UDP data flows.

SOCKS is an Internet service that comprises a client library and a daemon. The SOCKS daemon is typically executed in a firewall, to allow for secure (that is, relayed from a secure host) TCP and UDP client access to a network. The daemon supports both outbound and inbound TCP connection requests and UDP packet flows. To use SOCKS, a client application must be "socksified" to support encapsulation of key sockets functions using the SOCKS client library. The result of this encapsulation is that a client application can transparently connect via the SOCKS daemon to a server, or a server can transparently connect to a listening client application. Most web browsers and FTP, SSH, and telnet client applications already support SOCKS since it is a popular service used in firewalls. The SOCKS standard supports IPv6 addresses allowing for future migration to IPv6.

4.2 High level view of the NCP

The NCP covers for a sleeping host. This requires the NCP to know the power state of the host (for example, awake or asleep) and to be able to transfer state between the host and NCP. Figure 2 shows the key functional steps for an NCP, which are:

1. The host determines that it is time to go to sleep (for example, based on an operating system inactivity time-out or explicit action by a user).

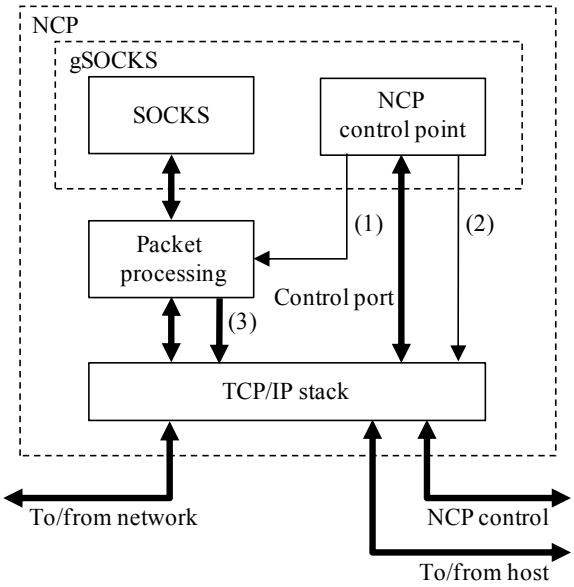


Note: Queued packets are delivered to the host when it wakes-up.

Figure 3. Packet processing by the NCP

2. Notice and state are passed to the NCP, and the host goes to sleep.
3. The NCP maintains full network presence responding to, and generating, protocol and application packets as needed.
4. The NCP determines when a packet requiring the full resources of the host has arrived and signals the host to wake up. The host could also wake up on user activity or from an internal timer.
5. Once the host has fully woken up, state is passed back from the NCP to the host, and the host returns to normal fully powered-up operation.

When the NCP is covering for a sleeping host it receives packets intended for the host. Each received packet results in one of the following actions, directly respond to the packet, wake-up the host, discard the packet, and/or queue the packet for later processing by the host when it is awake. Figure 3 shows the flowchart of actions for each received packet. Low-level packets such as ARP and ICMP packets can be responded to directly. Other packets, such as an SNMP GET, may require the host to be woken-up if the host is running the service corresponding to the received packet. An SNMP GET has a finite lifespan, it cannot be queued for later response (that is, it does not make sense to respond to an SNMP packet many minutes later). Some application packets can be queued for later processing by the host when it wakes-up. We specifically consider two network applications where queueing of packets makes sense for



Note: Control (1) communicates host sleep state to NCP packet processing. Control (2) is used to change the TCP retransmission time-out. Data path (3) is used to send response and wake-up packets to the host.

Figure 4. Architecture of the NCP

later processing and where proxying can enable a host to sleep. The applications are SSH and IM.

SSH is used as a secure telnet replacement for remote access to a terminal console. SSH uses a TCP connection. If the TCP connection is dropped, then application state is lost in the remote computer. In addition, a lengthy re-login process is required to reconnect. As a result, users with SSH connections typically disable power management in their local (client) hosts. During an active SSH connection, messages can flow from the remote computer to the console. For the client host to be able to sleep, the TCP connection must be preserved and any messages queued for later display on the console window in the host (that is, for when the host wakes-up).

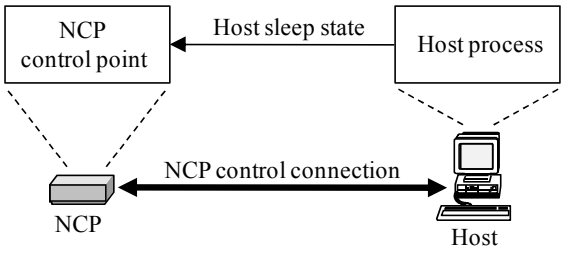
IM is often used in multi-way chat sessions. IM uses a TCP connection from IM client to an IM server. If a client TCP connection is dropped, the client is removed from the multi-way chat and any IM messages sent during the disconnected period will not be received (by the disconnected client). For an IM client to be able to sleep and not lose incoming messages during its sleep period, the TCP connection must be preserved and any messages queued for later display on the IM client application in the host similar to the above described SSH case.

Beyond SSH and IM are a broad range of future applications under the rubric of Rich Internet Applications (RIA). RIAs are web-based applications where the web client executes the user interface and the back-end application server executes the application itself. RIAs support both pull and push of data via TCP connections and potentially split state between a server and client. RIAs have the potential to have major impact on power management in desktop PCs. An NCP can support energy efficient operation of RIAs by maintaining connections and buffering data to enable the client host to sleep.

4.3 The green SOCKS component of the NCP

NCP requirements (6) and (7) are to allow new inbound TCP connections to be established and to preserve TCP connections. To meet these requirements the SOCKS service is used. Figure 4 shows the architecture of the NCP with the "green SOCKS", or gSOCKS, component. The packet processing component handles packet discard, packet response generation, and wake-up (that is, the first three decision blocks of the flowchart in Figure 3). The gSOCKS component consists of an unaltered SOCKS daemon and a new NCP control point daemon. The SOCKS service relays TCP connections (similar to what is shown in Figure 1) and contains buffering. The use of gSOCKS requires that the NCP be fully powered on at all times so that all TCP connections to be preserved during host sleep are relayed at all times. Short-lived TCP connections (for example, HTTP connections as part of web browsing that occurs only when client host is awake and in active use) that do not need to be preserved when a host sleeps are not relayed through the NCP gSOCKS component.

For the SOCKS service to be used to preserve TCP connections for sleeping hosts one change must be made to TCP in the NCP to prevent a time-out and disconnect of a connection. When a host goes to sleep, any TCP connections from the NCP to the host trying to deliver a packet to the host will go into an exponential backoff period (due to time-out caused by delivery failure). If the host wakes-up in the middle of a backoff period, it may still take many tens of seconds before the current backoff period completes and the retry packet is successfully resent and received. First at this time can any queued packets flow on the connection (as a result of the ACK from the host for the successfully delivered retry packet). In addition, after a fixed number of backoffs (which varies with TCP implementation), a backed-off connection will "give up" and close. To meet NCP requirements (7a) and (7b) the backoff of all NCP-to-host connections should be "frozen" when the NCP determines that the host is asleep (done via control (2) in Figure 4). Then, when the NCP determines that the host is again awake, the backoff timer for each NCP-to-host connection is reset to zero. This will cause an immediate



Note: Host process catches wake-up and sleep interrupts and communicates them to the NCP.

Figure 5. Architecture of power state signaling

resend of any unacknowledged packets and then all data flows will immediately resume.

4.4 Signaling of host sleep state to the NCP

NCP requirement (2) is to know the power state of the network host that the NCP is covering for. How this is done depends on the location of the NCP. If the NCP is located outside of the sleeping host (for example, in a switch in the network), then host sleep state signaling can be accomplished via a new daemon or process running in the host that captures operating system sleep and wake-up interrupts (for example, via ACPI in Windows and Linux), and communicates these interrupts to the NCP via a packet-based protocol (note that if the NCP is located within a NIC, this signaling is not needed). A notification packet is defined that contains a field that can contain two values signifying that 1) the host is now entering a sleep state, or 2) the host is now awake after having been in a sleep state. Figure 5 shows the architecture for power state signaling where a TCP connection is established between the host process and the NCP control point. The NCP listens for these connections on a pre-defined port.

If a host being proxied physically disconnects from the network, the NCP must stop covering for the host and clean-up NCP resources including closing any TCP connections that it may be relaying via the gSOCKS component. There are several possible ways for an NCP to determine that a host has disconnected, they are:

- The host periodically wakes-up (for example, based on an internal timer) and "checks in" with the NCP by using the sleep and wake-up notification packets described above.
- The NCP periodically polls the host to determine its presence. The poll could be to the host NIC (for example, via an ARP if the NIC can directly respond to an ARP) or it could be a full wake-up of the host and its operating system and applications (for example, by sending a Magic Packet) followed by the sending of notification packets.

Figure 6. The target system: Linksys WRT54G

- The NCP determines that the host is disconnected the first time that the NCP needs to wake-up the host and the host fails to wake-up thus signaling its absence.

In our design of an NCP we require a periodic timer-based wake-up of the host to exchange wake-up and sleep notification packets with the NCP control point. A process in the host can implement this periodic wake-up using a hardware timer that is always running, even when the host is asleep. Such a timer capability exists in x86-based PCs. If a host fails to wake-up and communicate with the NCP at its designated time-out, then the NCP will assume that the host is disconnected. We used a ten minute periodic wake-up in our prototype.

5. Development of a Prototype gSOCKS

We prototyped the gSOCKS component of the NCP in a low-end router. Specifically, we used a Linksys WRT54G version 2.2 SOHO router (see Figure 6) and replaced the original firmware with the open source WhiteRussian distribution of OpenWrt [19]. The WRT54G has a processor running at 216 MHz and 16 MBytes of RAM. The router has a NAT service, so applications running on remote servers connected to a host behind the router only see the router's IP. The router consumes about 8 W when all four Ethernet ports are connected to a link. Network equipment such as this router would typically remain fully powered-on at all times. Our addition of gSOCKS to the router did not change the power use (of the router), so NCP requirement (1) was met.

The WRT54G router does not have a SOCKS daemon included, however the Srelay package has already been developed for OpenWrt [26]. We used Srelay for our prototype gSOCKS implementation. We implemented an

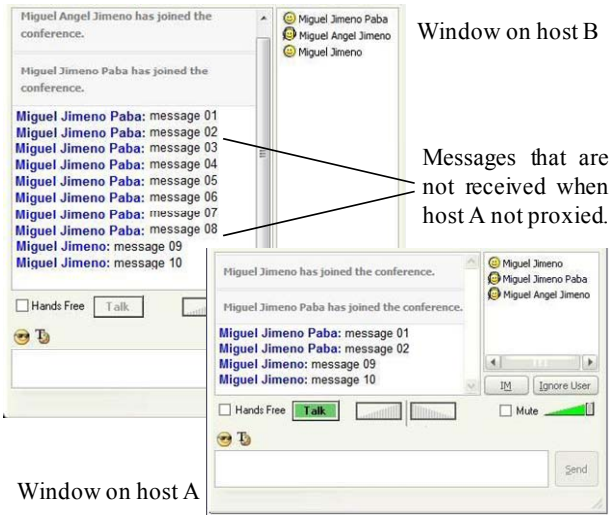


Figure 7. Results from IM experiment

NCP control point daemon for the router and a power state signaling application to run in a Windows-based client host. The NCP control point is a sockets application that executes in the router. The control point application listens on a pre-determined port for a connection and message from the client host. The only two messages supported are:

- Host is going to sleep.
- Host is now awake.

The client host application generates these two messages in response to an operating system interrupt (an ACPI interrupt) signaling power state changes in the host. We developed the client applications for Microsoft Windows. A TCP connection is established and closed just to send the one control message. The message sent by the client contains the IP of the client host. The control program in the client host resumes execution when it wakes up and detects another message from the operating system signaling that the system just woke up. This triggers a "host is now awake" message that is sent to the NCP control point.

We were unable to implement a freezing and reset of TCP back-off in the NCP as described in the previous section. To implement NCP requirement (7b) we instead found a way to change to TCP_MAX_RTO to 1 second and TCP_MAX_RETRIES2 to very large (we made it 10,000). Thus, when a client host goes to sleep the gSOCKS will retry any undelivered packets every second and otherwise not "give up" and close the connection. To implement this method, we needed to modify the TCP implementation in the Linux kernel to expose the key variables. We used the patch in [8] to achieve this. This method of achieving requirement (7b) generates much more overhead traffic than the stated design (of freezing the time-out), but achieved our goal. This approach works well for a single client supported by the gSOCKS. For a gSOCKS supporting many clients (some sleeping and some awake at any given time), this may not be a good approach. Implementing the back-off freeze is future work. We used a timer to give a sleeping host a maximum sleep time. If the client does not wake-up before the timer expires, all TCP connections are reset and the TCP_MAX_RTO and TCP_MAX_RETRIES2 variables are reset to default values. This satisfies requirement (7c).

6. Evaluation of the Prototype gSOCKS

In this section we describe our evaluation of the prototype implementation. Specifically, we address that our gSOCKS meets requirement (7). We also discuss how requirement (9) could be met, but constraints in our Linksys router prevented full implementation.

6.1 Test bed configuration

The test bed consisted of a Linksys router WRT54-G with our gSOCKS (Srelay SOCKS plus NCP control point) implemented. The client host to be proxied (called "host A") was a Dell Optiplex desktop PC with a 3.2 GHz Pentium IV and 1 GByte RAM running Windows XP. The PC had a Broadcom NetXtreme 5755 NIC supporting ASF 2.0 (and thus the ability for the NIC to respond to ARP packets when the PC was asleep). When necessary, two other PCs with the same configuration labeled as host B and C were also used. All hosts were connected to the router, but only host A was covered by the NCP.

6.2 Definition of experiments

Two experiments were conducted to show that TCP connections could be preserved for real applications – SSH and IM – when the client host was sleeping.

SSH experiment: The session-oriented application used for this experiment was the SSH client in the well known Putty program. Putty has SOCKS support built in. For the purpose of the experiment we developed a simple test application to output messages to the console every few seconds (the messages were text strings such as, "Message #1", "Message #2", and so on). This application was run on a remote server (host B) and messages were sent to host A through the SSH connection to appear on the SSH console window on host A. For the experiment, host A was put to sleep for 30 minutes during which the application was running in host B (and outputting messages on the console window in host A). The experiment was run with and without SOCKS support enabled in Putty. When host A was returned to a

fully powered-on state, the contents of the console window were examined for lost messages.

IM experiment: The Yahoo IM client was used for this experiment. The Yahoo IM client had SOCKS support built in. For this experiment we used three hosts (host A, B, and C with users Miguel Angel Jimeno, Miguel Jimeno, and Miguel Jimeno Paba, respectively) to participate in a three-way chat session. Only host A had its connection to the Yahoo IM server relayed through the gSOCKS. Host A was put to sleep in the middle of a three-way chat for 30 minutes. On wake-up of host A, the IM client window (in host A) was examined for lost messages.

6.3 Results from the experiments

The results from both experiment showed that no messages were lost when the host A connection was relayed through gSOCKS. The results also showed that the application – the SSH console window and IM client window – updated immediately on host wake-up. Figure 7 shows the results from the IM experiment. The messages that would not be received when host A was not covered by the NCP are shown.

6.4 Other evaluations and discussion

We also measured the throughput achievable on a relayed TCP connection (for FTP) through the Linksys router. A relayed connection is handled by the router control processor. The throughput was about 12 Mb/s, which is far less than the throughput of a non-relayed connection. This suggests that performance needs to be a future consideration for applications that have long-term TCP connections and have high throughput needs. IM typically does not require high throughput. Most SSH connections are for low data rate console messages, but large file transfers can also done via SSH.

We also noticed that some client applications (such as, Windows Live Messenger) are able to detect when the client host goes to sleep and signal the server of this state change via an application message. This would cause a logoff of the client from the server – this is an undesirable condition (the whole purpose of proxying is to prevent logoff – to make sleeping invisible to the server) and needs to be further investigated and addressed.

Finally, our evaluation showed that the NAT capability of the Linksys router caused a conflict with SOCKS support for UDP data flows. Specifically, UDP flows bypassed SOCKS and went directly to host A. This should be resolved if we can get NAT "turned off", or disabled, in the Linksys router.

7. Related Work

The wireless community has long been interested in methods to reduce energy consumption of mobile wireless hosts in order to increase battery life. One approach that has been explored is to have a hierarchy of hardware components ranging from high-power and high-function at the bottom to low-power and low-function at the top. The low-power components do work for the high-power lower tiers, and thus allow the high-power components to sleep. One example of this is "Wake on Wireless" where a low-power out-of-band wireless component is used to wake-up a (relatively) high power PDA [24]. Another example is a laptop with three levels of hardware where the lower-power components perform simple network tasks (such as keeping email up to date) allowing the higher power components to sleep [25]. The term "proxy-based architecture" is used in [25]. Our work is more general in scope than that of [24] and [25].

The use of "performance enhancing proxies" (PEP) to mitigate link-related degradations is described in RFC 3135 [4]. A means of handing TCP connections during periods of disconnection is described whereby a proxy would "zero window" a server host during the disconnection period. Handling optionally reliable links and hosts is further considered under the scope of Delay-Tolerant Networks (DTN) [11]. Our NCP goes beyond the scope of PEP and DTN. Our gSOCKS does not alter the behavior of a server. None-the-less, many of our ideas for gSOCKS are strongly motivated by RFC 3135.

Proxying for maintaining networking connectivity for sleeping hosts was first explored by the authors and colleagues in the mid-1990s for shared Ethernet networks [6]. This work was later refined for IP networks in general [7], [13], [18]. A specific goal of being able to proxy low-level tasks including ARP, DHCP, and ICMP was explored. A simple prototype was developed and is described in [13]. In [3] an initial exploration of the architectural constructs required to support selective connectivity was presented. Selective connectivity is the notion that a host can choose the degree to which it maintains a network presence, rather than today's binary "connected" or "disconnected". A key architectural construct to support selective connectivity is an assistant that stands in for a host that is sleeping. Thus, [3] begins a more mature thinking of the long-term implications of network presence and how to support it in future networks.

Recently, Microsoft has begun to prototype a scheme called Somniloquy whereby a secondary low-power processor covers for the main processor of PC [1]. The prototype has been developed on a USB-based gumstix device. Somniloquy appears to be a general purpose

Table 1. Mapping of NCP requirements to related work and our work

Requirement	How requirement has been met
1	NCP can be implemented in low-end router (Section 5), or within a co-processor or NIC that contains a processor that consume a very small incremental power increase (see ref. [1]).
2	Power signaling semantics guarantee that NCP is aware of power state of the host (Section 4.4).
3	Gratuitous ARP can be used to support two hosts with the same IP address on one subnet with only one host operational at a time (see ref. [5]). Implementation of NCP in a NIC does not require gratuitous ARP.
4	Low level protocol proxying can be implemented for ARP, ICMP, DHCP, etc. (see ref. [1], [2], [6], [7], and [13]).
5	NAT support has not yet been fully considered.
6	SOCKS protocol supports port binding which allows the NCP to detect a TCP SYN directed to a sleeping client and trigger a wake-up (Section 4.3). However this was not evaluated in our gSOCKS implementation.
7	Modifications to the TCP implementation allows for buffering packets (Section 5). The NCP control point uses a timer to consider a client as disconnected, thus cleaning-up abandoned connections (Section 4.4).
8	Wake-up of a sleeping host as a function of gSOCKS buffer state has not yet been implemented.
9	The SOCKS protocol supports port binding that can listen for UDP packets. We were unable to get this to work, however (see discussion in Section 6.3).
10	Application stubs running on a proxy can process and respond packets on behalf of an application (see ref. [1]). Dedicated proxies can be implemented for specific applications (see ref. [14] and [15]).

architecture that can also support applications via application stubs. Somniloquy builds upon many of the ideas in [7], [13], and [18]. Somniloquy is not able to preserve TCP connections between host awake and sleep; our gSOCKS does address this important capability.

The future direction for Somniloquy, and possibly for our work, is to host the service on a NIC. NICs are becoming more capable, many now include onboard processors. TCP Chimney is a Microsoft NDIS 6.0 architecture for full TCP offload [12]. TCP Chimney provides a direct connection between applications and an offload-capable NIC to reduce TCP-related processing load on the host CPU. This enables the NIC to perform TCP processing for offloaded connections, including maintaining the protocol state. As such, TCP Chimney is the closest to our gSOCKS approach but is limited to a NIC-based solution. Also, TCP Chimney does not explicitly address network connectivity beyond that of TCP connections. Our SOCKS-based approach is broader in scope and can be targeted for other network equipment, as well to a NIC (with onboard processor).

8. Summary and Future Work

In this paper we have identified network connectivity as a major problem for power management of network connected hosts. A Network Connectivity Proxy (NCP) was defined, with reference to previous work including [1], [6], [7], [13], [18]. The previous work has not, however, fully addressed how to maintain TCP connections during periods of host sleep. Thus, the key motivation of our work presented in this paper was in addressing how established TCP connections could be maintained whole an end point (host) sleeps. We designed, prototyped, and evaluated a SOCKS-based approach to solve this problem. Table 1 summarizes both related work and our contributions in terms of meeting NCP requirements. The table shows the NCP requirements in the first column and describes how they are met in the second column. All NCP requirements have effectively been met – with the exceptions of requirements (5) and (9) – in this and in related work. Thus, we believe that the development of a fully operational NCP is feasible.

The energy savings potential of adopting proxying to solve network connectivity is considerable. If even a tenth of the estimated energy waste ($5 billion per year in the US described in Section 2) can be saved, this is still a considerable savings.

Future work consists of 1) implementing a gSOCKS that can support multiple clients (the current prototype supports only one client), 2) supporting UDP data flows, 3) investigating adaptive methods based on the buffer fill rate for gSOCKS to wake-up a sleeping client, and 4) better understanding the requirements of future network applications including RIA. We also need to better understand performance requirements for the NCP and the interaction of client applications that signal a server when the host client goes to sleep.

Acknowledgment

The development of this material is based upon work supported by the National Science Foundation under grant CNS-0721858 (Christensen), Cisco Collaborative Research Initiative (Jimeno), and California Energy Commission – Public Interest Energy Research program (Nordman).

Note

Miguel Jimeno is on leave from Universidad del Norte, Colombia.

References

[1] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta, "Augmenting Network Interfaces to Reduce PC Energy Usage," To be submitted, 2008. URL: http://research.microsoft.com/users/ranveer/docs/somniloquy.pdf.

[2] Alert Standard Format (ASF) Specification, Version 2.0, April 23, 2003. URL: http://www.dmtf.org/standards/documents/ASF/DSP0136.pdf.

[3] M. Allman, K. Christensen, B. Nordman, and V. Paxson, "Enabling an Energy-Efficient Future Internet Through Selectively Connected End Systems," *Sixth Workshop on Hot Topics in Networks (HotNets-VI)*, November 2007.

[4] J. Broder, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, June 2001.

[5] S. Carl-Mitchell and J. S. Quarterman, "Using ARP to Implement Transparent Subnet Gateways," RFC 1027, October 1987.

[6] K. Christensen and F. Gulledge, "Enabling Power Management for Network-Attached Computers," *International Journal of Network Management*, Vol. 8, No. 2, pp. 120-130, 1998.

[7] K. Christensen, C. Gunaratne, B. Nordman, and A. George, "The Next Frontier for Communications Networks: Power Management," *Computer Communications*, Vol. 27, No. 18, pp. 1758-1770, 2004.

[8] Customizable TCP Backoff Patch by B. Woodard, 2006. URL: http://lwn.net/Articles/202398/.

[9] Energy Information Administration, "US Household Electricity Report," July 2005.

[10] "EPA Announces New Computer Efficiency Requirements," EPA Newsroom, Release date: October 23, 2006, Contact: Enesta Jones.

[11] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," *Proceedings of ACM SIGCOMM*, pp. 27-34, August 2003.

[12] "Full TCP Offload," Microsoft, 2008. URL: http://msdn.microsoft.com/en-us/library/aa503758.aspx.

[13] C. Gunaratne, K. Christensen, and B. Nordman, "Managing Energy Consumption Costs in Desktop PCs and LAN Switches with Proxying, Split TCP Connections, and Scaling of Link Speed," *International Journal of Network Management*, Vol. 15, No. 5, pp. 297-310, September/October 2005.

[14] M. Jimeno and K. Christensen, "A Prototype Power Management Proxy for Gnutella Peer-to-Peer File Sharing," *Proceedings of the IEEE Conference on Local Computer Networks*, pp. 210-212, October 2007.

[15] J. Klamra, M. Olsson, K. Christensen, and B. Nordman, "Design and Implementation of a Power Management Proxy for Universal Plug and Play," *Proceedings of the Swedish National Computer Networking Workshop (SNCW 2005)*, September 2005.

[16] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, "SOCKS Protocol Version 5," RFC 1928, March 1996.

[17] S. Lohr, "An Energy Diet for Power-Hungry Household PCs," *The New York Times*, August 6, 2008.

[18] B. Nordman and K. Christensen, "Improving the Energy Efficiency of Ethernet-Connected Devices: A Proposal for Proxying," White Paper, Version 1.0, Ethernet Alliance, October 2007.

[19] OpenWrt: Linux Distribution for Embedded Devices, 2008. URL: http://openwrt.org/.

[20] "PC Energy Report 2007, United States," 1E, Inc. 2007.

[21] Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431, U.S. Environmental Protection Agency ENERGY STAR Program, August 2, 2007.

[22] K. Roth, F. Goldstein, and J. Kleinman, "Energy Consumption by Office and Telecommunications Equipment in Commercial Buildings Volume 1: Energy Consumption Baseline," Arthur D. Little Reference No. 72895-00, January 2002.

[23] M. Sanchez, R. Brown, C. Webber, and G.. Homan, "Savings Estimates for the United States Environmental Protection Agency's ENERGY STAR Voluntary Product Labeling Program," *Energy Policy*, Vol. 36, No. 6, pp. 2098-2108, 2008.

[24] E. Shih, P. Bahl, and M. Sinclair, "Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Device," *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking*, pp. 160-171, 2002.

[25] J. Sorber, N. Banerjee, M. Corner, and S. Rollins, "Turducken: Hierarchial Power Management for Mobile Devices," *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, pp. 261-274, 2005.

[26] Srelay: A Free SOCKS Server for Unix by S. Tomo, 2003. URL: http://socks-relay.sourceforge.net/.

[27] C. Webber, J. Roberson, M. McWhinney, R. Brown, M. Pinckard, and J. Busch, "After-Hours Power Status of Office Equipment in the USA," *Energy*, Vol. 31, No. 14, pp. 2823-2838, November 2006.