Enabling an Energy-Efficient Future Internet Through Selectively Connected End Systems

Mark Allman[†], Ken Christensen[‡], Bruce Nordman[¶], Vern Paxson[†,¶]
[†]*International Computer Science Institute*, [‡]*University of South Florida*
[¶]*Lawrence Berkeley National Laboratory*

Abstract

The Internet's architecture largely and implicitly assumes full-time connectivity, a notion that is embodied in key networking principles including fate sharing, soft state, and the end-to-end principle. In contrast, efforts to allow for more graceful operation in the presence of forced disconnectedness have recently been undertaken that change the underlying style of networking used by applications to accommodate both host-level and hop-by-hop disconnectedness (*e.g.*, for deep space networks where connectivity depends on orbital mechanics). In this paper, we offer an initial exploration of the architectural constructs required to support *selective connectivity*, whereby a host can *choose* whether to be "connected" or "disconnected". While we keep our notion of selective connectivity general, the driver behind our thinking is to allow hosts to go to sleep to realize energy savings while not sacrificing their standing in the network. Studies show that enabling such sleeping offers large potential energy savings. Specifically, we explore ideas related to assistants, soft state, host-based control, and application primitives.

1 Introduction

Many architectural elements of today's Internet implicitly assume that hosts remain attached to the network—*i.e.*, maintain IP connectivity—over extended periods of time. These assumptions manifest in the reliance upon such basic principles as:

Fate sharing. When possible, store state in the end systems that rely upon it so it is only lost if the system itself is lost.

Soft state. Elsewhere, use ephemeral state that the end systems must periodically refresh.

The end-to-end principle. When designing functionality to place inside the network, consider the degree to which such placement is unnecessary or at best a performance optimization since the end systems will themselves have to implement a version of the functionality.

While some might take issue with how we specifically frame these principles, clearly they all assume active end systems that sustain network connectivity over the period of use of the network. Although the network is designed to accommodate failures that cause disruptions in connectivity, these disruptions are viewed as problems to work around, rather than routine operation.

An alternative architectural approach has been pursued in the development of *delay/disruption tolerant networking* (DTN) [6]. Here the assumption is that connectivity is necessarily intermittent not only end-to-end but hop-by-hop, requiring a switch to a fundamentally different degree of store-and-forward operation. DTN-style operation opens up possibilities for networking in contexts previously prohibited by Internet-style assumptions of end-to-end connectivity, but at the cost of major reworkings of how applications use the network.

In between these two architectural styles lies a view of connectivity that we term *selectively-connected networking*. By "selectively-connected" we mean an end system knowingly manages the extent of its network connectivity in response to internal or exterior events. Here, "knowingly" refers to the end system anticipating changes in connectivity (which might be voluntary, or instead about to be imposed upon it) such that it can respond to them. For example, end systems might anticipate losing connectivity entirely because they are moving into a region that lacks layer 2 connectivity (*e.g.*, a cell tower signals that the end system's current trajectory has it soon reaching a no-coverage zone), or their owner is powering them down in some fashion (*e.g.*, every time a user closes their notebook).

One highly significant form of selective connectivity, and one that is much richer than simply "no connectivity," concerns placing end systems into some degree of "sleep" in order to *conserve energy*. Clearly, such sleeping yields major benefits for portable devices by greatly extending their battery lifetime. Less widely appreciated, however, is the degree to which energy savings at *much* greater scales could be realized by employing sleep states for regular desktop systems and other end systems powered directly by the electrical grid.

The past two decades have seen a significant rise in energy consumption due to the accelerating use of computing and networking. Today, in offices, desktop systems at a minimum remain powered up all day long—whether being actively used or not. Further, field assessments show two-thirds of such systems to be fully on after work hours, with only 4% operating in sleep mode [14]. It appears plausible that network connectivity drives much of this [12]. These systems remain available to facilitate *sporadic, occasional* activity, such as:

- User remote access.
- Administrator access for maintenance (backups, patch management).
- Service provider access, for (primarily home) systems that serve as the on-site access point for entertainment and communication services, such as set-top boxes or VOIP systems.
- Preservation of network state.

In the residential sector, the average desktop PC is on 34% of the time and spends only 4% of the time in an some form of "sleep" state [15]. Further, [15] notes that more than half the time a PC is on no one is actively using the machine.[1] [5] quantifies the potential savings for just the United States to be in the range of $0.8–2.7 billion per year (based only on desktops and notebooks).

Of the reasons systems stay fully powered, preserving network state has the greatest implications for network architecture. When a system sleeps, it can recover its *local* state upon waking, but its network peers—due to their reliance upon architectural principles such as fate-sharing and soft state—will often have discarded their associated state on the assumption that the system's prolonged outage reflects a serious failure. For example, a session can fail to function after the system reawakens due to: *(i)* TCP reliability (remote peer sends new data and eventually gives up in the absence of its acknowledgment); *(ii)* TCP keep-alives, often enabled by default to probe for liveness; *(iii)* session-layer activity (*e.g.*, routine SSH key rollover); or *(iv)* management activity (*e.g.*, DHCP lease expires in the absence of renewal). Thus, idle hosts are often left fully powered because network protocols and mechanisms fail when the host is not able to conduct basic state-keeping operations.

What we desire for such hosts, however, is a degree of selective connectivity such that they continue to have a limited presence on the network, but with the semantics of that presence under the control of the end system, so that it can trade off on a fine-grained basis the utility of maintaining connectivity versus the benefits of sacrificing it (*e.g.*, to enable a range of power management decisions).

For today's Internet, one can pursue a series of "one-off" work-arounds for the various protocols and applications that most frequently thwart the sort of selective connectivity necessary for energy efficiency (*e.g.*, installing a proxy to respond to TCP keep-alive probes on a host's behalf). In this paper we advocate a different approach that starts an exploration of the basic network architecture and abstractions required to allow a network to support selective connectivity.

We emphasize that enabling energy efficiency serves as a driver for our thinking, but does not limit the overall scope. We anticipate that the architectural elements will prove significantly more general, providing abstractions for secure task-delegation, the evolution of soft state to a notion of "proxyable state," and a framework for network elements such as middleboxes to aid absent end systems while allowing those systems to secure their data communications end-to-end (more generally, layering security to include more than the two ends of a conversation).

The remainder of this paper is organized as follows. In § 2 we discuss related work. In § 3 we present a quick summary of naturally-occurring network traffic captured in the middle of the night on an enterprise network as an example of the "chatter" our current networking technologies produce. In § 4 we describe our initial designs of new architectural components for supporting selectively-connected networking. We conclude in § 5. We stress that this is an initial foray into selective connectivity and the ideas presented in this paper are not fully worked out designs. A significant reason for writing this paper is to solicit feedback from the community.

2 Related Work

First, we note that a number of efforts have been undertaken to make hardware more energy efficient. For instance, by using Dynamic Power Management for components [3] and manufacturing computers that follow the U.S. EPA Energy Star program [11].

Several efforts have also directly tackled the problems imposed by networks on host energy consumption. [9] highlights the potential for reducing the energy use of the Internet by putting devices to sleep during idle periods, and proposed both uncoordinated and coordinated sleeping. In uncoordinated sleeping, devices make their own decision to go to sleep (*e.g.*, during extended inter-packet times). For coordinated sleeping, they proposed a network-wide approach based on routing protocols that aggregate traffic into fewer routes. [8] shows that a LAN switch that can enter a sleep state and be woken from it by packets queueing in a buffer that retains power can result in significant energy savings. The role of proxying of connections for power management has been previously studied (*e.g.*, in [7]); we explore some of the findings and implications below.

[1] Of course, this does not mean the machine is not doing useful work.

Additionally, many of today's end systems have a "Wake on LAN" functionality built into the NIC. Here, a system's NIC remains powered, and if a special packet arrives, the NIC wakes the host to carry out a given task. Such operation is suggestive for our purposes. However, we stress that we do not intend to limit ourselves to only this style of sleeping and wakeup. First, doing so would presuppose too much of the architectural solution in terms of where state can reside and what sort of processing goes into the decision to awake the host. Wake-on-LAN tends to result in either far too few or far too many wakeups. Second, if we want our mechanisms to extend to other selective-connectivity scenarios, such as a host anticipating its entrance into a region that lacks connectivity, then clearly we cannot rely upon the NIC mediating on the end system's behalf.

In summary regarding energy efficiency, the literature includes a number of efforts that touch on some of the ideas we explore. However, to our knowledge there has been little work on *architectural* components to facilitate selectively-connected networking. The main related architectural work of which we are aware concerns the development of Delay/Disruption Tolerant Networks (DTNs) [6, 1]. The general idea behind these networks is that uncontrollable connectivity issues dictate when a particular node will be able to access the network, and at those points full end-to-end connectivity likely might not exist; therefore, a store-and-forward architecture is used to allow data to propagate opportunistically as connectivity subsequently allows.

One instance of such a network is a deep-space network whereby orbital mechanics dictate when two nodes (say on a planet and on an orbiting satellite) will have connectivity. Therefore, during connected periods "bundles" are pushed from one node to another, with the receiving node taking the responsibility to further propagate the data to its final destination. The situation we tackle is significantly different in that (i) nodes can *voluntarily* alter their connectivity to the network, (ii) conceptually, connectivity is diminished (some forms still function) rather than absent, (iii) nodes can re-establish full connectivity if needed for a pressing task, and (iv) we don't aim to facilitate communication in the face of very large delays—when connectivity exists, we presume it has the same sort of delays as present in today's Internet. With DTN networks, in contrast, disconnected hosts are gone until the next connected interval. The components to address each of these situations are quite different.

Finally, a number of efforts within the wireless networking context are interested in developing techniques to allow mobile devices to conserve battery power. One theme of the work is to develop multiple tiers of power consumption, with lower-power-level tiers handling small, mundane tasks and higher-power tiers taking on the more complicated functions. Examples of work in this area include [16, 17, 2].

3 Ongoing Network Chatter

A major hurdle for end systems desiring to enter a sleep state arises from the large degree of network activity that more or less continuously seeks to engage the system. Each such type of chatter has different implications for the degree to which end systems can either ignore it or partially engage in some fashion (*e.g.,* via an agent acting on their behalf).

Previous work assesses the packets received over a 12+ hour period by an idle PC connected to a campus network [7]. Over the entire period, the system received on average more than 6 packets each second. Within this traffic, the activity is identified as reflecting more than 20 network protocols (*e.g.,* ARP, service location, switch and bridge maintenance, Windows name service, DHCP broadcasts, routing protocols, port scans, network time, network management). This data is quite suggestive at the scope of the "chatter" problem, but offers only a single vantage point, and reflects measurements over the bulk of a day, during which some periods will naturally see much more chatter activity than others.

To look a bit more deeply at the nature of such activity, we recently analyzed snippets from a large collection of traces recorded internal to an enterprise, as follows. The underlying datasets for our analysis are a collection of 72 traces recorded off the ports of switches internal to the Lawrence Berkeley National Laboratory, a medium-sized enterprise (8,000 hosts) engaged primarily in basic research and open science. Traces each generally span 23–24 hours and captured the traffic (400+ GB total) of 5 ports off of a given switch, with each trace monitoring a different set of ports. Many of the ports are directly connected to end systems, but some instead connect to other switches, hubs, or broadcast Ethernets, so a single port may include multiple end systems.

Detailed analysis of these traces remains for future work. Here, we present a brief, phenomenological look at the character of network "chatter" manifested within them. To this end, we extracted a 60-second slice from each trace, starting at 3:18 AM local time (selected arbitrarily). Each slice shows significant chatter, with the least busy containing 171 packets, and the median rate of 1,348 pkts/min reflecting an average per-port rate of 4.49 pkts/sec.

In terms of types of traffic, 20% of the packets are non-IP, including ARP, Novell, Appletalk, IPX, and NetBEUI. Interestingly, the site does *not* route these last three, and attributes their presence to legacy equipment and/or devices such as printers that enable numerous protocols to facilitate turn-key operation [4]. Of the IP traffic, the bulk consists of TCP (primarily backup services, SSH and

Windows NetBIOS, but a number of other services) and UDP (NFS, SNMP, DHCP, DNS, NTP, and again quite a few additional services). A small proportion of the UDP traffic is multicast (service location, IGMP, PIM) and broadcast (DHCP, printing).

Thus, the middle-of-the-night "chatter" reflects a wide range of types of activity. To get a sense of the degree to which chatter reflects established two-way communication (and therefore a fully powered on machine), and thus necessarily involves an end system, we computed the number of distinct, two-way flows present in each trace. To do so, we defined a two-way flow as observing packets both from MAC address A transmitted to MAC address B, and also from B to A. This definition allows us to determine when a given host is engaged in bidirectional communication even for unknown network-layer or transport protocols. The definition also condenses multiple transport connections between the same hosts into a single instance, so our counts emphasize distinct communicating host-pairs rather than the size or structure of the sessions between them.

Using this definition, we find the 66 of the 72 one-minute snippets include two-way communication, with a median of 3 two-way flows in a given trace. Our traces span 5 switch ports, so the presence of typically 3 such flows in a trace does not mean that every end system is busy during our one-minute slices. However, it does indicate that a non-negligible portion of the end systems are indeed busy; and for traces with multiple two-way flows, usually no one MAC address accounts for all of the flows, indicating multiple busy end systems.

Thus, our brief assessment suggests that "chatter" both comes in disparate forms and for many systems is an ongoing activity. Clearly, however, it will require much more extensive analysis of our traces to adequately model the different types of chatter in order to fully understand their implications for enabling selective connectivity.

4 Architectural Elements

Before discussing some of our initial ideas for new architectural elements, let us first frame a perspective on potential architectural notions as seen from the viewpoint of energy efficiency. For purposes of power management, we can view end systems as having three basic states: *on*, *off*, and *asleep*. In networking terms, such systems: have connectivity; have no connectivity; or operate in a selectively-connected fashion, as we defined in § 1. The *asleep* state is not as crisply defined as the other two states and can manifest itself in varying degrees, depending on the end system's desired behavior and policies. For example, an *asleep* host might request to be woken for certain "important" incoming activity, but shielded from "mundane" activity. Therefore, while we informally discuss a single *asleep* state, the specifics of such a state will vary by environment and user/operator policy decisions. The thrust of our design is to allow end systems in the *asleep* state to continue to have a presence ("standing") in the overall network, though this might require that they occasionally return to a fully powered *on* state when certain tasks need to be performed.

In addition to the state of a host we can classify activity as spanning a spectrum from *low-power tasks* to *high-power tasks*, in terms of the resources they require: both computing power, and what we might term network "presence," *i.e.*, the necessary forms of network interaction and state. For example, responding to a keep-alive or renewing a DHCP lease should require fairly modest resources since the interaction is tightly scoped, and therefore does not require a fully attached host to interact with the peer; these are relatively low-power tasks. Even less demanding would be tasks that the host can wholly delegate to some form of "assistant" that handles these tasks while the host itself sleeps (see below), such as responding to ARP queries (though in this specific case these usually will be followed by communication requiring more resources).

High-power tasks, on the other hand, are those which likely require a host to be fully attached. For example, a request for a file would be a high-power task if it entails finding a file on disk, conducting reliable delivery across the network, logging, reporting errors, and so forth.

We now discuss the architectural concepts and components that are part of our initial thinking. We doubt this list is complete and are interested in engaging with the community to discuss both the structures we describe below and additional components and principles that would help realize an architecture that supports selective connectivity.

Assistants. The first component we discuss is an *assistant*, which is a general mechanism to help a host while the host is in an asleep state.[2] The assistant's job is to perform some of the routine and mundane operations that an end system normally handles. For example, perhaps an assistant keeps connections alive by responding to keep-alives on the end system's behalf. Or, as noted below, an assistant might handle the maintenance of soft state of some form. Also, an assistant could vet incoming traffic to determine whether it is important enough to wake a given host. Such a determination could be made based on a policy that the host itself provides to the assistant (*e.g.*, incoming messages from the enterprise's backup process are cause for wakeup). Assistants could also return "soft errors"—such as "try again later"—so that a remote host can understand that the resource they are seeking is only temporarily unavailable.

[2] More speculatively we might allow assistants to aid systems in the *off* state, as well, even though such systems cannot be awakened on demand. For instance, an assistant could provide the rest of the network with the notion that some device does in fact exist even if it cannot be readily reached at the moment.

We note that we make no presumption of where an assistant sits in the network. It may be on the powered-on NIC of an asleep host; it might be an independent system that sits in the network and acts on behalf of many end hosts; or it might be built into the switch or wireless access point from which end system's get their connectivity in the first place. In addition, a host may interact with several assistants that handle different tasks.

Finally, we note that involving an assistant introduces another point of potential failure. If the assistant simply provides a performance gain, then its failure may not be a large concern. However, to the extent that the assistant takes over functionality from an end system and holds state that the end system will ultimately require, the new failure modes introduced by assistants will need to be analyzed and taken into account.

Exposing Selective Connectivity. A foundational notion in our architecture is that of exposing a host's level of connectivity across different layers of the local protocol stack and potentially to peers with which the host is or might be communicating. Here, our use of energy efficiency as a driver for the exploration immediately points up some particular considerations: it can be valuable for other systems to understand not only that a given system is selectively-connected, but that it is in that state due to power management decisions, because the underlying *reason* of energy efficiency might alter how the peer reacts to the selective connectivity. For example, the peer's policy might be to encourage energy savings, and thus it will also enter a reduced-power state rather than wake the end system to get some low-priority work done. Thus, our mechanisms for exposing selective connectivity will also include some form of accompanying explanation for the end system's decision.

While we want to expose the reasons for a host's selective connectivity as broadly as useful, there is a tussle in doing so. For example, identifying a host as in a low-power state over a long period of time (or simply that it remains only selectively-connected) may expose too much information to would-be criminals, *e.g.*, identifying possible targets to burglarize because computer inactivity suggests that nobody is home. Our design therefore calls for the ability to scope to whom the network provides a host's selective-connectivity status and explanation. For example, to turn off a movie download service while away on vacation a host does not need to publicly expose its state. However, if a host is part of a public peer-to-peer network and is exposing state within that context, then scoping who receives the information is not as easy.[3] It is also tempting to consider some sort of "virtual lamp timer" to stimulate some activity and perturb a long-running connectivity state (again, at a user's discretion); however, it is not clear that this is a winnable arms race.

Evolving Soft State. As discussed in § 1 one of the architectural successes of the Internet has been the use of soft state, which provides major benefits in terms of robustness and flexibility. However, maintaining soft state across end systems that may be asleep and not able to actively renew state poses a problem. We do not advocate reverting to more hard state, rather we wish to evolve the basic notion of soft state to a form that supports persistence for selectively-connected end systems.

We offer two ways to evolve soft state:

- *Proxyable State:* This state calls for the maintenance of soft state by an assistant. Such a system will naturally have both pros and cons over traditional end-host management of soft state. These pros and cons will have to be fully explored for each delegated task.

- *Limbo State:* This state occupies the middle ground between traditional soft state and statelessness. While soft-state assumes a host or resource is no longer available if the state times out before being renewed, a notion of "limbo state" would allow enough information exchange so that participating machines could recognize the distinction between being inexplicably "gone" and simply "asleep".

Host-based Control. Another underpinning of our architecture is leaving the end system in control of how others in the network react to the system's selective connectivity. When designing new mechanisms, we want these to remain flexible enough to support a range of user/operator-designated *policy decisions*. Specifically, when a host moves to a state of selective connectivity, how that host then wishes to be treated by other actors in the network, and which tasks should be delegated to others, should be governed by the user/administrator.

As an example, one user's policy might be to simply drop off a peer-to-peer network at night and rejoin in the morning. Another user, however, may wish to retain their standing in such systems and continue to provide files that are not available elsewhere to get "upload credit" of some sort, while also not keeping the host fully powered between such transactions. This second approach may require both help from a third-party assistant and the host going to sleep sharing some information about which files are being provided across the peer-to-peer network with that assistant. Therefore, users need control over the decision of whether or not to engage the third-party.

Application Primitives. As a way to empower assistants over a range of network activity, our design calls for the development of generic primitives that could be shared across applications. For example, a generalized "keep-alive" message would both convey the status of a host and

[3] As discussed below, this may be a case where a host can delegate certain tasks to an "assistant" and therefore does not need to expose its asleep status to the entire peer-to-peer network.

allow an assistant to act on a host's behalf without explicitly understanding a myriad of protocols. Such a message could go beyond the binary yes/no answer used by today's protocols by including specific information about a host's state, per the notion of exposing the reason behind a host's selective connectivity, as described above.

As briefly discussed above, another example primitive might provide an assistant with an understanding of the files a host exports across a peer-to-peer network. This information would allow the assistant to participate in peer-to-peer searches and wake the end system only when an actual file transfer is required. A version of this sort of functionality has been developed for a Gnutella-like network [10]; however, our goal here is a generalized primitive that works across application types. Additionally, we envision exploring how to use the redundancy built into some applications (*e.g.,* peer-to-peer applications) to aid selective connectivity. For example, many times on peer-to-peer networks a given file is available from multiple places, and so a particular host would not necessarily merit awakening to serve such a file. Of course, redundancy is part of the power of such networks, and thus we must take care not to remove all redundancy in the system.

We also note that resources may be replicated such that when the primary source of some resource is unavailable (*e.g.,* due to a host sleeping), other trusted sources may provide the resource. For example, we could use anycast [13, 18] for such a service (coupled with some new primitives for a host to easily populate anycast servers).

A fundamental question, of course, is whether there are in fact a set of primitives that we can use to cover a wide variety of situations, or whether the primitive we actually require is to provide a *program* to some assistant that encodes the needed functionality. If this latter, then a number of thorny questions about the allowed semantics of the program come into play.

Security. As is often the case, issues relating to secure operation cut across all of the architectural components sketched above. Security concerns raise a myriad of questions: How can end systems securely delegate tasks to assistants? How do remote hosts trust that an assistant indeed has authorization to act on behalf of an end system, and is not an imposter? How do we layer our use of cryptography mechanisms such that we can expose some portions of communications to assistants, while not exposing sensitive data to intermediaries? To what degree do our generalized notions of soft state expose network elements to denial-of-service state-holding attacks? If an adversary wishes to undermine the use of selective connectivity to conserve energy, to what degree will the resulting system be vulnerable to such "denial-of-money" attacks?

We appreciate the import of raising such questions at the forefront of our design efforts, to ensure that the mechanisms they call for become first-class elements of our designs, rather than later attempting to fit them retroactively.

5 Final Remarks

Above we discuss how the current network architecture embodies an underlying assumption that hosts retain IP connectivity over long time periods. In addition, in § 3, we illustrate that current networks do in fact experience a large amount of "chatter" during periods when most hosts are not being actively used by people. Studies have shown that a great deal of energy savings can be realized by putting these mostly-idle hosts in some sort of low-powered sleep mode. To accomplish this while allowing a host to retain its network standing—often the reason for remaining fully powered in the first place—we describe a number of new architectural components and abstractions. We stress that our designs are preliminary and quite likely incomplete. Our goals in writing this paper are both to generally increase the awareness of the issues surrounding so-called selective connectivity and to get feedback on our initial architectural ideas.

6 Acknowledgments

Many thanks to Mike Bennett and Jason Lee for their efforts in recording the traces analyzed in § 3, and Mike for his help in understanding the menagerie of activity contained within them. Discussions with Scott Shenker helped this work, as did the comments from the anonymous HotNets reviewers. This work was funded in part by NSF grants NSF-0721933, ITR/ANI-0205519 and NSF-0520081 for which we are grateful. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of the National Science Foundation.

References

[1] Delay tolerant networking research group, 2007. `http://www.dtnrg.org/wiki`.

[2] N. Banerjee, J. Sorber, M. Corner, S. Rollins, and D. Ganesan. Triage: Balancing Energy and Quality of Service in a Microserver. In *5^{th} International Conference on Mobile Systems*, 2007.

[3] 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, June 2000.

[4] M. Bennett. Personal communication, July 2007.

[5] K. Christensen, B. Nordman, and R. Brown. Power Management in Networked Devices. *IEEE Computer*, 37(8):91–93, Aug. 2004.

[6] S. Farrell. *Delay- and disruption-tolerant networking*. Artech House, 2006.

[7] 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*, 15(5):297–310, September/October 2005.

[8] M. Gupta, S. Grover, and S. Singh. A Feasibility Study for Power Management in LAN Switches. In *Proceedings of the 12th IEEE International Conference on Network Protocols*, Oct. 2004.

[9] M. Gupta and S. Singh. Greening of the Internet. In *ACM SIGCOMM*, pages 19–26, Aug. 2003.

[10] M. Jimeno, K. Christensen, and A. Roginsky. A Power Management Proxy with a New Best-of-N Bloom Filter Design to Reduce False Positives. In *IEEE International Performance Computing and Communications Conference*, pages 125–133, 2007.

[11] B. Johnson and C. Zoi. EPA Energy Star Computers: The Next Generation of Office Equipment. In *Proc. of the American Council for an Energy Efficient Economy Summer Study on Energy Efficiency in Buildings*, 1992.

[12] B. Nordman, A. Meier, and M. A. Piette. PC and Monitor Night Status: Power Management Enabling and Manual Turn-off. In *Proceedings of the American Council for an Energy Efficient Economy (ACEEE) Summer Study on Energy Efficiency in Buildings*, 2000.

[13] C. Partridge, T. Mendez, and W. Milliken. Host anycast service, Nov. 1993. RFC 1546.

[14] J. Roberson, C. Webber, M. McWhinney, R. Brown, M. Pinckard, and J. Busch. After-hours Power Status of Office Equipment and Inventory of Miscellaneous Plug-Load Equipment, 2004. Technical Report LBNL-53729-Revised, Lawrence Berkeley National Laboratory.

[15] K. Roth and K. McKenney. Energy Consumption by Consumer Electronics in U.S. Residences. *Final Report to the Consumer Electronics Association (CEA)*, Jan. 2007.

[16] E. Shih, P. Bahl, and M. Sinclair. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Device. In *8th Annual International Conference on Mobile Computing and Networking*, 2002.

[17] J. Sorber, N. Banerjee, M. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In *3rd International Conference on Mobile Systems, Applications, and Services*, 2005.

[18] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Service. *IEEE/ACM Transactions on Networking*, 8(4):455–466, Aug. 2000.