Assessing the Potential for Saving Energy by Impersonating Idle Networked Devices

Bolla, R., Khan, R., & Repetto, M. (2016). Assessing the Potential for Saving Energy by Impersonating Idle Networked Devices. IEEE Journal on Selected Areas in Communications, 34(5), 1676 - 1689. DOI: 10.1109/JSAC.2016.2545414

Published in:
IEEE Journal on Selected Areas in Communications

Document Version:
Peer reviewed version

Assessing the Potential for Saving Energy by Impersonating Idle Networked Devices

Raffaele Bolla, *Member, IEEE,* Rafiullah Khan and Matteo Repetto

***Abstract*—The idea of proxying network connectivity has been proposed as an efficient mechanism to maintain network presence on behalf of idle devices, so that they can "sleep". The concept has been around for many years; alternative architectural solutions have been proposed to implement it, which lead to different considerations about capability, effectiveness and energy efficiency. However, there is neither a clear understanding of the potential for energy saving nor a detailed performance comparison among the different proxy architectures.**

In this paper, we estimate the potential energy saving achievable by different architectural solutions for proxying network connectivity. Our work considers the trade-off between the saving achievable by putting idle devices to sleep and the additional power consumption to run the proxy. Our analysis encompasses a broad range of alternatives, taking into consideration both implementations already available in the market and prototypes built for research purposes. We remark that the main value of our work is the estimation under realistic conditions, taking into consideration power measurements, usage profiles and proxying capabilities.

R. Bolla is with the Department of Electrical, Electronic and Telecommunication Engineering and Naval Architecture (DITEN), University of Genoa, Italy (e-mail: raffaele.bolla@unige.it).

R. Khan was with the Department of Electrical, Electronic and Telecommunication Engineering and Naval Architecture (DITEN), University of Genoa, Italy (e-mail: rafiullah.khan@unige.it).

M. Repetto is with the National Inter-University Consortium for Telecommunications (CNIT), Research Unit of Genoa, Italy (e-mail: matteo.repetto@cnit.it).

***Index Terms*—Green networking, energy efficiency, network connectivity proxy.**

I. Introduction

Nowadays, electronic devices are often left switched on even when idle, just to maintain their "presence" on the network. Typically, this happens to remotely access the device (e.g., with SSH/RDesktop), to refresh the on-line status on Skype, Messenger and social networks, to maintain the priority in waiting queues for file sharing, and so on [1].

The regrettable consequence of this behavior is a large waste of energy. Reducing the power drawn in idle states is not an effective solution, due to the well-known non-linear power profile of electronic equipment. In fact, despite the large hardware improvements achieved in the last years, recent data and estimates show that end-user devices still account for 60% of energy usage in ICT [2]–[4], and more than 20% of this share is wasted by idle and standby states [5]. However, by delegating background activity to some connectivity proxy, devices could enter sleep states, while the proxy maintains their network presence, hence bringing the potential for larger energy savings.

Different architectural solutions have already been devised to proxy network connectivity, but only few experimental testbeds have been realized so far. Standardization in this area leaves the possibility to adopt different architectural solutions [6], but a common interface for controlling the operation is still missing [1]. Some commercial implementations are already available on the market; however, little effort has been devoted to evaluate the actual saving achievable in practice and to compare alternative architectures.

This paper fills the gap by evaluating efficiency and effectiveness of proxying network connections in real environments. We consider previous studies and experimental testbeds in this field, and we carry out additional performance measurements that are currently missing. In particular, we consider both on-board and external proxy implementations. For on-board proxying, we take into account current high-end network interface controllers (NICs) and experimental NICs equipped with general-purpose processors. For external implementations, we consider proxies running both on low-end devices (home gateways, access points, small and tiny computers) and on high-end servers. To perform uniform comparison between the two classes, we consider the energy saving per device in our analysis.

The potential for energy saving is computed by taking into account several factors beyond the mere power difference between idle and sleep states. We consider available statistics, models and studies to build usage profiles, and we also take into account the power drawn to run the network connectivity proxy. Our work is the first attempt to answer the following questions, which are still open [1]: *i)* what are the real potential to save energy by reducing idle periods?, and *ii)* what is the trade-off between the energy to run the proxy and the potential energy/cost savings? Despite many years of discussion on alternative proxy architectures, and the availability of both standards and commercial implementations, a clear and realistic answer to such questions is not available yet.

The paper is organized as follows. Section II briefly reviews related work and the current state of the art on this topic. Section III briefly discusses the main concept, different implementations and target platforms. Section IV reports performance analyses and points out scalability problems; Section V builds on these data to estimate the potential for saving energy in different environments. Finally, Section VI gives our considerations about the results and highlights challenges still to be addressed by research.

II. Related work

The concept of proxying network connectivity has been around for many years, although with different names and architectural solutions. Research in this field has been devoted to identify basic requirements and functionality [7], [8], to design suitable architectures [9], [10], to investigate the feasibility of managing various network protocols and applications [7], [9], [11]–[14], and to provide implementations for specific hardware platforms [8], [12]–[19].

A reference framework for proxying network connections was standardized as ECMA-393 [6]. This document describes the main functionality that should be implemented to cover for sleeping devices, and sets mandatory and optional normative requirements. However, it is neutral about implementation choices, thus allowing different deployment alternatives. Due to this positioning, a standard to communicate with an external proxy is currently missing [1]. Latest specifications by the ENERGY STAR initiative for electronics and office equipment also include the notion of Full Network Connectivity, which can be implemented internally or externally to covered devices. Such specifications explicitly refer to ECMA-393.

The interest for proxying connections is also evident in market products. Several chipset vendors (e.g., Intel and Broadcom) already ship NIC with embedded proxy functionality (mainly, the mandatory functions required by ECMA-393). Apple has included a Bonjour Sleep Proxy[1] in its network devices and equipment (access points, backup and web TV servers) and Mac OS (Lion) since several years. Microsoft provides InstantGo[2] (formerly, Connected Standby), a set of guidelines and hardware requirements that computer vendors must conform to, in order to run background routines in low-power modes. Intel announced its Ready Mode Technology[3], which enables computers to remain active, instantly ready and always connected while sipping power; this feature exploits the new C-7 low power state in the Haswell architecture of processors.

Despite the availability of both experimental and commercial implementations, a clear assessment of the potential energy saving by proxying network connectivity is still missing. Previously, the effectiveness of proxying has been estimated by real data collected in different environments (university campus, of-

[1]About Wake on Demand and Bonjour Sleep Proxy. URL: https://support.apple.com/it-it/HT201960.

[2]InstantGo: a better way to sleep, by Kevin A Chin. URL: https://blogs.windows.com/bloggingwindows/2014/06/19/instantgo-a-better-way-to-sleep/.

[3]Intel Ready Mode technology. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/intel-ready-mode-technology-brief.html

fice employees) and by reasonable assumptions about the increased usage of power management features [9], [11], [12], [18]. However, important issues like performance constraints that might limit the number of covered devices, the incremental power consumption to run the proxy and the trade-off between these factors have only been considered occasionally. In this context, measurements have encompassed packet classification against the kinds of managed protocols and the number of registered rules [20], latency in answering application's traffic [15] and the overhead of starting and stopping to cover for devices [18]; further, hardware consumption and CPU/memory utilization have also been considered in the same papers. Till now, performance evaluation has only been carried out for NICs [15], [20] and high-end computers [18], thus neglecting all the relevant hardware that lies in the middle (network equipment, low-power devices) and the relationship between the number of covered devices and the additional power consumption to run the proxy.

III. Proxying network connectivity

Proxying network connectivity mainly implies to carry out some tasks on behalf of sleeping devices. Such tasks entail managing low-level connectivity (e.g., ARP, DHCP, NetBIOS) and network management protocols (e.g., ICMP, SNMP), waking devices up when necessary (for example, on new connection requests) and refreshing soft-states for applications (e.g., by sending 'keep-alive' and 'heartbeat' messages). This way, sleeping devices appear active and reachable to all other hosts.

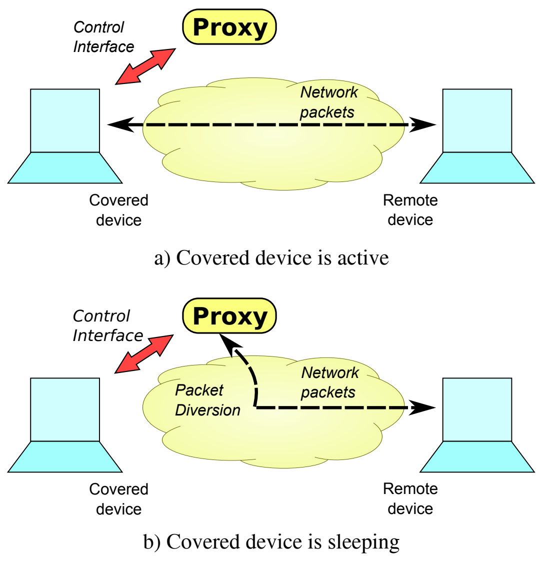


Figure 1. Overview of operations for proxying network connectivity.

A high-level overview of operation is depicted in Fig. 1. When the covered device is fully powered, it directly interacts with remote peers (Fig. 1.a); however, when it is sleeping, packets are caught and managed by the proxy (Fig. 1.b). We call the function that redirects packets towards the proxy as "packet diversion". There is also a "control interface" (red arrow) that is used to configure the proxy (i.e., to select which routines to carry out and when it should start/stop to cover for the device) and to wake up the device when needed.

There are two different ways to operate a connectivity proxy. A *uncoordinated* or *invisible* proxy does not have a control interface. It does not advertise its presence in the network, and it guesses both the routines to be run (or it runs fixed pre-defined routines) and the hosts' power state. This approach does not require to make any changes to hosts and/or applications. On the other hand, a *coordinated* or *cooperative* proxy has a control interface, which allows both registration of required routines and notification of power state transitions.

The proxy function could be either internal or external to the covered devices, with different considerations about additional power consumption and processing capabilities [10]. In both cases, a method to wake-up sleeping devices is required (e.g., Wake-on-LAN or internal circuit).

Internal proxies are typically located 'on-board' of Network Interface Controllers (NICs). Running the proxy on smart NICs is the simplest architectural solution [12], [13], [15]: no packet diversion is required and the control interface (if present) can be easily integrated into the device driver (this already happens for commercial NICs). However, managing complex protocols may require additional computation and memory resources and could lead to excessive power consumption for this kind of devices. This solution is mostly suitable for maintaining basic network presence only (which includes ARP, ICMP, IGMP, DHCP).

External proxies can be deployed in network equipment (like switches, access points, home gateways, routers), or in standalone devices (like streaming devices, set-top boxes). Running the connectivity proxy on network equipment is a good compromise between performance and energy consumption [8], [16], [19]. In fact, such devices are always-on and the possible additional energy consumption to run the proxy is shared among many covered devices. On the other hand, dedicated hardware usually provides higher processing capabilities and more ease in software implementation [14], [17], [18]. In this case, the power to run the additional device might not be negligible and should be carefully compared to the energy saved by clients (in general, a large number of devices must be covered).

Traffic diversion is a concern for external proxying only. If the proxy runs on network equipment that the covered device is directly attached to, there is no need for traffic diversion: all traffic passes through the proxy, which can easily catch packets intended to sleeping devices. In all other cases, traffic diversion in local networks can be implemented by a sort of ARP 'spoofing'. The proxy makes use of the ARP mechanism to bind the IP address of sleeping clients to its own MAC address, and it also uses Gratuitous ARP to update the cache of other hosts. When the covered client wakes up, the proxy binds the IP address back to the client's MAC address with another Gratuitous ARP message. This solution is compliant with RFC 826 [21] and RFC 5227 [22]; it is already used by other networking protocols (e.g., Mobile IP).

The control interface is an element that has almost completely been neglected both in the literature, in commercial implementations and in standards. It is used by the client devices to load and withdraw their routines, and to notify their transition to/from sleep. Zero-configuration protocols are the most suited for this purpose; for instance, the Universal Plug-and-Play standard (UPnP) may be used [19].

The most important task for proxying network connectivity is packet classification, in order to detect packets that trigger background networking routines. In this respect, the choice between software or hardware implementations has different implications on flexibility and performance. Packet classification in hardware may outperform the same operation in software by an order of magnitude [20]; however, software implementations usually provide a higher degree of flexibility in setting and modifying different pattern matching criteria.

Table I lists some proxy implementations available as commercial products or experimental tools. We have both internal (I) and external (E) proxies, and packet classification is usually done by software filters. Supported features include:

- *ARP and Neighbor Solicitation (NS)*. The proxy answers to ARP/NS queries and provides its own MAC address.
- *ICMP Echo-request*. The proxy answers to ICMP echo-request messages.
- *DHCP*. The proxy periodically renews IP address leases with the DHCP server.
- *Wake-on-Connection (WoC)*. The proxy wakes clients up when a connection request (or a new packet in case of UDP) is received on a given port.
- *Wake-on-Pattern (WoP)*. The proxy wakes clients up when a packet contains a given data pattern.
- *TCP Keep-Alive (TCP-KA)*. The proxy answers Keep-Alive messages that TCP implementations might send to check for the presence of their peers (this mechanism is usually exploited by servers to avoid wasting resources for broken connections).
- *Heartbeating (HrtBt)*. The proxy builds and sends solicited or unsolicited messages to refresh the application status. Solicited heartbeats are sent in response to incoming requests, whereas unsolicited heartbeats are triggered at periodic time intervals.
- *Application routines*. The proxy runs simplified background routines provided by applications. Usually, these routines carry

out similar tasks to hearbeating, but they might also integrate more complex logic.

Most proxies are compliant with ECMA-393, although research implementations usually deal with IPv4 only.

The efficiency of an implementation mostly depends on the hardware platform, which establishes the maximum number of devices that can be covered simultaneously and the energy consumed to cover for them. Table II lists hardware platforms used both for commercial products and for research purposes. We consider a wide range of hardware, with different capabilities in terms of processing power and memory resources. There are both on-board devices (O) and external devices, which we classify as network equipment (E) or standalone devices (S). For every platform, we provide indication of the specific proxy implementation they run.

IV. Efficiency of proxying network connectivity

Proxying network connectivity allows devices to save energy by spending more time in sleep mode, but it also brings additional consumption by the proxy hardware. We define as '*efficiency*' the additional energy consumption per client to run the proxy service. In this respect, *efficiency* mainly concerns the hardware platform and the number of devices that can be covered simultaneously.

On-board proxies only cover for a single host, thus there is no need for further discussion; instead, external proxies can cover for multiple clients. An analysis about the maximum number of devices that can be covered simultaneously is available for SleepServer [18]; however, no evaluation has been done for implementations on low-end devices. We fill this gap by investigating performance issues with our NCP implementation [19]; then, we compare the main implementations under consideration in Section IV-C.

A. Performance evaluation for an external NCP on low-power hardware

To evaluate the performance of an external connectivity proxy, we consider a testbed that consists of our NCP implementation [19]. Different hardware platforms have been considered, in order to account for alternative deployment scenarios (see Table II): a home gateway (Lantiq), a tiny computer (Raspberry) and a low-power computer (Jetway).

The NCP architecture is made of several components for packet filtering (classification) and processing, scheduling of periodic events, registration of background routines and a con-

[4]Protocol offloading is mainly defined by the NDIS interface. See https://msdn.microsoft.com/en-us/library/windows/hardware/ff566804\%28v=vs.85\%29.aspx

[5]About Wake on Demand and Bonjour Sleep Proxy. URL: https://support.apple.com/en-us/HT201960

Table I
PROXY IMPLEMENTATIONS.

Name	Type	Control	Pkt class.	ECMA-393	Features
Somniloquy [15]	I	Y	Sw	Y (IPv4)	ARP, ICMP Echo-request, DHCP, WoC, WoP, App. routines.
Protocol offloading[4]	I	Y	Hw	Y	ARP, NS, MLD, WoC, WoP.
SmartNIC [20]	I	Y	Hw	?	ARP, ICMP, TCP and UDP packet classification.
Network Connecvitiy Proxy (NCP) [19]	E	Y	Sw	Y (IPv4)	ARP, ICMP Echo-request, DHCP, WoC, TCP-KA, HrtBt.
Bonjour Sleep Proxy[5] (BSP)	E	Y	Sw	N	WoC for Bonjour applications.
SleepServer [18]	E	Y	Sw	Y (IPv4)	ARP, ICMP Echo-request, WoC, App. routines.

NS=Neighbor Solicitation, WoC=Wake on Connection, WoP=Wake on Pattern, TCP-KA=TCP Keep-Alive, HrtBt= Heartbeating.

Table II
TARGET HARDWARE PLATFORMS.

Vendor	Model	Type	Processor	Memory	Power consumption		Implementation	Notes
					Idle	Processing		
Gumstix	–	O	XScale PXA255 200 MHz	64 MB	1.073W	1.162W	Somniloquy	USB computer-on-module.
Intel	I350	O	–	–	703mW	–	NDIS	Ethernet controller.
Rice Univ.	RiceNIC	O	Virtex-II Pro FPGA + 2 embedded PowerPC	256 MB	–	180mW	SmartNIC	Hardware packet classification.
Lantiq	EASY XWAY VRX288	E	ARM VR9	64 MB	6W	6.4W	NCP	Home gateway.
Raspberry	Pi	S	ARMv6 700MHz	512 MB	3.6W	3.8W	NCP	Tiny computer.
Jetway	JNC9C-550-LF	S	Intel Atom N550 1.50GHz	2 GB	24.8W	27.3W	NCP	Low-power computer.
Apple	Apple Tv 3.2	E	Chip Apple A5 (PowerVR SGX543MP2)	512 MB	0.7W*	2W	BSP	Proxy also in sleep (0.21W).
Apple	AirPort Extreme	E	Atheros AR9344	64 MB	8.6W	10.6W	BSP	
Dell	PowerEdge PE2050	S	2 XEON 5550	32 GB	213W	308W	SleepServer	High-end server.

* Apple declares 0.21 W in sleep, but several independent reviews agree the device consumes 0.7-0.8 W in that mode.

trol interface [19]. The software is entirely written in C++ and runs on Linux.

Packet filtering uses the Pcap library[6]. It works in user-space and filters packets by optimized Berkeley Packet Filters (BPFs), which are derived from a text string in human-readable format. We have taken into account two different strategies to manage the Pcap filters. The first one sets one filter for every kind of routine. The filter includes specific parameters like IP addresses and port numbers; every packet selected by a filter triggers the execution of the corresponding routine. The second strategy provides a more generic filter, which does not use any dynamic information from the set of active routines. This version is termed "Lightweight Pcap" (LwPcap) in the following, because it does not require to continuously update the filters; however, it puts more processing burden on the NCP code, which must do a second stage filtering of all the packets with the actual parameters from active routines. The reason behind the two strategies will be clear when comparing the results in the next subsections.

[6]Tcpdump & Libpcap. Web site: http://www.tcpdump.org.

Our implementation provides the ARP, PING, DHCP, WoC, TCP-KA and HrtBt routines, as described in Section III.

We have chosen UPnP[7] for communication between the devices and the NCP [23]. Our implementation provides an NCP service which exposes several 'actions' to clients. The actions are used to register new routines, to withdraw existing routines, to notify power state transitions, and to transfer protocol or application states (e.g., in case of TCP sessions).

To study scalability issues, we need a large number of clients, but this is unpractical in a real experimental setup. For this reason, we carried out the analysis in a synthetic scenario, by emulating client software; this is perfectly acceptable for our purpose, since only the NCP performance is under investigation.

The maximum number of devices that can be covered simultaneously by a single proxy instance depends on the CPU and the memory available on the specific hardware platforms. Our previous work [19] already showed that the most critical activities are storage of background routines, state transitions, packet filtering and processing.

1) Memory: Memory is mostly used by the NCP to store background routines registered by client devices. Fig. 2 shows the memory taken by the software to cover for an increasing number of devices. Every client device registers 1 PING, 1 DHCP, 1 WoC, 1 TCP-KA and 1 HrtBt routine[8]. We consider both the Virtual Memory Size (VMS, i.e., the total address space taken by the NCP process) and the Resident Set Size (RSS, which is the actual space allocated in the RAM).

Fig. 2 clearly shows that the same code brings to very different results. The VMS is similar for Raspberry and Jetway, because the same compiler was used; however, different kernels and underlying hardware result in very different RSS. Lantiq is the more memory-constrained device, and it cannot register more than 120 devices.

Memory slowly increases with the number of registered clients, although the trend is not clearly visible for Raspberry and Jetway due to the graph scale. This relationship is very good, because it means the software scales well for a large number of devices. The lightweight version consumes slightly less memory, because the filtering string does not depend on the number of registered devices. The difference is more visible for Lantiq, due to the different scale.

Starting from the data shown in Fig. 2, we can estimate the maximum number of coverable devices, by considering the amount of physical memory installed on each platform (see Section IV-B). The estimation also con-

[7]Universal Plug-n-Play. Web site: http://www.upnp.org/.

[8]The ARP routine is automatically registered for each device, without explicit request, because it is needed to implement traffic diversion.

siders the amount of memory taken by the Operating System and other essential services.

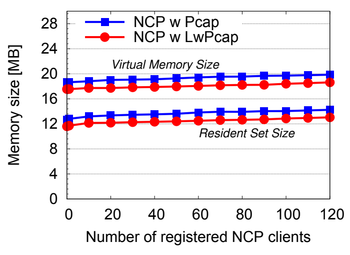
2) State transitions: When a device goes to sleep, the NCP carries out two operations: *i)* it updates the filtering engine by including all information to catch packets intended to that device, and *ii)* it performs any preliminary operation to cover for that device (e.g., inferring dynamic parameters like sequence numbers for TCP connections). Fig. 3 shows the time taken by the NCP to start to cover an additional device. In this test, each device registers 1 Ping, 1 WoC, 1 TCP-KA and 1 HrtBt.

The time to set up the filtering engine quickly rises with a larger number of covered devices (Fig. 3(a)); this latency is mostly due to the compilation of filter (text) strings into optimized BPFs. The delay is unacceptable when the number of devices and/or routines increases. With the lightweight version, the latency drops to a few milliseconds, because the filter strings are very simple (so it is not worth showing the graph). However, the CPU usage increases, as discussed in Section IV-A3.

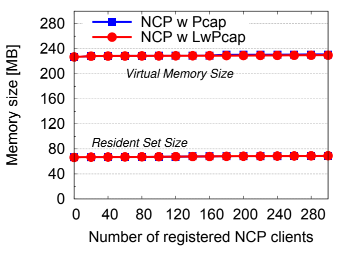
Fig. 3(b) shows the total time to start the routines; it includes traffic diversion (i.e., sending the Gratuitous ARP) and other preliminary operations. For example, the NCP immediately carries out a DHCP rebind (if the DHCP routine is registered), sends TCP Acks to infer the current sequence numbers (if TCP KeepAlive routines are registered), and sends HeartBeat messages (if HeartBeat routines are registered). Usually, the time taken to start the routines is within a few milliseconds and does not depend on the number of covered devices. However, Fig. 3(b) shows higher values because the experiments were conducted in a synthetic scenario, where we did not emulate remote peers; in this situation, some routines wait for given timeouts, because no response is received to their packets (this is the case for TCP KeepAlive).

3) Filtering: Performance of the filtering engine is evaluated by measuring latency and packet loss for a PING routine, while flooding the NCP with other ignorable traffic (which we consider as “noise” for NCP operation). We analyze how the buffer allocated to the Pcap filter affects the performance. As a matter of fact, a small buffer steals less memory, but gets filled earlier and packets may be dropped; a larger buffer consumes more memory, but is less prone to losses.

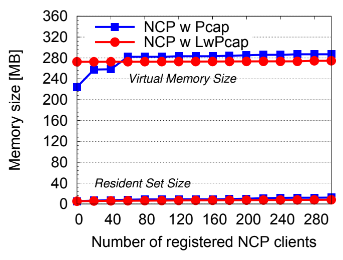
Fig. 4 shows that the packet loss is higher for the lightweight version of our implementation, apart for the Lantiq device. However, the latter is an experimental board, which might need some additional enhancements and tuning from the vendor. In general, the results confirm our expectations: lightweight mode demands more processing to our code, which is not as optimized for filtering as the kernel built-in engine. Latency is usually smaller for the



(a) Lantiq

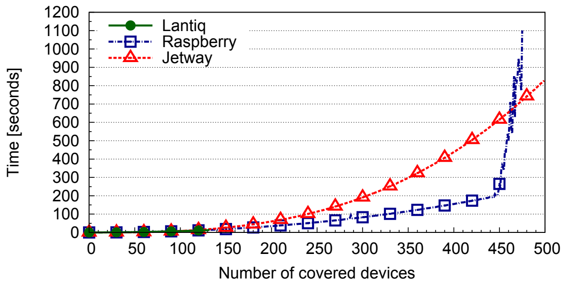


(b) Raspberry

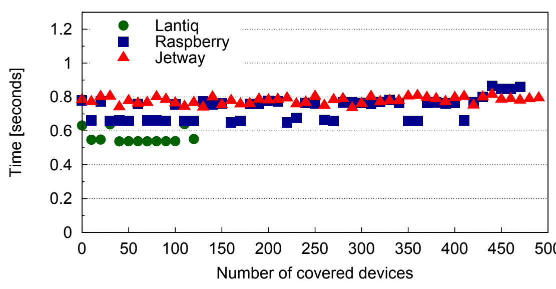


(c) Jetway

Figure 2. Memory usage vs number of registered client devices, with different filtering techniques. Virtual Memory Size is the total address space of the application; Resident Set Size is the amount of memory currently allocated in RAM.

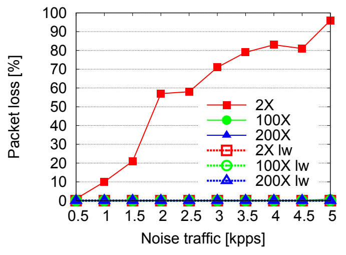


(a) Setup of the filtering engine

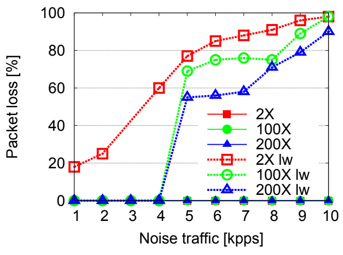


(b) Activation of routines

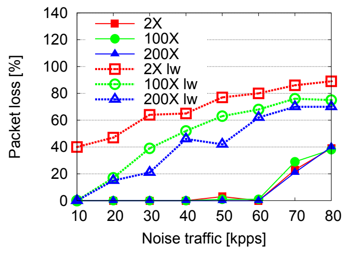
Figure 3. Time to start to cover an additional device, for a different number of already covered devices. Every device registers 1 Ping, 1 WoC, 1 TCP KeepAlive and 1 Heartbeating. Data are missing when the platform was not able to cover for that number of devices.



(a) Lantiq



(b) Raspberry

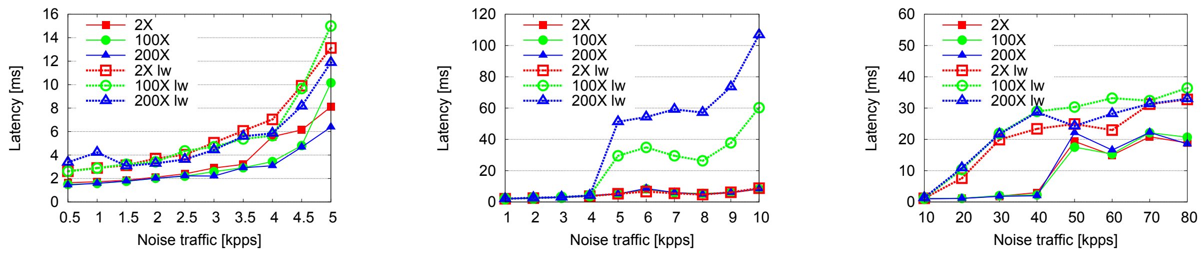


(c) Jetway

Figure 4. Packet loss for ICMP echo-requests, with noise traffic of the same kind. Lw stands for lightweight mode. The buffer size is 2, 100 and 200 times the base size of 65536 bytes.

lightweight version (see Fig. 5); indeed, when more packets are dropped, more CPU time is devoted to other packets, which are thus served quicker. Finally, all figures show that there is not a linear relationship between performance and buffer size for Pcap.

Quite oddly, the latency for Raspberry follows a “wave” shape, although the amplitude is different for different buffer sizes. In our understanding, this trend is due to the different slopes of the packet loss curves, and related to some complex effects that occur in the internal



(a) Lantiq (b) Raspberry (c) Jetway

Figure 5. Average latency for ICMP echo-requests, with noise traffic of the same kind. Lw stands for lightweight mode. The buffer size is 2, 100 and 200 times the base size of 65536 bytes.

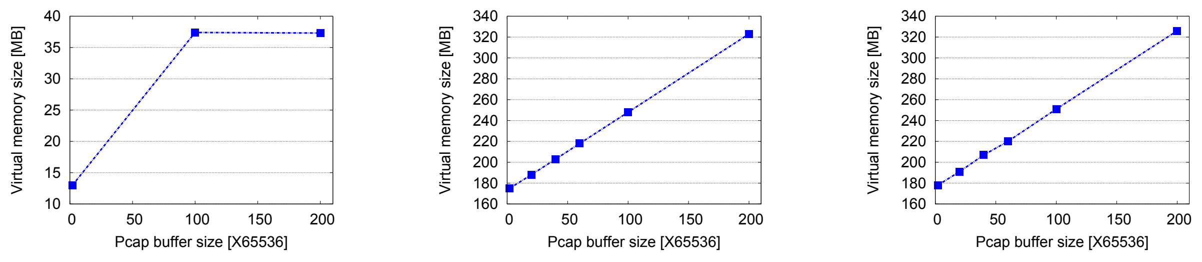
Pcap buffer.

It is worth viewing the corresponding CPU and memory usage as well. Memory increases linearly with the size of the Pcap buffer in all cases but the Lantiq board (probably there is a different memory allocation strategy in the kernel of this device), as shown in Fig. 6. We note the virtual memory size is very similar for Raspberry and Jetway, according to the results shown in Fig. 2. The CPU usage in general confirms the expectations (see Fig. 7): more processing is needed with the lightweight implementation, due to less optimization in filtering[9]. However, the Lantiq board has an opposite behavior; we believe this is due to the particular kernel used, which lacks many features with respect to other devices. We also show for comparison the CPU taken by the kernel to process the traffic when the NCP is not present (all packets are discarded in this case).

[9]The CPU load raises over 100% for the Jetway because the processor has four cores, and we plot the sum of the utilization for all of them (thus the maximum utilization would be 400%).

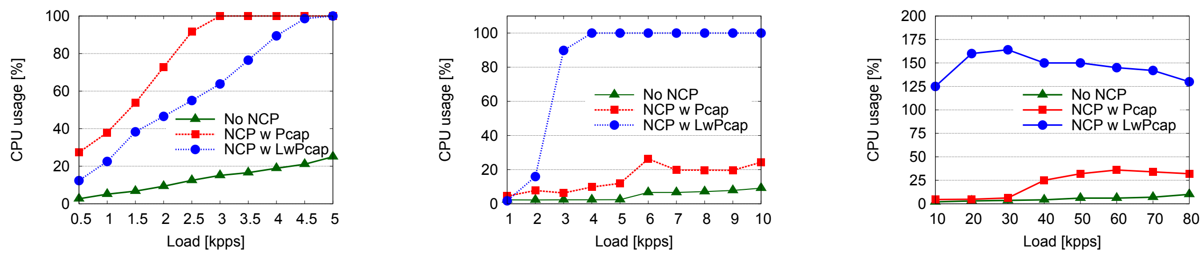
4) *Processing:* The last aspect under evaluation is processing of an increasing number of packets. To conduct this experiment, we generate Heartbeat messages, which are expected to be sent every few minutes or even less, thus leading to far more load than other rules.

We consider unsolicited heartbeats to exclude the contribution of filtering to CPU usage. Every client registers 10 HrtBt routines, and each routine schedules one packet per second; this load is rather unrealistic, but is necessary to generate a high processing load. We measure the interarrival time between consecutive heartbeat messages; ideally it should equal 1 second (called the 'Reference interval'), but it becomes longer when the processing power is not enough for the generation rate of all devices/routines. We report the mean interarrival times and their standard deviation averaged over at least 300 measurements in Fig. 8, together with the corresponding CPU load. The generation becomes unstable (larger



(a) Lantiq (b) Raspberry (c) Jetway

Figure 6. Virtual memory size allocated to the NCP process, with different buffer sizes for the Pcap filter. Values are taken with no clients registered.



(a) Lantiq (b) Raspberry (c) Jetway

Figure 7. CPU usage while varying the traffic load. The curve for 'no NCP' means the device is receiving the traffic but the NCP service is not runnig; the other curves depict the usage when the NCP is running with the basic (w Pcap) and lightweight (w LwPcap) filtering implementation.

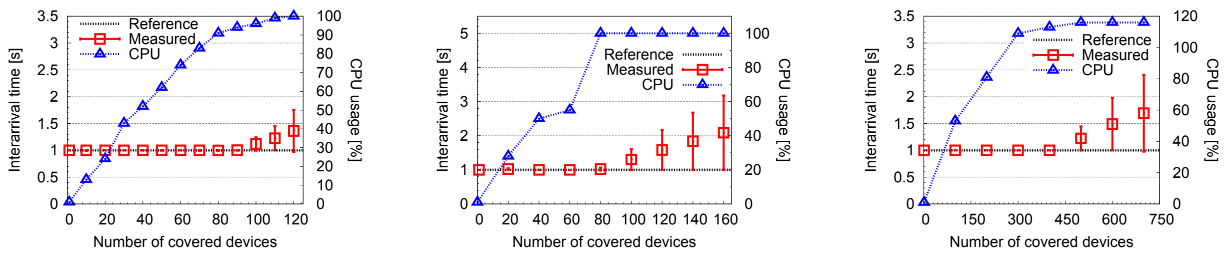
variance) when the CPU comes close to saturation and the scheduled heartbeats cannot be sent in time. Note that the multi-core and hyper-threading architecture of the Atom N550 CPU is only partially exploited, because in the current implementation there is a single task serving all scheduled events. With a larger number of concurrent tasks, we expect the number of covered devices to increase almost fourfold.

B. *Efficiency of the external NCP*

Table III reports the maximum number of devices that can be covered for simultaneously by our external NCP implementation, for each of the three hardware platforms that we deployed in our experimental testbed and with different filtering techniques. We computed such values by setting reasonable performance thresholds for NCP operation, starting from the results shown in Section IV-A.

Table III
MAXIMUM NUMBER OF DEVICES THAT CAN BE COVERED BY THE NCP, ACCORDING TO DIFFERENT IMPLEMENTATION CONSTRAINTS.

Factor	Lantiq		Raspberry		Jetway	
	Pcap	LwPcap	Pcap	LwPcap	Pcap	LwPcap
Filter setup latency	25	–	27	–	41	–
Physical memory	120	140	8560	12430	5920	186850
High network traffic	150	225	114	200	5033	2564
High computation load	90	90	80	80	400	400
Max number of clients	25	90	27	80	41	400



(a) Lantiq (b) Raspberry (c) Jetway

Figure 8. Interarrival times of heartbeat packets, compared to the reference interval at which they are generated. The graphs also show the CPU usage for the generation of the heartbeat packets.

The *Filtering engine setup latency* brings very hard constraints in case of the standard Pcap method (see Fig. 3); however, there are no constraints for the 'lightweight' version. The values in Table III were derived by imposing the filter setup latency to be less than 500 ms.

The *Physical memory* of the hardware platform is a constraint for the number of routines than can be registered and can be active simultaneously (see Fig. 2). We estimated the number of devices under the hypothesis that each client device registers each type of behavioral routine only once.

High network traffic causes packet loss and latency (see Figs. 4–5). We set our performance threshold at the traffic load corresponding to full CPU utilization (see Fig. 7); then, the number of clients was estimated by assuming 20 packets each[10].

CPU utilization was also used to estimate the constraints by *High computation load* (see Fig. 8); in this case there is a straight relation with the number of covered devices.

[10]This value is quite large in our opinion; we are anyway pursuing a worst-case analysis.

Given the results shown in Table III, one may argue the 400 devices that can be covered by the Jetway platform are usually not found in a single LAN. Indeed, such platform has several Ethernet interfaces and can cover for devices in different subnetworks; hence, it is a proper solution for medium-sized networks.

C. Comparison among different implementations

We consider the connectivity proxies listed in Table I, together with their target hardware platforms listed in Table II, and compare their efficiency in Table IV. Unfortunately, precise data are not available for all of them. In particular, commercial products often do not disclose design information, and no performance have been measured about proxying. On the other hand, research prototypes have often been conceived to demonstrate specific issues (such as fast packet classification for SmartNIC), and do not consider the full range of meaningful performance metrics for proxying.

The additional energy consumption may be the most controversial parameter to compute. For internal NICs, we take the power consumption of a commercial NIC with WoL capability[11] as baseline, and consider the remaining power as fully dedicated to run the connectivity proxy. SmartNIC does not implement all Ethernet functions as other NICs, so we take its power consumption as fully additional; further, we believe the power consumption for this device is a bit optimistic, because it was computed by simulation and only accounts for packet classification (thus excluding all other packet handling functions). Network devices would be anyway active even if the connectivity proxy were not run; for this reason, we compute the power drawn by the connectivity proxy as the difference between the ”processing” and ”idle” values shown in Table II. For other external devices, we take their entire processing consumption; for Apple TV only, we consider the standby power, as Apple states the Bonjour proxy runs in that state as well.

Though many data about performance metrics are missing, we could collect enough information to compare the efficiency of such implementations (i.e., the additional power consumption and the maximum number of covered devices). Fig. 9 shows the additional power per client consumed by the every implementation listed in Table I. The figure zooms in on the most relevant part of the graph, cropping out the low-efficient parts (where some implementations draw an excessive amount of power). To carry out a uniform comparison over a wide range of covered clients, we envisage the usage of multiple proxy instances to cover for more devices than those supported by each single platform[12]. This assumption leads to the sawtooth profile for some curves; each rising edge corresponds to the insertion of an additional instance.

As Fig. 9 clearly shows, external implementations usually bring better efficiency when the proxy covers for multiple clients. External proxies are more effective when they run on low-power devices, since they provide the best efficiency even when using multiple instances; further, running the connectivity proxy on network equipment proves to be more efficient than on NICs just for few covered devices. Implementations on high-end servers look less efficient, however, they are able to carry out very complex routines, thus allowing covered

[11]We chose the Intel 82583V GbE Controller, since data reported for internal proxies usually only consider the power drawn by controllers, and not the whole card. The 82583V datasheet reports 167 mW for operating the card in D3 cold with WOL at 10 Mb/s. We rounded such figure to 170 mW.

[12]The usage of multiple proxy instances in useful in practice to cover for devices in different subnets. For hardware platforms that support a large number of clients, we assume the proxy has multiple network interfaces, connected to every different subnetwork. Our NCP implementation supports this architecture.

Table IV
COMPARISON AMONG PERFORMANCE OF DIFFERENT CONNECTIVITY PROXIES.

	Offloading	Somniloquy	SmartNIC	NCP		Bonjour Sleep Proxy	SleepServer
				Pcap	LwPcap		
Hardware platform	Intel i350	Gumstix	Rice NIC	Lantiq, Raspberry, Jetway		Apple TV/AirPort	Dell PE2050
Additional power consumption	533 mW	992 mW	180 mW	400 mW/3.8 W/27.3 W		700 mW/2 W	308 W
Memory per client	∼ 3 KB	64 MB	–	∼ 60 KB	∼ 7 KB	–	64 MB
Registration latency	–	–	–	< 200 ms	< 200 ms	–	120 s (+/- 10)
Activation latency	–	–	–	1 s*	700 ms*	–	11 s (+/- 1)
Response time	< 60 s	<2 s‡	< 50 ms	< 100 ms	< 100 ms	–	–
Max number of clients	1	1	1	25–41†	80-400†	20	200

* Worst case analysis; typical values are 500 ms less (see Section IV-A).
‡ Response time is large because this implementation does not buffer packets that trigger wake-up.
† Exact figures depend on the hardware platform (see Table III)

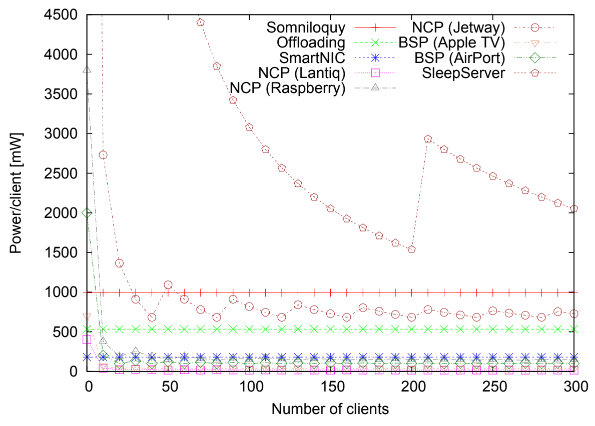


Figure 9. Efficiency of different connectivity proxy implementations, by considering their target hardware platforms.

devices to potentially sleep for longer periods of time; this aspect will be considered in the following.

V. ESTIMATING THE POTENTIAL FOR ENERGY SAVING

Idle devices can sleep while the connectivity proxy maintains their network presence. Hence, there is a huge potential for saving energy, given by the much lower power drawn in the sleep state with respect to running idle. Obviously, this approach is only meaningful for devices with IP connectivity, which anyway account for two-thirds of electronics energy usage [24].

As a preliminary rough evaluation of the potential for energy saving, we suppose to be able to put devices in sleep mode for all their idle periods. There are several surveys that report estimations of the time spent in different power states for various networked devices in homes and offices [2], [24]–[26]; we take into consideration computers (desktops, laptops, all-in-ones), set-top boxes, media streaming devices, Network-Attached Storages (NAS), VoIP phones, imaging devices (printers, scanners, copiers). The results of our analysis are shown in Fig. 10[13]

It is worth noting that our rough estimation is indeed a *ultimate* bound. In fact, energy saving

[13]We set the power consumption for sleep mode to 1W for all devices but computers, according to the energy efficient target of the IEA (*1W initiative*); such target has been demonstrated feasible by recent studies [27]. For computers, we consider the current power consumption for S3 state.

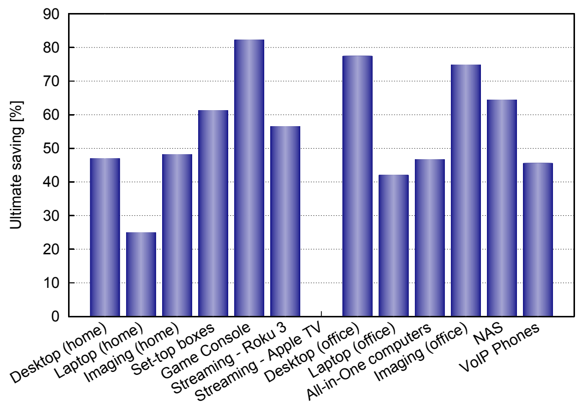


Figure 10. Ultimate potential for energy savings allowed by proxying network connectivity for a wide range of consumer electronics. Estimations consider to replace all idle periods with low-power sleeping.

is computed under the hypothesis of removing all idle periods; however, the usage of some devices intrinsically envisages idle periods (for instance, when computer users read something on the monitor).

Fig. 10 shows a large potential for energy savings. Apple TV is the only device without potential for energy saving, because it already has a very efficient sleep mode (700mW) and automatically sleeps when idle. In monetary terms, with the current electricity prices, typical households can save more than 50$ in US and 80€ in Europe, a small office with 10 employees can save more than 130$ or 200€, and a medium office with 30 employees can save more than 400$/600€[14].

[14]We made these estimations by considering data from previous surveys about density of consumer electronics in homes and offices [5], and by reasonable assumptions for missing data.

To provide a more realistic estimation, we need to consider three additional factors. First, usage patterns of devices, which indicate the percentage of idle periods in which the device is really unused. Second, the effectiveness of the connectivity proxy, which is its ability to carry out background activity on behalf of covered devices without having to wake them up. Third, the power consumption of the proxy itself, which consumes a non-negligible amount of energy even when it is not covering for devices. The rest of this Section considers all these factors and derives more precise energy saving estimations for computers, which are the only kind of devices with more detailed analyses and data available.

A. *Usage patterns*

Though there are many studies that point out the power consumption of computers and other electronic devices, just a few of them break such data down and provide usage patterns. These patterns are quite different for home and office environments: at home, users are more prone to shut down their computers (perhaps because they only use these devices occasionally), whereas, at office, computers are often left on the whole day.

We reviewed several surveys about computer usage patterns [2], [5], [25], [26], [28], [29], and derived a concise summary for the above environments. Table V shows the av-

Table V
USAGE PATTERNS FOR HOME AND OFFICE ENVIRONMENTS.

		In-use		Idle		Sleep		Off	
		Power	%	Power	%	Power	%	Power	%
Home	Desktops	70 W	14.3	56 W	17.6	3.4 W	23.8	1.6 W	44.3
	Laptops	30 W	13.5	17 W	9.5	1.6 W	29	1.1 W	48
Office	Desktops	80 W	10.8	60 W	59.1	2.5 W	3.1	1.5 W	27
	Laptops	30 W	10.8	17 W	18.5	1.6 W	13.3	1.1 W	57.4

erage power consumption for each state (first column) and the average percentage of time spent in that state (second column).

The 'in-use' state includes very heterogeneous activities (e.g., compiling code, browsing the web, downloading files), so it typically leads to large variance with respect to the mean power consumption reported in the Table. The idle state is sometimes broken into short-idle and long-idle; the first usually corresponds to user activity that does not entail computation (e.g., reading documents or emails, input by keyboard), whereas the second is often symptom that the user is no more attending the computer. Long-idle periods could be removed by proxying network connectivity.

B. Effectiveness of proxying network connectivity

The effectiveness of proxying network connectivity relates to the set of routines provided by the proxy, i.e., its capability to deal with background tasks on behalf of clients, without having to wake them up. In the ideal case, devices should not waste energy in the idle state (this is the ultimate bound already estimated in Fig. 10); in practice, some short-idle periods are intrinsic in computer usage.

We define as '*effectiveness*' the percentage of additional time that devices could sleep when they are covered by a connectivity proxy. Table VI shows estimations of effectiveness taken by past works that have analyzed real data about computer usage and network activities [11]. Such analysis compares the following hypothetical proxying behaviors[15] *i) Wake-up*: the proxy discards ignorable traffic and wakes the hosts up for any other packet; *ii) Full availability*: the proxy only issues simple mechanical responses and wakes the hosts up for any other packet; *iii) Selective availability*: the proxy behaves as in the previous case, but wakes the hosts up only for specific traffic; and *iv) Network presence*: in addition to the previous behavior, the proxy also schedules periodic operations. The values shown in Table VI were computed by considering the device idle after 15 minutes of mouse/keyboard inactivity, and by assuming 10 seconds as the total time for a device to wake up, process a packet and go to sleep again. Further details about the network traces and the outcomes from the analysis are available in the original work by Nedevschi et al. [11].

[15]We changed the original terminology used in the referenced work (which simply numbers the different options) to better recall the actual behavior of the connectivity proxy.

Table VI
ESTIMATED SLEEP DURATION (AS PERCENTAGE OF IDLE TIME) UNDER DIFFERENT PROXY BEHAVIORS FOR HOME AND OFFICE ENVIRONMENTS.

	Home	Office
Wake-up	57%	19%
Full availability	78%	48%
Selective availability	99%	92%
Network presence	97%	88%

C. Potential energy savings

The total energy E_T consumed in the reference period T by a single device is given by:

$$E_T = P_A T_A + P_I T_I + P_S T_S + P_O T_O \quad (1)$$

where P_i and T_i are the power and the time spent in the i state (i=Active/in use, Idle, Sleep, Off), respectively.

The operation of the connectivity proxy allows reducing the idle period by T_g (where g stands for 'gain') and moving the same amount of time to the sleep period:

$$T'_I = T_I - T_g \quad T'_S = T_S + T_g$$

The time gain T_g can be expressed in terms of the original idle period:

$$T_g = \alpha T_I$$

where α is the "effectiveness coefficient", whose estimated values are given in Table VI for different operational patterns and environments.

The power budget must account for the energy drawn by the connectivity proxy (E_{CP}), although such factor is shared by the number of covered devices (N). Hence, the total energy per device becomes:

$$E'_T = P_A T_A + P_I T'_I + P_S T'_S + P_O T_O + E_{CP}/N \quad (2)$$

and the potential energy saving per device (E_{sav}) is given by:

$$E_{sav} = E_T - E'_T = (P_I - P_S)\alpha T_I - E_{CP}/N \quad (3)$$

We believe that the relative percentage of energy saving ($\%E_{sav}$) for each device is a better indicator for our purposes. In fact, it allows comparing the saving for devices that draw different amount of power (e.g., desktops and laptops) in a uniform way, and it also provides an immediate estimate of money saving for the user[16]. Hence, we consider:

$$\%E_{sav} = \frac{E_{sav}}{E_T} \times 100 = \\ = \frac{(P_I - P_S)\alpha T_I - E_{CP}/N}{P_A T_A + P_I T_I + P_S T_S + P_O T_O} \times 100 \quad (4)$$

Figs. 11 and 12 show the potential savings for the different implementations and their relative target platforms; they also show the 'ultimate' bound for comparison, which is the theoretical saving achievable if all the idle periods were removed (as shown in Fig. 10).

[16]Indeed, the percentage of energy saving is the same percentage as cost saving.

We show both the percent energy saving per device (on the left vertical axis) and the annual energy saving per device (on the right vertical axis). The monetary cost saving per device could be computed directly by multiplying the right vertical axis scale by the average cost of electricity (currently, about 0.2€ in Europe and 0.12$ in North America).

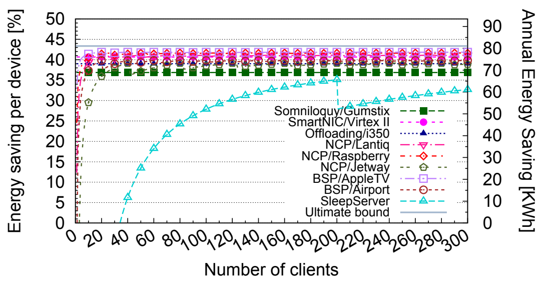
Our analysis targets a worst-case scenario, i.e., the proxy consumes the highest amount of energy (see Table II); hence we do not take into account the *additional* energy consumption reported in Table IV. Further, we consider the Pcap version of our NCP implementation. Discontinuities in the graphs correspond to the need for an additional proxy instance to cover for that number of devices.

Embedding the connectivity proxy on NICs provides the best saving with very few devices; however, running the proxy on low-power platforms provides better performance with just a small number of clients. If we considered the *additional* power consumption alone, we would even get better results. As expected, offices have the larger potential for saving, due to the different usage pattern (as shown in Table V).

Interestingly, providing full support for applications by running virtual images (as done with SleepServer) is far less efficient than all other architectural solutions; many devices must be covered to reach a positive energy balance (i.e., energy saving greater than zero). However, the main potential of SleepServer is the underlying concept of moving applications to the cloud, allowing 'thin' clients to draw very little energy and complex computing infrastructures to run millions of applications in a very effective and efficient way [24].

By taking into account data about the installed base in the US [5], we can roughly estimate a potential saving per year of about 6.6 TWh and 1.1 TWh in homes for desktops and laptops, respectively. At the current average cost of electricity, this corresponds to a total saving of 925 millions of dollars for the US alone. For US offices [24], we can roughly estimate a potential saving per year of about 32.2 TWh and 2.3 TWh for desktops and laptops, respectively, which corresponds to a total saving of about 4 billions of dollars.

Figs. 13–14 show how the potential for energy saving changes for the different kinds of proxy behaviors (see Section V-B). In this case, we limit the maximum number of devices, in order to provide better focus on the relevant part of the graphs. The real advantage comes from waking up the hosts only when it is really necessary, and not for all incoming traffic. In other words, it is worth giving client devices the possibility of dynamically choosing their proxying behavior. This motivates the need for a configuration interface. In addition, waking up also for periodic operations only results in negligible performance degradation.


(a) Home

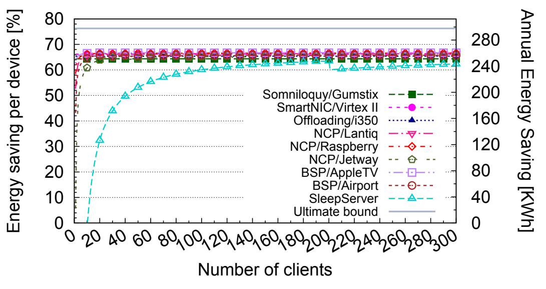
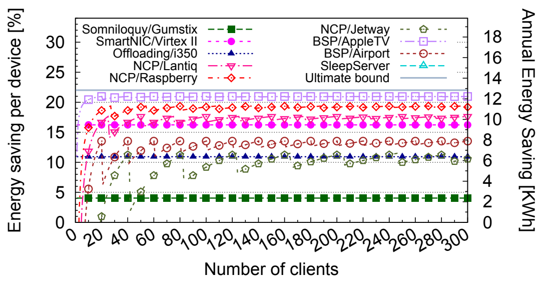

(b) Office

Figure 11. Energy saving for typical desktop computers by running different connectivity proxies on their target hardware platforms.


(a) Home

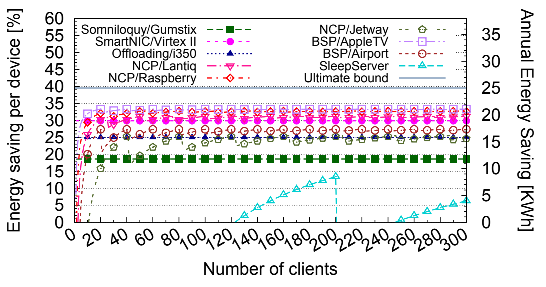
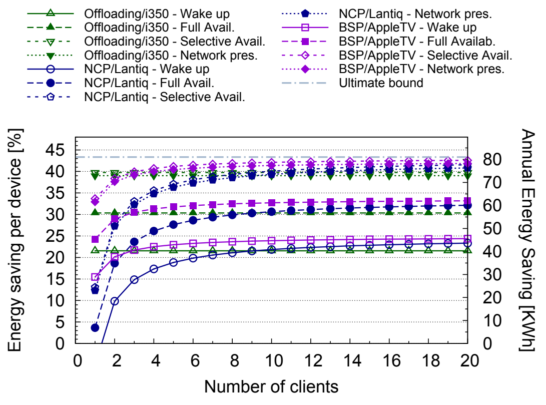

(b) Office

Figure 12. Energy saving for typical laptop computers by running different connectivity proxies on their target hardware platforms.


(a) Home

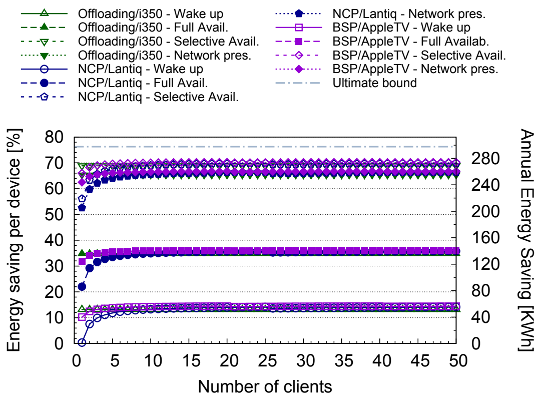
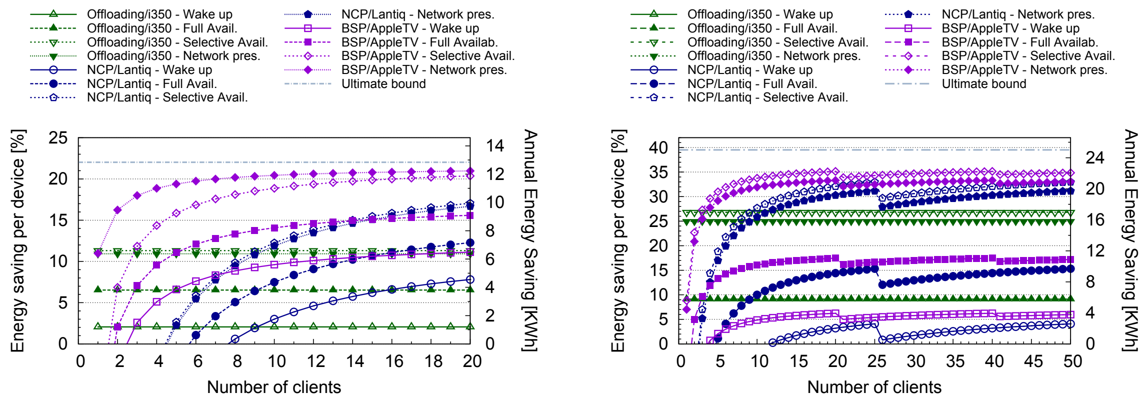

(b) Office

Figure 13. Energy saving for typical desktop computers by providing different proxying behaviors.

By comparing the NCP/Lantiq and BSP/AppleTV curves in Figs. 13–14, we conclude that it is more convenient to cut down the hardware's power consumption rather than covering for a larger number of devices. However, we cannot conclude that a standalone low-power device is a better solution than networking equipment, because



(a) Home (b) Office

Figure 14. Energy saving for typical laptop computers by providing different proxying behaviors.

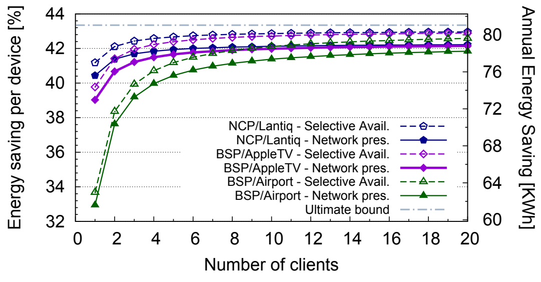
networking equipment mainly implements other functions. As a matter of fact, if we consider the *additional* energy consumption of the connectivity proxy as computed in Table IV, we clearly see that running the connectivity proxy on networking equipment may be the most efficient solution. Figs. 15–16 show such comparison for the most interesting implementations in this context: the NCP on the Home Gateway (Lantiq) and the BSP on both the AppleTV and Airport.

Looking again at the graphs in Figs. 13–14, we note that sometimes the energy saving is negative (i.e., running the proxy consumes more energy than saved by putting devices to sleep). Table VII summarizes the minimum number of covered clients to achieve a positive energy balance; this gives a clear indication of which platform may be used for specific conditions and environments. It is worth noting that some solutions never save energy (those combinations where a hyphen is present instead of a figure), as they would require a number of clients larger than what they could effectively cover for simultaneously.

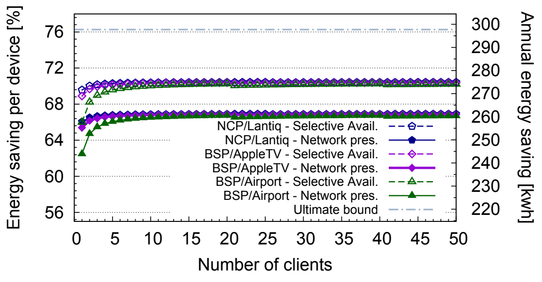
VI. Conclusions

This paper provides an extensive and detailed analysis about the effectiveness of proxying network connectivity to enable idle devices to sleep. Our work has analyzed the full spectrum of deployment solutions, bringing together available commercial products, previous architectures, and prototypes with novel implementations that fill the gap in this field.

The main outcome from our work is the clear and realistic assessment of the potential for energy saving. We have taken into account the influence of different hardware platforms for proxying network connectivity; furthermore, we have brought together a number of surveys

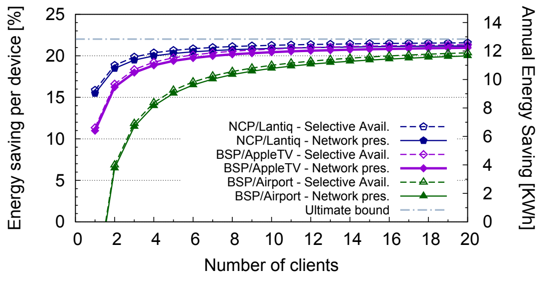


(a) Home

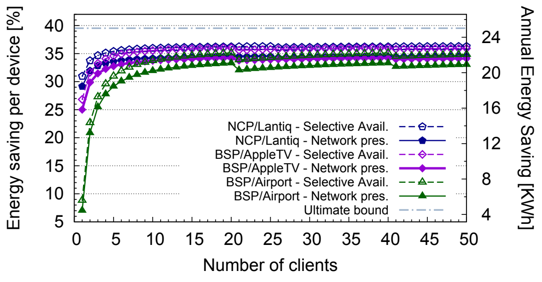


(b) Office

Figure 15. Energy saving for typical desktop computers by considering the additional energy to run the proxy.



(a) Home



(b) Office

Figure 16. Energy saving for typical laptop computers by considering the additional energy to run the proxy.

Table VII
MINIMUM NUMBER OF CLIENT DEVICES REQUIRED TO ACHIEVE POSITIVE ENERGY SAVINGS.

	Wake up		Full availability		Selective Availability		Network presence	
	Home	Office	Home	Office	Home	Office	Home	Office
Desktops								
NCP/Lantiq	2	1	1	1	1	1	1	1
NCP/Raspberry	1	1	1	1	1	1	1	1
NCP/Jetway	6	5	5	2	3	1	4	1
Bonjour Sleep Proxy/Apple TV	1	1	1	1	1	1	1	1
Bonjour Sleep Proxy/Airport	2	2	2	1	2	1	2	1
SleepServer/PE2050	59	47	43	19	34	10	35	11
Laptops								
NCP/Lantiq	8	12	6	5	5	3	5	3
NCP/Raspberry	5	7	4	3	3	2	3	2
NCP/Jetway	33	–	24	21	19	11	20	11
Bonjour Sleep Proxy/Apple TV	1	2	1	1	1	1	1	1
Bonjour Sleep Proxy/Airport	13	–	10	8	8	5	8	5
SleepServer/PE2050	–	–	–	–	–	117	–	123

and data to estimate the energy saving in realistic scenarios.

Our work has pointed out some aspects that have often been neglected by past works in this field; in particular, how the hardware platform affects both the effectiveness and the efficiency of the connectivity proxy. We have investigated the trade-off between implementation constraints and energy saving, showing which architectural solutions best fit specific environments.

Future work on this topic should include additional aspects not covered so far, like function virtualization in the cloud, an emerging aspects that could shift most computation in energy efficient infrastructures, allowing 'thin' clients to consume much less power than today's terminals.

REFERENCES

[1] B. Nordman, S. Lanzisera, and R. E. Brown, "Energy efficient digital networks," Lawrence Berkeley National Laboratory, Tech. Rep., January 2013, chapter 3: Network Presence Proxy.

[2] K. Roth, B. Urban, V. Shmakova, and B. Lim, "Residential consumer electronics energy consumption in 2013," in *2014 ACEEE Summer Study on Energy Efficiency in Buildings*, Pacific Grove, California, USA, Aug. 17-22, 2014, pp. 9–308–9–320.

[3] W. V. Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, "Trends in worldwide ICT electricity consumption from 2007 to 2012," *Computer Communications*, vol. 50, no. 1, pp. 64–76, September 2014.

[4] "Gesi smarter2020: The role of ict in driving a sustainable future," Global e-Sustainability Initiative, December 2012. [Online]. Available: http://gesi.org/portfolio/report/72#

[5] B. Urban, V. Shmakova, B. Lim, and K. Roth, "Energy consumption of consumer electronics in u.s. homes in 2013," Fraunhofer USA, Tech. Rep., June 2014. [Online]. Available: http://www.ce.org/CorporateSite/media/environment/Energy-Consumption-of-Consumer-Electronics.pdf

[6] "ProxZZZy for sleeping hosts," Standard ECMA-393, June 2012. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-393.pdf

[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*, vol. 15, no. 5, pp. 297–310, Sep.-Oct. 2005.

[8] M. Jimeno, K. Christensen, and B. Nordman, "A network connection proxy to enable hosts to sleep and save energy," in *Proceedings of the IEEE International Performance Computing and Communications Conference (IPCCC 2008)*, Austin, Texas, USA, Dec. 7–9, 2008, pp. 101–110.

[9] J. Klamra, M. Olsson, K. Christensen, and B. Nordman, "Design and implementation of a power management proxy for Universal Plug and Play," in *Proceedings of the Swedish National Computer Networking Workshop (SNCNW 2005)*, Halmstad, Sweden, Nov. 23–24, 2005, pp. 23–24.

[10] R. Bolla, R. Bruschi, M. Giribaldi, R. Khan, and M. Repetto, "Smart proxying for reducing network energy consumption," in *Proceedings of the 2012 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS2012)*, Genoa, Italy, Jul. 8–11, 2012.

[11] S. Nedevschi, J. Chandrashekar, J. Liu, B. Nordman, S. Ratnasamy, and N. Taft, "Skilled in the art of being idle: reducing energy waste in networked systems," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09)*, Boston, Massachusetts, USA, Apr. 22–24, 2009, pp. 381–394.

[12] K. Christensen, P. Gunaratne, B. Nordman, and A. George, "The next frontier for communications networks: Power management," *Computer Communications*, vol. 27, no. 18, pp. 1758–1770, December 2004.

[13] M. Jimeno and K. Christensen, "A prototype power management proxy for Gnutella peer-to-peer file sharing," in *Proceedings of the IEEE Conference on Local Computer Networks*, Dublin, Ireland, Oct.15–18, 2007, pp. 210–212.

[14] P. Werstein and W. Vossen, "A low-power proxy to allow unattended Jabber clients to sleep," in *Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, (PDCAT 2008)*, Dunedin,

New Zealand, Dec. 1–4, 2008, pp. 390–395.

[15] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, V. Bahl, and R. Gupta, "Somniloquy: Augmenting network interfaces to reduce PC energy usage," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09)*, Boston, Massachusetts, USA, Apr.22–24, 2009, pp. 365–380.

[16] W.-K. Park, C. Choi, I. Lee, and J. Jang, "Energy efficient multi-function home gateway in always-on home environment," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 1, pp. 106–111, February 2010.

[17] K. Christensen and F. Gulledge, "Enabling power management for network-attached computers," *International Journal of Network Management*, vol. 8, no. 2, pp. 120–130, March-April 1998.

[18] Y. Agarwal, S. Savage, and R. Gupta, "SleepServer: a software-only approach for reducing the energy consumption of PCs within enterprise environments," in *Proceedings of the USENIX annual technical conference (USENIX ATC'10)*, Boston, Massachusetts, USA, Jun. 23–25, 2010.

[19] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Network connectivity proxy: Architecture, implementation and performance analysis," *IEEE Systems Journal*, vol. PP, no. 99, pp. 1–12, June 2015, pre-print.

[20] K. Sabhanatarajan, A. Gordon-Ross, M. Oden, M. Navada, and A. George, "Smart-NICs: Power proxying for reduced power consumption in network edge devices," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI '08)*, Montpellier, France, Apr. 7–9, 2008, pp. 75–80.

[21] D. C. Plummer, "An ethernet address resolution protocol – or – converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware," RFC 826, November 1982. [Online]. Available: http://tools.ietf.org/html/rfc826

[22] S. Cheshire, "Ipv4 address conflict detection," RFC 5227, July 2008. [Online]. Available: http://tools.ietf.org/html/rfc5227

[23] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Design and implementation of cooperative network connectivity proxy using Universal Plug and Play," in *The Future Internet Future Internet Assembly 2013: Validated Results and New Horizons*, ser. Lecture Notes in Computer Science, A. Galis and A. Gavras, Eds. Springer Open, May 2013, vol. 7858, pp. 323–335.

[24] M. Malinowski, J. Clinger, R. Unger, B. Nordman, and K. Kaplan, "Light at the end of the tunnel for electronics energy use?" in *2014 ACEEE Summer Study on Energy Efficiency in Buildings*, Pacific Grove, California, USA, Aug. 17-22, 2014, pp. 9–262–9–273.

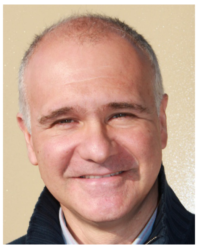
[25] J. A. Roberson, C. A. Webber, M. C. McWhinney, R. E. Brown, M. J. Pinckard, and J. F. Busch, "After-hours power status of office equipment and inventory of miscellaneous plug- load equipment," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-53729, January 2004. [Online]. Available: http://enduse.lbl.gov/info/LBNL-53729.pdf

[26] K. Kawamoto, Y. Shimoda, and M. Mizuno, "Energy saving potential of office equipment power management," *Energy and Buildings*, vol. 36, no. 9, pp. 915–923, September 2004.

[27] "Benchmarking of the standby power performance of domestic appliances," report by the Energy Efficient End-Use Equipment (4E) - International Energy Agency, July 2012.

[28] B. Nordman and C. Christensen, "Greener PCs for the enterprise," *IT Professional*, vol. 11, no. 4, pp. 28–37, July-Aug 2009.

[29] M. A. Piette, M. Cramer, J. H. Eto, and J. G. Koomey, "Office technology energy use and savings potential in new york," Lawrence Berkeley National Laboratory, Tech. Rep. LBL-36752, January 1995. [Online]. Available: http://emp.lbl.gov/sites/all/files/lbnl-36752.pdf



R. Bolla (S'89-M'95) received the Ph.D. degree in Telecommunications in 1994 from the University of Genoa. He is currently Full Professor at DITEN in the same University, where he is also leading the Telecommunication Networks and Telematics (TNT) laboratory. He has been Principal Investigator of many important research projects and contracts with both public institutions and private companies. He is also involved in many standardization activities in ITU-T, ETSI and IEEE (he is currently the rapporteur of a Working Item of the ETSI EE-TC). He has co-authored over 200 scientific publications in international journals and international conference proceedings. His current research interests are in: i) networking architectures, protocols and techniques for the smart energy grid; ii) mechanisms and techniques for energy consumption reduction in IP networks, iii) advance platforms for Future Internet nodes (Flexible Software Router) iv) Software Defined Network and Network Function Virtualization Based architectures, v) advance management of user mobility in packet networks.

Rafiullah Khan was born in Kohat, Pakistan, in 1987. He received his B.Sc degree in Electrical Engineering from NWFP University of Engineering and Technology, Pakistan, in 2009. He received his master degree in Satellite Navigation and Related Applications from Politecnico Di Torino, Italy, in 2010. He received his Ph.D. degree from University of Genoa, Italy, in 2015, under the European Commission's funded Erasmus Mundus program. He is currently a Postdoctoral Research Fellow with the Institute of Electronics, Communications and Information Technology, at Queen's University Belfast in United Kingdom. He has co-authored several papers in international conference proceedings. His Ph.D. research area was Green Networking, with particular focus on Energy-Aware Home Networking. His research interests include Ad Hoc Networking, Self-Organizing Networks and Green Networking.

Matteo Repetto received the Ph.D. degree in Electronics and Computer Science in 2004 from the University of Genoa. From 2004 to 2009 he was a postdoc at University of Genoa. Since 2010, he is a Research Associate at CNIT. He has been teaching many courses in telecommunication networks and network security. He has co-authored over 20 scientific publications in international journals and conference proceedings, and he has been involved in many different national and European research projects in the networking area. His current research interests include wireless networks, estimation of freeway vehicular traffic, pervasive communications and mobility management, energy-efficient networking.