A Client-Side Architecture to Support Energy Efficiency for Upcoming Networks

Raffaele Bolla,
Marco Chiappero
DITEN – University of Genoa, Italy
Email: raffaele.bolla@unige.it,
marco@reti.dist.unige.it

Maurizio Giribaldi
Infocom Srl
Genoa, Italy
Email: maurizio.giribaldi@infocomgenova.it

Matteo Repetto
National Inter-University Consortium
for Telecommunications (CNIT)
Research Unit of Genoa, Italy
Email: matteo.repetto@cnit.it

***Abstract*—Upcoming telecommunication networks are expected to lower energy requirements of current infrastructures, especially at the network edge. That necessarily entails cutting off energy wasted when devices are active, yet idle, just to maintain their presence on the network. To this purpose, the concept of delegating network activity has been introduced to allow devices to enter low power states without breaking their connectivity.**

This paper addresses the client-side architecture for delegating routine network tasks to the proper service. It depicts the logical framework for this purpose and identifies its main logical elements and their function; further, it discusses their integration with power management capability today available in most computing devices.

I. Introduction

Telecom operators report they are among the biggest nation-wide consumers of energy, due to their communication infrastructures [1]. Breakdown of energy consumption shows that today most of energy is drawn at the network edge, due to its capillarity and to the large number of installations [2]. The above estimations only consider network equipment, but the situation is even more alarming if one also takes into account ICT devices at home. The most contradictory aspect is perhaps the fact that most of the power drawn by such devices gets wasted when they are idle and no one is using them [3]: this habit is dramatically common for several reasons, including remote access to devices, nightly updates, user presence in multimedia applications (VoIP, Instant Messaging), keeping the priority in downloading queues (e.g., in file-sharing), quick availability to restore the working session, and wrong beliefs about power saving systems [3], [4]. At it's said, hosts maintain their presence on the network.

Effective and efficient management of peripheral equipment and users' devices is therefore essential to cut down energy waste; the whole network could be involved in this challenging process, as most of matters that hinder the usage of low-power states come from networking issues. One straightforward solution is the delegation of routine network activity to specific agents that consume very low power and hopefully cover for many devices simultaneously. This approach has the immediate advantage of making any other network element unaware of what is happening, hence it does not affect the way all Internet protocols and applications behave today.

There are several tasks a 'connected' host is expected to carry out: it must handle link-local protocols (like ARP, IGMP, DHCP, NetBIOS) and network management protocols (like ICMP and SNMP), it must answer to incoming packets (e.g., new TCP connection or sporadic UDP packets), and it should refresh protocols' and applications' soft states (usually referred to as keep-alive or heart-beating messages). In many cases, these are very simple operations that only need elementary or no data at all from the host in question. Therefore, managing these tasks needs very limited storage and processing capability and could be demanded to any kind of device, including Network Interface Cards (NICs) and network equipment [5]. Delegations of network activity to NICs offers best and easier integration within the device, whereas delegation to external equipment brings more opportunities and larger saving when many devices are covered. When complex routines are involved (as may happen for application heart-beating), NICs need general-purpose processors that might rise the power consumption up to the same level of other network equipment, though they cover a single device.

The shift of the legacy operational and management paradigms towards more in-network programmability and flexibility will impact the concepts described so far. The management of networking routines will become a network service, which could be implemented anywhere in the network (either in a centralized or distributed fashion) according to emerging paradigms like Software Defined Networking (SDN) and Network Function Virtualization (NFV). The service is thus likely to be moved inside the network in edge devices or in green data centers, thus taking advantage of high computation and storage resources and sharing them with many other processes, eventually enhancing their efficiency. That would provide the ability to perform networking tasks for ever more complex applications and protocols, and even to support novel processing paradigms where applications are run directly inside the network and 'thin' client devices act merely as rendering and interaction interfaces. In this perspective view, applications will always be available and connected, though the thin client will also have to cope with basic connectivity tasks when in low-power mode.

Nevertheless, irrespective of the vision about the evolution, client devices are expected to be compliant to the delegation paradigm, in order to fully take advantage of its capability and save energy by a most intensive resort to low-power states. Till now, several works have addressed requirements, challenges, functions and architectures for carrying on network routines on behalf of other devices; however, no effort has still been

978-0-9836283-9-2

undertaken on the client side.

This paper fills the gap by proposing a logical architecture to integrate the delegation of network routines with power management functions already available in off-the-shelf devices, and by giving preliminary guidelines for its implementation. The design of the client-side architecture explicitly targets our delegation service, named Network Connection Proxy (NCP). The choice is motivated by the facts that our NCP is currently the most complete implementation to this purpose, supports a general set of functions and offers a communication interface to control its behavior (registration of functions to activate, transferring of protocols' and applications' data, notification of power state changes).

The paper is organized as follows. Section II reviews the main requirements and architectures to delegate network activity, so to highlight capability and requirements for the client side. Section III provides a brief understanding of the Network Connection Proxy, by discussing its operation, the functions it currently supports and the communication interface. Section IV sketches the logical framework that enables devices and their applications to control the NCP behavior, and Section V presents the corresponding software architecture we are designing, together with preliminary details about its implementation. Finally, Section VI gives our conclusions and next steps of our work.

II. Related work

Fostering the usage of low-power states at the network edge implies the delegation of network activity to a proper agent or service. This problem has been studied for over a decade, bringing to a set of requirements, expectations, challenges and several implementations.

In a nutshell, maintaining network presence essentially includes three main issues [6]:

- link-layer presence, namely maintaining physical connectivity (synchronization, state, topology, etc.); this is quite easy for Ethernet but more complex in WLANs;
- network-layer presence, which is active participation in networking protocols to maintain end-to-end reachability;
- application-layer presence, which includes all messages and functions to let other peers believe specific applications are active and ready to answer.

The service may run as a network proxy (indeed, it is often referred to as Network Connection Proxy); it inspects all packets intended to covered devices and classifies them according to the treatment they need [7]:

- *minimal response required*, when the answer is easily predictable and no heavy computation is required;
- *delayed response required*, when packets can be buffered and processed later without affecting the remote application/protocol (IM messages and notifications, emails, etc.);
- *wake-up required*, if the involvement of an application or of the OS is unavoidable, such as with new TCP connection requests, SNMP requests, and so on; wake-up is mostly based on the Wake-on-LAN (WoL) feature already supported by most NICs.

Most effort has been devoted to management protocols like ARP, ICMP, IGMP [4], [7]–[9]; TCP has a complex state machine and initially only the advertising of the Zero-Window condition was envisaged, to delay packet delivery by the remote peer [8]; however, support of keep-alive messages and session migration was recently added [10]. There have been a few proposals for proxying both high-level protocols as UPnP [11], [12] and specific applications as Gnutella [13] and Jabber [14].

A further step was done in Somniloquy, where "stubs" for specific applications can be added to the base framework [4]; the implementation covered for *wget* (an application for web downloads), *bittorrent* (a well-known protocol for file sharing) and *finch* (an IM client that supports several protocols as MSN, AOL, ICQ). A more generalized solution was proposed for heart-beating, by defining a common and general framework that allows applications to specify a message template and dynamic fields to be filled in when the packet is sent [10].

The interface to control the behavior of the service and the internal architecture for client devices have been neglected for many years, sometimes arguing the service could have inferred by itself what covered devices had needed and when they had entered and exited low-power states. Recently, the usage of zero-discovery auto-configuration protocols like UPnP was proposed, together with a suitable abstraction of general and common services that could be offered to clients [15]; this paper carries on that work by proposing a client-side architecture that integrates with native power management.

III. Network Connection Proxy

The Network Connection Proxy (NCP) is a logical function that maintains network presence for temporally unconnected devices, for example those which are in low-power states ('sleeping' devices). The NCP processes packets intended to such hosts and carries out specific tasks on their behalf. This operation implies four main issues:

1) the knowledge of client devices and of the kind of behavior they need while asleep;
2) the notification of any change in the power state of client devices;
3) the abstraction, the retention, the update and the exchange of internal status for various protocols and applications;
4) the ability to catch traffic intended to covered hosts.

The basic operation of an NCP includes the following steps:

1) client devices register with the NCP what it should do when they are unavailable;
2) client devices notify the NCP they are going to sleep; maybe some state information is passed to the NCP;
3) the NCP maintains network presence for covered devices, according to the behavior they have registered. Additionally, it may update state information

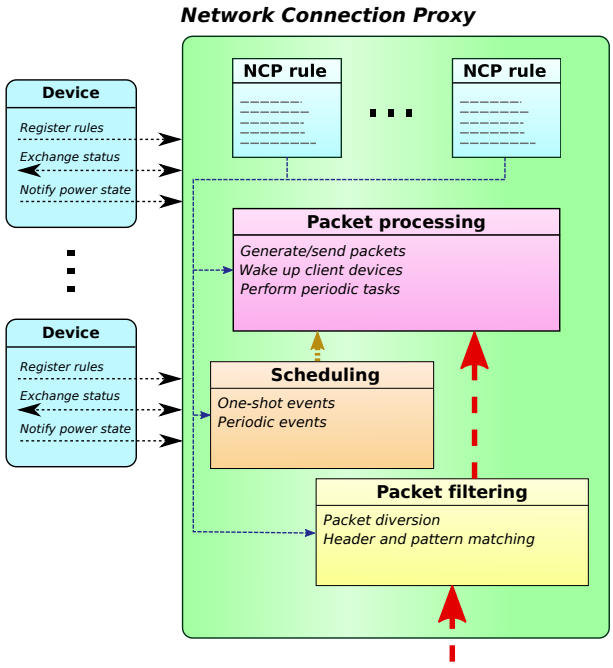

Fig. 1. Reference architecture for NCP operation.

for some protocols/applications (e.g., TCP sequence numbers, counters, timestamps);

4) client devices awake and notify the NCP, which stops covering for them. Updated state information, if any, is returned to them.

Usually, step 1 is carried out only once, or maybe it is updated occasionally when new applications are started or running applications are closed. Step 2–4 are indeed repeated every time a host goes to sleep.

A. Reference architecture

Figure 1 depicts the reference architecture for the Network Connection Proxy. The internal components are the database of behavioral *rules*, *packet processing*, *scheduling* and *packet filtering*; external interfaces enable client devices to control its operation by registering their behavioral rules, by exchanging status information and by notifying their power state transitions.

NCP rules contain the description of the behavior requested by clients. They consist of two parts: the *condition* and the *action*; the former specifies the event that triggers the execution of the latter. *Conditions* may involve packets or time schedules. In the first case, they are given as matching criteria on packets content; in the second case, they consist of time intervals. *Actions* the NCP is expected to carry out include processing, buffering, generation and sending of network packets, and waking up sleeping devices. A behavioral rule is active when the corresponding client is sleeping.

Packet filtering inspects packets and catches those that match the conditions in active rules. The inspection process considers header information (source and destination addresses, protocol, source and destination ports, protocol-specific flags and options) as well as packet content (bit patterns, application-specific headers and data).

Scheduling triggers actions as the time elapses. Triggering might be one-shot (i.e., just one trigger occurs) or periodic (i.e., many triggers are generated).

Packet processing performs the action of each NCP rule, when triggered by the relative condition. The NCP should account at least for the following kinds of operations: waking devices up, parsing and/or modifying incoming packets, building new packets.

B. Supported operations

The NCP currently carries out the following operations on behalf of client devices [15]

- at the link layer: ARP management;
- at the network layer: DHCP renewal, ICMP echo-request answer;
- at the transport layer: wake up on incoming connections, TCP keep-alive;
- at the application layer: generic heart-beating.

The NCP answers ARP requests on behalf of the sleeping host, by providing its own MAC address in ARP responses, so to get all traffic addressed to that node; further, it also sends Gratuitous ARP packets when it starts and when it stops covering each device, in order to update the caches of other hosts on the network. That implements the traffic diversion feature needed to intercept traffic addressed to sleeping clients in local networks.

The NCP wakes client devices up when it detects new connections to given TCP/UDP port numbers; to speed up operation, the NCP also buffers packets that triggered the action and forward them as soon as the device gets awake. For established connections, the NCP answers TCP keep-alive message, in order to avoid the remote peer's closing the session.

For what concerns the generic heart-beating, applications provide a bit template, which includes variable fields that are filled in by the NCP. Variable fields can be counters, timestamps, random numbers and other kinds of information that is easy to infer and is updated by the NCP. Heart-beating messages can be sent in a "solicited" way, by looking for a given bit pattern in incoming packets, or "unsolicted", namely at periodic intervals.

C. Interaction with client devices

Client devices control the NCP behavior by its external communication interface. Three kinds of operations are allowed:

- registration of behavioral rules, which includes all data and parameters needed for performing the operations described in Section III-B;
- transfer of protocols' and applications' status, for example the TCP control block when heart-beating is done over TCP;
- notification of power state transitions.

Currently, our implementation includes a UPnP interface [15], [16], which also works in combination with the UPnP Low Power architecture [12]. However, additional protocols can be easily defined.

D. Deployment

The reference architecture shown in Fig. 1 only brings one major requirement: the ability to intercept packets intended to sleeping hosts. In local networks, the NCP implements traffic diversion for sleeping devices by ARP, as described in Section III-B. That means the NCP can run on any device, both network equipment (switches, routers, home gateways) and stand-alone PCs; in particular, we have successfully ran our implementation on standard low-power devices (low-power Atom PCs, Raspberry ARM board) and home gateways (Lantiq EASY 80920 XWAY VRX288 evaluation board) [17].

Our implementation shows the NCP service could be easily moved among network equipment, thus providing all the flexibility needed to virtualize this function.

The adoption of the NCP service in the access/core network opens the door to new opportunities, but also raises new challenges and hindrances. Beyond the LAN boundary, the NCP could cover for a larger number of devices, yielding more energy efficiency; further, running the service in clouds also offers a virtually unlimited number of computing resources, which enables to covers for much complex protocols and applications than envisaged so far. Additionally, the service could also extend battery lifetime of smart devices that moves across different network segments [18]. Proxying connections in the network will allow the distribution of virtual functions over different network elements, which cooperate to build a distributed service that offers a single controlling interface to client devices. This way, different application vendors can provide stubs on their own servers, without the need to disclose their applications' internals to third party developers.

However, a better network orchestration would be needed to successfully and effectively tackle traffic diversion in this case; hopefully, this could be integrated with mobility management in next-generation software-defined networks.

IV. A CLIENT-SIDE FRAMEWORK TO DELEGATE NETWORK ACTIVITY

Despite the location and the specific architecture of the service that carries out routine network tasks, client devices must be able to interact and to control it; this implies a few operations. First of all, they need to collect the behavior the system and the applications expect while the host is asleep and to register it with the service. Second, they must detect the transition to/from the low-power state; transitions may happen automatically (when no activity or no time-critical tasks are running on the host) or may be triggered manually by the user, and they must be notified to the service so it can start covering for the device.

The reference architecture for the NCP shows there are different kinds of operations for covering a device, targeting networking protocols as well as applications (mainly, keep-alive and heart-bearting). Further, the power management subsystem should be involved in the process as well, to catch the power state transitions.

Although the integration of such new features in the OS would only involve a few different products (Windows, Linux, Mac OS X, *BSD), the same operation would be much less feasible for applications. Indeed, there is no reason why every single application on the system should deal with the many details of the communication with the proxy agent (e.g., UPnP) and with the power management subsystem; instead, each application should only care about its logic. What applications really require is functionality, a mean to know about the presence of the NCP and, whenever available, ask for its services by sending commands.

We argue there are a few matters to consider:

- many applications might be running on the same system, but system-wide policies are often needed or just a single communication channel can be used (for example, because the same port should be used);
- applications can register the behavior they need at any time, but status information can be exchanged just before the transition to the low-power state; that implies a tight integration with the power management system, since applications must have enough time to transfer any information between the hardware get frozen;
- there are different interfaces and facilities for the power management subsystem, even on the same OS (e.g., Linux);
- several protocols might be used to delegate connections to different agents.

The most effective way to integrate delegation of network connections into devices and to foster wide usage of them by applications is by building a software framework that hide any details about the service interface and the OS's power management subsystem; this approach abstracts away the communication protocol and the OS's internals and reduces the need for coding, hence acting like a sort of middleware for applications.

Different architectural solutions could be considered for the client-side framework. The most straightforward approach is the realization of a single monolithic library providing the business logic for every task. Although simple and enough effective, this solution is not well-suited for system-wide actions that are not ascribable to single applications, like those entailing link and network layer operations (ARP, ICMP, IGMP, DHCP). Moreover, it does not take advantage of sharing part of the code and other resources among several applications. Indeed, if we consider the current communication protocol for our NCP (see Section III-C), even if it is possible to run concurrently multiple UPnP instances by binding and listening to the same multicast group, it is still preferable to have a single process for both security and efficiency reasons (being the UPnP protocol quite complex and chatty); additionally, it would be difficult to change, at a later point in time, the communication protocol into a unicast based one without affecting the applications or their setups. Yet having the power management event handling spread among many different applications can overly complicate the definition of a single coherent policy.

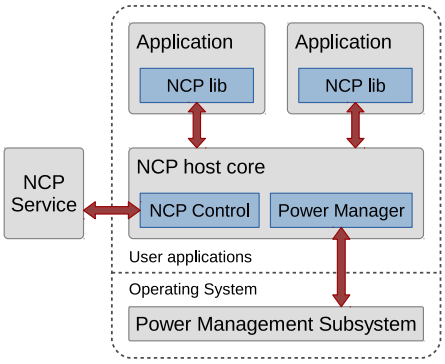

Fig. 2. Client-side framework to delegate network activity to the NCP.

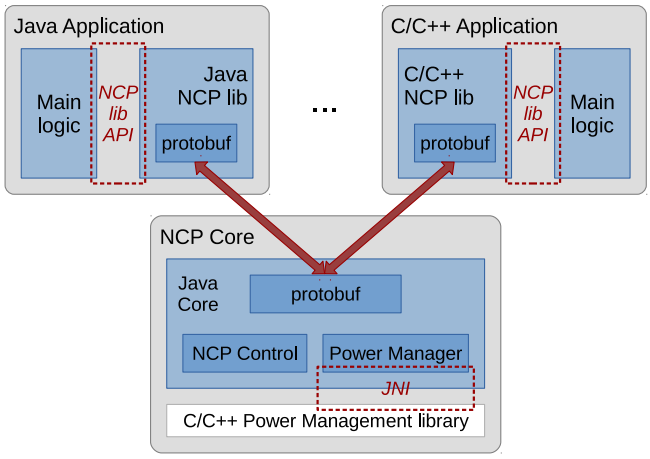

Fig. 3. Software architecture of our implementation of the client-side framework to delegate connections to the NCP agent.

Our design has thus considered a different approach, shown in Fig. 2. The framework is built of two main parts: a core process and tiny libraries for applications. The core is a common and system-wide process, which deals with the communication protocol(s) to delegate network activity and which interacts with the power management subsystems; it could also take charge of handling system-wide tasks at the link/network layers. The libraries are just a small layer that will easily integrate with applications and provide a transparent gate towards the core block in terms of a streamlined and intuitive Application Programming Interface (API) to control the service (the NCP in our case).

The architecture depicted in Fig. 2 removes all the complexity away from the applications and confines any future development (mostly) in a single place, that is the core. That leads to two benefits:

- power management events are handled by one entity only, which can orderly control the pre-suspend (post-resume) procedures, with a single coherent policy;
- since the application library is very simple and thin, multiple implementations for different programming languages are ease to be provided.

V. Implementation

Running on low-power resource-constrained hardware is a key challenge for the NCP; however, portability and flexibility are the main drivers for the client-side framework. With this consideration in mind, we started the implementation of our client-side framework targeting the NCP service described in Section III; Fig. 3 sketches the main software components we have currently selected and the logical relationships among them.

NCP client software must run on the most widespread OSes and must support a broad range of applications, written with different programming languages. We found Java to be the best trade-off between portability, use and protocol support; therefore we used Java to develop the core of the client-side framework. Java is a mature language, which code is usually run by a Java Virtual Machine (JVM). Many different JVM implementations are available for the vast majority of OSes and they are often already included in PCs with Microsoft Windows pre-installed and in most Linux distributions. Moreover, being one of the most adopted programming language ever, many valid UPnP implementation are readily available; among them Cling, an open source project that we choose over the others because of its quality.

In addition to the communication interface, power management is the other core feature. Unfortunately, every OS comes with specific power management subsystems and APIs, and no programming language seems to provide some form of portable support or common API, Java included. However, Java provides straightforward native code execution by means of the Java Native Interface (JNI) specification. Thanks to JNI, any Java class can ask the JVM to load and execute C/C++ code, simply by marking class methods as “native”. This way, the code written in Java will make use of a single and unique interface while the native dynamic library will take care of implementing that interface for a specific OS.

The usage of C/C++ bindings into the core of the framework somehow limits the portability, yet non-portable code is confined to a very narrow part of the whole project; supporting a new OS is just a matter of developing a small C/C++ library.

At this stage of our project, we have considered Microsoft Windows and Linux. The former has been providing power management APIs since long, whereas the latter still does not supply a common and agreed power management interface; we are now considering and evaluating D-Bus[1], as it looks like the most complete and used subsystem for power management notification and control.

Finally, we have to consider the interface between the core and the libraries for applications. Although not part of the business logic, the communication between the two part must guarantee portability and multi-language support. Portability affects the choice of the Inter Process Communication (IPC), which is the way different processes communicate with each other; multi-language support entails an agnostic serialization method to convey the information.

Internet sockets are an example of portable IPC: they are nowadays available on every OS, though they are not the fastest IPC. Data must be structured and encoded from class instances to data streams, more concisely serialized, in

[1] http://www.freedesktop.org/wiki/Software/dbus/

order to be transmitted over sockets; this process is hidden to applications, but it affects the efficiency of the implementation. There are few alternatives available to this purpose, being Google's Protocol Buffers[2] the most suitable solution.

Protocol Buffers is a solid, efficient and extensible technology for encoding structured data. It can undoubtedly be considered a very stable and tested technology. The main driver for the selection of Protocol Buffers is, not only its quality and extensibility, but also the availability of implementations in C++, Java and Python languages. This simplify the development of the applications' libraries for different programming languages, either with a native implementation or by using bindings out of the C++ library.

As a closing note, while the core module can be executed as common application, it is intended to provide a service, thus should be executed as a Windows service or Unix daemon. Many tools provide system specific binding for controlling its lifecycle, freeing the user from its management.

VI. Conclusions

This paper has addressed a client side architecture to delegate the management of basic networking routines to the network. Although our work targets a marginal aspect for future network architectures, it still is complementary to energy efficient mechanisms that are expected to be present inside the network (thanks to network function virtualization and service mobility) and it is a fundamental building block to cut off energy consumption at the network boundary, which is ascribable for most of today's waste.

Our architecture provides a common framework to boost the integration of delegation functions into applications, by hiding all stuff about the network location and interface of the connection agent; software developers can easily modify their applications by calling simple libraries APIs to set what connections are to be handled by the external agent and to transfer any state for that purpose. Further, the integration with power management features already available in the OS enables to automatically notify all interested applications and the network when the device enters and exits low-power states.

Next steps will be the implementation of the software framework and the release of libraries for most used programming languages (C, C++, C#, Java, Python). Functional validation and performance measurement will be conducted to assess the correctness of our implementation and its ability to fulfill timing requirements imposed by the OS.

References

[1] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 2, pp. 223–244, May 2011.

[2] R. Bolla, F. Davoli, R. Bruschi, K. Christensen, F. Cucchietti, and S. Singh, "The potential impact of green technologies in next-generation wireline networks: Is there room for energy saving optimization?" *IEEE Commun. Mag.*, vol. 49, no. 8, pp. 80–86, August 2011.

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

[4] 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.

[5] R. Bolla, R. Bruschi, M. Giribald, 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.

[6] B. Nordman and C. Christensen, "Proxying: The next step in reducing IT energy use," *IEEE Computer*, vol. 43, no. 1, pp. 91–93, January 2010.

[7] 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.

[8] 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.

[9] 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.

[10] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Network Connectivity Proxy: Maintaining virtual presence of sleeping devices to exploit full potential of their power management features," 2014, submitted to Computer Networks.

[11] 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.

[12] "UPnP Low Power architecture," August 2007, version 1.0. [Online]. Available: http://www.upnp.org/specs/lp/UPnP-lp-LPArchitecture-v1.pdf

[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] R. Bolla, M. Giribaldi, R. Khan, and M. Repetto, "Network connectivity proxy: An optimal strategy for reducing energy waste in network edge devices," in *The 24th Tyrrhenian International Workshop on Digital Communications (TIWDC 2013 "Green ICT")*, Sep. 23–25, 2013, pp. 23–25.

[16] ——, "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.

[17] ——, "Design of home energy gateway boosting the development of smart grid applications at home," in *The 4th International Conference on Energy Aware Computing Systems & Applications (2013 ICEAC)*, Istanbul, Turkey, Dec. 16–18, 2013, pp. 103–108.

[18] ——, "Smart proxying: An optimal strategy for improving battery life of mobile devices," in *2013 International Green Computing Conference*, Arlington, Virginia, USA, Jun. 27–29, 2013.

[2] https://code.google.com/p/protobuf/