Achieving Energy Saving through Proxying Applications on behalf of Idle Devices

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 187 – 194

Procedia Computer Science

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

Achieving Energy Saving through Proxying Applications on behalf of Idle Devices

Rafiullah Khan[a,*], Sarmad Ullah Khan[b]

[a]*Queen's University Belfast, Belfast, United Kingdom, Email:rafiullah.khan@qub.ac.uk*
[b]*CECOS University of IT and Emerging Sciences, Peshawar, Pakistan, Email: sarmad@cecos.edu.pk*

Abstract

Several studies in the past have revealed that network end user devices are left powered up 24/7 even when idle just for the sake of maintaining Internet connectivity. Network devices normally support low power states but are kept inactive due to their inability to maintain network connectivity. The Network Connectivity Proxy (NCP) has recently been proposed as an effective mechanism to impersonate network connectivity on behalf of high power devices and enable them to sleep when idle without losing network presence. The NCP can efficiently proxy basic networking protocol, however, proxying of Internet based applications have no absolute solution due to dynamic and non-predictable nature of the packets they are sending and receiving periodically. This paper proposes an approach for proxying Internet based applications and presents the basic software architectures and capabilities. Further, this paper also practically evaluates the proposed framework and analyzes expected energy savings achievable under-different realistic conditions.

Keywords: Green networking; energy efficiency; network connectivity proxy; power measurement; Universal Plug & Play.

1. Introduction

The world-wide energy consumption by computers and other ICT equipments is continuously growing[1]. Some previous studies have revealed that most of this energy is wasted as network devices are most of the time idle or under-utilized[2,3,4]. Fig. 1 depicts the world-wide energy consumption by different ICT sectors[5]. The energy consumption by PCs is estimated 30 GW which is continuously increasing depending on its penetration rate in the world. Beyond PCs, other network devices (e.g., copiers, scanners, game consoles, printers etc) also constitute about 40 GW. In short, ICT consumes more than 168 GW which was more than 8% of the global energy consumption during the year 2008[5].

The Advanced Configuration and Power Interface (ACPI) defined several low power states as an effective mechanism for reducing energy waste. However, ACPI low power states are rarely used due to their inability to maintain network presence for remote access, Instant Messaging (IM), VoIP and other Internet-based applications. The Network Connectivity Proxy (NCP) has recently been proposed to impersonate the connectivity of devices during their sleeping

[*]Corresponding author. Tel.: +44-747-180-2366.
E-mail address: rafiullah.khan@qub.ac.uk

doi:10.1016/j.procs.2016.04.115

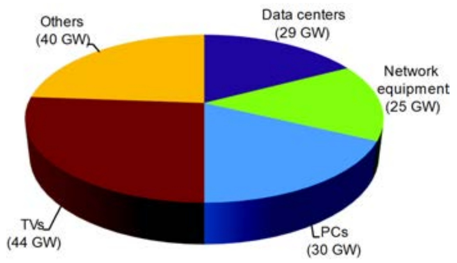


Figure 1. Worldwide energy consumption by different ICT sectors[5].

periods[2,6]. The NCP hides low power states of covered devices by impersonating their link, network, transport and application layers presence on their behalf. It wakes up a covered device only when its resources are required[7].

Till now, the NCP concept has been quite successful in proxying basic networking protocols[6,8]. However, proxying of applications have always been quite challenging due to nature of packets sent/received by different applications. The proxying of applications become almost impossible when proprietary closed-source applications are concerned[9,10,11]. This paper presents a future oriented approach for proxying any open or closed source application. Further, this paper also validates the proposed framework through implementations and evaluation of proxying functions for xChat and kadu-chat applications in experimental test-bed.

2. Related Work

Jimeno et. al. presented the most initial work on NCP concept[6]. The authors have focused their work on NCP design for on-board NIC and LAN scenarios. The generic NCP design on on-board NIC and strategies to increase the device sleep periods have also been addressed in several other studies[3,12,13]. A detailed state of the art work on NCP concept is presented by Khan et. al.[2]. The authors have highlighted key challenges in design and implementation and have suggested possible future directions for extending NCP capabilities[11,14,15,16]. To have better understanding of the importance of NCP concept, Christensen et. al.[1] performed traffic analysis inside a university dormitory and estimated possible savings. S. Nedevschi et. al. have classified achievable savings from NCP into four categories based on the NCP capabilities[17]. Although, the authors have not implemented a complete NCP but estimated potential savings.

It is worth mentioning that most NCP implementations have focused on proxying of basic networking protocols (ARP, PING, DHCP etc) while few have addressed proxying of applications[18,19,20]. Further, previous approaches in literature are only limited to open-source applications and cannot be adopted for proprietary closed-source applications (such as, use of application stubs[10,21]). The application stubs are light-weight version of original applications developed from their source code.

3. Network Connectivity Proxy

The NCP impersonates sleeping devices such that they appear as fully operational and connected to their remote peers. This requires the NCP to carry out basic network presence and management tasks on their behalf. It wakes a sleeping device up only when its resources are required by sending the Wake-On-LAN (WOL) packet. Generally, the NCP functionalities can be divided into four sub-categories: (i) link-layer proxying (i.e., managing ARP requests), (ii) network-layer proxying (i.e., keeping sleeping device's IP address alive, renewing DHCP leases, replying to ICMP PING etc), (iii) transport-layer proxying (i.e., preserving TCP connections alive/open) and (iv) application-layer proxying (i.e., generating/responding to applications periodic presence messages).

3.1. Basic Requirements

The basic requirements of a NCP include[2,6]: (i) it should be always accessible and available in the network, (ii) it should be aware of the power state transitions of covered devices (e.g., by developing a kernel module[4]), (iii) it should be able access the packets intended for sleeping devices (can be achieved through ARP spoofing), (iv) it should be able to wake-up a sleeping device when necessary (i.e., sending WOL packet), (v) it should keep the sleeping devices IP addresses alive (i.e., renewing DHCP leases), (vi) it should be able to proxy basic network management and presence protocols (e.g., ICMP PING), (vii) it should be able to keep TCP connections alive on-behalf of sleeping devices or establish a new connection, (viii) it should be able to proxy any running application on its client devices.

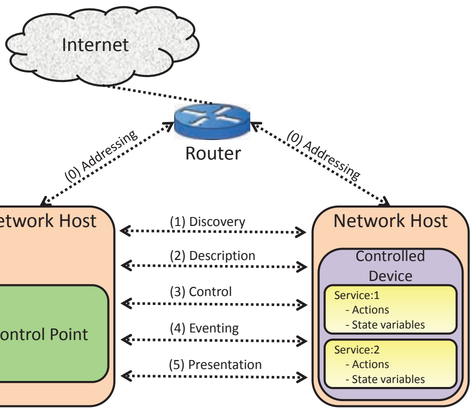


Figure 2. Generic communication scenario between two UPnP devices.

3.2. *Communication Protocols*

Communication protocols are required for communication between the NCP and its client devices. Our proposed framework (presented in Section 4 - Fig. 5) consists of two communication protocols; Hyper Text Transfer Protocol (HTTP) and Universal Plug & Play (UPnP). HTTP protocol is based on simple client-server architecture. However, UPnP is a complex communication system and offers auto-discovery, zero configurations and invisible networking features. UPnP based communication consists of two types of devices: Control Point (CP) and Controlled Device (CD). The CD implements one or more services which can be requested by the CP.

Fig. 2 shows generic communication scenario between two UPnP devices. During addressing, UPnP devices acquire IP addresses through DHCP/auto-IP. Discovery is a unique and distinguishing feature of UPnP for discovering the presence of other UPnP devices. After discovery, the CP retrieve the description of CD and its embedded services. During the control phase, CP sends instructions/commands to the CD which may result in state change of CD (e.g., power state transitions). Thus, the CD updates the CP about such updates using Eventing. The CP may also load CD description in a browser using its presentation URL. This phase is known as presentation. In NCP framework depicted in Fig. 5, both NCP and its client devices implement a CP as well as a CD to achieve two way command and control features. The detailed design of UPnP-based NCP communication framework is presented in[8,9,22,23].

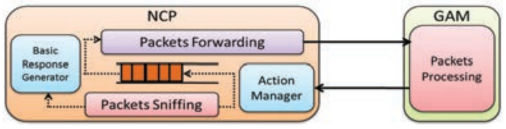
3.3. *Proxying Basic Networking Protocols*

The NCP is responsible to proxy sleeping devices at all layers of TCP/IP stack. There are several networking protocols which network devices use quite frequently e.g., ARP, DHCP etc. The ARP protocol associates a device IP address to its MAC address. This helps in the final delivery of packets inside the local network which is based on MAC address. If a device goes to sleep mode, the NCP needs to associates its own MAC address with the sleeping device IP address. This process is known as ARP spoofing or traffic diversion. The NCP also needs to keep sleeping device IP address alive. Unless static addressing is used, the NCP needs to periodically renew DHCP lease with the DHCP server on behalf of sleeping device. With successful DHCP lease renewal and ARP spoofing, the NCP will be able to receive all its traffic. ICMP PING messages are normally used to check presence/availability of a device. Therefore, the NCP must be capable to respond to PING messages on behalf of a sleeping device.

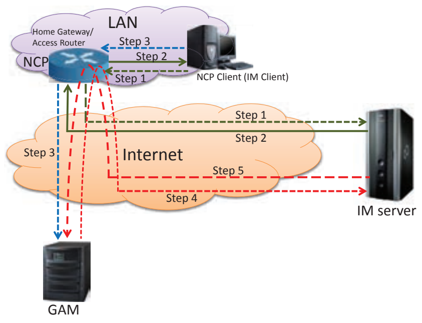
The most common reason due to which users normally keep their PCs powered-up 24/7 is the remote access. Remote access applications establish a new connection over TCP or UDP with their remote peers. The NCP must be able to recognize such packets and take appropriate actions (e.g., buffering of new connection request packets, waking-up of sleeping device and transferring of buffered packets to device after wake). The NCP rules and actions for proxying ARP, DHCP, ICMP PING and Wake-On-Connection for remote access applications are addressed in[11,23] and are out of scope for this paper.

3.4. *Proxying Applications*

Proxying applications is the most challenging task for NCP especially if they operate over TCP. This requires the NCP to generate/respond to application-specific periodic messages as well as preserve TCP connections alive on



(a) Proxying conceptual architecture



(b) Overview of proxying operation.

Figure 3. Proxying of IM Applications.

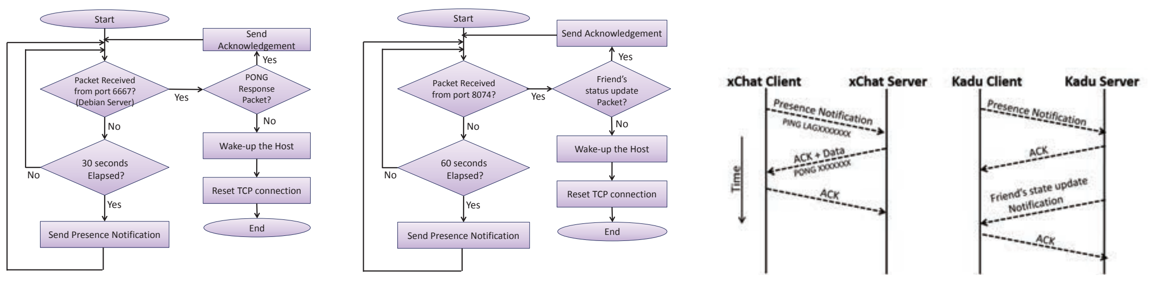
behalf of sleeping devices. The same TCP connection will be resumed back on sleeping device along with updated application status after it wakes-up. The network applications periodically transmit/receive heartbeat messages that indicate the specific application is active and responding. An application is considered disconnected/offline if it fails to do so. Generally, applications can be classified into two categories where NCP plays its rule differently:

1. Stateless Applications: Such applications do not maintain/store the status information during the course of their use (i.e., no persistent sessions with remote peers). Such applications either do not send/receive periodic messages or the messages are easily generatable by third parties (e.g., counter or random value in payload). The stateless applications can be sub-categorized into: (i) proxiable applications and (ii) non-proxiable applications. Proxiable stateless applications require simple response to be generated by the NCP which will be accepted by the remote peers. Non-proxibale stateless applications require the sleeping device resources e.g., Remote Desktop or SSH request, incoming file sharing request, request to web server, request to video server etc.

2. Stateful Applications: The operations of such applications depend on the status information. They normally maintain long active sessions or keep transport connection open. Stateful applications periodically transmit/receive keep alive or heartbeat status messages. Thus, proxying stateful applications is comparatively more complex and requires NCP to understand the syntax and semantics of their specific heartbeat messages. Examples of stateful applications include IM applications, VoIP clients, P2P applications etc.

This paper focuses on proxying of IM applications based on using application-specific routines and TCP snooping. Different IM applications have different type of heartbeat messages exchanged at different time intervals and use different ways to exchange such messages with their IM server. Thus, specific routines can be designed that periodically send status messages and handle received messages in accordance with each specific IM application. All that is required for proper functioning of IM routines is the last exchanged IM specific heartbeat message. The NCP then uses these routines to maintain the IM status on behalf of sleeping devices. The generic scenario for proxying an IM application is shown in Fig. 3(b). The NCP client sniffs and stores the status packet that is sent to the IM server. Every time a packet is exchanged, the NCP client deletes the previous status packet and stores the new one. Before going to sleep, it transfers the latest status packet to the NCP to snoop the connection. The NCP then starts sending periodic status packets. The conceptual architecture is depicted in Fig. 3(a). The NCP sniffs packets intended for sleeping devices, analyzes them and generates their response if they belong to supported networking protocols (addressed in Section 3.3). If packets belong to any non-supported application, they are queued to be forwarded to Global Application Manager (GAM). The GAM processes the received packets and sends back the response packets to the NCP to take appropriate action. To clarify the proxying operation, Fig. 3(b) represents the simplified process: *Step 1:* NCP client stores and sends the status packet to the IM server. *Step 2:* The IM server acknowledges the status packet. *Step 3:* NCP client is going to sleep and transfers the last status packet it has exchanged with IM server to GAM through NCP. *Step 4:* The GAM uses IM specific routine to generate and send the status packets to the IM server with the help of NCP. *Step 5:* The IM server acknowledges the GAM about the status packets through NCP.

The step 4 and step 5 are periodically repeated as long as the NCP client is sleeping. The NCP maintains the same TCP connection which was initiated by its client (IM client) and terminates it when its client wakes-up. The IM



(a) Proxying XChat. (b) Proxying Kadu-chat. (c) XChat and Kadu-chat heartbeat messages.

Figure 4. Proxying of XChat and Kadu-chat Applications.

client detects the de-synchronization with the IM server and re-establishes a new connection and starts repeating step 1 and 2. The termination of TCP session does not create any problem for IM applications as the client re-established connection upon wake-up and immediately receive all IM updates. During the whole process (including step 4 & 5 in Fig. 3(b)), the IM client user appears online to its remote peers/friends. This approach of proxying IM applications has been successfully applied to proxying of XChat and Kadu-chat IM applications.

3.4.1. *XChat Instant Messenger*

XChat is an Internet Relay Chat (IRC) based open source chat software. It allows user to join multiple IRC channels with different chat rooms. The proxying functionality has been successfully tested for Debian Server (irc.debian.org/6667). The NCP client sniffs the packets directed to or coming from port 6667. The packet analysis was performed on payload to differentiate status packets from others. The client application periodically sends status packet with fixed payload containing the text 'PING LAG' which precedes a random number. This number in payload continuously increase in the upcoming status packets. The response packet contains the text 'PONG' following by the same number in payload. The client acknowledges the IM server on receiving this packet. The basic XChat proxying scheme is depicted in Fig. 4(a) which operates according to XChat normal functioning depicted in Fig. 4(c).

3.4.2. *Kadu Instant Messenger*

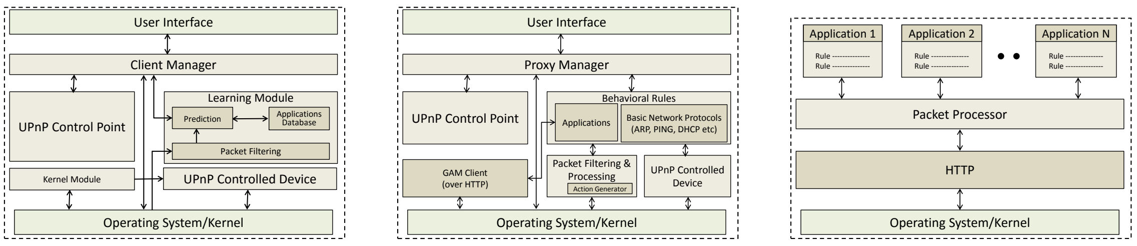
Kadu is an open source multi-platform messaging software that supports Gadu-Gadu and Jabber/XMPP protocols. It allows user to change his status, add new contacts, view friends status and have private one-2-one chat conversation. The NCP client sniffs status packets on port 8074. Once again, packet analysis was performed on payload to differentiate status packets from others. The client application periodically sends a fixed payload status packet which is acknowledged by the Kadu server. The basic Kadu chat proxying scheme is depicted in Fig. 4(b) which operates according to Kadu normal functioning depicted in Fig. 4(c).

3.5. *Global Applications Manager*

Writing application stubs[10,21] for proxying is tough job due to thousands of available applications (mostly proprietary closed-source). Also, application-specific routines and TCP snooping (used for XChat & Kadu) can also be complicated due to many applications have encrypted and dynamic payloads. The encrypted status packets can still be snooped if the data field does not change in the subsequent status notification messages. The trust relationship between the NCP and its clients for proxying applications can be established only in case of a global trusted standardized proxying entity. Thus, the GAM can be designed to this aim in cooperation with different application developers. If GAM is a global trusted entity, application developers will not hesitate to supply their application-specific proxying techniques/code to the GAM. This is the main motivation behind the use of GAM in this paper.

3.6. *Proxying Persistent TCP Connections*

Re-establishment of proxied TCP session after client device wake-up is not necessary for XChat and Kadu-chat applications. The client gets updates immediately upon wake-up by establishing a new connection. However, some applications demand the re-establishment of same proxied TCP session (e.g., a P2P client in queue for downloading, VoIP, video calling applications, etc). The most convenient approach to achieve this is the migration of updated/latest TCP state from NCP/GAM to its client device upon wake-up. The client device re-establishes the same TCP session by correcting the values of different TCP socket parameters. This approach is known as TCP session migration[24].



(a) NCP Client (b) NCP (c) GAM

Figure 5. Basic software structures in the proposed framework.

4. Implementations

The implementation of proxying system consists of three software entities as shown in Fig. 5; NCP client, NCP and GAM. The NCP and its client interact with one another using UPnP while the NCP and GAM interact over HTTP. All three software entities were developed in Linux OS using standard C/C++ programming language along with the help of Boost libraries, PCAP libraries, and libupnp libraries. Fig. 5(a) depicts the basic structure of the NCP client software. It consists of a Kernel Module to track the changes in power state of the device and immediately inform the NCP. To achieve two-way control, it implements UPnP CD as well as CP (see Section 3.2). It also implements a Learning Module to determine the applications being used on the client device by capturing packets and comparing their structure with the Application Database. It plays a key role in capturing latest XChat and KaduChat heartbeat/presence messages and supplying to the GAM.

A quite simplified structure of the NCP software is depicted in Fig. 5(b). Likewise NCP client, it also implements UPnP CD as well as CP. Further, it also implements GAM Client which interacts with the GAM over HTTP protocol. Packet Filtering & Processing is an important block that sniffs packets intended for sleeping devices and generates response packets (either itself or with the help of GAM). Behavioral Rules specify the proxying capabilities for different protocols and applications. It is classified into two types: Basic Network Protocols and Applications. Sniffed packets matching any protocol in the Basic Network Protocols are processed by the NCP itself while packets matching any application are processed with the help of GAM.

Fig. 5(c) depicts basic GAM structure. It implements HTTP client and server for communication with the NCP. The GAM processes each packet received from the NCP based on the application specific routines/rules. For each supported application, it contains proxying techniques for generating/responding to different types of packets.

5. Evaluation of the Proxying Framework

The experimental testbed for the evaluation of the proxying framework is shown in Fig. 6. It consists of NCP client (running two applications: XChat & Kadu-chat), the NCP, the GAM and a remote peer. The remote peer is also running XChat & Kadu-chat applications for messaging with the NCP client. The NCP and GAM softwares are running on low power device i.e., Raspberry Pi v2 (CPU: ARMv6 700 MHz, memory: 512MB, power consumption: 3.8W) to avoid any significant energy waste. The NCP impersonates its client device as long as it is sleeping. When NCP receives a text message (either on XChat or Kadu-chat), it wakes-up its client device using WOL packet. This section evaluates the effectiveness of the proxying framework based on resource requirements, communication latencies and achievable energy savings. The experimental results focus on the GAM software. Detailed evaluation of the NCP software is reported in[11,23].

Since, NCP and GAM softwares are running on low power devices, they have limited available memory and processing power. Both these factors can limit the scalability in terms of the number of proxied devices and applications. Fig. 7(a) and Fig. 7(b) depicts the CPU and memory usage for GAM, respectively. It can be observed that the resource requirement is quite low and has no significant impact on GAM scalability. Another important factor limiting scalability is the communication and processing latencies. The observed UPnP communication latency between the NCP and its client devices is very low as reported in Table 1. The HTTP communication latency between the NCP and GAM is a bit high. This is not due to the fact that communication passes through the Internet but a significantly portion of this latency is contributed by NO-IP dynamic DNS service. HTTP communication latency will be low if

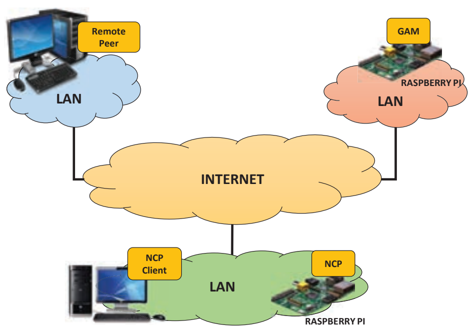


Figure 6. Experimental Testbed.

no third-party DNS service is used. The GAM processing latency in Table 1 is the time required to process a received packet and is quite low even on Raspberry Pi. Based on experimental tests, the GAM scalability is up to 400 applications considering 1 second as maximum latency threshold. It is worth nothing that maximum NCP scalability is 23 when considering 1 second as latency threshold[23].

Table 1. Communication and GAM processing latencies.

UPnP Communication	HTTP Communication	GAM Processing
96.33 ms	1.29 sec	2.49 ms

The expected energy savings depend on the power consumption of NCP clients in awake and sleep states, their sleep durations and power consumption of NCP and GAM hosting devices. Mathematically, it can be expressed as:

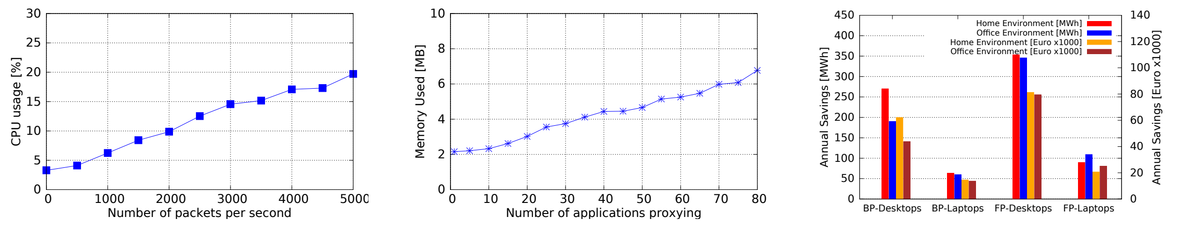
$$E_{Savings} = \sum_{i=0}^{N} (P_{i,Awake} - P_{i,Sleep}) t_{i,Sleep} - P_{(NCP+GAM)} t_{24h} \quad (1)$$

where $P_{i,Awake}$ and $P_{i,Sleep}$ represent the power consumption of NCP client i in awake and sleep states, respectively. $P_{(NCP+GAM)}$ represents the combined power consumption of the devices hosting the NCP and GAM services which are powered-up 24/7 (t_{24h}). It also takes into account NCP and GAM scalability as multiple NCP and GAM instances will be used for proxying large-size networks. $t_{i,Sleep}$ represents the time NCP client i stays in sleep state.

Fig. 7(c) depicts expected savings considering 1000 PCs under Basic Proxying (BP) and Full Proxying (FP) conditions. The BP involves functionalities of only NCP (i.e., proxying only basic network protocols) whereas FP involves functionalities of both, NCP and GAM (i.e., proxying all protocols and applications). Based on the network traffic analysis[17], NCP clients under BP can sleep for maximum 10.94 and 6.91 hours/day in home and office environments, respectively. In case of FP, NCP clients can sleep for maximum 13.94 and 12.52 hours/day in home and office environments, respectively. The reported savings in Fig. 7(c) consider desktop PC with 80 W consumption in idle and 4 W in sleeping state. The laptop PC is assumed consuming 25 W in idle and 0.8 W in sleeping state. It can be observed that energy savings in home environment is lower than office environment for FP-Laptop which is in contrast with the sleep durations. It is due to the fact that in home environment, each NCP proxy just one PC (i.e., 1000 NCPs required) while in office environment, each NCP can proxy 23 PCs (i.e., 44 NCPs required). As depicted in Fig. 7(c), several hundreds MWh energy savings can be achieved when considering 1000 homes or an organization with 1000 PCs. If NCP is adopted worldwide, Billions of Euro savings can be achieved considering millions of PCs in the world.

6. Conclusions

The NCP is quite useful approach for encouraging devices to sleep when idle without losing network connectivity. It has been quite successful in proxying basic networking protocols, however, proxying of application has no absolute solution especially when the applications are proprietary closed-source. This paper proposed a proxying framework which can be adopted for both, open-source as well as closed-source applications. A global trusted application proxying entity brings more opportunities and potential for large energy savings through possible collaboration with application developers. This paper presented the basic proxying requirements and addressed software architectures in



(a) CPU usage. (b) Memory usage. (c) Expected savings considering 1000 PCs.

Figure 7. Analysis of required resources for GAM and expected energy savings (considering 23 Euro cents per kWh as average cost of electricity).

proposed system. Additionally, the proposed system was practically evaluated through proxying of two applications (XChat and Kadu-chat). It was analyzed that significantly higher energy savings can be achieved if applications are also proxied compared to proxying of only basic networking protocols. In short, proxying can provide great amount of economic savings while indirectly reducing CO_2 footprint in the atmosphere.

References

1. K. Christensen, P. Gunaratne, B. Nordman and A. George. *The Next Frontier for Communications Networks: Power Management,* in Computer Communications, Vol. 27, No. 18, pp. 1758-1770, Dec. 2004.
2. R. Khan, R. Bolla, M. Repetto, R. Bruschi and M. Giribaldi. *Smart Proxying for Reducing Network Energy Consumption,* in SPECTS 2012.
3. 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,* in IEEE Communications Surveys and Tutorials (COMST), vol 13 no. 2, pp. 223-244, May 2011.
4. R. Bolla, M. Giribaldi, R. Khan and M. Repetto. *Network Connectivity Proxy: An Optimal Strategy for Reducing Energy Waste in Network Edge Devices,* in 24th Tyrrhenian International Workshop on Digital Communications, 2013.
5. M. Pickavet, W. Vereecken, S. Demeyer, P. Audenaert, and et. al. *Worldwide Energy Needs for ICT: The Rise of Power-Aware Networking,* in 2nd International Symposium on Advanced Networks and Telecommunication Systems, 2008. ANTS 08, Dec. 2008.
6. M. Jimeno, K. Christensen and B. Nordman. *A Network Connection Proxy to Enable Hosts to Sleep and Save Energy,* in IPCCC conf. 2008.
7. K. Sabhanatarajan, A. G. 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, 2008.
8. R. Khan, R. Bolla and M. Repetto. *Design of UPnP based Cooperative Network Connectivity Proxy* in The Second IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2012), Oct. 2012.
9. R. Bolla, M. Chiappero, R. Khan and M. Repetto. *Saving Energy by Delegating Network Activity to Home Gateways,* in IEEE Transactions on Consumer Electronics, 2015.
10. Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl and R. Gupta. *Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage,* in 6th ACM/USENIX Symposium, 2009.
11. R. Bolla, R. Khan and M. Repetto. *Assessing the Potential for Saving Energy by Impersonating Idle Networked Devices,* in IEEE Journal on Selected Areas in Communications, series on Green Communications and Networking, 2016.
12. 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,* in Int. Journal of Network Management, Vol. 15, No. 5, pp. 297-310, Sept. 2005.
13. K. Christensen and F. Gulledge. *Enabling Power Management for Network-Attached Computers,* in Network Management Journal, 1998.
14. R. Bolla, M. Giribaldi, R. Khan and M. Repetto. *Smart Proxying: An Optimal Strategy for Improving Battery Life of Mobile Devices,* in 4th International Green Computing Conference (IGCC), 2013.
15. R. Bolla, R. Khan and X. Parra and M. Repetto. *Improving Smartphones Battery Life by Reducing Energy Waste of Background Applications,* in 8th International Conference on Next Generation Mobile Applications, Services, and Technologies (NGMAST), 2014.
16. R. Bolla, M. Giribaldi, R. Khan and M. Repetto. *Design of Home Energy Gateway Boosting the Development of Smart Grid Applications at Home,* in 4th International Conference on Energy Aware Computing Systems & Applications (ICEAC), 2013.
17. 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 USENIX symposium, 2009.
18. 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 SNCW, 2005.
19. P. Werstein and W. Vossen. *A Low-Power Proxy to Allow Unattended Jabber Clients to Sleep,* in PDCAT conference, 2008.
20. M. Jimeno and K. Christensen. *A Prototype Power Management Proxy for Gnutella Peer-to-Peer File Sharing,* in 32nd IEEE LCN, 2007.
21. 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 USENIX conference, 2010.
22. R. Bolla, M. Giribaldi, R. Khan and M. Repetto. *Design and Implementation of Cooperative Network Connectivity Proxy using Universal Plug and Play,* in 10th edition of Future Internet Assembly (FIA Book 2013), 2013.
23. R. Bolla, M. Giribaldi, R. Khan and M. Repetto. *Network Connectivity Proxy: Architecture, Implementation, and Performance Analysis,* in IEEE Systems Journal, 2015.
24. R. Bolla, M. Chiappero, R. Rapuzzi and M. Repetto. *Seamless and transparent migration for TCP sessions,* in IEEE PIMRC 2014.