ADVERTIMENT. La consulta d’aquesta tesi queda condicionada a l’acceptació de les següents condicions d’ús: La difusió d’aquesta tesi per mitjà del servei TDX (www.tesisenxarxa.net) ha estat autoritzada pels titulars dels drets de propietat intel·lectual únicament per a usos privats enmarcats en activitats d’investigació i docència. No s’autoritza la seva reproducció amb finalitats de lucre ni la seva difusió i posada a disposició des d’un lloc aliè al servei TDX. No s’autoritza la presentació del seu contingut en una finestra o marc aliè a TDX (framing). Aquesta reserva de drets afecta tant al resum de presentació de la tesi com als seus continguts. En la utilització o cita de parts de la tesi és obligat indicar el nom de la persona autora.

ADVERTENCIA. La consulta de esta tesis queda condicionada a la aceptación de las siguientes condiciones de uso: La difusión de esta tesis por medio del servicio TDR (www.tesisenred.net) ha sido autorizada por los titulares de los derechos de propiedad intelectual únicamente para usos privados enmarcados en actividades de investigación y docencia. No se autoriza su reproducción con finalidades de lucro ni su difusión y puesta a disposición desde un sitio ajeno al servicio TDR. No se autoriza la presentación de su contenido en una ventana o marco ajeno a TDR (framing). Esta reserva de derechos afecta tanto al resumen de presentación de la tesis como a sus contenidos. En la utilización o cita de partes de la tesis es obligado indicar el nombre de la persona autora.

WARNING. On having consulted this thesis you’re accepting the following use conditions: Spreading this thesis by the TDX (www.tesisenxarxa.net) service has been authorized by the titular of the intellectual property rights only for private uses placed in investigation and teaching activities. Reproduction with lucrative aims is not authorized neither its spreading and availability from a site foreign to the TDX service. Introducing its content in a window or frame foreign to the TDX service is not authorized (framing). This rights affect to the presentation summary of the thesis as well as to its contents. In the using or citation of parts of the thesis it’s obliged to indicate the name of the author.
Erasmus Mundus Joint Doctorate in Interactive and Cognitive Environments [EMJD ICE]

PhD Dissertation
Energy-Aware Home Area Networking

Rafiullah Khan

Supervisors
Prof. Raffaele Bolla
Università degli Studi di Genova

Prof. Xavier Parra
Universitat Politècnica de Catalunya BarcelonaTech

August 2014
ICE PhD Acknowledgements

This PhD Thesis has been developed in the framework and according to the rules of the Erasmus Mundus Joint Doctorate on Interactive and Cognitive Environments EMJD ICE [FPA n° 2010-0012] with the cooperation of the following Universities:

- Alpen-Adria-Universität Klagenfurt – AAU
- Queen Mary, University of London – QMUL
- Technische Universiteit Eindhoven – TU/e
- Università degli Studi di Genova – UNIGE
- Universitat Politècnica Catalunya – UPC

According to ICE regulations, this PhD Thesis has been awarded also of Italian PhD title by the Università degli Studi di Genova.
I wish to express my thankfulness to Prof. Raffaele Bolla for his supervision and guidance throughout my Ph.D activities. He offered me numerous opportunities including collaboration with National Inter-universities Telecommunication Consortium (CNIT) to contribute to the European Commission Framework Program 7 (FP7) funded project called low Energy COnsumption NETworks (ECONET). Besides supervision, Prof. Raffaele Bolla also offered me the opportunity to work as a visiting researcher for 3 months at Portland State University (PSU) in United States under the project Future InterNet Energy Efficiency (FINE2). I also offer thanks to Prof. Suresh Singh for his kind supervision during my stay at the PSU.

I wish to express my sincere gratitude to Dr. Matteo Repetto from National Inter-universities Telecommunication Consortium (CNIT) whose endless help and unbounded support made me able to write this dissertation. His supervision and guidance always motivated me to explore new solutions and always kept me on the right track. I express my thankfulness to Maurizio Giribaldi from Infocom s.r.l, whose research experience helped me in achieving my objectives. I am also thankful to Prof. Xavier Parra for his supervision and proving me a peaceful research environment during my stay at Universitat Politècnica Catalunya (UPC).

I am thankful to Lantiq company for providing necessary hardware support during my Ph.D. Finally, I express my thankfulness to Telecom Italia for providing a professional experimental environment to demonstrate the research activities.
This dissertation was produced under Erasmus Mundus Joint Doctorate Program in Interactive and Cognitive Environments. The research was conducted towards a joint/double PhD degree between the following partner universities:

Università degli Studi di Genova (UNIGE)

Universitat Politècnica de Catalunya (UPC)
Abstract

Green networking is one of the important research topics due to ever rising cost of the electricity, limited available natural resources and environmental concerns. Briefly, green networking is the practice of selecting energy efficient networking techniques and strategies with the aim of reducing unnecessary network energy consumption while maintaining the same quality of service.

A study by Lawrence Berkeley National Laboratory (LBNL) revealed that about 60% of the office PCs are left powered-up 24/7 only to maintain the network connectivity for remote access, Voice-over-IP (VOIP) clients, Instant Messaging (IM) and other administrative management reasons [1]. The Advanced Configuration and Power Interface (ACPI) proposed several low power states for PCs as effective mechanism to reduce energy waste, but unfortunately low power states are seldomly used due to their incapability to maintain network presence. Thus, Billions of dollars of electricity is wasted every year to keep idle or unused network devices fully powered-up only to maintain the network connectivity.

This dissertation addresses the Network Connectivity Proxy (NCP), a concept recently been proposed as an optimal strategy to reduce energy waste due to idle network devices. The NCP is a software entity running on a low power network device (such as home gateway, switch or router) and impersonates presence for high power devices (such as PCs) during their sleeping periods. The NCP wakes-up a sleeping device only when its resources are required (e.g., remote access connection request). In short, the NCP impersonates link layer, network layer, transport layer and application layer presence on behalf of sleeping devices.

Research Contributions

In this dissertation, we presented the design and implementation of our NCP prototype for reducing the energy waste due to idle network devices. The NCP concept faces several issues and challenges that we tried to address in the most effective way in our implementations. Knowing when to start or stop proxying presence on behalf of sleeping devices is critically important for the NCP operations. To achieve this objective in a seamless way without requiring any user intervention, we developed a kernel module that monitors the power state transitions of the device and immediately informs the NCP over a suitable communication protocol in case of any update.

An important challenge for the NCP is its ability to proxy a huge and ever increasing number of applications and networking protocols on behalf of sleeping devices. This requires the knowledge of specific applications/protocols and the packets they exchange periodically with their remote peers (only possible for open-source applications). To tackle with this challenge in an efficient way, we implemented a quite generalized set of behavioral rules in our NCP framework that can be suitable for any protocol or application (including closed-source). Our developed NCP prototype is also capable to proxy transport layer protocols (associated with different applications) and keeps them alive on behalf of sleeping devices.

A communication protocol is required for information exchange between the NCP and client devices.
e.g., power state notifications, registration or de-registration of client devices and behavioral rules for proxying different protocols and applications etc. To avoid any configuration issues, we developed a flexible and reliable communication framework based on the Universal Plug & Play (UPnP) architecture that provides interesting features such as auto-discovery, zero-configuration and seamless communication between the NCP and client devices.

It is important that the NCP runs on a low power hardware entity with enough memory and processing resources to impersonate presence on behalf of specific number of high power devices without causing any incremental power consumption. To aim this, we incorporated deployment flexibility in our NCP software that enables us to operate it on on-board NIC, switch/router or on a standalone PC. This flexibility enables us to deploy the NCP software on any device that meets our needs. For example, on-board NIC and switch/router are the optimal locations for the NCP software in home/small office environment (very limited number of devices) or a standalone PC with enough resources is a good choice if high scalability is desirable e.g., medium or large size organizations.

The NCP concept is also quite important for mobile devices to reduce the idle energy waste as they are battery powered. Thus, we extended the NCP concept for mobile devices to help in improving the battery life. We mainly proposed different possible architectural solution for supporting client devices mobility and highlighted issues and challenges in different deployment considerations of the NCP software. Further, we expanded the NCP coverage beyond LAN boundaries in order to exploit its full potential in terms of energy savings by covering for thousands of client devices. A single global powerful NCP instance located anywhere in the Internet can make easier the implementation of complex tasks and boosts up the energy savings by also shutting down the unused access links and the packets forwarding equipments whenever possible.

Another important contribution of this dissertation includes the extensive evaluation of the NCP performance on different low power hardwares. We performed large number of experiments and evaluated the effectiveness of NCP prototype in different realistic scenarios. Further, we proposed, implemented and tested several design optimizations to improve the overall performance such as improving NCP scalability, reducing processing latencies, improving effectiveness of NCP packets filtering engines etc.

Outline of Dissertation

In this dissertation, we presented the design, implementation and evaluation of our NCP prototype. We mainly highlighted issues and challenges in its realization and addressed in detail its capabilities and supported features. Following this, we presented our efforts in extending NCP coverage beyond LAN boundaries in order to enable it to proxy very large number of devices globally. We also presented the design of our communication protocol and our efforts in incorporating NCP support for mobile devices. To have better understanding of the outline of this dissertation, the chapter wise details are as follows:

- **Chapter 1** briefly addresses the importance of green technologies and provides different strategies to reduce energy waste in ICT. It particularly focuses on the NCP concept and motivates its importance. Further, it also briefly presents the main research contributions being addressed in this dissertation.

- **Chapter 2** presents the background and previous works on NCP concept from the literature. It also presents different available NCP implementations from the literature and addresses their supported features.

- **Chapter 3** presents in detail the NCP concept. In particular, it provides basic requirements for designing the NCP framework, addresses different modes to operation and types based on the location of the NCP service in the network.

- **Chapter 4** presents the design of a cooperative NCP framework. It particularly presents a very flexible NCP architecture that can operate on a number of low-power devices in the local network such
as on-board NIC, switch/router or standalone device. It analyzes different NCP functional blocks and supported features in terms of proxying different protocols and applications. Further, it also presents a communication model for information exchange between the NCP and its client devices.

- **Chapter 5** addresses our efforts in expanding the NCP coverage beyond LAN boundaries by presenting different possible architectural solutions. It addresses opportunities and benefits of having a global Internet NCP instance. Mainly, the issues and challenges have been highlighted in its realization.

- **Chapter 6** addresses key issues and challenges in incorporating NCP support for mobile devices. Further, it presents different architectural solution for supporting client devices mobility in the NCP framework.

- **Chapter 7** presents the implementation efforts to develop the NCP and its client device softwares.

- **Chapter 8** presents detailed experimental evaluation of our NCP framework in different realistic scenarios. It provides large number of experimental results which are mostly performed under extreme or worst case scenarios.

- **Chapter 9** summarizes the accomplishments in this dissertation and also proposes possible future directions.
Reference Work

Reference work related to this Ph.D dissertation is available below.

International Publications

3. R. Khan, R. Bolla and M. Repetto, Design of UPnP based Cooperative Network Connectivity Proxy, in 2nd IEEE IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT), Pisa, Italy, October 2012.

Live Demonstrations/Papers

List of Acronyms

ACPI Advanced Configuration and Power Interface
ARP Address Resolution Protocol
ALG Application Layer Gateway
ASP Application Service Provider
API Application Programming Interface
APM Advanced Power Management
CD Controlled Device
CP Control Point
CN Corresponding Node
CH Corresponding Host
CoA Care-of Address
EIA Energy Information Administration
EPA Environmental Protection Agency
ECONET low Energy CONsumption NETworks
FA Foreign Agent
GeSI Global e-Sustainability Initiative
GENA General Event Notification Architecture
GP-INCP General-Purpose Internet-wide NCP
HTTP Hyper Text Transfer Protocol
HAN Home Area Network
HG Home Gateway
HN Home Network
HA Home Agent
HoA Home Address
IEA International Energy Association
INCP Internet Network Connectivity Proxy
IGD Internet Gateway Device
ISP Internet Service Provider
IGD Internet Gateway Device
IM Instant Messenger
INS Intentional Naming System
LAN Local Area Network
LNCP Local Network Connectivity Proxy
LP Low Power
LBNL Lawrence Berkley National Laboratory
mDNS multicast DNS
MIP Mobile IP
MN Mobile Node
MH Mobile Host
NCP Network Connectivity Proxy
NSF National Science Foundation
QoS Quality of Service
RWT Remote Wake Technology
RA Remote Access
RAS Remote Access Server
RAC Remote Access Client
RAT Remote Access Transport
RAAS Remote Access Application Server
RATA Remote Access Transport Agent
RADA Remote Access Discovery Agent
RDP Remote Desktop Protocol
SSDP Simple Service Discovery Protocol
SOAP Simple Object Access Protocol
STUN Session Traversal Utilities for NAT
SS-INCP Service-Specific Internet-wide NCP
SNMP Simple Network Management Protocol
SOHO Small Office/Home Office
SIP Session Initiation Protocol
TCP-KA TCP Keep-Alive
TCB Transmission Control Block
UPnP Universal Plug & Play
UDA UPnP Device Architecture
URA UPnP Remote Access
UUID Universally Unique IDentifier
VNAT Virtual NAT
VPN Virtual Private Network
VoIP Voice-over IP
WoC Wake On Connection
WLAN Wireless LAN
WOL Wake On LAN
WoWLAN Wake-on-Wireless LAN
# Contents

## Acknowledgements

## 1 Introduction

1.1 Importance of Green Technologies .............................. 1  
1.2 Strategies for Reducing Energy Waste ............................ 2  
1.3 Importance of Network Connectivity Proxy  
  1.3.1 Motivation for NCP  
  1.3.2 The NCP Potential: Achievable Benefits  
1.4 Novelty and Contributions  
  1.4.1 Generalized Set of Behavioral Rules  
  1.4.2 Deployment Flexibility in Local Network  
  1.4.3 Flexible & Reliable Communication Protocol  
  1.4.4 NCP Support for Mobile Devices  
  1.4.5 Coverage Beyond LAN Boundaries  
  1.4.6 Performance Evaluation on Different Hardware Entities  

## 2 Background and Literature Review

2.1 Power Management Facilities  
2.2 Related Work  
2.3 Implementations  
2.4 Key Open Issues and Challenges in NCP Concept  

## 3 Network Connectivity Proxy

3.1 Overview  
3.2 Modes of Operation  
  3.2.1 Uncoordinated or Invisible Proxying  
  3.2.2 Coordinated or Cooperative Proxying  
3.3 Key Requirements  
  3.3.1 Requirements for the NCP  
  3.3.2 Requirements for NCP Client Devices  
  3.3.3 Communication Requirements  
3.4 NCP Deployment Scenarios  
  3.4.1 On-board or Self Proxying Scenario  
  3.4.2 LAN Scenario  
  3.4.3 Point-to-Point Scenario  
  3.4.4 End-Point Proxying Scenario  
  3.4.5 Internet Scenario  
  3.4.6 Mixed Scenario  

4 Design of Local NCP
4.1 NCP Architectural Design ........................................... 27
4.2 Behavioral Rules ......................................................... 29
  4.2.1 ARP Rule ...................................................... 30
  4.2.2 PING Rule .................................................... 30
  4.2.3 DHCP Rule .................................................... 31
  4.2.4 Wake-On-Connection Rule ................................... 31
  4.2.5 TCP Keep-Alive Rule ......................................... 31
  4.2.6 Heartbeating Rule ............................................. 31
4.3 Device Wake-up Mechanisms ........................................ 33
4.4 Network Traffic Diversion .......................................... 33
4.5 Managing TCP Connections ........................................ 35
  4.5.1 Maintaining TCP Connections Alive ......................... 36
  4.5.2 Handing off TCP Connections ................................ 37
4.6 Communication Protocol ............................................ 41
  4.6.1 Choice of Communication Protocol ......................... 41
  4.6.2 Basic UPnP model for NCP .................................. 41
5 Opportunities and Challenges in Internet NCP ..................... 45
  5.1 Single or Dual Proxy Architectures ............................... 46
  5.2 Architectural Solutions based on Capabilities of Internet-NCP 46
    5.2.1 Service-Specific Internet-wide NCP ....................... 47
    5.2.2 General-Purpose Internet-wide NCP ....................... 48
  5.3 Issues and Challenges concerning General-Purpose INCP ....... 49
    5.3.1 NAT/Firewall Issues ................................... 49
    5.3.2 Traffic Diversion in Public Infrastructure ............... 50
  5.4 Communication Protocol ........................................ 54
6 Proxying for Mobile Devices ......................................... 57
  6.1 Motivation ......................................................... 57
  6.2 Supporting Clients Mobility in NCP Framework ................ 58
    6.2.1 Main Issues Concerning Devices Mobility ............... 58
    6.2.2 Architectural Solutions for Mobility Management ........ 60
    6.2.3 Concluding Remarks ................................... 63
  6.3 Reducing Energy Waste Through Intelligent Coordination Among Daily Used Devices . . 64
    6.3.1 Overview ................................................. 64
    6.3.2 Devices & Applications Control Logics ................... 66
    6.3.3 Possible Architectural Solutions ......................... 66
    6.3.4 Communication Interface ................................ 69
    6.3.5 Concluding Remarks ................................... 69
7 Software Framework & Implementations ............................ 71
  7.1 Basic Proxy Architecture ....................................... 71
    7.1.1 Communication with External Devices ................... 72
    7.1.2 Internal Communication .................................. 73
    7.1.3 Tasks Scheduler ........................................... 73
    7.1.4 Abstraction of Devices ................................... 74
    7.1.5 Internal Messages ........................................ 75
    7.1.6 Abstraction of Services .................................. 76
  7.2 The NCP Service .................................................. 76
Chapter 1

Introduction

Green technology is one of the main challenges for the today’s world to step into a sustainable society, that can fulfill the needs of today and has the capability to meet those of tomorrow. Main motivation for green technology is reducing the global CO₂ footprint [2] and reducing the utilization of natural resources for energy generation which may exhaust some day. Since green technology is the efficient generation, transmission and utilization of electricity; it finds applications in many sectors including networking. However, it mainly concerns reducing energy consumption or waste in the networking sector. Energy efficiency in networking sector became a high priority objective due to increase in energy price, continuous growth of Internet-connected edge devices, high data rates and increase in the number of services offered by telecoms and Internet Service Provider (ISP). Thus, green networking is a primary research topic for network engineers and operators to maintain the same performance and Quality of Service (QoS) of current Internet infrastructures while reducing the energy consumption [3].

This chapter addresses the importance of green technologies by presenting the ICT energy requirements today and forecasting it in the future. It also presents different strategies which can be adopted to reduce energy waste in the ICT. In particular, this chapter focuses on the Network Connectivity Proxy (NCP) concept as an optimal strategy to reduce energy waste due to idle devices. It motivates the need for the NCP concept and highlights its importance. Finally, this chapter briefly presents the main research contributions being addressed in this dissertation.

1.1 Importance of Green Technologies

With the great advancement in the field of Information and Communication Technology (ICT) and continuous growth of Consumer Electronics (CE), the aggregate energy consumption became a prime concern for the operators. The increasing energy consumption has three negative impacts: (i) economic, (ii) limited natural resources and (iii) environmental. Economic impact is obvious as the energy prices are continuously increasing. The natural resources used in power plants are also limited that must be utilized intelligently. Environmental impact comes from the power plants that generate the electricity. These power plants consume precious natural resources and most of them emit Green House Gases (GHG) in the environment [2].

In order to understand the importance of green technologies in the ICT sector, it is important to have knowledge of the ICT power consumption and its yearly growth rate. Fig. 1.1 presents the current ICT power consumption and forecasts the requirement in the future [4]. It can be observed that currently ICT consumes around 275 GW which is about 8% of the annual electricity production. However, the ICT power consumption will reach closer to 500 GW by 2020 which will be about 14% of annual electricity production. It is important to note that this power consumption only refers to usage of ICT equipment and does not account for the manufacturing process. Further, considering different ICT devices, PCs, network
equipments, data centers, TV and other devices have estimated annual power consumption growth rate of 7.5%, 12%, 12%, 9% and 5%, respectively [4].

The increase in ICT energy consumption can have several reasons including continuous growth of the ICT devices, high data rates and increase in the number of services offered by telecoms and ISP. The increase in energy consumption has also negative impact on the environment due to emission of GHG. Fig. 1.2 depicts the growth rate in GHG emissions for data centers, voice & data networks and end-user devices. It can be observed that the growth rate from 2002 to 2011 was 6.1 percent every year. The growth rate for end user devices and data centers is relative high and roughly around 6.1% and 8.6%, respectively. However, it is forecasted that the growth rate will reduce in the future to 2.3% and 7.1% for end user devices and data centers, respectively (or overall ICT growth rate will reduce to 3.8% from 6.1%). The significant decrease in growth rate for end user devices is due to expected energy efficiency gains due to green technologies and a shift to low power devices such as laptops and tablets.

1.2 Strategies for Reducing Energy Waste

Recent surveys revealed that most of the energy consumed by the ICT devices is wasted as they are mostly under-utilized than their full capacity or stay idle but powered up which causes considerable energy wastage [2, 5]. Thus, huge amount of energy savings are possible (accounting to Billions of Euro annually) if the devices performance is scaled up or down dynamically (and some parts of it are switched ON/OFF or use low power states) based on the current load conditions or idle devices can sleep whenever possible [6]. Thus, energy use can be considerably reduced by incorporating some energy saving strategies in the ICT devices.

Several strategies have been proposed in the literature to reduce ICT energy consumption. These strategies can be classified into three broad categories: (i) Re-engineering, (ii) Dynamic performance scaling and (iii) Smart proxying.
Re-engineering

Re-engineering focuses on introducing and designing more energy efficient elements for device architectures. It consists of new silicon (e.g., ASICs, FPGAs, network/packet processors, etc) and memory technologies for packet processing engines and adoption of pure optical switching architectures for replacing the current electronic based devices [2]. Thus, re-engineering focuses on suitably dimensioning and optimization of internal devices structure as well as reduction of intrinsic complexity levels.

Dynamic performance scaling

Dynamic performance scaling is based on varying the device capabilities (e.g., link bandwidths, computational capacity, power of packet processing engines, etc.) according to current loads and service requirements. Thus, dynamic performance scaling makes device energy efficient that consumes energy proportional to its utilization [6]. It reduces working rate by changing system clock frequency and/or by changing voltage of processors or by throttling the CPU clock. Thus, it can be performed by using two power aware capabilities of the devices called idle logic and dynamic scaling, both of which allow the dynamic trade-off between performance and power consumption. In detail, idle logic reduces power consumption by rapidly turning off some hardware sub-components in case of no activity and re-waking them up when some new activities are received. The wake-up instants may be triggered by external events in a pre-emptive mode (e.g., wake-on-packet) and/or by a system internal scheduling process (e.g., the system wakes itself up every certain time periods, and controls if there are new activities to process). On the other hand, dynamic scaling capabilities allow dynamically reducing the working rate of processing engines or of link interfaces. This is usually accomplished by tuning the clock frequency and/or the voltage of processors, or by throttling the CPU clock (i.e., the clock signal is gated or disabled for some number of cycles at regular intervals).

Smart Standby (Proxying)

Smart standby or proxying approaches smartly and selectively put network/devices portions into low power standby modes and wakes them up when it is truly necessary [7]. However, when a device (or a part of it) goes to standby state, its applications and services stop working and lose their network connectivity.
1 – Introduction

Since today's networks and related services and applications are designed to be continuously and always available, standby modes have to be explicitly supported with special proxying techniques able to maintain the 'network presence' on behalf of sleeping nodes/components.

In this dissertation, we focus on a specific smart standby technique called the NCP which can be designed to maintain network presence on behalf of sleeping devices such as PCs [8]. Notwithstanding that PCs architecture already include low power standby features, networking functionalities and applications have often interfered with their effectiveness. Due to inability of today's PCs to enter into standby state without losing their network connectivity, a growing number of PCs are continuously left fully powered-up, even though there is no user demand for their resources most of the time. This is mainly because of the applications and services running on the PCs and users demand to have their PCs remotely accessible at any time. The NCP concept encourages the users to put their PCs into standby state whenever possible without losing their network connectivity. This is the main research topic being addressed in this dissertation.

1.3 Importance of Network Connectivity Proxy

We have analyzed the different sources of ICT energy consumption both directly and from the literature review that significant portion of it is ascribable to end user devices and terminals. US Environmental Protection Agency (EPA) estimated that PCs alone consume about 2% of the overall US electricity requirement [9]. Beyond PCs, the number of other Internet-connected edge devices like smartphones, tablets, IP-phones, set-top boxes, game consoles and other network-based multimedia equipment are growing at a fast rate. Reducing the energy footprint of such devices would have a positive and non negligible effect on total energy consumption. Low-power states are effective mechanisms for energy saving, exploiting the fact ICT devices are often powered on but idle. However, for the purpose of maintaining network presence, low-power states are often disabled. To foster wide usage of low-power states, network presence should be maintained on behalf of sleeping devices.

1.3.1 Motivation for NCP

Recent survey [1] revealed that about 60% to 70% of the offices PCs are left powered on all the time even outside work hours just to maintain network connectivity. Most of network applications and protocols require full-time network connectivity to receive/respond to routine periodic heartbeat messages [10]. Present power saving sleep states such as Advanced Configuration and Power Interface (ACPI) S3 and S4 states cannot maintain network connectivity, thus network devices are left powered on to maintain full-time connectivity even when they are idle, which leads to significant wasted energy consumption [6, 11].

To have an estimation of energy consumed by PCs, Fig. 1.3(a) presents the energy consumption by different sectors of ICT. It can be observed that a large portion of power consumption is ascribable to PCs within ICT. Keeping PCs powered up all the time leads to large energy waste: they spend most of the times in the idle state, which consumes much more than sleeping or powered off states. Fig. 1.3(b) shows the share of energy consumed by different states averaged over a large number of enterprise PCs. Just to have an idea about why the time spent in the idle state is so prevalent, consider that studies by Lawrence Berkeley National Laboratory have revealed that about 60% of the office computers are left powered-up 24/7 with existing power management features disabled, 36% are turned off and only 4% have power management enabled [12, 13]. That means most of energy is simply used to make the devices present and available and not for actual computation [7]. Roughly speaking, a typical last generation PC can save on average up to 60 W from shifting from idle to sleep, which may lead to 400 kWh per year if it is expected to be available 24/7 and can sleep for three fourths of the time.

Some main reasons we investigated which forces users to keep their PCs powered-up even when idle are the following:

1. Remote access (e.g., SSH, RDP protocols etc).
2. Maintaining presence for applications running in the background such as Instant Messenger (IM) and Voice-over IP (VoIP) clients. Network applications require preserving connectivity for sending/receiving periodic presence or heartbeat messages to/from remote peers. Application is considered disconnected if the device fails to send/receive periodic heartbeat messages.

3. Some administrative management reasons such as to access PCs at night-time for applying software patches, performing back-ups and other maintenance operations.

4. Automatically retrieving and installing updates.

5. Quick availability, avoiding long boot times and lost of the working session every time the device shut down.

6. Sometimes, it becomes a habit to leave PCs powered-on when idle for no reasons.

The simplest solution to address the network connectivity issue is to design a low power entity that maintains the presence on behalf of high power devices and allow them to sleep when idle. This low power entity should be able to wake-up the device when its resources are required e.g., a remote access request is received. This low power entity can be named as NCP. The NCP service should be hosted on the network device which always remain powered up such as switch or router so that it don’t cause any incremental energy consumption. This dissertation is focused on the concept of NCP that has the capabilities to maintain full network presence for sleeping PCs. The NCP allows the network devices to sleep and still maintain their standing over Internet by responding to applications/protocols heartbeat messages on their behalf.

1.3.2 The NCP Potential: Achievable Benefits

The NCP is impersonating presence on behalf of sleeping devices. The real achievable benefits in term of energy savings depend on two parameters:

1. How much power is consumed in idle and sleep states.

2. How long the devices can stay in low power sleep state.

The first question about the power consumption in active and sleep states depend on the device. Its true that idle power consumption of device is much higher than when it is suspended. However, every device has different power requirements; thus the achievable savings may vary from device to device. Usually,
desktop PCs consume about 60-80 W while idle, whilst laptops and mini PCs draw about 15-30 W. Power drained in standby is a few watts for all kinds of devices (2-5 W). Power saving thus depends upon the devices under consideration and ranges from a ten to several tens of watts; if one also considered high-end equipment like servers, the saving would be even higher, up to more than a hundred of watts.

The second factor that determines the achievable energy savings depend on the NCP capabilities in terms of proxying different types of network traffic. This factor determines how long the devices can stay in low power sleep state. However, such estimation is more completed due large number of available network protocols and applications, some of them may not be supported by NCP. The devices use patterns also determine how long the devices can sleep. Many experimental measurements showed that PCs spend most of their time in the idle state; in theory, this brings the opportunities for large savings, but, in practice, this mostly depends on the length of the single periods of inactivity. A device could profitably sleep if the idle period is larger then the total time to suspend and resume it, otherwise no saving is possible but the responsiveness is affected anyway.

The design of NCP should help in maximizing the sleep periods by proxying basic network presence and management protocols and applications on behalf of sleeping devices. Most of the network traffic may be silently ignored by the NCP which is not necessary for the device during idle/sleeping periods (such as traffic addressed to other hosts, some multicast and broadcast traffic, browser related traffic etc). In short, if the NCP proxy only protocols and applications in which the users are interested, their devices can sleep for most of the idle times and provide large amount of energy savings [16].

1.4 Novelty and Contributions

The NCP concept has recently been proposed in the literature as an ideal strategy for reducing network energy waste [3, 10, 17, 18]. However, the concept faces several critical issues and challenges in its realization which we tried to address in the best possible way in this dissertation. In this section, we briefly present the main research activities related to the NCP concept, particularly related to capabilities and features of the developed NCP prototype and its performance evaluation.

1.4.1 Generalized Set of Behavioral Rules

The NCP is expected to proxy large number of protocols and applications which are continuously increasing. Proxying a protocol or application requires its detailed knowledge and the nature and content of the packets that it exchanges with the remote peers. Thus, it is very difficult to implement proxying capability for large number of applications and protocols. Therefore, we realized the need for a generalized NCP framework that can be suitable for any application or protocol. To aim this objective, we have implemented very generalized set of behavioral rules and features in the NCP prototype which can be requested by the heterogeneous devices. The features offered by the developed NCP prototype are enough to impersonates basic network presence on behalf of sleeping devices.

1.4.2 Deployment Flexibility in Local Network

We have incorporated deployment flexibility in the NCP software that is capable to work on on-board NIC, switch/router as well as on a standalone PC. Thus, the NCP software can be deployed on any network device depending on the number of client devices. For the simple home environment where usually single PC exists in the home network, the on-board or switch/router can be the optimal location to deploy the NCP software. However, standalone PC will be a good choice to deploy the NCP software in small or medium size organizations where the client devices range from few hundreds to thousands.
1.4.3 Flexible & Reliable Communication Protocol

The communication protocol is required for information exchange between the NCP and client devices e.g., power state notification, registration or withdrawal different behavioral rules at the NCP etc. To avoid any configuration issues, the communication protocol should support auto-discovery and seamless communication between the NCP and client devices. Thus, we developed a very flexible and reliable communication framework based on the Universal Plug & Play (UPnP) architecture. Some interesting features of the developed communication framework include auto-discovery, zero configuration and seamless communication.

1.4.4 NCP Support for Mobile Devices

It’s true that the desktop computers consume more power than laptops and other mobile devices. However, reducing energy waste in mobile devices can be more important as they operate on batteries power which have limited capacities. Thus, we extended the NCP concept also for mobile devices.

1.4.5 Coverage Beyond LAN Boundaries

The Local Area Network (LAN) usually consists of few network devices especially in the home environment. The real potential of NCP in terms of energy savings can only be exploited if it covers for very large number of devices globally. Thus, we shifted the NCP concept beyond the LAN boundaries. A single global NCP located anywhere in the world can provide huge amount of energy savings by covering thousands or even millions of client devices. The Internet NCP concept can provide additional benefits e.g., support for mobile devices, additional savings from the access link etc.

1.4.6 Performance Evaluation on Different Hardware Entities

The NCP performance especially the scalability depends on the device hosting the NCP service. Different devices have different memory and processing capabilities and have different limitations on scalability and overall performance e.g., latency, filtering activations, packet loss etc. Thus, we performed detailed evaluation of the developed NCP prototype on different low power devices including Raspberry Pi (pocket PC), Lantiq experimental home gateway, low power Jetway PC and normal laptop.
Chapter 2

Background and Literature Review

In this chapter, we present previous work from the literature in order to understand open issues and research challenges in the NCP concept. The NCP exploits existing power management facilities in the network devices in order to reduce or eliminate the idle energy waste. Thus, this chapter firstly address the available power management facilities in network devices and their basic characteristic in Section 2.1. Following this, Section 2.2 present previous work from the literature on the NCP concept. In particular, it describes different strategies used by the researcher in order to proxy different types of networking protocols and applications on behalf of sleeping devices. Afterwards, Section 2.3 presents available NCP implementations and their supported features from the literature. Finally, Section 2.4 extracts key open issues and challenges from the previous works in literature.

2.1 Power Management Facilities

The power management facilities particularly include the low power states supported by the network devices and the ability to wake them up on a network activity. Since, the NCP intelligently exploits the low power states available in the PCs in order to reduce the energy waste during idle periods; it is important to understand them and their characteristics before designing the NCP [19]. The ACPI proposed several low power management states (ranging from S0 to S5) which are available in today’s modern PCs [20]. Briefly, ACPI low power states are the following:

- **State S0**: During this state, the PC is fully powered up and consuming maximum power.
- **State S1**: This low power state has the lowest latency for the PC to fully wake-up. During this state, the CPU stop executing instructions and the processor caches are flushed.
- **State S2**: This low power state has also the low latency for the PC to fully wake-up. This state is somehow similar to S1 except CPU and system cache context is lost and the OS is responsible to maintain the caches and CPU context.
- **State S3**: During the S3 sleeping state, most of the system context (CPU, cache, chip set) is lost except RAM. This state is most commonly referred as standby or suspend to RAM state. The RAM remains powered up. The cache content is flushed to RAM and hardware restores some CPU configurations after wake-up.
- **State S4**: This low power state is most commonly known as hibernation or suspend to disk. It has the highest latency for the sleeping devices to fully wake-up. During this state, the content of the RAM is saved to non-volatile memory such as hard-disk from where the content is loaded after wake-up. This state provides maximum possible savings as it switch OFF most of the hardware components.
• **State S5:** This state is somehow similar to S4 state except the OS does not maintain or save any context. The system is considered as in soft OFF state and requires complete boot for wake-up. The power consumption during this state is close to zero. This state is different from the previous states as the system is not restored from the saved memory image.

The network devices use these ACPI states as power management features. If the power management features are enabled on the device, the OS in powered ON state (ACPI state S0) checks a time-stamp when the last user input was detected. The device enters into sleep state (ACPI state S3) when the inactivity timer expires. The inactivity timer is called as power management delay time in [21] in which the authors proposed a technique to shorten this delay timer. The short delay implies that the PCs will wait less time from last user input before entering into sleep state. However, the authors in [21] also concluded that shorter delay timer is usually annoying for the users and they usually encourage to have delay timer of at-least 60 minutes.

Besides the availability of low power states on the NCP client devices, it is important that the device cleanly and quickly suspends and resumes from low power states. Third-party software usually running on desktop PCs makes this task hard, as all applications should be ready to deal with system changes during the standby (for example, disk or interface removal). To build reliable systems, Microsoft has developed its OnNow technology, a set of design specifications for system hardware and software applications which let them participate in the system-wide power management decision process. For example, applications are notified when the system is going to sleep; they can take the appropriate actions (e.g., close open files or network connections) and even prevent the system to suspend if they are not ready to face this event. In the OnNow architecture, power management of individual devices is the responsibility of a policy owner in the OS; The policy will operate in conjunction with a global system power policy implemented in the operating system. In general, the power conservation policy strives to reduce power consumption while the system is working by transitioning amongst various available power states according to device usage. Microsoft and AMD jointly published a set of specifications defining the behavior of devices with respect to power management, including network devices as Ethernet and Token Ring adapters [22]. Such specification defines four power states and envisions wake-up from link events (for example, a link-on indication may trigger the transition to a power state where wake-up frames can be received) and from network packets (MagicPacket, pattern matching).

Further efforts from Microsoft enhances power management capability in Windows 7. It supports the following feature:

• **Wake patterns:** packets matching a given set of patterns can wake the host up.

• **Bitmapped patterns:** this provides a flexible way of specifying packets that wake the host up. Bitmapped patterns are made of a bitmap and a bit pattern; the bitmap is applied to incoming packets and the result is compared to the pattern; if a match is found, the NIC wakes the host.

• **Magic packet:** this is the standard MagicPacket used for Wake On LAN (WOL).

• **Network presence:** Windows 7 adds capability for ARP and Neighbour Solicitations (NS) offload: these are the ability of the network adapter to respond to ARP requests without waking up the host.

• **Low Power on media disconnect:** this is the ability of the network adapter to go to sleep when it is not in use, for example because the media is disconnected. In such situation, no frame can be received to wake up the host, thus the hardware can fall into a deeper sleep.

• **Wake on Wireless LAN:** this is superset of the Wake on LAN functions, as the NIC should be able to keep wireless connectivity while the host is sleeping.

Apple provides an even more advanced framework in its Wake on Demand solution, which is part of Apple’s energy efficiency policy for environmental protection. This solution provides a transparent
2 – Background and Literature Review

and zero-configuration way for waking up hosts when applications using the Bonjour protocol need to be accessed. Lot of effort has been undertaken to bring more intelligence into network cards, including proxying functions to maintain network presence. Indeed, many NICs today include on-board processors to allow the host to offload some frame-processing tasks (checksum calculation, packet fragmentation, etc.). Proxying of ARP requests in Ethernet NIC is envisioned in the DMTF Alert Standard 2.0 specification [23], some NICs currently support it, for example the Broadcom NetXtreme 57xx series controller.

Most NICs are currently able to wake their host up upon receipt of a magic packet (known as WOL feature); further, some NICs are also able to wake up on direct packet matching. For example, Intel Pro 100 series and Realtex PCIe FE Ethernet adapters can wake up the host when they receive a packet with the host’s IP address, or an ARP packet addressed to the system. Intel provides Remote Wake Technology (RWT) on some desktop boards; it allows RWT-enabled applications to wake up remote hosts at home, including media and streaming servers.

2.2 Related Work

The interest towards efficient use of network devices first came out during the mid ’90, with the rapid proliferation of personal computers for both commercial and personal applications; some studies argued about these device’s being one of the major cause of electrical consumption in buildings, mainly because of the habits of leaving them always powered up [11]. Beside the number of different proposals and the large number of research articles discussing the issue, the term NCP only came out recently [18].

The network connectivity is sought as one of the main hindrance to power management; early studies already proposed the use of a network proxy to maintain network presence for sleeping devices; the main focus was on TCP, although other management and control protocols were also considered [8]. Here, the main idea was to exploit the ‘zero window’ advertising feature of TCP to inhibit the remote peer from sending any data during sleep periods. Then, other approaches to deal with TCP connections have been proposed. In the ‘Green TCP’ proposal, a new option is included in the protocol to advertise the next power state (standby or wake up) to the remote peer [24]. Another solution was splitting TCP connections by inserting a ‘shim’ layer between applications and the legacy socket interface [16]. The shim layer sets up and closes TCP connections at each power state transition but hides this behavior to applications, which indeed always see a connected socket; further, this layer wakes up the sleeping peer whenever an application sends data. Finally, the use of an external proxy was again envisioned for splitting TCP connections at a SOCKS proxy [18]. Local hosts connect to the SOCKS server and indicate the remote peer to which they wish to connect; then the SOCKS server establishes another connection with that peer. The SOCKS proxy relays data between the two connections; the local host tears down the connection with the SOCKS server when it goes to sleep while the connection between the proxy and the remote peer remains active. The SOCKS server also buffers packets when the local host is unavailable. Main drawbacks in this approach are scalability issues, incoming connection management, address management concerns, data transfer synchronization, packet losses and privacy concerns.

Most of the proxy implementations till now have focused on TCP and management protocols like ARP, ICMP, IGMP. However, there have been a few proposals for proxying both high-level protocols as UPnP [25] and specific applications as Gnutella [26, 27] and Jabber [28]. A further step was done in Somniloquy, where ‘stubs’ for specific applications can be added to the base framework [29]. In Somniloquy [29] a secondary low-power processor is used to maintain network presence while the device is sleeping. Network state (IP address, ARP table, DHCP lease, SSID for wireless networks, etc.) is transferred to the secondary processor when the device is going to sleep; from now on, the secondary processor behaves just like it was the main device. When active, the secondary processor is in charge of waking up the main host on receipt of packets matching a given set of filters (including header and payload fields) and running application stubs. Packet filters were defined for applications that require resources of the main host (telnet and SSH, remote file access, remote desktop access); filtering criteria included transport ports and patterns in the
application payload. Stubs are tiny code that perform routine operations mainly aiming at maintaining network presence for specific applications. The prototype included stubs 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). The concept of using application specific stubs is also used in [30]. The authors in [30] proposed using virtual machines running on a standalone PC to maintain presence on behalf of sleeping devices. When a client device goes to sleep, its specific image is loaded in the virtual machine. This approach faces issues with the resources required to run a virtual machine. Therefore, the approach hardly scales to limited number of client devices.

Along with the idea of proxying different applications and protocols, there are also several proposals for the device hosting NCP service such as on-board NIC, switch/router or standalone PC. The main difference between them is where and how the communication between the client devices and the proxy happens: on the network, in case of external proxy (switch/router or standalone device), or on internal buses, in case of embedded devices (e.g., on-board NIC). Apart preliminary works that emulated the internal device with an external PC connected on a dedicated interface [9], all later proposals have followed the same architectural design of embedding the NCP function within the device’s NIC. Given the unquestionable difficulty in developing firmware for real hardware, several experimental and research tools have been used to this purpose that feature an embedded microprocessor [26, 29]. If the NCP service is provided by the on-board device NIC, there will be no need to develop any special communication protocol or paradigm for communication with the client device [29]. Modern NICs have increasing processing and storage capability; further, they consume very low power and are expected to be always on (perhaps with a lower rate) for waking up the device by network packets when it sleeps. Placing the NCP service in NICs is therefore a good solution and brings other benefits: for example, the same IP and MAC addresses are being used for proxy operations without any network change. However, it may come out that NIC’s memory and computation power are not enough to implement a complete NCP service covering many applications and protocols; In this case, new NIC designs should include additional storage and computation capability (CPU, packet classification hardware) to be activated just when needed or additional hardware should be integrated in the device, as proposed in the Somniloquy project [29]. Many researchers have also addressed the design of NCP for smart NICs [9, 10, 31]. On the other hand, a communication protocol will be required for communication between the NCP and client devices if the NCP service is deployed on a standalone entity such as a dedicated PC or network switch/router. The network switches and routers are likely to be always powered up; thus they are another suitable location for the NCP service [32]. Indeed, switches/routers often provide hardware accelerators for packet filtering and classification, greatly simplifying the most resource-starving tasks of the NCP. Deploying the NCP on a standalone PC can only be beneficial in terms of energy savings if it is covering for large number of client devices. However, a number of mini or micro PCs are already commercially available with very low power consumption and enough memory and processing capabilities to enable the NCP service to proxy large number of client devices, applications and protocols [8, 33].

Since the main objective of the NCP is to reduce idle energy waste; it is important to understand whether proxying is indeed necessary and to what extent it could help in reducing energy waste. Some evaluations have been carried out in the literature to analyze the network traffic in realistic scenarios and to determine the amount of achievable savings from the NCP concept. The very first attempt in this direction was done by K. Christensen et. al., in [9]. The authors have analyzed the network traffic for the university undergraduate dormitory over the period of 24 hours. The authors evaluated the busy and idle intervals for the computers in dormitory and concluded that most of the computers have very long idle intervals; however, the average packets inter-arrival time was just few seconds. This information is quite useful in order to determine the time-out policy to automatically put the computers into sleep state. It was observed that 77% of the packets have size up to or less than 1KB, about 8% have less than 2KB and so on. The authors also breakdown the protocol messages (e.g., NetBIOS, ARP, RARP, DHCP, HTTP, etc) for a single computer in order to classify them according to the treatment/action required from the NCP during the sleeping period (e.g., ignore, simple response, wake-up, periodic heartbeat, state update etc) [16]. The authors further estimated the possible energy savings from the NCP concept for both residential and commercial environments by
considering that 20-70 percent PCs are always powered ON in residential while 50-70 percent PCs are always powered on in commercial environment. A more thorough investigation was conducted among Intel’s employees in [7] over the period of 5 weeks. The data was collected from both desktops and laptops in the home as well as office environments. This study also estimated idle states when no user activity (mouse or keyboard input) is detected for a given period; coarse results agreed there are long inactivity periods. Breakdown of protocol messages showed that most of the network traffic is due to broadcast and multicast frames for address resolution, service discovery and networking issues; however, most of them could safely be ignored or answered by simple response functions. If all the broadcast and multicast traffic is ignored in the home and office environments, the possible sleep period for a computer is about 80% of idle time in home while approximately 53% of idle time in office. Unicast traffic is more complex to evaluate, as mainly depends upon specific (often unknown) applications and user’s preference; in general, outgoing connections are responsible of most traffic and many of them are set up in idle periods (thus would not be present or would be delayed and grouped together during sleeping period). Finally, an evaluation of benefits in terms of sleeping time was also reported by authors for four proxy’s architectures (each proxy is considered to have different capabilities in terms of proxying network traffic), with different hypothesis about transition times for the low-power state.

Beyond the concept of proxying, there also was the interest in embedding management of intermittent connectivity into the Future Internet technologies. To this purpose, the concept of Selective Connectivity [17] was introduced to indicate several degrees of network connections, ranging from the disconnection to full connection. Selective Connectivity means a device may only be willing to carry out a limited number of network tasks; these could be basic management operations (answering ARP and ICMP requests, renewing DHCP leases), keep-alive routines, wake-ups on incoming connections, partial implementations of applications features. A number of basic architectural elements were proposed: assistant for sleeping devices, power management awareness, proxyable and limbo states for suspended applications, flexible control by users on connection management, abstraction of applications’ behavior for generic operations by assistants and secure and trustworthy infrastructures. In this context, the NCP concept fits very well the role of assistant.

Maintaining applications presence for mobile devices is still uncovered and disregarded issue. No NCP solution has ever been addressed that takes into account the mobility of client devices. However, there are some other strategies which have been proposed in the literature for improving the battery life of mobile devices. Most of the works targeted software based strategies [34–41]. Many researchers focused on traffic shaping strategies through a local proxy in order to create idle gaps during which the network interface can stay in low power mode [42–45]. Some further work targeted thin client-server approach that offloads the heavy computational tasks to the cloud and reduce load on the system resources (such as CPU, memory etc) which in turns improve the battery life [46–48]. However, the NCP concept is quite important for mobile devices and must be properly addressed. It will help in improving battery life by reducing energy waste due to background applications which usually consume major share of battery life through utilizing resources such as 3G/4G data connections or WiFi, CPU, memory and may be some built-in sensors (GPS, compass, gyroscope, proximity or health related sensors etc).

2.3 Implementations

There are few implementations available for the NCP concept with limited capabilities. One of the initially developed prototype focused on the proxying of TCP connections on behalf of sleeping devices [8]. The prototype was being developed for Linux, but there was not a clear understanding of the real implementation status. The Green TCP/IP implementation dates back to the same years; however, it introduced a new TCP option and does not use any proxy [24]. More recently, a stand-alone proxy was developed with the aim of emulating operations on a NIC [9]. It was able to answer PING requests (ICMP echo request messages) and to wake up the device when a new incoming TCP connection is seen (TCP SYN packet). Further, it
sent Gratuitous ARP messages to bind the IP address of the covered device to that of the proxy’s hardware. Network proxies were also implemented for specific applications: the UPnP [25] and the Gnutella peer-to-peer file sharing [26]. The former developed a stand-alone proxy for Windows OS, while the latter designed a proxy for a NIC and developed it on an Ethernet development kit. Both of them are quite simple realization of the NCP concept, as they process a subset of control messages and implement a predefined behavior.

A more general framework was implemented by exploiting SOCKS [18], using a switch proxying architecture (the NCP was located on a low-end SOHO router). In this implementation, the NCP answers ARP and ICMP packets and keeps TCP connections active for sleeping devices. SOCKS allows splitting TCP connections, thus the NCP could proxy it in order to prevent timeouts and disconnection (indeed, that was discussed but not really implemented). This NCP behaved quite simply: it buffered packets addressed to sleeping devices and forwarded them when the destination woke up. The prototype was tested with clients for SSH and IM applications. Interestingly, this work pointed out that a relayed connection has very poor performance as it is entirely managed by the control processor instead of being accelerated by the proper hardware components.

Another implementation is the Turducken [31] where different hardware tiers are used with different processing power and energy requirements. Lower layers check network availability and wake up to perform synchronization tasks; they wake up their upper layer when there are complex tasks that they cannot accomplish by themselves. The Turducken framework was designed to proxy simple and periodic operations by low-power devices, thus allowing higher tiers to sleep and save energy; it did not maintain network presence and sessions on behalf of running applications. However, there is no reason to believe this framework could not be extended for including NCP operations as well.

A stand-alone proxy implementation is presented based on Click modular router in [7], which shares the same broadcast domain as its client devices to sniff all network traffic. It included only a limited number of actions that does not require any state exchange with clients: it wake the client device up on incoming TCP SYN requests or NetBIOS Name Queries, answered ARP requests and dropped all other traffic. Further, it buffered packets that required the client device to wake up. No communication interface was developed between the proxy and the clients: power state of devices is inferred by querying them with ARP requests and waiting for possible responses.

Apple provides a Bonjour Sleep Proxy that runs on latest Apple’s network devices and equipment (access points, backup and web TV servers) and Mac OS. Devices running the Bonjour protocol automatically discover the Bonjour Sleep Proxy and registers its services with it. Once the host goes into a sleep state, the proxy answers multicast DNS queries on its behalf and wakes it up when a TCP/UDP packet addressed to any of the advertised Bonjour services is seen. Further, the Bonjour Sleep Proxy answers ARP requests on behalf of sleeping host by providing its own MAC address; this way it can get all successive connection requests. Yet being developed by Apple, Bonjour is an open source project and is freely available on Apple’s web site; in particular, a version for Windows OS is present.

A somehow more advanced and meaningful implementation of the NCP concept is the Somniloquy [29], where a secondary low-power processor is used to maintain network presence while the device is sleeping. For developing purpose, the secondary processor was placed on a USB Gumstix, although the architectural design envisaged it to be on the NIC. Network state (IP address, ARP table, DHCP lease, SSID for wireless networks, etc.) is transferred to the secondary processor when the device is going to sleep; from now on, the secondary processor behaves just like it were the main device. When active, the secondary processor is in charge of waking up the main device on receipt of packets matching a given set of filters (including header and payload fields) and running application stubs. Packet filters were defined for applications that require resources of the main device (telnet and SSH, remote file access, remote desktop access); filtering criteria included transport ports and patterns in the application payload. Stubs are tiny code that perform routine operations mainly aiming at maintaining network presence for specific applications. The prototype included stubs for web downloads, bittorrent and instant messaging; further it was tested with filters for remote desktop access (RDP), remote directory listing and remote file copy (SMB) and VoIP call (SIP).
The authors also carried out many performance evaluations including energy saving, CPU load and memory usage, application-layer latency.

### 2.4 Key Open Issues and Challenges in NCP Concept

Several architectural solutions have been proposed for the NCP; however, there is not a clear understanding whether one of them is winning over the others or whether the optimal solution should be tailored to each specific use case. Based on the previous works from literature, several open issues and challenges have been highlighted in this section. It particularly presents basic challenges concerning the proxying capabilities of the NCP and the characteristics of the client devices and the device hosting the NCP service [49].

#### Memory and Processing Power Requirements

The memory and processing capabilities of the device hosting the NCP service is the performance and scalability limiting factor. Proxying many client devices or complex applications will require lots of resources and thus powerful devices could be needed. Answering ARP and ICMP echo requests, renewing DHCP leases and waking up client devices require very few processing power and information status to be stored at the NCP; however, proxying TCP connections and network applications increases the amount of resources needed. Another important issue concerns inspecting all packets against pattern matches, which is a very processing-intensive task that usually cannot be accomplished by low-end processors.

The ever increasing processing power (both for packet handling and generic purposes) available on many modern adapter cards makes the solution of embedding NCP functions into the NIC appealing in the future. Even, offloading the whole TCP/IP suite to the NIC may become feasible. However, that would require a CPU processing power similar to that of the main client device, to avoid slowing down the performance for bulk transfer of data. If the amount of required resources by NCP rises, deploying the NCP into NICs becomes infeasible. It's true that a smart NIC with powerful hardware may consume very low additional power than a stand-alone low end network equipment (switch/router). However, if the number of devices is large, the total power consumption of their NICs may exceed that of a single entity (switch/router). On the other hand, offloading TCP/IP only during sleeping periods may become too onerous in terms of synchronization between software and hardware. The issue is therefore which functions could be delegated to the NIC given the limited power of the on-board processors and the small memory available.

In short, current NICs and network switches don't have enough memory and processing power to implement complete standalone NCP covering for many applications and network devices. Standalone NCP can solve this problem but it will increase the energy consumption as NCP is required to be powered on all time. Thus, standalone NCP is only useful if it is covering for large number of network devices. The power consumption can be reduced if the standalone device hosting the NCP service can also sleep during idle times and woken up by client devices before entering into sleep state.

#### Applications Independent Proxy

Although the number of network protocols is rather limited, there is a very huge and ever increasing number of applications that could take advantage from the NCP service. Every application has its specific heartbeat/keep alive messages that need to be received and responded at periodic intervals in order to maintain presence to remote peers over Internet. Despite the fact a few frameworks have been developed for specific applications, there is a general agreement the NCP service should be conceived as general as possible, with the capability of being dynamically configured with any kind of custom behavior for any application. E.g., previous work in literature used application stubs in Somniloquy [29] which is highly application dependent. Each application needs to have its own stub which is only possible for open source applications. Due
to very large number of network-based applications and their continuous increase, it would be practically impossible to implement application stub for each of them (especially not possible for closed source applications). Thus, the need arises for suitable languages to code the behavior expected by each application: pattern matching for incoming packets (headers field and body content), actions to be undertaken for each match (discard, respond, forward, wake up), periodic tasks (keep-alive notifications). Also, proper interpreters are needed to run stubs created with these languages on the NCP, for example virtual machines or something like that.

**Coverage over Different Networks/Subnets**

Most efforts until now have been devoted to local scenarios, where the NCP is located close to the covered devices (within the same device or in the same LAN). The NCP can only cover for network devices that are located in the same network/subnet. When the NCP receives a packet that requires the sleeping device’s resources, it wakes it up using a WOL packet. The WOL packet contains the target device MAC address in its payload. However, greater energy savings are possible if the NCP covers for large number of devices. Obviously, this objective will be achieved if the NCP coverage exceeds beyond the LAN boundaries. This approach would be really effective for single-homed sites (as usually SOHO users are), because the operation of the remote proxy will prevent forwarding any unnecessary traffic to its final destination, and that will enable shutting down network segments (mainly last mile) during long periods of inactivity, turning out into great savings for telcos operators. However, the major challenges in expanding NCP coverage beyond LAN boundaries are related to traffic diversion and Firewall/NAT issues which may interrupt communication between the NCP and client devices.

**Support for Mobile Devices**

Largely availability of ever smaller and amazing devices, as well as the almost ubiquitous coverage by wireless networks, have shifted large part of Internet traffic towards mobile devices. Laptops are today superseding desktops in many environments, and it is not infrequent that many employees bring their business computers to home (and vice versa); further, most tablets and smartphones are currently connected permanently to the Internet. Power saving for mobile devices is even more crucial than for fixed infrastructure because of their need of running on batteries most of the time. Mobility of asleep devices is a further challenge for NCP design; here, the main issue is inability of sleeping devices to notify the NCP about their movement. Indeed, no trouble should arise if the NCP keeps proxying the device on, but it can no more wake it up because it does not know how to get to it. Another issue related to mobility management is the non continuous radio coverage during the movement, and the following concept of delay tolerant networking. In this context, the NCP should be aware that the covered device may fall out of network coverage; if this happens while it is asleep, the NCP cannot wake it up but should continue answering on its behalf, whilst if this happens when it is awake, the NCP should autonomously begin proxying it until network connectivity is restored.

**Communication protocol**

More versatility is usually required for NCP to let client devices choose whatever they want to be proxied. Moreover, power state of covered devices should be easily accessible by the NCP. Thus, a standard communication interfaces should be available for configuring NCP services and notifying of power status changes; automatic discovery, zero-configuration and flexibility are key issues for this purpose.
Client Device Power State Awareness

The NCP should be aware of the client device presence and its power state. It is usually very difficult to know about the device presence in the network and its power state without directly communicating with it. Thus, the client devices should be responsible to keep NCP updated about transitions in their power state over suitable communication protocol. Further, the NCP should be able to know if the device permanently leaves the network and stops proxying for it.

Network Traffic Diversion

One of the main challenges for the NCP concept is the ability to get all traffic addressed to the covered devices when they are sleeping. This is implicitly fulfilled if the NCP lies in the network adapter. In single-homed sites (as most Small Office/Home Office (SOHO) scenarios), placing the NCP on the Internet access gateway could appear an equivalent solution; however, that would not work for internal devices, whose traffic is not forced to cross the gateway. Thus, an efficient traffic diversion strategy is required to divert network traffic intended for sleeping devices towards the NCP when they go to sleep and vice versa when they wake up.
Chapter 3

Network Connectivity Proxy

The NCP is quite useful approach to reduce energy waste due to idle network devices by allowing them to sleep and impersonates their virtual presence. Since the NCP operates on behalf of sleeping devices, it is necessary to understand what the devices usually do when awake. Based on the characteristics of an Internet connected device, this chapter firstly addressed (in Section 3.1) the activities that NCP needs to perform when a device enters into sleep state. The NCP can operate in different ways in the network based on whether it communicates or not with its client devices. Based on this, Section 3.2 addresses the possible modes of operation for the NCP service and highlights the benefits and limitations of each. Afterwards, Section 3.3 points out basic requirements that need to be addressed before designing the NCP. Following this, Section 3.4 presents possible architectural solutions for the NCP based on device hosting hosting the NCP service. Each architectural solution considers the NCP service being deployed on a different type of network device and has different coverage (e.g., NCP for only LAN devices or all devices over Internet).

3.1 Overview

The NCP uses a low power entity that can maintain the network presence for high power devices and smartly make the high power devices to transition into low power sleep and active modes [49]. It encourages the devices to enter into low power modes during idle periods that would otherwise be left powered up by the users only to maintain the network presence. The NCP allows automatic waking up of the devices from low power modes only when it is truly necessary [16]. Thus, considerable amount of energy savings are possible if the network devices can sleep when idle.

At present, the ACPI standard sleep modes such as S3 (standby: suspended to RAM) and S4 (hibernation: suspended to disk) states are not suitable to maintain network connectivity. As a result, these power saving features are usually hindered to be adopted by the users even during idle periods. Power consumption of devices can be reduced through slow clock (dynamic rate adaptation) or stopped clock (sleep) operation. Unfortunately, remote device access in stopped clock mode is not possible as it is effectively sleeping [9]. The NCP is designed for this purpose that implements key network presence capabilities and maintains network standing for the devices when they enter into sleep state [10]. The NCP wakes them up only when their resources are required by using some wake-up technology. The NCP needs to address the basic network connectivity requirements on behalf of sleeping devices in order to encourage users to utilize low power standby state during idle periods. Thus, it is important to understand the characteristics of a fully connected network device [18]. The NCP needs to address the following basic issues or characteristics of an idle network connected device:

1. Responds to ARP requests in order to allow MAC level communication in the network.
2. Maintains its IP address. In case of DHCP, it periodically generates DHCP lease requests in order to maintain its IP address.

3. Responds to network diagnostic messages such as PING that uses ICMP to detect the presence of device over the network.

4. Maintains application-level reachability and allow new TCP connections to be established by responding to TCP SYN packets.

5. Maintains existing TCP connections.

6. Generates/responds to the network-based applications and protocols periodic heart beat messages.

The generic functional diagram for NCP is shown in Fig. 3.1. The client device transfers its network presence to the NCP before it enters into sleep state. The NCP starts functioning by generating/responding to the routine network traffic on behalf of sleeping client device. It wakes up the sleeping client device when a packet or new TCP connection request is received that requires the device resources and transfers the presence state back to the device.

![Diagram of NCP functionality](image)

**Figure 3.1.** NCP overview: 1: client device is awake and performs over Internet, 2: client device is going to sleep and transfers its state to the NCP, 3: the NCP impersonates the client device’s presence, 4: NCP wakes-up the client device when required and step 1 repeats.

Maintaining network presence on behalf of sleeping devices mainly implies preserving reachability (ARP requests, NetBIOS name resolution, DHCP address leases), device manageability (answering control/management protocols as ICMP and SNMP), application reachability (responding to new connections and packets addressed to transport-layer listening ports) and application state (keeping alive transport connections and sending application-specific heart-beating messages). Summing up, network presence essentially includes three main issues:

1. Link-layer presence, namely maintaining physical connectivity (synchronization, state, topology, etc.).

2. Network-layer presence, which is active participation in networking protocols to maintain end-to-end reachability.
3. Application-layer presence, which includes all messages and functions to carry out to let other peers believe specific applications are active and ready to answer.

The NCP manages network connections on behalf of devices in low power states (usually indicated as sleeping devices); they enter these power states either automatically (based on the expiration of an inactivity timer or a more sophisticated predictor) or manually (e.g., by closing the laptop’s lid). Basically, packets addressed to sleeping hosts can be classified according to the treatment they need:

- No response required, i.e., packets that should be discarded/ignored by the OS (multicast, broadcast, port-scans, bridging, routing, etc.).
- 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, Simple Network Management Protocol (SNMP) requests, and so on.

The NCP performs four basic types of functions [3]: responding to routine requests (e.g., ARP and ICMP messages), automatically generation of routine protocol messages (e.g., keep-alive or hear-beating messages), waking up the device when its functions or resources are strictly necessary (e.g., hardware facilities for VoIP, access to data stored on disk, applications running on that host) and ignoring all other messages. Devices should instruct the NCP about the network services they need while sleeping and then should notify the NCP when they are going to sleep; the NCP operates on their behalf until the need to wake them up occurs or they autonomously exit the standby state.

3.2 Modes of Operation

There are two different ways to operate the NCP (i) uncoordinated/invisible proxying and (ii) coordinated/cooperative proxying [25].

3.2.1 Uncoordinated or Invisible Proxying

Uncoordinated or invisible proxy doesn’t advertise its presence in the network. It invisibly works in the network and guesses about the client device’s power state from the traffic analysis. When the client device goes to sleep, proxy invisibly starts functioning for it. The invisible proxy requires client device traffic to directly pass through it as it guesses device’s power state from the traffic analysis. It does not require any change to client devices and/or protocols or applications. Invisible proxy might appear as an optimal solution; however, it poses a number of limitations and practical constraints. As the device are not aware of the NCP, the NCP has to infer all information it needs to do its work: it has to detect the current power state of devices and determine when starting/stopping its operation; further, it has to choose which routines are necessary for the sleeping device. Since it doesn’t communicate with the client devices, it cannot verify if the client devices permanently leave the network while sleeping.

Deploying the NCP service on on-board device’s NIC can be a good choice based on the characteristics of invisible proxying. The on-board NCP will move along the device itself and can easily track its movements. Further, all the device traffic will pass directly through the NCP and the objectives of invisible proxy can be easily achieved. However, the main limitation will be the available memory and processing power on the NIC to support proxying for many applications. Deploying the NCP service on network switches/routers can be another possible design choice as they always remain powered up in the network, the NCP will not
cause any increase in the network power consumption. It is important to design the network topology such that it allows all the device traffic to pass directly through that switch/router where the NCP service is running. However, it will be challenging to detect device presence in the network after it enters into sleep state. Better than device NIC but still most of the routers/switches have limited memory and computational power which will make difficult to implement complete NCP covering many applications for many devices. To solve the limitation of memory and processing power, the NCP service can also be deployed on a standalone device e.g., PC in the local network. Although huge amount of memory and processing power will be available but fulfilling requirements of invisible proxying will be much more challenging such as accessing traffic of client devices or tracking their presence in the network after entering into sleep state. Further, a standalone device for NCP service can only be beneficial if it is covering for very large number of client devices.

3.2.2 Coordinated or Cooperative Proxying

To overcome the limitations of the invisible proxy, the cooperative NCP framework can be designed. Coordinated or cooperative NCP announces its presence in the network and communicates directly with the client devices. In cooperative proxying, NCP and client devices exchange two kinds of messages: application specific and wakeup/sleep messages. Application specific messages are used to register client device’s applications and services at the NCP. These contain information about the application connections and application routine messages. Wakeup/sleep messages are required to trigger the NCP when client device goes to sleep and wake-up the client device when NCP receives a new connection request or a packet that needs device resources. Thus, when a client device goes to sleep based on power saving policies or external user commands, it informs the NCP about the power state change and transfers all proxyable information like its MAC and IP address, port number of all open sockets and connection sessions information, TCP connection sequence and acknowledge numbers, etc to the NCP.

Unlike invisible proxy, the cooperative proxy can be located at any place in the network and requires a software on the client device that can be used for communication with the NCP using a suitable communication protocol. Thus, cooperative NCP can easily know when to proxy and what to proxy in any kind of deployment (on-board NIC, switch/router or a standalone PC). The available memory and processing power on the NCP service hosting device can be a factor decided based on the network size (number of client devices). On-board NIC and switch/router can be suitable locations for the NCP service if proxying is required for few applications and/or client devices. Standalone PC can be a good choice if the NCP is expected to cover for large number of client devices and applications.

3.3 Key Requirements

The concept of NCP brings a number of requirements and expected features for both the NCP and its client devices [18]. In the following, we address the basic NCP, client devices and communication requirements.

3.3.1 Requirements for the NCP

The NCP has some general requirements in order to allow network devices to sleep and save energy and still maintain their standings over the network. Some basic requirements of the NCP are the following:

1. It should always remain fully powered up and present in the network to take over any client device presence as soon as it enters into sleep state.

2. It should be in the same network/subnet as the host for which it is covering. This will make easy for the NCP to discover the power state of client devices and waking them up using suitable wake-up mechanism.
3. It should be present on network devices which always remain powered up such as routers/switches so that it will not cause any significant incremental network energy consumption when all the client devices are awake.

4. The NCP must be aware of the power state of its client devices, just to know whether it should process packets or not. Moreover, the NCP should be able to distinguish whether a device is entering a temporary low power state or it is permanently leaving the network; in the second case, no more operations should be undertaken by the NCP on behalf of that device. However, this requirement is quite challenging for invisible NCP but can easily meet in cooperative NCP design.

5. It should be able to act on behalf of sleeping client devices by impersonating the presence of their IP addresses.

6. It should be able to handle simple network discovery and management protocols such as ARP, ICMP, DHCP etc on behalf of sleeping client devices.

7. It should be able to preserve existing open TCP connections on behalf of sleeping client devices by generating and responding to control messages.

8. It should be able to allow new TCP connection requests for sleeping client devices.

9. It should be able to buffer incoming packets to give enough time for the sleeping client device to fully recover from their stand-by modes. Buffers can also be used to increase the client device sleep time specially during the P2P files download process in cooperative NCP design. When the NCP buffer is almost full, it wakes up the client device, transfers data to it and then the client device goes back to sleep state.

10. Generate/Respond to routine application/protocol messages as required by the application/protocol.

11. Given the large and increasing number of applications and protocols that may benefit from NCP presence, the NCP should be independent of applications and suitable Application Programming Interface (API) should be devised to automatically set up the desired behavior on the NCP.

12. A very critical issue with the NCP is security: the use of a network proxy should not introduce any new pitfall or weakness into existing protocols and applications; further, user privacy should be preserved.

In short, the NCP requirements concerning each network layer are the following:

- **Link layer:** The NCP should lie in the same broadcast domain as its client (this is necessary in order to answer ARP requests).

- **Network layer:** The NCP must use the IP address of sleeping client devices for packets sent on their behalf and should operate even if behind Firewalls and NATs.

- **Transport layer:** The NCP must detect packets addressed to listening sockets on the sleeping client devices, it must hold TCP connections with remote peers and it must prevent them from closing.

- **Application layer:** The NCP should maintain application presence by answering routine application messages from remote peers and by generating routine messages as if the local application were running. Moreover, its hardware should provide enough resources to deal with application requirements as data processing and temporary buffering.
Apart the basic requirements for the NCP, reference architectures and implementations should also care about reliable, stable and robust operation. Failure in the NCP service could result in losses of behavioral rules, thus leaving client devices sleeping forever if no periodic wake up is scheduled by the same client devices. To this purpose, redundant operation and back up of rules should be considered secondary feature in NCP design.

### 3.3.2 Requirements for NCP Client Devices

There are also some basic requirements for the client devices in order to be proxied by the NCP.

1. The client devices must support a sleep state with minimal power consumption.
2. The client devices must be able to exit from low power state when triggered by an external method in a very short time span (no more than few seconds).
3. The client devices must support a remote wake up method so that the NCP can wake it up whenever necessary.
4. The client devices must be able to set and unset the desired behavioral rules at NCP for different applications and protocols with highest degree of flexibility and security. However, it can only be possible in cooperative NCP design.
5. The client devices must be able to resume suspended applications and protocols and in shortest possible time after successful wake-up by the NCP.
6. The client devices should be able to detect if some applications are actively performing over Internet and prevent themselves from entering into sleep state.
7. The client devices should have unique identity and should be accessible from NCP at any time.

### 3.3.3 Communication Requirements

In the design of cooperative NCP, a communication protocol is required for communication between the NCP and client devices (for discovery, rules registration, rules de-registration, power state notifications etc). The following are some of the basic preferred requirements for the communication protocol:

1. The communication link should be established in shortest possible time and should have low latency and overhead.
2. Autonomous seamless communication requiring no or minimal configurations can be the optimal choice. This requirement can be easily met if both the NCP and client devices lie in the home network, however special techniques or strategies need to be adopted for communication over Internet.
3. Both the NCP and client devices need to have unique identity in order to communicate. This requirement can be easily met if both the NCP and client devices lie in the same local network, however NAT and Firewall issues needs to be addressed if the NCP is located anywhere else over the Internet.
4. The communication protocol should have proper security measures. This is especially important when the communication between the NCP and client devices pass through public infrastructure.

### 3.4 NCP Deployment Scenarios

There are several possible deployment scenarios for the NCP. Each scenario has its own benefits and drawbacks that we will explore in this section. Further, the basic NCP requirements or internal architectural components may vary slightly depending on the deployment scenario.
3.4.1 On-board or Self Proxying Scenario

In the on-board scenario, the NCP runs on a device’s peripheral (network adapter, USB stick, etc.). In this architecture, the NCP covers a single device; the communication with the main device may exploit some OS’s API or the TCP/IP suite. With the second approach the device hosting the NCP service is forced to implement a full networking stack; however, this case can be brought back to the LAN scenario. In the on-board scenario the OS could be charged of mutual discovery of the NCP service and client device. The on-board scenario is the simplest architectural choice; it allows a device to manage its network presence without relying on any network infrastructure. The basic scenario is depicted in Fig. 3.2. Further, as the NCP would be tightly bound to the device, few privacy concerns would arise and mobility would be a minor issue.

The on-board NCP can be invisible and can be cooperative. The NCP has the advantage of sharing with the device the same MAC and IP address. Modern NICs have an auxiliary power source from the motherboard which allows NICs to remain powered up even when the device is sleeping. NIC consumes very low power, thus on-board proxying will not cause any significant increase in network power consumption. Since NIC has limited memory and computational power, thus it will be difficult to implement a complete standalone NCP covering for many applications.

![Figure 3.2. On-board scenario. The NCP service is hosted on the client device’s NIC. When the client device goes to sleep, NCP impersonates presence of its applications with their respective application service provider/third-party device (depicted with red lines).](image)

3.4.2 LAN Scenario

In the LAN scenario, the NCP runs on a device located in the same LAN as covered devices, perhaps a network device (switch, router) or a stand-alone device. Fig. 3.3 shows a pictorial view of this scenario. Communication between the NCP and client devices happens by standard protocols over the TCP/IP suite; multicast and broadcast can be effectively used for mutual discovery of endpoints.

This scenario is mostly effective when an always-on device is present or when the number of client devices is large. Today, this scenario is representative of a huge number of installations: in fact, network devices for Internet access (home gateways in SOHO and routers) are often left always on to make devices reachable from the Internet. This scenario has little privacy concerns and no mobility implications just for fixed desktop computers, but it raises a number of issues for roaming users: mainly, the NCP service should be available in most local networks. In the LAN scenario a traffic diversion mechanism should bring packets addressed to sleeping client devices towards the NCP device. Traffic diversion is discussed in Section 4.4.

3.4.3 Point-to-Point Scenario

In the point-to-point scenario, the client devices are connected directly with an access gateway where the NCP is running on as shown in Fig. 3.4. This is a typical architecture for Internet access (dial-up, ADSL,
WiMax, GSM/GPRS/UMTS), where all users are connected to the access device owned by their ISP, but no direct communication among them is allowed. This scenario is simpler than the LAN scenario, because all traffic is forced to cross the access concentrator, which is therefore the best place where the NCP could run and does not require traffic diversion.

Communication between client devices and the NCP may happen by some protocol running on top of TCP/IP stack or some built-in mechanism in the link layer technology. The first approach provides a common and general solution across different access networks, whilst the second approach could optimize the behavior for sleeping devices by exchanging messages with modems. In the point-to-point scenario, privacy is usually an issue, as many proxy-able services might not be supplied by the ISP. On the other hand, as far as the device is attached to the same ISP, mobility should be handled at the link layer in a transparent way, thus providing a seamless NCP service. The point-to-point scenario could be deployed in the short/medium term; this mainly depends upon the capability of the access technology of waking up client devices in low-power state.

3.4.4 End-Point Proxying Scenario

In end-point proxying scenario, the proxying functions are located on the remote endpoints of the device. When the client device is going to sleep, it will inform the application endpoints about its power state change with proxying request. The endpoints will apparently disconnect application connections with the
client device but will preserve their state. When the client device wakes up, it will contact the endpoints and re-establish the proxied connections. Endpoint proxy does not require any addition or change in the network hardware but requires software modifications to both sides of the connection. Endpoint proxy can easily solve the general NCP challenge of preserving TCP connections but every application needs to have its own endpoint proxy.

### 3.4.5 Internet Scenario

In the Internet scenario, the NCP service is provided by a device located anywhere in the Internet. This deployment scenario has several technical issues and challenges, but can provide huge amount of energy savings by covering for thousands of client devices globally. The general Internet-NCP scenario is depicted in Fig. 3.5. The Firewall/NAT and traffic diversion in the public Internet are the main issues in this scenario.

![Figure 3.5. Internet scenario. The NCP service is located anywhere on the Internet. Main issues to be solved in this scenario are: traffic diversion, remote wake-up of sleeping hosts, privacy concerns, mobility. A communication protocol is required in this scenario for communication between the Internet-NCP and its client device (depicted with purple line). When the client device goes to sleep, Internet-NCP impersonates presence of its applications with their respective application service provider/third-party device (depicted with red lines).](image)

No local setup is required in the Internet NCP scenario, the NCP service can be availed as long as connection to the Internet is available. Thus, this scenario is also suitable for the mobile devices. The Internet-NCP service is expected to be offered by a very powerful device that has huge resources and processing capabilities to offer proxying for very large number of client devices. The power consumption of the NCP hosting device will not be a concern as it will spread of thousands of client devices causing negligible increment consumption. The Internet-NCP can be a single point of failure which can affect thousands of subscriber. However, this problem will be overcome if the Internet NCP works in a distributed fashion utilizing more than once physical device which will also provide further benefits especially in terms of load sharing and redundancy. Communication among the Internet NCP and client devices makes use of standard protocols running on top of the TCP/IP stack. Since, broadcast and multicast are not allowed in the Internet which makes the autonomous or seamless discovery of NCP service difficult. This deployment scenario is not suitable for local network devices as the packets intended for other local network devices will be forced to go back and forth the Internet. However, this scenario is suitable if only one device is present in the local site.

### 3.4.6 Mixed Scenario

The mixed deployment scenario uses two or more than two NCP instances which run concurrently and cooperate with each other to addressed the limitations of previously addressed scenarios. Fig. 3.6 depicts the general scenario where once NCP instance is running in the local network while the other is running anywhere in the Internet. Mixed deployment scenario is an extension of the Internet-NCP deployment scenario (addressed in Section 3.4.5). The local NCP and Internet NCP cooperate together to offer a flexible,
scalable, secure and reliable NCP service to the client devices. Both the NCP instances coordinate to overcome the limitations of LAN scenarios and Internet scenario.

Figure 3.6. Mixed scenario. A local NCP instance is used to overcome some limitations of Internet-NCP. A communication protocol is required for communication between the local NCP and client device (depicted with purple line) and between the local NCP and Internet-NCP (depicted with green line). When the client device goes to sleep, Internet-NCP impersonates presence of its applications with their respective application service provider/third-party device (depicted with red lines).
Chapter 4

Design of Local NCP

The design of NCP depends on its operational mode in the network (addressed in Section 3.2). We have targeted cooperative NCP design due to several limitations in the invisible NCP. Cooperative NCP allows client devices to communicate with the NCP about their power state transitions and what to proxy and when to proxy. Thus, the NCP is fully confident about the proxying needs of its client devices. Cooperative NCP can be of different types based on the location of NCP in the network. This chapter particularly target the NCP design for local network scenarios. The main objective is the development of a very flexible NCP software that can operate in several local deployment scenarios, particularly in on-board, LAN (switch/router or standalone device) and point-to-point scenarios.

This chapter presents a reference architectural framework for the NCP service in Section 4.1 with a number of functional blocks. Following this, Section 4.2 presents all of the NCP supported features and capabilities in terms of proxying network protocols and applications on behalf of sleeping devices. The NCP should be able to wake-up its client devices whenever necessary, thus a suitable wake-up mechanism for the NCP framework is addressed in Section 4.3. Since, the NCP performs on behalf of sleeping devices, it is important for the NCP to access all packets intended for its sleeping client devices. To aim this objective, Section 4.4 presents the basic traffic diversion mechanism used by the NCP to access packets intended for the sleeping devices. Since, many applications rely on the persistent TCP connections, it is important for the NCP to have the capability of preserving TCP connections alive on behalf of sleeping devices. Section 4.5 presents some novel techniques for managing or preserving TCP connections on behalf of sleeping devices and resuming them back after they wake-up. Following this, Section 4.6 presents a communication model for information exchange between the NCP and its client devices (a basic feature of the cooperative NCP).

4.1 NCP Architectural Design

We have designed cooperative NCP with different goals in mind. The first important goal was to propose a set of generalized behavioral rules to meet basic proxying needs of sleeping devices. Secondly, the NCP should enable client devices to dynamically request different actions/rules and notify the NCP when to start or stop proxying. Another important design goal was the deployment flexibility of NCP software in the local network which can be conceived through a traffic diversion mechanism to redirects packets intended for sleeping devices towards the NCP. Thus, the NCP must be able to process packets addressed to sleeping/suspended client devices and carry out specific tasks. Several architectural components are required to build the NCP service. Fig. 4.1 depicts the basic NCP architecture. There are three main elements within the NCP service: database of behavioral rules, packet processing and packet filtering.

The behavioral rules pilot the NCP’s operations; they bind network traffic addressed towards sleeping client devices to specific actions. Network traffic consists of packets or connections; their identification
is mainly made by a set of conditions that take into account header information (source and destination addresses, source and destination ports, protocol, protocol-specific flags and options) or packet content (pattern matching, application-specific headers and information). Actions specify the operation the NCP is expected to undertake: waking up sleeping devices, answering on behalf of the suspended device and sending periodic messages are the most meaningful examples. Rules are registered by client devices prior to their application and are stored internally as long as they are explicitly withdrawn, in order to follow the dynamic needs of client devices. The NCP distinguishes among active and inactive rules: a rule is active when the corresponding device has delegated the NCP to act on its behalf (i.e., when the device is in low power, standby or power off modes), inactive when no delegation has been made.

Packet filtering inspects packet header and content to identify connections present in active NCP rules. A basic prerequisite for this operation is the ability to get packets addressed to client devices at the NCP. This obviously happens if the NCP lays on the path between each of its client devices and all of their peers; although this hypothesis sounds reasonable for single-homed local networks (as the huge number of home and SOHO sites connected by xDSL technologies), the same is not likely if the NCP works in an Internet-wide scenario. Starting from this consideration, packet hijacking is an integrating issue in packet filtering (addressed in Section 4.4).

Packet processing carries out the operation specified in each NCP rule for each packet matching that rule’s filtering specifications. Different kind of operations should be considered, depending on the NCP’s processing power and on the complexity of transferring application/protocol status between the client devices and the NCP. The NCP framework accounts at least for three kinds of operations: waking up devices, replying to incoming packets and sending unsolicited packets. Starting from these basic types, a wide variety of more complex and specific operations may be derived. The NCP also has task scheduler that is responsible to execute or invoke actions by communicating to packet processor. This process can be single shot and can be periodic (e.g., application specific heartbeat messages). One shot events are executed once
the absolute time is reached. The periodic events re-schedule themselves at the specified interval.

When a client device is going to switch to a low power mode, it notifies its NCP, so that its set of rules are activated. Rule activation implies two tasks. First, traffic addressed to the client is diverted towards the NCP. Second, the filtering engine is set to pick up packets matching the traffic patterns contained in that rule (e.g., source/destination address, source/destination port, protocol, etc.). A packet matches a rule when it satisfies all of rule’s conditions (a logical AND operation among all conditions); a packet is picked up by the filtering engine when it matches any of the active rules (i.e., a logical OR operation among all active rules). After a device has returned to a higher power mode, it notifies its NCP, its set of rules are deactivated and traffic is diverted back to that device. The deactivation of a rule implies removing its set of conditions from the filtering engine. Thus, power state notifications are needed by the NCP to know the current status of covered devices and to decide whether it should act on their behalf or not. To make the process automatic, we have developed a KernelModule on client devices that tracks the kernel for power state transitions and immediately informs the NCP. Indeed, the time available to notify the NCP about the low power state is very limited (before the device goes to sleep). Thus, a suitable communication protocol with lowest possible latency is very important.

4.2 Behavioral Rules

Each NCP behavioral rule is made of four parts: device identification, power state, a filter specification and an action. Device should identify themselves when registering rules. Using an IP address or other network parameter for this purpose is not fine, as devices may have multiple NICs and thus multiple IP/MAC addresses and names. Thus, we used Universally Unique IDentifier (UUID) for unique identification of each device. The power state indicates when the rule should be applied. The NCP activates a rule when the device switch to a power state equal or lower than that held by that rule. The filter identifies packets for which the rule applies based on a set of conditions. Table 4.1 shows a summary of the preliminary definition of the NCP behavioral rules. Despite the generic purpose of maintaining network presence, the NCP behavioral rules can be classified with more details into a small number of sets:

- **Network Connectivity**: These are protocols and applications that maintain a device connected to the network, dealing with address assignment and resolution, host location, presence verification, etc.

- **Packet Management**: This includes dealing with incoming packets which requires some response from the client device; such packets can be generated by network protocols or by applications.

- **Heart-beating**: Heart-beating is the sending of periodic messages just to refresh some information (for example, to keep alive a network connection where no data are sent, to keep active a soft state, etc.).

The second part of a rule is the action the NCP is requested to carry out. There are four kinds of actions the NCP is supporting:

1. **Wake-up Devices**: The NCP is expected to do nothing but waking up devices when any packet matches the registered rules. However, overusing this kind of action could lead to shorter sleeping periods and a large numbers of transitions between the active and sleeping states for devices, thus wasting much of the potential the NCP offers.

2. **Send Predefined Packets**: If the reply to an incoming packet or a hear-beating message is foreseeable and does not depend upon any parameter. The client devices can create the body of these packets in advance and load them at the time of rule registration. Packet headers should be added by the NCP just when the sending is triggered so to have a generic rule that holds for every remote peer.
3. **Building Packets**: The NCP understands the protocol/application specified by the rule and creates packets on its own to reply incoming requests or for periodic heart-beating. This option is feasible for simple applications/protocols that do not require complex state to be transferred between the NCP and its clients. The main drawback here is that the NCP would in general need to be aware of a large number of protocols and applications.

4. **Building Packets by Templates**: the NCP builds packets starting from templates supplied by devices; templates are filled in and modified according to a set of patterns and instructions contained in the NCP rule. This approach let the NCP proxy many protocols and applications without knowing them.

<table>
<thead>
<tr>
<th>Action</th>
<th>ARP</th>
<th>PING</th>
<th>DHCP</th>
<th>WoC</th>
<th>TCP-KA</th>
<th>Hrt-Bt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wake-up</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Buffer packet</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Send predefined packet</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Build packet</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Build packet by template</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

4.2.1 **ARP Rule**

The ARP rule instructs the NCP to answer ARP requests on behalf of the sleeping devices. The ARP rule is unavoidable for NCP operations; thus, it is automatically activated when clients go to sleep without an explicit request by them. In theory, the IP address present in the ARP rule should be resolved to the device’s MAC address. However, packets sent upon the resolution will be delivered to the sleeping device and this could be an issue if the NCP does not lie on the path between the two devices. Indeed, when the NCP answers ARP it is acting on behalf of another device and thus it is quite natural it should get all traffic addressed to that device. Thus, the most general and flexible solution is to provide the NCP’s MAC address in ARP responses. This approach brings two issues. First, some devices on the LAN may already have gotten the MAC address by a previous resolution when the device was awake; in this case, their traffic would be sent directly to the sleeping device and could not be seen by the NCP. Second, when the client wakes up, it must take back the management of its network presence, and that implies getting the correct traffic delivery again. To solve both issues, traffic diversion is addressed in Section 4.4.

4.2.2 **PING Rule**

The PING rule instructs the NCP to answer ICMP echo request messages on behalf of the sleeping devices. ICMP echo requests can carry additional data in their payload. The NCP simply copies this data without any modification in the echo reply packet it generates.
4.2.3 DHCP Rule

The DHCP rule instructs the NCP to preserve the sleeping device IP address by periodically renewing its lease with a DHCP server. The DHCP rule periodically starts a renewal process, thus extending the lease time; after two failed attempts (with a deny or no answer) a rebind process is tried at most two times. If the NCP also fails both the rebind trials, it gives up and stops renewing the IP address for that client, yet the DHCP rule remains active and ready to be reactivated. Seamless transition between DHCP client operation on client device to DHCP rule on the NCP would require the knowledge of additional information like renewal time, rebind time, lease time, stint since last renewal, current renewal state, server address and maybe something else. Indeed, to keep the NCP’s client and communication interface lighter, the NCP rule starts by a rebind message to get all parameters it needs for following operations. The DHCP protocol allows getting such information easily: a specific option is provided to request a custom list of parameters to the server; the aforementioned rebind message is just used to this purpose.

4.2.4 Wake-On-Connection Rule

The Wake On Connection (WoC) rule instructs the NCP to wake up the sleeping client device when packets addressed to a given transport port are received. The same rule is used both for connection-oriented and connectionless protocols (e.g., TCP and UDP). Every time a packet addressed to the port number of the transport protocol of the sleeping device specified in the filter is seen, that device is woken up by a suitable routine. There is a side effect behind the way NCP operates that should be minded: packets that trigger the WoC rule should be delivered to their actual destination, yet this does not happen automatically because they are addressed to the NCP due to traffic diversion (indeed, even if ARP resolved the correct MAC address of the sleeping device, packets would reach their destination before it has woken up and thus would get lost). To overcome this issue, packets matching the WoC rule are buffered at the NCP and sent out once their target device has completely woken up.

4.2.5 TCP Keep-Alive Rule

The TCP Keep-Alive (TCP-KA) rule asks the NCP to maintain a given TCP session active on behalf of the covered client devices. It keeps a TCP connection alive while the client device is sleeping. TCP sessions are identified by the 4-tuple (src, dst, src port, dst port), so each NCP client can register different behaviors for each TCP session. The TCP-KA rule is mainly conceived to maintain TCP connectivity while no data have to be exchanged between the peers. As the name implies, this action mainly answers TCP keep-alive messages; however, it also implements simple management of data coming from the remote side. Three behaviors are possible when data is received on the connection:

1. Wake-up the client device.
2. Advertise zero-window to the remote peer to delay data reception.
3. Buffer packets and acknowledge them to the remote peer. No data is sent back.

Advertising the zero-window condition and buffering packets delay device wake up for a maximum timeout, after which the device is awoken in any way. This is because the inability to deliver data causes a failure for most applications, which could assume the unreachability of the device. The TCP-KA rule is thus a generic mechanism that can be used for any kind of application.

4.2.6 Heartbeating Rule

The Heartbeating rule provides a template based approach and requests the NCP to generate/respond heartbeat messages related to a specific application. This rule is smart enough to manage both solicited and
unsolicited presence messages for any application. Solicited heartbeating is invoked if the incoming application presence message matches certain filter pattern. The pattern matching is done from the beginning of data. It is also possible to ignore some opaque or dynamic fields in the pattern matching which change in every heartbeat message as shown in Fig. 4.2. On the other hand, unsolicited heartbeating sends the application presence messages at periodic interval. The period must be specified during the rule registration. Further, the applications can also specify when the first heartbeat message should be sent.

![Pattern Diagram](image1)

Figure 4.2. Solicited heart-beating is sent in response to incoming messages that matches a predefined pattern at a specific position in the data fields. Some part of the pattern (i.e., opaque fields) may be ignored to account for dynamic fields that change in every packet.

The heartbeating rule is based on the template based approach that allows client devices to specify all required fields (e.g., pattern, variable/dynamic fields and method to compute them etc) during the rule registration (see Fig. 4.3). The variable/dynamic fields are part of the data template that should change in each heartbeat packet, hence providing the NCP the ability of generating dynamic packets. The definition of dynamic fields include the following information:

![Template Diagram](image2)

Figure 4.3. Heart-beating messages are generated dynamically by the NCP starting from a template provided by the client device and filling in variable fields.

- The position of variable field inside the packet i.e., offset from the beginning of data field.
- Variable field length.
- Variable field type i.e., the method to compute variable field.
- Any additional data specific to the variable field.

Our NCP software supports the following types of variable fields:

- **Counter**: This type of variable field is incremented by a given step in successive heartbeat messages starting from an initial value.
- **Timestamp**: The Timestamp field carries the time value. It is calculated by adding the time value supplied by client device during the rule registration to the time elapsed since the rule is registered. This way, even if the NCP and client device clocks are not synchronized, the timestamp value remains congruent with those calculated on client device.
- **RandomNumber**: This type of variable field requires a random number and no additional parameters required during rule registration.
4.3 Device Wake-up Mechanisms

Depending on the communication medium, the NCP may use different types of wake-up mechanisms. The wake-up mechanisms are required to wake-up the client device whenever required. The wake-up mechanism can be based on WOL packet which is also known as 'magic packet'. The WOL was invented by IBM in 1996 for the remote waking up of a network connected PC through its NIC. All modern NICs have an auxiliary power connection from the motherboard. This allows NICs to remain powered up even when the device is sleeping. NICs also have the connection to the wake up interrupt line on the motherboard [9] used for the device wake up. These NICs have the ability of pattern matching for the received packets to trigger the device wake-up. The magic packet is used for this purpose and it is sent in a MAC frame to the sleeping device or broadcasted to all network devices using the network broadcast address. Thus, WOL packet wakes up the device from ACPI S3 or S4 sleep states to fully powered on state (S0). It contains in its payload six bytes of all 1’s followed by the target device’s MAC address (48-bit) repeated 16 times. The generic format of the magic packet is shown in the Fig. 4.4. The following are some of the basic requirements of magic packet:

1. It requires the target device MAC address.

2. It does not provide any delivery confirmation.

3. It requires the target device to be in the same subnet/network. WOL packet can also be sent over Internet but requires to create new inbound firewall rule for the router to accept packets directed to specific port and broadcast them within the network. This increases security risks such as denial of service attacks.

4. It requires the target device NIC to support pattern matching and WOL.

![Figure 4.4. Format of WOL packet.](image)

4.4 Network Traffic Diversion

Network traffic diversion is the basic prerequisite for the NCP operation in order to get the packets intended for the sleeping devices. The requirement of network traffic diversion also depends on the location of the NCP. If the NCP service is being offered by the Home Gateway (HG), there will be no need to implement any traffic diversion mechanism for packets coming from outside and intended for local sleeping devices. However, in complex scenario, the NCP will not be able to maintain presence of sleeping client device for another local device until the traffic passes through HG where NCP is located. Thus, the traffic diversion mechanism is not required as long as all network traffic intended for sleeping devices passes through a common point where the NCP service is located. This can only be true for home networks where usually one or two PCs are present at the local site. However, this is not always the case especially in small or medium size office environment where NCP has to cover for hundreds or thousands of client devices. The HG usually have limited resources and cannot implement complex proxying tasks and cannot scale beyond
few client devices. Thus, the NCP is usually also considered as running on a standalone device located in
the local network or anywhere in the Internet. Thus, traffic diversion mechanism is quite important for the
NCP which will divert packets from their normal path to sleep client devices towards the NCP.

The final delivery of the packets in the local network is based on the MAC address, a 48-bit address
that uniquely identifies a physical device attached to the local network. Source and destination addresses
are included in packet headers and are used by intermediaries (bridges, switches, APs) to forward packets.
The MAC addresses are only used locally (although they are usually globally-unique identifiers). The
destination MAC address must be automatically retrieved to build Ethernet/WiFi frames. This operation is
called ‘address resolution’ and for IP packets the Address Resolution Protocol (ARP) manages it. The ARP
broadcasts request messages including the IP address of the destination and asking the device that owns that
address to provide its physical or link-layer (i.e., MAC) address; the destination then answers providing its
MAC address. The ARP resolutions are stored in caches by each host and removed after a time-out occurs;
this limits the amount of traffic generated by this protocol and speeds up packet forwarding.

To achieve traffic diversion in the local network, it is important to bind the NCP’s MAC address to the
IP addresses of sleeping client devices: when a device falls into a sleep state, the NCP’s MAC replaces the
client’s MAC, and when the device comes out of the sleep state, the client’s MAC is brought back. Thus,
traffic diversion in the local network implies two actions:

1. The NCP must answer to the ARP request packets intended for the sleeping client devices by provid-
ing its own MAC address.

2. The NCP must update ARP caches of local network devices when its client device enter or exit the
sleep/standby state. This action is critical important to avoid sending packets to the obsolete MAC
addresses in caches.

In the local network, devices are allowed to update their binding between IP/MAC addresses through
a so called ‘Gratuitous ARP’ packet: an unsolicited ARP packet (usually an ARP request) holding the IP
address and the associated MAC address. RFC 826 [50] states that any device receiving an ARP packet
(both request and reply, even if the latter do not match any previous request by that device) update its
local ARP cache with the MAC address provided by that packet if such device already has an entry for the
associated IP address; otherwise, the device insert the information in its ARP table only if it is the target of
the ARP packet (this implicitly implies the packet be a response). Implementations that conform to RFC
826 does not have any problems with packet redirection between sleeping devices and the NCP. Resolving
an IP address into a different MAC could also be viewed as a mechanism to manage device’s movement.
RFC 826 already envisioned this possibility; quoting from its ‘Related issues’ section: ‘A host could move if
a host name (and address in some other protocol) were reassigned to a different physical piece of hardware.’

Based on the Gratuitous ARP, Fig. 4.5 presents the basic traffic diversion scenarios in the local net-
work for the NCP framework. Fig. 4.5(a) depicts the scenario when the NCP client device is awake and
responding to packets. When the client device wants to go to sleep state, it immediately informs the NCP
about its power state transition (as depicted in Fig. 4.5(b)). After receiving the power state transition notifi-
cation, the NCP implements traffic diversion by broadcasting Gratuitous ARP packets in the local network
(as shown in Fig. 4.5(c)). The Gratuitous ARP updates the ARP caches of local network devices and binds
the NCP’s MAC address with the IP address of the sleeping client device. Thus, the future packets intended
for sleeping client device will arrive at NCP and get responded according to the NCP behavioral rules (as
depicted in Fig. 4.5(d)). If the client device wakes-up either by user activity or NCP operations, it once
again updates the NCP about it new power state (depicted in Fig. 4.5(e)). After the NCP receives wake-up
acknowledgment from its client device, it re-divert packets towards the client device using Gratuitous ARP
(as depicted in Fig. 4.5(f)). Now the new packets will arrive directly at the client device and get responded
as in Fig. 4.5(a).

The Gratuitous ARP is the most feasible approach for traffic diversion in the local network. There are
several other networking protocols/techniques that take the advantage of assigning dynamically given IP
and MAC addresses based on the Gratuitous ARP. The following are some of the examples utilizing ARP probes:

- A well known architecture ‘Mobile IP’ where the Home Agent gets packets addressed to its mobile devices while they are away from the home network. Thus, the Home Agent answers ARP request packets on behalf of those devices and sends out Gratuitous ARP to update the caches of local network devices. Details are available in RFC 3344 [51].

- Stateless link-local address configuration randomly assigns IPv4 addresses in the subnet 169.254.0.0/16; once an address is chosen, ARP probes are sent to detect possible conflicts and then an announcement is done by a Gratuitous ARP request to bind that IP to the host’s MAC address. Details are available in RFC 3927 [52].

- Statefull address configuration by DHCP suggests clients to send ARP probes to detect possible conflicts after they have been assigned an IP address and then, if no responses are received, to send out a reply (Gratuitous ARP in a reply message) to clear stale ARP caches on the local link. Details are available in RFC 2131 [53].

- Apple’s Bonjour Sleep Proxy answers ARP/ND requests on behalf of sleeping devices addresses giving its own MAC address as the current (temporary) owner of that address.

Finally, a side effect of binding the IP addresses of a sleeping device to a different MAC address comes out because ARP is also used to detect whether an IP address is already used in the subnet. RFC 5227 envisions IP address conflict detection by means of ARP probes. According to this specification, a host can check if an address is already used by another device by sending ARP probes; these are ARP requests sent by an unspecified IP address (the special all zeroes address: 0.0.0.0). If the NCP answered such packets, the awaking devices would believe its address has been taken by somebody else and would not use it anymore; we found that Windows 7 behaves this way. Thus, an additional step implemented in the NCP software is to detect these ARP probes from its client devices and does not answer them.

4.5 Managing TCP Connections

This section addresses how the NCP can preserve TCP connections on behalf of sleeping client devices. Th TCP transport protocol is most commonly used as many applications need a reliable connection without
having to be worried about network related issues such as error recovery, congestion control and packet ordering. The TCP maintains persistent transport connections even when no data is being transferred between the peers. The TCP protocol builds end-to-end connections relying on the (unreliable) service provided by the underlying IP. Being a reliable carrier, TCP guarantees no data loss, but it also avoids connections to stall, by detecting any network and peer failure. There are two main issues that need to be addressed for managing TCP connections on behalf of sleeping client devices.

1. Maintaining TCP connections alive. That concerns keep-alive or presence messages, which can be generated periodically by one peer and must be answered by the other one; if either of them fails to do that, the connection is dropped. Keep-alive are void TCP messages sent when no data has been exchanged for a given amount of time just to verify that the other peer is still present and connected, and to avoid starving if it does not. The NCP should be able to generate/respond to these periodic keep-alive messages on behalf of sleeping devices in order to avoid their TCP connections being dropped.

2. Handing off TCP connections. That offers the opportunity to continue exchanging data on an existing session, without the overhead due to breaking it and setting up a new one. Simply, a sleeping device should be able to use same TCP session after wake-up which it was using before sleeping.

This section particularly addresses how the NCP deals with the above two issues. It addresses how NCP keeps TCP connections alive by probing the periodic TCP keep-alive messages and the invoking actions when a new data is received on a connection. Following this, different techniques have been addressed for handing off TCP connections to client devices after wake-up.

4.5.1 Maintaining TCP Connections Alive

Keeping TCP connections alive means letting the peer believe the device is still awake and the connection is still open. Two issues can be identified to this purpose:

1. Dealing with periodic keep-alive probes, sent to check the connection is available.

2. Dealing with new data on the connection.

To successfully acknowledge data on an active TCP connection, the NCP must know the current TCP status; indeed, only the knowledge of sequence and acknowledge numbers is really required to generate valid packets. Such information could be easily inferred by TCP packets exchanged with the remote peer. In particular, a simple ACK packet may be sent with wrong sequence number (for example, both fields set to zero) and the reply could be parsed to find out the next byte the peer will send and the next byte it expects to receive. Sleeping devices are likely to be in a steady-state situation, as they are sleeping just because there is no activity to carry out; in this situation, the sequence number of the received packet is the byte to acknowledge and the acknowledgment number is the next byte to use when sending data or acknowledging packets. This trick was successfully used to infer sequence numbers without an explicit indication by the covered devices.

Keep-alive Probes

The NCP needs to answer keep-alive probes from the TCP peer of the covered host. That implies this kind of messages should be recognized. This task is quite simple, as of the way keep-alives are generated: their sequence number is one-byte less the expected next sequence number and they usually carried no data. However, the best way is to check only the sequence number of the packet that was already acknowledged and, in case, answer with an ACK confirming the next expected byte. The NCP also checks if the packet carries no data. Sending of keep-alive messages on behalf of sleeping hosts is not meaningful. Remember
the keep-alive mechanism was conceived to avoid servers to waste resources while the peer is no more connected. However, if the server is sleeping, there is no need to wake it up if the remote side did not answer to a keep-alive message: indeed, resources are frozen on asleep hosts and the management of keep-alives could be safely postponed to when the host resumes to the awake state, thus extending the sleeping period and the energy savings.

**New Data on Connection**

Management of new data on the connection is important to hide the power status of the covered client device to its remote peer. To this aim, three different behaviors can be taken into account: (i) hang up data reception, (ii) buffer data, and (iii) wake-up the client device. Hanging up data reception is based on advertising the zero-window condition. This is an optimal solution to extend the time spent in low-power modes, but there are practical issues that limits its effectiveness e.g., delivery of data are subject to certain timeouts, after which the application fails and may close the connection because it considers its peer unreachable. Buffering data implies the need to acknowledge them to the remote peer. Data will then be delivered to the client device once it awakes. Waking-up the client device seems the most reasonable option, although it may reduce sleeping times.

**4.5.2 Handing off TCP Connections**

As stated earlier, handing off a TCP connection simply implies enabling sleeping devices to use same TCP sessions after wake-up which they were using before sleeping. There are three approaches to hand off TCP connection between two devices; the NCP and its client device [49]: (i) End-to-End Management, (ii) Proxying TCP Connections and (iii) Migrating TCP Sessions. These approaches are described in the following:

**End-to-End Management**

The simplest solution to get around the problem of maintaining TCP state and presence for suspended devices would be rewriting applications so that they do not use persistent TCP connections. That means TCP connections should be always set up and torn down for each data transfer, like HTTP 1.0 does by default. However, this approach will inevitably make the efficiency of TCP worse, as of (i) the overhead brought by the three-way handshake at connection set up and (ii) the slow-start phase of the congestion control. A variant approach is wrapping TCP sockets in order to hide applications the burden of non-persistent connections. The wrapper emulates a legacy socket interface for applications, which are free to use persistent connections; however, the wrapper sets up and tears down TCP connections at each data transfer. Reduced efficiency still remains an issue with this approach. This solution was already proposed by using a ‘shim’ layer between applications and the socket interface [16]. The third end-to-end solution is changing the TCP protocol to account for power saving options. In this proposal, a new option is included in the protocol header fields to advertise the next power state (standby or wake up) to the remote peer [24].

**Proxying TCP Connections**

Proxying TCP connections is useful when the intermediary should generate and receive data on behalf of covered applications. Different approaches can be proposed depending on how the NCP takes the control of a TCP session. Splitting TCP connection is one such approach to preserve TCP connections on behalf of sleeping devices. Two different connections are set up: the first between the client device or Local Host (LH) and the NCP, the second between the NCP and the Remote Peer (RP) as shown in Fig. 4.6. Inside the NCP, a relay agent receives data on each connection and sends them on the other. When the client device goes to sleep, the connection with the NCP may be closed or frozen while the NCP maintains active the connection
with the remote peer by standard keep-alive mechanisms. This way, power state transitions of the client
device are hidden to the remote peer. Main drawbacks in this approach are scalability issues (for managing
many client devices and applications), new incoming connection management (remote peers are not aware
of proxy), data transfer synchronization (two connection segments most likely have different characteristics
such as bandwidth, delay, reliability etc), packet losses (packets may be lost on one connection while being
acknowledged by proxy to the other peer) and privacy concerns.

![Figure 4.6](image)

Figure 4.6. The TCP connection can be split in two segments, with the Proxy relaying messages between
them. Red circles represent TCP sockets.

Some issues of connection splitting can be addressed with relaying TCP connections at the NCP as
shown in Fig. 4.7: in this case, instead of setting up two different connections, the NCP simply takes over
the control of the connection while the client device is sleeping. The NCP always processes packets in a
transparent manner; it could infer the connection status (sequence numbers, window size) by inspecting
packets, although this would require a heavy burden if many connections are active concurrently. As the
remote peer is unaware of the power state of the client device, it may send data while the latter is sleeping;
in this case the NCP can wake up the device or buffer packets until the device autonomously wakes up or
some predefined criterion is met (maximum amount of data stored, urgent data, time-out). However, if the
NCP buffers packets on behalf of sleeping devices, it must acknowledge their reception to remote peers.
Thus, a variation in the Transmission Control Block (TCB) occurs and requires clients the capability to get
back this information when they are awake and to dynamically change the internal TCP operation. An easy
approach to avoid modification in the TCB would be preventing the remote peer from sending data during
sleeping periods by advertising a zero-window condition.

![Figure 4.7](image)

Figure 4.7. All traffic crosses the proxy during the lifetime of the TCP session. Red circles represents TCP sockets.

Another approach would be to have two separate spaces for sequence and acknowledge numbers: be-
tween the client device and the NCP and between the NCP and the remote peer. Before the device goes to
sleep, the two spaces fit together neatly. However, when data are received by the NCP while the client de-
vice is sleeping, the number in the second space are increased. Once the client device is again available, the
NCP starts transferring the buffered packets and increases the numbers in the first space. When all buffered
data have been retrieved from the NCP, the two sets are again aligned and the NCP may stop intercepting
and relaying packets on that connection. This approach is known as TCP splicing as shown in Fig. 4.8 and
is somehow in the mid between splitting and relaying TCP connections. TCP splicing makes it appear to
the endpoints of two separate connections but it is in fact one connection. The splicing only switches some
fields within the TCP headers (sequence and acknowledge numbers) and relays packets between the two connections.

Figure 4.8. TCP splicing acts on TCP packets, not on data carried. Red circles are TCP sockets.

**Migrating TCP Sessions**

This approach mainly implies the capability of client device to delegate to the NCP the use of an active TCP session for the whole duration of its sleeping period and to take it back once it wakes-up. Fig. 4.9 depicts the basic scenario. When the client device is awake, it maintains the TCP session with the remote peer (shown in Fig. 4.9(a)). The client device before going to sleep transfers TCP session to the proxy where it is resumed in seamless and transparent way with the remote peer (shown if Fig. 4.9(b)). Precisely, TCP migration allows the client devices to freeze TCP session (save TCP state) before sleeping and migrate the TCP session to the NCP. The TCP session is resumed on the NCP after the client device enters into sleep state. Likewise, the TCP session is returned back from the NCP to the client device after it wakes-up.

Figure 4.9. TCP migration: Red circles represents TCP sockets. (a) Direct connection between local host and remote peer, (b) Proxy takes over the connection during sleeping periods.

TCP migration requires exchanging the current state to restore the TCP session at a different device; the TCB is the main information that holds all information and parameters about the TCP’s state machine, but in-flight packets that have not yet been acknowledged must be included as well, to avoid data losses and
preserve the reliability of the mechanism. Indeed, two issues immediately come out with TCP migration approach:

1. The possibility of saving the current TCP state and resuming the connection back at another host from that information; TCP is usually implemented within the OS and changing the default behavior may be hard to implement.

2. The possibility of keep using same address and transport port numbers at a different device. The transport port number of the TCP session may be already used on the proxy by another application of other client devices.

Recent developments in the Linux kernel make possible the TCP migration feature. The TCP_REPAIR feature is added to the Linux kernel which was originally developed under the Criu project. It is a socket option that allows to put TCP socket in the repair mode and can be modified. The data required for saving and restoring TCP session include: (i) the TCP information (TCB contains all the information about the current status of connection), (ii) send and receive queues lengths, (iii) send and receive sequence numbers, (iv) content of send queue (packets already wrote by the application to the socket but not yet sent or waiting for acknowledge) and receive queue (packets already received but not yet read by the application) and (v) TCP options such as timestamp, window size, max segment size etc.

In addition to TCP repair feature, the incorrect addresses issue at the proxy must be resolved (binding local sockets to non-local addresses and possibly already-in-use TCP ports). The NAT is the best option to address this issue. The translation is done between the original address/transport port number and the local address/transport port number dynamically associated to the socket. This objective can be easily achieved in Linux by configuring the translation in the pre-routing and post-routing chains of NAT table as shown in Fig. 4.10.

![Figure 4.10. Natting for TCP migration. The picture shows the Netfilter chains and the rules needed to translate the LHs address/port into those of the local socket.](image)

---

1Checkpoint/Restore In Userspace (CRIU), website: www.criu.org.
4.6 Communication Protocol

Since, this chapter targeted the design of a collaborative or cooperative NCP framework, suitable communication protocol is required for the effective communication between the NCP and its client devices. Communication protocol is particularly required for registration/withdrawal of different types of behavioral rules and notification of power state transitions. This section first motivates the choice of communication protocol by highlighting the preferred communication features and characteristics for the NCP framework. Following this, detailed description of the proposed communication model is presented in accordance with the behavioral rules being offered by the NCP.

4.6.1 Choice of Communication Protocol

Several requirements need to be met to build a flexible, secure and reliable communication framework. Since, the NCP impersonates presence of different protocols and applications on behalf of sleeping client devices, it is important to realize a secure communication framework that can protect from malicious operation such as connection hijacking, spoofing and denial of service etc. Further, flexibility and reliability of NCP operations require correct setup of communication protocols. E.g., if rules registration fail, the NCP will not be able to impersonate presence of its sleeping clients (further, client devices may not be aware of this fact).

There can be different choices for communication protocol based on the expected communication features such as capability to manage high throughput, auto-discovery, zero-configuration and seamless networking. Such features usually depend on the location of NCP service and the type of communication medium. It is unlikely that different communication protocols meet all of the required features. Two good options for the design of communication framework include UPnP and multicast DNS (mDNS). The UPnP is an emerging standard. It is a complete architecture operating on the top of several networking protocols to provide interesting features of auto-discovery, zero-configuration and seamless communication which make it quit suitable for the NCP framework. On the other hand, mDNS is just a mechanism for auto-discovery of devices of interest in the local network. After discovery, it requires support of other protocols to enable communication among the network devices. The mDNS service is mainly used by Apple devices to discover each others presence in the local network.

We have designed and implemented the communication framework using UPnP protocol specifications [54]. The choice of UPnP protocol is well motivated due to its interesting features and rapid popularity for future network devices. Further, it can be easily supported by heterogeneous network devices including PCs, printers, scanners, copiers, Internet gateways, storage devices (such as Western Digital My Book Live) and smartphones etc [54].

4.6.2 Basic UPnP model for NCP

The UPnP architecture allows seamless autonomous communication between the devices in local network which makes it the most suitable choice for the NCP framework. The UPnP technology will allow the NCP and its client devices to communicate autonomously and in a seamless way without any need of configurations. The UPnP Device Architecture (UDA) specified two main roles for the UPnP devices: Controlled Device (CD) and Control Point (CP). The CD represents the physical entity that implements one or more services. Each service consists of one or more actions that build remote procedure calls. The CP runs on another network device that sends commands to specific service of the CD and invokes particular actions. Simply, the CD plays the role of server that receives and processes requests sent by the client (CP in this case).

The basic UPnP based communication scenario in the NCP framework is depicted in Fig. 4.11. In the general UPnP communication scenario, one physical device implements CP and the other implements CD with one or more services. E.g., network printer implements CD with printing service which receives
commands from the CP on computer. However, to allow two way communication in the NCP framework, both physical devices (the NCP and its client devices) implement a CP as well as a CD as shown in Fig. 4.11. The CP is required on NCP clients to send proxying commands for different applications and protocols to the NCP service of the CD implemented by the NCP hosting device. Similarly, the CP is required on the NCP hosting device to send commands to the Low Power (LP) service of the CD implemented by the NCP clients. Such commands may be used for power management of NCP clients e.g., instructing a client device to go to sleep mode for the specified periods during the day.

[Diagram of UPnP model for NCP]

The NCP service on NCP hosting device provides a list of remote procedure calls for its client devices to register or invoke different types of actions. Each action corresponds to a behavioral rule the NCP is offering for different applications and protocols (presented in Section 4.2). In short, NCP actions can be classified into two categories:

1. Actions that enable client devices to subscribe/unsubscribe different behavioral rules at the NCP which are automatically activated when the client devices enter into sleep state.

2. Actions to transfer data between the NCP and its client devices which is required for the proxying of some applications or protocols (e.g., the TCP migration feature provided by the NCP framework).

The registration of each action returns an identifier which can be used for the un-subscription of that particular behavioral rule at the NCP. Each action registration by the NCP service requires client device identification and power state information. The client device identification is based on the UUID which is unique for each device and is supplied during action registration. The power state defines when to activate/execute a particular registered action. Since the NCP maintains the presence on behalf of sleeping devices, it responds to ARP packets as soon as the power managed device enters into low power state. The ARP rule is automatically activated for sleeping client devices and don’t require any registration through UPnP interface. Table 4.2 summarizes the list of actions that client devices can request from the NCP service over UPnP communication framework.

Most of generalized set of behavioral rules offered by the NCP framework do not require any additional data. However, the TCP migration feature of the NCP framework requires exchange of additional data between the NCP and its client devices. The application specific heart-beat messages are usually sent over TCP protocol, thus, the client device must transfer the TCP state to NCP before going to sleep and get it back from the NCP when it wakes up. Therefore, the NCP offers two types of actions which the client devices can invoke over the UPnP communication framework: SetTCPState and GetTCPState.
Table 4.2. Summary of actions that client devices can request from the NCP over UPnP communication framework.

<table>
<thead>
<tr>
<th>Action</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PING</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>string</td>
<td>Client device IP address.</td>
</tr>
<tr>
<td></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID return by NCP. Can be used for un-subscribing action.</td>
</tr>
<tr>
<td><strong>DHCP</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>string</td>
<td>Client device IP address.</td>
</tr>
<tr>
<td></td>
<td>MacAddress</td>
<td>string</td>
<td>Client device MAC address.</td>
</tr>
<tr>
<td></td>
<td>Hostname</td>
<td>string</td>
<td>Host name used to identify client device in DHCP messages.</td>
</tr>
<tr>
<td></td>
<td>ClientId</td>
<td>string</td>
<td>In case an identifier other than MAC address is used.</td>
</tr>
<tr>
<td></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID return by NCP. Can be used for un-subscribing action.</td>
</tr>
<tr>
<td><strong>WakeOnConnection</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>string</td>
<td>Client device IP address.</td>
</tr>
<tr>
<td></td>
<td>Protocol</td>
<td>string</td>
<td>Transport protocol on which new connection attempt is received.</td>
</tr>
<tr>
<td></td>
<td>Port</td>
<td>ui2</td>
<td>Transport port on which new connection attempt is received.</td>
</tr>
<tr>
<td></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID return by NCP. Can be used for un-subscribing action.</td>
</tr>
<tr>
<td><strong>TCP KeepAlive</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>string</td>
<td>Client device IP address.</td>
</tr>
<tr>
<td></td>
<td>Peer</td>
<td>string</td>
<td>Peer IP address in order to identify a TCP session.</td>
</tr>
<tr>
<td></td>
<td>LocalPort</td>
<td>ui2</td>
<td>Local port used in the TCP session.</td>
</tr>
<tr>
<td></td>
<td>RemotePort</td>
<td>ui2</td>
<td>Remote port used by the TCP peer.</td>
</tr>
<tr>
<td></td>
<td>OnData</td>
<td>string</td>
<td>Wakeup, buffer or delay when new data is received on TCP session.</td>
</tr>
<tr>
<td></td>
<td>OnDataTimeout</td>
<td>ui4</td>
<td>Timeout to wake-up client when new data is received.</td>
</tr>
<tr>
<td></td>
<td>WakeOnPust</td>
<td>boolean</td>
<td>Wake-up client when new data arrives with push flag.</td>
</tr>
<tr>
<td></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID return by NCP. Can be used for un-subscribing action.</td>
</tr>
<tr>
<td><strong>Heartbeating</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
<tr>
<td></td>
<td>Address</td>
<td>string</td>
<td>Client device IP address.</td>
</tr>
<tr>
<td></td>
<td>Peer</td>
<td>string</td>
<td>Peer IP address to identify a TCP session.</td>
</tr>
<tr>
<td></td>
<td>Protocol</td>
<td>string</td>
<td>Transport protocol (TCP or UDP).</td>
</tr>
<tr>
<td></td>
<td>LocalPort</td>
<td>ui2</td>
<td>Local port used in the TCP session.</td>
</tr>
<tr>
<td></td>
<td>RemotePort</td>
<td>ui2</td>
<td>Remote port used by the TCP peer.</td>
</tr>
<tr>
<td></td>
<td>MsgTemplate</td>
<td>bin.base64</td>
<td>The template for heart-beating message.</td>
</tr>
<tr>
<td></td>
<td>VariableFields</td>
<td>bin.base64</td>
<td>Type and position of variable field in heartbeat message.</td>
</tr>
<tr>
<td></td>
<td>Period</td>
<td>ui4</td>
<td>Time period for heartbeat messages.</td>
</tr>
<tr>
<td></td>
<td>FirstRun</td>
<td>ui4</td>
<td>Time to start sending first heartbeat message.</td>
</tr>
<tr>
<td></td>
<td>FilterPattern</td>
<td>bin.base64</td>
<td>A pattern to check in the incoming messages to detect the heartbeat message.</td>
</tr>
<tr>
<td></td>
<td>FilterOffset</td>
<td>ui2</td>
<td>Position of the pattern from beginning of data.</td>
</tr>
<tr>
<td></td>
<td>OpaqueFields</td>
<td>bin.base64</td>
<td>Fields that must be ignored for filtering.</td>
</tr>
<tr>
<td></td>
<td>OnData</td>
<td>string</td>
<td>Wakeup, buffer or delay when new data is received.</td>
</tr>
<tr>
<td></td>
<td>OnDataTimeout</td>
<td>ui4</td>
<td>Timeout to wake-up client when new data is received.</td>
</tr>
<tr>
<td></td>
<td>WakeOnPust</td>
<td>boolean</td>
<td>Wake-up client when new data arrives with push flag.</td>
</tr>
<tr>
<td></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID return by NCP. Can be used for un-subscribing action.</td>
</tr>
<tr>
<td><strong>Unsubscribe</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
<tr>
<td></td>
<td>SubId</td>
<td>ui4</td>
<td>Subscription ID of action to unsubscribe.</td>
</tr>
<tr>
<td><strong>UnsubscribeAll</strong></td>
<td>UUID</td>
<td>uuid</td>
<td>Unique identification of client device.</td>
</tr>
</tbody>
</table>

resumes a TCP session on the NCP side which is frozen on the client device. GetTCPState gets the TCP state back from the NCP and resumes once again on client device. The details of these actions are provided in Table 4.3.
### Table 4.3. List of actions offered by the NCP for its TCP session migration feature.

<table>
<thead>
<tr>
<th>Action</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SetTCPState</strong></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID for which TCP migration feature is requested.</td>
</tr>
<tr>
<td></td>
<td>SrcIP</td>
<td>string</td>
<td>The source IP address of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>DstIP</td>
<td>string</td>
<td>The destination IP address of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>SrcPort</td>
<td>ui2</td>
<td>Source port number of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>DstPort</td>
<td>ui2</td>
<td>Destination port number of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>InQLen</td>
<td>ui4</td>
<td>Input queue length.</td>
</tr>
<tr>
<td></td>
<td>InQSeq</td>
<td>ui4</td>
<td>Input queue sequence number.</td>
</tr>
<tr>
<td></td>
<td>InQBuff</td>
<td>bin.base64</td>
<td>Input queue data.</td>
</tr>
<tr>
<td></td>
<td>OutQLen</td>
<td>ui4</td>
<td>Output queue length.</td>
</tr>
<tr>
<td></td>
<td>OutQSeq</td>
<td>ui4</td>
<td>Output queue sequence number.</td>
</tr>
<tr>
<td></td>
<td>OutQBuff</td>
<td>bin.base64</td>
<td>Output queue data.</td>
</tr>
<tr>
<td></td>
<td>OptMask</td>
<td>ui4</td>
<td>TCPI_OPT_bits.</td>
</tr>
<tr>
<td></td>
<td>SndWScale</td>
<td>ui4</td>
<td>Sender window scale.</td>
</tr>
<tr>
<td></td>
<td>MaxSegSize</td>
<td>ui4</td>
<td>Maximum segment size.</td>
</tr>
<tr>
<td></td>
<td>HasRcvWScale</td>
<td>boolean</td>
<td>Receiver window scale in use. True if the RcvWScale field is valid.</td>
</tr>
<tr>
<td></td>
<td>RcvWScale</td>
<td>ui4</td>
<td>Receiver window scale.</td>
</tr>
<tr>
<td></td>
<td>HasTimeStamp</td>
<td>boolean</td>
<td>Timestamp option in use. True if the TimeStamp field is valid.</td>
</tr>
<tr>
<td></td>
<td>TimeStamp</td>
<td>ui4</td>
<td>Timestamp value for the connection.</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td>string</td>
<td>Indication about success or failure of SetTCPState operation.</td>
</tr>
<tr>
<td><strong>GetTCPState</strong></td>
<td>SubId</td>
<td>ui4</td>
<td>Action subscription ID for which TCP migration feature is requested.</td>
</tr>
<tr>
<td></td>
<td>SrcIP</td>
<td>string</td>
<td>The source IP address of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>DstIP</td>
<td>string</td>
<td>The destination IP address of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>SrcPort</td>
<td>ui2</td>
<td>Source port number of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>DstPort</td>
<td>ui2</td>
<td>Destination port number of the TCP session.</td>
</tr>
<tr>
<td></td>
<td>InQLen</td>
<td>ui4</td>
<td>Input queue length.</td>
</tr>
<tr>
<td></td>
<td>InQSeq</td>
<td>ui4</td>
<td>Input queue sequence number.</td>
</tr>
<tr>
<td></td>
<td>InQBuff</td>
<td>bin.base64</td>
<td>Input queue data.</td>
</tr>
<tr>
<td></td>
<td>OutQLen</td>
<td>ui4</td>
<td>Output queue length.</td>
</tr>
<tr>
<td></td>
<td>OutQSeq</td>
<td>ui4</td>
<td>Output queue sequence number.</td>
</tr>
<tr>
<td></td>
<td>OutQBuff</td>
<td>bin.base64</td>
<td>Output queue data.</td>
</tr>
<tr>
<td></td>
<td>OptMask</td>
<td>ui4</td>
<td>TCPI_OPT_bits.</td>
</tr>
<tr>
<td></td>
<td>SndWScale</td>
<td>ui4</td>
<td>Sender window scale.</td>
</tr>
<tr>
<td></td>
<td>MaxSegSize</td>
<td>ui4</td>
<td>Maximum segment size.</td>
</tr>
<tr>
<td></td>
<td>HasRcvWScale</td>
<td>boolean</td>
<td>Receiver window scale in use. True if the RcvWScale field is valid.</td>
</tr>
<tr>
<td></td>
<td>RcvWScale</td>
<td>ui4</td>
<td>Receiver window scale.</td>
</tr>
<tr>
<td></td>
<td>HasTimeStamp</td>
<td>boolean</td>
<td>Timestamp option in use. True if the TimeStamp field is valid.</td>
</tr>
<tr>
<td></td>
<td>TimeStamp</td>
<td>ui4</td>
<td>Timestamp value for the connection.</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td>string</td>
<td>Indication about success or failure of GetTCPState operation.</td>
</tr>
</tbody>
</table>
Chapter 5

Opportunities and Challenges in Internet NCP

The NCP can provide huge amount of energy savings if its coverage is extended beyond LAN boundaries and cover for very large number of client devices. The possible solution is the design of a global flexible, scalable, reliable and powerful Internet-wide NCP that can offer proxying service to client devices at anytime located anywhere on the Internet. This will also provide easy support for mobile devices. The Internet-wide NCP service is expected to be offered by a very powerful device with enough memory and processing capabilities to proxy large number of applications for thousands of client devices. The power consumption of the NCP hosting device will not be a concern as it will split over thousands of client devices causing negligible incremental power consumption. Further, the Internet-wide NCP can also be trusted by the application developers if it becomes a standard global entity. The application developers would not hesitate to provide their application stubs/routines (methods of generating application specific heartbeat messages) and strategies for sharing users privacies or authentication credentials. However, the Internet-NCP can be a single point of failure which can affect thousands of subscriber. The possible solution can be to operate the Internet NCP in a distributed fashion utilizing more than once physical device which will also provide further benefits especially in terms of load sharing and redundancy.

This chapter extends the NCP coverage beyond LAN boundaries and particularly addresses the opportunities and challenges in its realization. It is important to note that the design of Internet-wide NCP is not from scratch. It is merely an extension of the local NCP (addressed in Chapter 4) by operating it on a powerful device that is accessible over Internet. However, extending NCP coverage beyond LAN boundaries raises several issues and challenges especially related to NAT/Firewall (which may hinder Internet NCP communication with its client devices) and traffic diversion in public infrastructure to divert packets intended for sleeping devices towards the Internet NCP. Proposing different possible architectural solutions for the Internet-wide NCP and addressing their pros and cons is the main scope of this chapter.

This chapter is organized as follows: Section 5.1 presents possible architectural solutions for the Internet-wide NCP based on whether a local NCP entity in the client devices network is being used or not in assisting the Internet NCP. Following this, Section 5.2 classifies the Internet-wide NCP based on its capabilities and required duties. Some challenges in the Internet NCP depends on the way it operates and its proxying capabilities. Section 5.3 addresses issues and challenges that come out in the most appealing architectural solution of the Internet-wide NCP. It particularly addresses issues and challenges concerning NAT/Firewall and traffic diversion in the public infrastructure. Finally, Section 5.4 describes the communication framework for the Internet-wide NCP based on the UPnP remote access specification.
5.1 Single or Dual Proxy Architectures

There can be two possible architectural solutions for the Internet-wide NCP depending on the presence of local NCP instance. Fig. 5.1 presents the architecture considering only single NCP instance that is located anywhere in the Internet. Thus, no local setup or infrastructure is required. This single NCP architecture faces some issues if multiple devices are present in the local network. The Internet Network Connectivity Proxy (INCP) instance will be able to impersonate presence of sleeping client device for all remote peers external to the local network. However, if a device is trying to access a sleeping device in its local network, the INCP instance will not be able to maintain its presence as the packets intended for sleeping client device will travel back and forth the Internet according to the standard networking policies.

![Single proxy architecture of Internet-wide NCP.](image)

The limitation of single proxy architecture can be overcome by considering also a local NCP instance as shown in Fig. 5.2. In this dual proxy architecture, the Local Network Connectivity Proxy (LNCP) is responsible to manage all the local network generated traffic on behalf of sleeping devices while the INCP manages all the traffic from external peers/devices which is intended for the sleeping client devices. The client devices are only aware of the LNCP and communicate with it. The LNCP effectively collaborate with the INCP making a more powerful service. It is the responsibility of LNCP to keep the INCP updated about the client devices, their operational status and requested behavioral rules. The dual proxy architecture also effectively address the NAT/Firewall issues which may be present in the local network site and hinder the communication with client devices as well as block the wake-up packets sent by the INCP. The local security policies will not be comprised if the connection is always initiated by the LNCP with INCP. Further, the LNCP based on its processing decision or based on the instruction of the INCP, will manage the waking up of sleeping client devices whenever desirable.

5.2 Architectural Solutions based on Capabilities of Internet-NCP

This section presents two possible architectural solutions for the INCP based on its mode of operation and proxying capabilities; (i) Service-Specific Internet-wide NCP (SS-INCP) and (ii) General-Purpose Internet-wide NCP (GP-INCP). Both architectural solutions have their own benefits and challenges.
5.2.1 Service-Specific Internet-wide NCP

The SS-INCP considers multiple INCPs, each of them is responsible to cope with a specific application/service or subset of applications. Fig. 5.3 considers proxying service provided by each Application Service Provider (ASP) network (e.g., Skype, Yahoo IM, MSN messenger etc) to its client devices. Thus, the SS-INCP allows application developers to have their own proxying functionality without revealing details of their protocols and internal specifications to the LNCP or client devices. This approach will solve the issues related to proxying of proprietary applications (close source/unavailable source code). Further, the SS-INCP provides more easy management of security and cryptographic issues by exploiting trustworthy relationship already established between the ASPs and their clients.

The SS-INCP is the simplest possible architecture of INCP without any complex requirements. The main objective of SS-INCP is to avoid NAT/Firewall issues and issues related to traffic diversion in the public infrastructure. These objectives can be easily achieved if the LNCP always initiates connection with INCP, forwards packets to it for processing and receives back the response packets on the same connection. The generic scenario during NCP client sleeping period is depicted in Fig. 5.3. It can be observed in Fig. 5.3 that all of the traffic addressed to sleeping client (both from local devices as well as external third-party
devices) is managed by LNCP hiding details of having multiple INCP instances. The LNCP checks if it can generate the response (when response is simple such as ARP, ICMP PING, DHCP, etc) otherwise forwards the packet to the respective INCP for processing (e.g., applications heart-beat messages). The main benefit of the SS-INCP is ease the proxying of thousands of applications which could not be implemented locally by the LNCP.

Since all network traffic from local hosts and external third-party hosts is managed by LNCP, it is easier to wake-up sleeping NCP client device whenever desirable (e.g., new incoming TCP connection request on specific protocol and port). The LNCP forwards only those sniffed packets to INCP which qualify certain conditions based on protocol, port number, IP address and/or data field and should be intended to the sleeping NCP clients. The packets screening is done in accordance with the applications requested to be proxied by the client devices. The client devices may be behind NAT and two or more clients may have same private IP address at the INCP. However, the INCP can be able to differentiate between them if a UUID is assigned to each client device. The SS-INCP is incapable to support client devices mobility because all of the traffic is received and managed (respond directly or get response from respective INCP) by the LNCP because the INCPs are not the complete replica of LNCP. The INCPs are only supposed to perform complex proxying functionalities related to proprietary closed source applications/services.

### 5.2.2 General-Purpose Internet-wide NCP

The GP-INCP architecture considers only a single global INCP instance capable of proxying all the network based applications and protocols. Unlike SS-INCP, here the INCP is complete replica of LNCP. The LNCP shares the whole set of rules registered by client devices with the INCP (except those rules only manageable by LNCP such as ARP, DHCP etc). The LNCP can be designed to maintain the sleeping client devices presence only inside the local network. While the INCP impersonates the presence of sleeping client devices to all third-party peers/hosts external to the local network. In short, the LNCP receives and processes packets coming from the local network devices (shown with purple line Fig. 5.4) while the INCP receives and processes packets from external devices (shown with green line Fig. 5.4). Similar to SS-INCP, the client devices only interact with the LNCP and are unaware of the presence of INCP. The LNCP shares the registered client devices and their set of requested behavioral rules with the INCP.

![General-purpose INCP architecture](image)

*Figure 5.4. General-purpose INCP architecture. LNCP only manages traffic coming from local devices (purple line), whilst INCP manages traffic coming from external devices over the Internet (green line).*

The GP-INCP faces several critical challenges in practical realization and has more complex requirements but from the perspective of energy savings, it provides higher energy savings compared to SS-INCP.
by reducing traffic in the access link and boost-up its build-in power savings policies. Usually, in home environments there only is a single host per site; while the host is sleeping, no traffic is expected to flow on the ISP link connecting the HG with the provider’s access concentrator (shown as the red dashed line in Fig. 5.2). That link could therefore be put in a low power mode. A similar consideration also holds for small offices at night-time, when no one is at work. Shutting down last mile links is an effective solution for cutting off the consumption of telecommunication networks. Further, for vast majority of complex applications, the interaction of remote peers only take place with the INCP. In particular, the client may be awake and the LNCP may thus not covering it, while the INCP may still be acting on behalf of that client: this could be useful when the client device is woken from a local peer, to avoid restoring the access link if it is not going to be used. The power consumption in access link per subscriber is quite small (ADSL: 2-4 W, VDSL: 6-10 W in [55]), however, savings from each subscriber link can accumulate to huge amount of savings considering millions of subscribers around the world.

5.3 Issues and Challenges concerning General-Purpose INCP

However, the GP-INCP (addressed in Section 5.2.2) is the most appealing Internet-wide NCP architectural solution in terms of achieving energy savings; it faces several critical issues and challenges. The issues are specifically related to NAT (how INCP knows the client devices external public IP addresses and local NAT mapping?), Firewall (blocks INCP communication with the local network or blocks wake-up packets from INCP) and traffic diversion in public infrastructure required by the INCP to access packets intended for sleeping devices. This section particularly addresses these issues and analyzes or proposes possible solutions to tackle them.

5.3.1 NAT/Firewall Issues

Firewalls are usually configured to block all incoming packets but those belonging to connections previously initiated by an internal host. Often users are allowed to connect to any external service (that is, any port on any server), although in many offices and corporate environments external access could be limited to a few protocols (e.g., only HTTP/HTTPS for visitors); new incoming connections may be only allowed to internal public servers. NATs translate internal IP addresses and port numbers (usually falling into a private space) into external IP addresses and port numbers. Two main issues arise from this:

1. How to map the connections from the Internet to the local IP addresses and port numbers.

2. If the INCP do not have the knowledge of NAT mapping, how the client devices or LNCP will be able to register suitable behavioral rules with the INCP.

The first issue is quite common in every NATted system; the main solution is to have manual and static mapping for all of the applications and services which are accessible from outside the local network. Several solutions can be proposed for the second issue. First, remote devices need some mechanism to get the knowledge of the IP address and port number used by every application and service (for example, DNS or other kind of service registration). The NCP client devices can use also the same mechanism as the remote devices in order to get the knowledge of external IP address and port number used for each of its listening service and application. However, this strategy can only be used for incoming connections. The second possible solution can be the INCP-aware NAT which take care of updating/changing the behavioral rule’s IP address and transport port number when the registration messages traverse the NAT (meanwhile, a NAT rule could be added automatically for accepting incoming connections on that transport port). To get access to the current entries in the NAT table, a NAT component could be developed. The third possible solution is delegating the NATting job to the INCP and each NCP client device establish a tunnel to pass packets to the INCP.
Address space collision is another common issue with the NAT. Any address space can be used behind a NAT, and very often private ranges are chosen. If the INCP is serving more than one subnets (as reasonably happens for an Internet service) some of them could use the same private IP addresses range, hence the INCP must avoid confusing the overlapping spaces. The Virtual Private Network (VPN) already faces and solves this issue. The simple possible solution is to use a unique identifier for each site and pre-fixing all private addresses with that identifier. Obviously, this process may make the internal data structure more complicated. Another solution is using unique ID e.g., UUID for each client device along with its IP address. Thus, the INCP will distinguish between client devices not just based on IP address but also on the UUID.

Another common issue is the state persistence on behalf of sleeping devices at the security gateways. Firewalls and NATs identify packets related to a specific connection by using a set of rules such as external and internal IP addresses and port numbers. This process decides whether to allow packets to pass through or discarded. If no packet is seen for a given time period, the connection is closed and the set of rules for that specific connection are removed. This process is strictly important to avoid wasting resources due to dead or stalled connections. For sleeping client devices, this timeout is most probably to occur and their connection will be closed. To avoid this issue, the INCP must send periodically some bogus packets on active connections related to sleeping client devices in order to prevent connection suspension and to refresh the timer for those connections. This will prevent the security gateways from dropping the connections related to sleeping client devices.

Finally, the wake-up packets sent by INCP will also be blocked by Firewalls in order to avoid any security threats or issues (as hackers may make the power management policies ineffective by attempting to keep the devices powered-up). A better and safer solution would be sending wake-up packets on the connections already established between the LNCP or NCP client devices and INCP.

### 5.3.2 Traffic Diversion in Public Infrastructure

Traffic diversion in public infrastructure is required for the INCP in order to get access to the packets intended for the sleeping client devices. However, traffic diversion in public infrastructure is much more challenging task due to the routing policies and numerous routers and possible end points. Traffic diversion in public infrastructure implies working on the IP protocol as the IP addresses are the only way of packets forwarding over the Internet. This section presents the possible architectural solutions to achieve the objective of traffic diversion at the IP layer.

**Diversion by Routing Protocols**

Changing routing policies is most simple solution for traffic diversion across the INCP. Unfortunately, it is not easy to divert traffic path over the Internet as the routing policies are controlled by different administrative domains. The traffic diversion will become a liable solution if the INCP is placed at the ISP and cover only for devices associated with that ISP. Thus, the traffic will be routed according to standard policies when the client device is awake (see the red line in Fig. 5.5(a)) and will be diverted towards the INCP by modifying the routing tables when the client device is sleeping (see the green line in Fig. 5.5(b)).

There are several issues in this traffic diversion strategy. The first issue concerns with the multiple client devices located in the local network. Some of the client devices may be sleeping while some are active. The traffic diversion strategy must guarantee that the active devices receive their packets in the normal way. This issue is not a big concern if NATting is not used in the local network. Some possible solutions to this issue could be: (i) selectively routing the packets towards the INCP which are related to the sleeping client devices, (ii) the INCP could divert or forward all the packets which are intended for the active devices other than its sleeping client devices, (iii) replicate the traffic towards the INCP which are addressed to the local network. Each of the solution has its own drawbacks in terms of routing protocols overhead, network overhead or the scalability of the INCP service (depending on the device hosting the INCP service).
The second issue in this traffic diversion strategy is related to the NAT. Usually, NATing is used to save IP addresses and multiple devices of the local network share the same external IP address. In this case, traffic diversion strategy must also inspect the port numbers which brings performance degradation and more technical challenges. It is not easy for the INCP to use the port numbers used during the rule’s registering and to infer from it the port numbers used in the public domain. The possible solution could be to make the INCP NAT-aware. This may be easier if the NAT service is used from the ISP access gateway which makes the translation table available to the INCP. Thus, the INCP rules would be aware of the public networking parameters if the user NAT provide such information to it. If there is no way to access user NAT translation table, one possible alternative could be inspecting all network traffic with the hope that the internal port numbers are conveyed within the packet’s body either sooner or later; however, this technique will not work if the network traffic is encrypted.

![Diagram](image)

Figure 5.5. Diversion by routing protocols: a) When the NCP client is awake, traffic follows the normal path towards access gateway of the local network. b) When the NCP client is sleeping, traffic is diverted to the INCP location.

**Diversion by Source Routing**

The Internet Protocol (IP) provides a source routing option which enables the devices to specify the packet route in the header. This feature can be used for the INCP service where client devices specify packet route forcing packets to pass through the INCP (see the green line in Fig. 5.6). This strategy might be useful for the connection oriented protocols where the packets follow symmetric paths. For example, IETF communication requirements for Internet devices [56] mandates TCP implementations to do that:

“When a TCP connection is OPENed passively and a packet arrives with a completed IP Source Route option (containing a return route), TCP MUST save the return route and use it for all segments sent on this connection. If a different source route arrives in a later segment, the later definition SHOULD override the earlier one.”

and also UDP applications are expected to answer back incoming packets using symmetric paths

“An application based on UDP will need to obtain a source route from a request datagram and supply a reversed route for sending the corresponding reply.”
The source routing may not be suitable for INCP service based on some important considerations. First, the source routing is disabled in IPv4 routers by default as it is normally considered as a security threat. Second, it may not be useful for INCP since transparency with respect to third-party device is an important requirement for the INCP operations. Thus, the source routing cannot be used for incoming connections while can only be used by client devices in outgoing connections. Third, the INCP will face all the previously discussed issues if the client devices are behind Firewall/NAT.

![Diagram of INCP](image)

**Figure 5.6.** Diversion by source routing: The green line represents outgoing connections; the INCP takes over the connection when the client device is sleeping. The red line represents incoming connections when source routing could not be used; the INCP cannot take over the connection as it does not lie in the path of these packets.

**Diversion by Connection Splitting**

The INCP may be used an intermediate node that split every connection related to any application between the client devices and their remote peers. Thus, first the connection will be established with the INCP which will then establish connection with the final destination. This way, all packets will be addressed to INCP and will pass through it. Since the INCP will be covering for thousands of client devices, it must need to change the source port used for the connection. This can bring side effects if some applications are also carrying this information in their payload.

This approach requires some changes to the applications which require proxying service. Such applications need to connect to the INCP instead of directly connecting with the remote peers; otherwise connection splitting can be made as default behavior by modifying the standard socket API. This approach will not scale well for large number of client devices due to resources required for splitting connection oriented protocols (i.e., TCP). However, more flexibility and scalability can be achieved if the job of connection splitting is assigned to the applications. Thus, burden on INCP will be reduced by splitting connection only for applications which require the proxying functionality.

The address of the client devices should be resolved to the INCP hosting device for the incoming connections. Thus, each client device should have the knowledge of the port numbers used by the INCP for its listening applications.

**Diversion by Tunneling and NATing**

Another possible solution is to establish a tunnel between each of the client device and the INCP. Thus, the INCP can behave as the default gateway and NAT all of its client devices with its own IP address (see Fig. 5.7). When Firewalls are present, this solution is unlikely to be feasible: tunnelling is usually not allowed because it could be used to get around or bypass security policies. However, this solution does not require splitting all connections at the INCP; indeed, several approaches could be used for managing TCP flows. This approach can potentially bring more scalability than the previous solution; this approach also avoids modifications to all applications that need network presence during sleeping periods.
Establishing tunnel between the INCP and its client devices will make the rule registration process easier. Even if the tunnel endpoint is changed by the NAT, encapsulated packets will have their original header, thus the INCP can communicate directly with the client devices as if they are located in the same network. Registration of rules will be a simple matter of converting local IP addresses and transport port numbers of the client devices according to the NAT translations within the proxy. Since the INCP based NATing is away from the local networks of client devices, some additional efforts will be required to keep the remote peers aware of the transport port and IP address to be used for each application or service. Further, there will be additional network overhead as the tunneling is based on the packets encapsulation. The overhead of VPN varies based on the protocol layer used (IP, UDP, application-specific).

![Diagram] Figure 5.7. Diversion by tunneling and NATing: The client devices establish a tunnel with the INCP. The INCP proxy their connections during their sleeping periods and NATs all traffic coming from them.

**Diversion by Mobile IP**

The concept of Mobile IP (MIP) can also be used for the objective of traffic diversion in the public infrastructure. The NCP client is considered as the Mobile Node (MN) and is assigned a Home Address (HoA) which is used for the outgoing connections. The INCP is co-located on the same physical device that acts as the Home Agent (HA). Thus, all internal and external traffic will pass through the INCP. When the NCP client is sleeping, the INCP/HA directly answers to the packets intended for it. The general scenario is depicted in Fig. 5.8. The protocol specification also envisions the presence of a Foreign Agent (FA). However, for the purpose of generality we assume no FA is present and all MIP tasks are executed by the client device itself.

The MIP architecture for traffic diversion is somehow similar to the previous approach (diversion by tunneling and NATing) but the HoA of the client device is not NATed. The MIP architecture also brings the mobility management of nomadic devices automatically as it hides movements to the remote peers. However, similar security concerns arises as in tunneling because the MIP solution also goes around the Firewall. Further, there are some common issues related with the MIP architecture including, inefficient network paths, triangular routing and additional overhead due to encapsulation and tunneling. Moreover, the outgoing traffic of the NCP client is likely to be dropped by the Internet gateways (the source address does not belong to local network space). However, there will be no issues with the anti-spoofing filters and triangular routing if the client devices send packets in the tunnel. The requirement of public routable IP addresses for MIP architecture is another drawback in this traffic diversion solution. In the current Internet, indeed NAT is widely used just to save IP addresses. However, this issue will be solved in IPv6 where the address space is almost unlimited and each client device could be assigned a unique identifier with no problems. Further, MIPv6 also solves other issues of the MIP such as security and triangular routing.

In the basic Mobile IP framework, packets travel from the Corresponding Host (CH) to the MN across the HA and this usually results in longer paths than needed (as obvious in Fig. 5.8); additionally, if reverse tunneling is not used, packets sent by the Mobile Host (MH) to the CH follow another path (triangular
routing). The MIPv6 enables MhS to update their CHs with their current Care-of Address (CoA), thus, direct tunneling can be used between the MN and the CH and best path will be chosen. The MIPv6 can be used to optimize the behavior in INCP architecture. Direct tunneling will be used when the client device is awake and it updates its remote peer about its CoA (shown with red line in Fig. 5.9). The client device will send another update that its CoA is now the HoA when going to sleep state. This means that tunneling is not required any more and the client device or MN is present at home (shown with green line in Fig. 5.9). Thus, the client device is sleeping and INCP impersonates its presence as if the client device is at home.

Figure 5.9. Diversion by Mobile IP: Optimization of INCP behavior with MIPv6 architecture.

5.4 Communication Protocol

Since, this dissertation targeted the design of a cooperative NCP framework, suitable communication protocol is required for information exchange between the client devices and LNCP, LNCP and INCP or between the client devices and INCP (single proxy architecture). There can be several choices for flexible and reliable communication protocol, however, we selected UPnP architecture for communication in the NCP framework due to its several interesting features such as zero-configuration, auto-discovery and seamless networking. The UPnP protocol has been originally proposed for seamless autonomous communication.
between devices in local network. The UPnP forum has recently introduced UPnP Remote Access (URA) specification for secure communication between devices located in two different networks [57]. In simple words, the URA specification extends the UPnP coverage beyond LAN boundaries and enables the CP/CD located in one network to securely communicate in a seamless way with another UPnP CD/CP located in a remote network. The URA specification also addresses key challenges and issues arise during communication between devices in two remote networks [57].

Based on the architectural choice of the Internet-wide NCP, the client devices may communicate with LNCP or INCP. In case of dual proxy architectural choice, client devices communicate with the LNCP in the similar way as addressed for the local network NCP in Section 4.6. However, communication between LNCP and INCP will pass through Internet and will be based on the URA specification [57]. Similarly, URA specification can also be used for communication between client devices and INCP in case of single proxy architectural choice of Internet-wide NCP. Since, URA specification allows devices to communicate which are located in two different networks, it may face several issues especially related to NAT/Firewall. It is possible that the client devices/LNCP and INCP may use same private IP address located in two remote networks. The URA specification proposed different approaches to avoid such address collision. These approaches include: (i) Using a randomization function to assign a private address space by the HG instead of using the manufacturer default, (ii) Reduce changes of IP address collision by transitioning to IPv6, (iii) Using an Application Layer Gateway (ALG) on both end-points of the remote connection that can translate the original IP address to a different IP address only when the IP address collision is detected. Usually, ISP allocates dynamic private or public routable IP addresses to end users. The current URA specification only works if public IP address is assigned to home network by the ISP. The URA specification proposed an approach to find external routable IP address in case of private IP address assigned by the ISP. The proposed approach is based on the Session Traversal Utilities for NAT (STUN) protocol to traverse the NAT boxes at ISP. STUN protocol is addressed in RFC5389 [58]. The STUN is a lightweight clientserver network protocol that allows devices behind NAT to discover the NAT mapping. The protocol operates essentially as follows: the client sends a message (known as a binding request) to a STUN server in the public Internet. The STUN server responds with a success containing NAT information in its payload. Simply, the STUN protocol finds public IP of the NAT boxes, Firewall settings and the open ports that can allow incoming traffic.

![Figure 5.10. UPnP Remote Access scenario.](image)

The URA specifications proposes usage of Remote Access Server (RAS) running on a device (e.g., PC, home gateway etc) in the home network [59]. First, the RAS in one network establishes a secure tunnel with the RAS in other network. Afterwards, it exposes the devices and services available in the local network to another RAS in the remote network/device [60]. Thus, both the client devices/LNCP and INCP need to implement RAS. The RAS also embeds a STUN client for discovering NAT mapping. The typical scenario for URA is depicted in Fig. 5.10. The remote access connection can be initiated from either side, the client device/LNCP RAS or the INCP RAS. The LNCP RAS exposes its embedded UPnP CD and CP to the RAS of INCP. Similarly, the RAS on INCP hosting device exposes its embedded or local UPnP CD and CP to LNCP. For successful establishment of connection, the client device RAS needs to know the remote NCP.
RAS IP address and other security associations (e.g., open ports for UPnP) and vice versa.

The quite generic Internet communication paradigm between the client device (single proxy architecture of Internet-wide NCP) or LNCP (dual proxy architecture of Internet-wide NCP) and the INCP is depicted in Fig. 5.11 [57]. The URA specifications suggest the use of dynamic DNS for RAS which may be running on the gateway device or the UPnP device itself. This allows UPnP devices to have fixed identity over Internet even if their IP addressed is changed. As the first step in the NCP framework, the RAS on client device/LNCP and INCP establish a secure transport channel for secure communication across the Internet. Following this, the ‘Discovery Agent’ discovers the local available UPnP devices or services in the network (depicted by purple line in Fig. 5.11) and control their visibility and advertisement in the remote network (depicted by red line in Fig. 5.11). Fig. 5.11 illustrates a two way UPnP communication scenario where both the INCP and its client device or LNCP implement a CP as well as a CD. The CP on client device or LNCP can register proxying request for different applications or protocols with the INCP (depicted by brown line in Fig. 5.11) while the CP on INCP may send some power management commands e.g., instructing LNCP to wake-up a client device when certain application updates are received (depicted by green line in Fig. 5.11). In short, the remote UPnP communication specification is primarily different from local UPnP communication specification in establishing a secure channel/tunnel between two networks (i.e., between RAS-RAS). After a secure channel is established, the UPnP devices in both networks communicate in a similar way as if they are located in the same network (with the help of Discovery Agent in Fig. 5.11).
Chapter 6

Proxying for Mobile Devices

The mobile devices are usually equipped with low power processors as they normally operate on battery power and cannot provide comparable energy savings that can be achieved from static desktop PCs. However, reducing energy waste in mobile devices can be more important as the battery capacities are quite limited. This chapter investigates possible proxying strategies for maintaining presence on behalf of mobile devices. First, Section 6.1 motivates the importance of proxying for mobile devices. The rest of the chapter addresses two possible proxying solutions. In particular, Section 6.2 addresses the first solution that is the NCP concept and its applicability for mobile devices. Specifically, it analyzes the key issues and challenges arise due to client devices mobility and presents different possible architectural solutions. Following this, Section 6.3 addresses the second proxying solution considering intelligent coordination among daily used devices. This second solution is based on the fact that a user normally use different types of fixed and mobile devices (e.g., desktop PC, laptop, tablet, smartphone etc) in his daily life and all of them have the capability to run almost similar applications.

6.1 Motivation

Today, the energy efficiency for battery powered mobile devices has become one of the hot research topic due to continuous users interest in them (especially because of portability and mobility ease that they can offer). Furthermore, the mobile devices are increasing becoming more and more affordable and powerful and run many different type of applications and services. These applications mostly demand continuous Internet access and transmission over WiFi or 3G/4G which contributes to a large share of overall battery consumption [44]. Further, applications also utilize hardware resources such as CPU, memory and may be some built-in sensors. Previous studies revealed that the Internet connectivity constitutes about 62% of power consumption for a mobile device in idle state [61], [62]. Unfortunately, the battery capacities are significantly restricted for mobile devices due to size and weight constraints.

Proxying can be the most effective solution to reduce energy waste due to background applications and improve the battery life of mobile devices. The NCP is one of the useful proxying approach whose importance has been motivated from the previous chapters. However, its applicability for mobile devices is quite challenging. First, the mobile device may move around from one network to another while sleeping and the NCP may not have its location information to wake it up whenever necessary. Secondly, the NCP concept is based on smart standby which automatically freezes the client device activities when idle by simply putting it into low power sleep state. Smart standby may be possible for some mobile devices (such as laptops), but cannot be possible for mobile devices demanding always ON status (e.g., smartphones for receiving calls, sms etc). Thus, some strategies need to be adopted to freeze activities of applications on such mobile devices while the NCP is proxying their presence and to resume them back when instructed.
by the NCP or the mobile user itself. Communication between the NCP and mobile devices may also be an issue due to Firewall if they are located in different networks. More details about the NCP applicability for mobile devices are provided in Section 6.2.

Another possible proxying strategy can be based on intelligent collaboration among user devices. In daily life, we normally use multiple devices which run the same applications concurrently (especially the smartphone always stays connected). Thus, the devices can collaborate and guarantee to run applications on one and only one user device at a given moment. Simply, the applications will run on either a smartphone or tablet or laptop or desktop PC or any other device currently under active use. For example, if the user is using a desktop computer, it will run all applications while other devices will reduce energy waste by stopping the applications and/or putting the device into sleep state. Since smartphones or tablets are always ON devices, stopping applications and their activities will be quite convincing in improving battery life as they utilized continuously the hardware resources (WiFi, 3G/4G, CPU, memory, sensors etc). Now, when the user is leaving home or office and not using anymore desktop computer, it can enter into sleep mode while a smartphone will maintain the presence of applications. However, to meet the basic need of user for remote access to his daily used fixed network devices (e.g., PC in lab or home), a very basic NCP with simple features or behavioral rules (such ARP, PING, DHCP, Wake-On-Connection) may be running on the gateway device. Proxying of applications by such NCP will no longer be required as they will always be running on one of the user device. Further, the NCP don’t need to care about the remote access issues for mobile devices as they normally move around with the users. The basic challenge in this energy saving strategy will be autonomous and seamless communication among different user devices without requiring any user input or consideration. However, the possible solution can be the UPnP architecture which will guarantee zero configuration and seamless communication among devices.

6.2 Supporting Clients Mobility in NCP Framework

Generally, the NCP concept for mobile devices has the same implications as for any other fixed network device, but there are two additional concerns:

- Mobile devices may change their point of attachment when in sleep mode, thus affecting both NCP operation and active sessions.

- Mobile devices may move out of wireless range quickly and might not have time to start NCP operation.

This section addresses the NCP applicability for mobile devices. In particular, it is organized as follows: Section 6.2.1 addresses the issues due to mobility of the client devices especially during their sleeping periods. Further, it also lists the mobility implications in different NCP architectures (such as on-board, standalone, endpoint etc). Section 6.2.2 presents architectural solutions for supporting devices mobility in the NCP framework. Finally, Section 6.2.3 presents the concluding remarks.

6.2.1 Main Issues Concerning Devices Mobility

Mobility management in data networks is a topic that has been debated for more than twenty years and includes several aspects: personal mobility, service portability, terminal handover and session migration. A broad range of approaches has been proposed to deal mobility at the different layers of the protocol stack; most of them deal with the specific issue of handover [63–67].

Many aspects of mobility could involve a device in low power state: the user can bring a different device with him or may wish to move its running multimedia application, the device can move across different networks, and so on. In most of the conceivable scenarios about personal, session and service mobility, device interaction is required to start applications, to shift current running sessions, to register
users and so on; thus the device must be awake. In a few number of cases, an NCP could manage simple mobility operations, for example registering user’s presence at a new device and could delay wake up until a new connection arrives.

Network hosts are assigned IP addresses that in the Internet infrastructure are used both as identifier and locators. If a device moves around and changes its point of attachment, it shall change its network address to stay compliant with the Internet architecture. That leads to several implications:

- Devices in sleep mode are not aware of movement and thus cannot update network configuration.
- The NCP has out-dated information about device location, thus cannot wake up the device if required.
- The suspended sessions with remote peers will be lost after device wake-up. If mobile device changes its IP address, there is no way to update the endpoints of existing connections at the remote side, hence all sessions would be lost and that would waste the NCP’s work.

As the proposed NCP framework claims to maintain network presence transparently for third parties, it must be ready to deal with such issues. Prerequisite for this feature is the capability of tracking the location of sleeping hosts. Tracking location of sleeping devices allows being aware of their current position within the network (e.g., current IP address).

**NCP Clients Location Updates**

The capability of tracking the location of sleeping clients (e.g., IP address) is one of the prerequisite of the NCP operation. The NCP needs location information when it has to wake up the device, but would not normally need to be updated timely at each movement. Thus, the tracking and location information can be managed by another server and the proxy could query it when necessary.

Many times waking up mobile devices might not be desirable and thus movement detection would not be necessary. The user may prefer no external wake up to prevent battery from discharging; further, if the device is carried around enclosed in bags or cases, awaking it may be harmful and can damage it (due to insufficient ventilation may cause it to get overheated). The mobile device’s position could be updated with two approaches. The device periodically wakes up, checks its current link and, if different from the previous one, gets a new address and registers this information with the NCP or another location server. However, to have up-to-date information, frequent wake-ups are necessary which may result in shorter battery lifetime. The second solution would be to integrate simple network detection features in NICs: that will trigger device’s wake up on link attaching/detaching operations and then the device takes care of registering the current location with NCP. The second solution is undoubtedly more efficient than the previous one, as the client is at most awake only when necessary.

**Mobility Implications in Different NCP Architectures**

Mobility of sleeping devices affects two network parameters: IP addresses and link-layer forwarding mechanisms. Depending on the architectural choice (e.g., on-board, switch/router, standalone etc), the NCP may be affected by none, one or both of these elements. In end-point proxying scenario, the NCP on remote peer knows the current location if it gets updated by an agent working on the NIC of sleeping device. Thus, the NCP can wake up the device when needed; further, it can NAT between the current IP address and that used to set up current connections, thus hiding mobility to the above applications. Incoming connections cannot wake the device up, but this is a general limitation of this architecture. In case of on-board proxying scenario, the NCP moves together with the device. This is undoubtedly an advantage for movement detection; moreover, there is no problems for waking the host. However, once the NCP’s address has changed (usually the same as the device) it cannot get anymore packets addressed to the sleeping device. New incoming connections can be seen if the new device’s location is made available to all other devices, that is quite simple with dynamic DNS.
In a stand-alone/router/switch architecture the NCP does not move, thus it can behave like an ‘anchor’ for traffic addressed to nomadic sleeping devices. Once it gets the up-to-date location of its client, it can wake it up and manage incoming connections; further, when the client awakes, the NCP can translate between the new and old addresses for connections set up with a previous address. In a dual-proxy architecture, the most convenient choice is having an on-board proxy and a stand-alone proxy. This solution brings all benefits of on-board and stand-alone proxying: devices can be woken up, incoming connections can be managed at both proxies (answering both the old and new addresses) and connections can be NATted. A brief summary of these main features is addressed in Table 6.1.

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Wake-up</th>
<th>Incoming Traffic</th>
<th>NATting</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-board</td>
<td>Yes</td>
<td>Only new connections</td>
<td>No</td>
</tr>
<tr>
<td>Stand-alone</td>
<td>Yes</td>
<td>Yes, new connections must use old address</td>
<td>Yes</td>
</tr>
<tr>
<td>Dual proxy</td>
<td>Yes</td>
<td>Yes, both new and old addresses</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Link layer forwarding mechanisms should have little or no impact on wake-up. Wake-up packets are broadcasted because the MAC address of sleeping device is likely to be expired in forwarding caches of network equipment. On-board NCP keeps the device’s address active; however, such NCP is likely to send traffic and thus updates the forwarding caches with the new location (in any case, the proxy can be easily configured to send some void datagrams to force this update, e.g., Gratuitous ARP packets). Placing the NCP in LAN makes it a good ‘anchor’ for mobile devices, both within the local network and in the Internet. However, all traffic addressed to mobile devices should make use of their ‘home’ address (where NCP lies), otherwise NCP will not be able to catch traffic addressed to sleeping devices. It is important to note that link layer issues may not apply for some types of mobile devices (such as smartphones and tablets) which are always ON devices. Thus, wake-up broadcasted packets cannot be used. Further, they may be using 3G/4G connections instead of lying in some local network where NCP is located. However, some strategy must be used to stop/freeze applications and their activities on such mobile devices and to resume back when requested by the NCP. Stopping applications or their activities will help improve battery life considerably as they continuously utilize hardware resources (CPU, memory, 3G/4G, sensors etc). Placing the NCP in the Internet makes it an ideal solution for all mobile users that either do not have a home network or do not wish to deploy their own NCP. Thus a global Internet-based NCP located anywhere in the Internet could simplify mobility management (presented in Chapter 5).

### 6.2.2 Architectural Solutions for Mobility Management

This section proposes possible architectural solutions for maintaining network presence in case of mobile sleeping devices while keeping remote peers unaware of any issue concerning low-power management and mobility. In particular, it addresses architectural solutions based on mobile IP and virtual NAT.

**An Architecture Based on Mobile IP**

This solution fits very well all scenarios where a mobile device most of the time lies in a home network. Exploiting the MIP [68] framework keeps all layers above the network one unaware of mobility issues, thus making this framework suitable for any service and application. Here, two architectural designs are considered: (i) HA and the NCP running on different devices, (ii) HA is integrated into the NCP.

The NCP and the HA are located in the same network; no FA is required to keep the architecture as general as possible and suitable for the current Internet (as shown in Fig. 6.1). When the device is at home,
the HA does nothing while the device and the NCP behave like in a usual LAN scenario. In particular, if the device is asleep, the NCP catches its traffic and acts on its behalf as shown in Fig. 6.2. When the device moves to a foreign network, it keeps its IP address and additionally it gets a local one; a tunnel is used to exchange packets with the HA. When the device is awake, the NCP does nothing while the HA and the mobile device operate according to MIP framework (as shown in Fig. 6.3. In MIPv6, mobile device can use the route optimization procedure to divert connections directly between the Corresponding Node (CN) and themselves. This is not shown in the picture for brevity; however, that simple means there are also tunnels with some corresponding peers in addition to that with the HA. When the device goes to sleep, the NCP starts catching traffic addressed to the mobile device and acting on its behalf. This scenario is depicted in Fig. 6.4. In MIPv6, traffic must be diverted back to the HA to enable the NCP to see packets addressed to the mobile device.
The main matter in this architecture concerns what happens when the MN is away from home and is sleeping. In this case, both the HA and NCP should catch packets addressed to the MN, both of them will try to resolve the host’s IP addressed into their own MAC. To avoid such conflict, two approaches could be followed depending on whether the MIP behavior is changed or not to account for the presence of sleeping devices. In any case, the NCP takes the preeminence in standing for the host.

In first approach, the NCP announces the MN at home. The MIP operation is not affected by sleeping devices and the standard behavior is preserved. However, as the HA thinks the MN be at home, it tears down the tunnel and the NCP must become aware of the MN’s current position and all following movements in order to send wake-up messages if needed. In MIPv6, the NCP should also update any CN that has previously carried out route optimization with the MN; the opposite should be done when the MN wakes up. There are some major drawbacks and limitations with this approach. According to the standard behavior, the MN would keep the tunnel with the HA up when asleep, and that would be inconsistent with the state at the HA. Further, the MN registers with the HA when awake, but notifies its position to the NCP while asleep. The MN should also share with the NCP its current status about route optimization, so that the NCP could divert traffic back to the Home Network. Finally, when the MN wakes up or is awoken, the HA has to be updated like the MN was gone away and has to set up again the tunnel; this could delay communication and may negatively impact the interactiveness with sleeping devices.

In second approach, the HA and the NCP cooperate for handling sleeping devices in foreign networks. In this case, an extension is added to standard MIP to make it aware of the ‘sleep’ condition (that could be inferred by seeing the MN’s traffic or, better, it could be notified by either the NCP or the MN). When this condition occurs, the HA stops catching packets for that MN and keeps the tunnel alive for delivering wake-up messages and it continues to update the node’s position. The NCP keeps working as usual, with no additional tasks; wake-up messages are sent locally. The HA is also responsible to detect wake-up messages and to tunnel them to the CoA of the sleeping host. In MIPv6, the NCP should anyway take care of diverting optimized paths back to the home network when the MN goes to sleep, and the MN should accordingly update its state: When the MN comes out from sleeping mode and the NCP stops proxying it, it should be ready to get all packets from the HA’s tunnel.

The second approach seems better, although it requires slight modification to MIP behavior. Further, the MIP framework allows extensions for the standard signaling to be added as new features are introduced; supporting low-power devices could indeed be an interesting add-on for this protocol. In second architectural design, the HA is integrated into the NCP. This would make operation even easier, as no conflicts arise between the two functions. The NCP would catch packets addressed to MNs both to maintain network presence and to forward them to their current position. Functional description is the same as for the previous design.
An Architecture Based on Virtual NAT

The Virtual NAT (VNAT) [69] was mainly designed for end-to-end applications, namely the remote host is aware of mobility and work together with the local mobile device. The prerequisite for keeping remote peers unaware of mobility is making use of an anchor which is Home Network (HN) as in MIP and where the local device is looked for. Packets addressed to a MN are hence forwarded to its HN; when the MN is present, packets are delivered as usual; if it is asleep, the NCP covers it. When the device moves away from its HN, its VNAT Migration informs the peer entity on the NCP. The NCP then starts NATting packets to/from the MN, as shown in Fig. 6.5. In the basic end-to-end paradigm of VNAT, new connections should be set up by using the current IP address as virtual identifier, thus avoiding the NAT overhead as long as possible; however, the transparent proxy paradigm used in this design requires all packets to cross the NCP for mobility and sleep support. Thus, all new sessions (both incoming and outgoing connections) should use the HoA of the MN, as in the MIP framework.

![Diagram of VNAT architecture](image)

Figure 6.5. Proxifying mobile hosts with VNAT. Packets from/to travelling hosts are NATted both at the NCP and at the client.

In VNAT no tunneling is required for mobility management. That implies less overhead in packet headers, but may rise more issues for firewall and NAT traversal: with a tunnel, just a single connection need to cross the firewall while in VNAT approach each single connection should be allowed, and this is very unlikely in almost all scenarios. Waking remote hosts is just a matter of sending a Magic Packet to their current location. The presence of security intermediaries can be an issue for this task too. Most issues concerning VNAT does not apply when it is integrated in the NCP framework: in particular, the NCP does not move and applications always use the same HoA which is uniquely assigned to their device.

6.2.3 Concluding Remarks

In this section, we have addressed a challenging topic about the NCP applicability for mobile devices. Achieving this objective is quite challenging but can provide benefits especially in terms of improving the battery life of mobile devices [70,71]. It is important to understand the issues and challenges that will arise if the NCP client goes out of the local network. Thus, the main objective of this section is to point out issues and propose possible solutions especially related to client’s mobility in low power states. Running the NCP on-board of mobile devices makes the NCP service available anywhere; the NCP can easily detect movements, can change network configuration and can wake the host to do the same. However, the basic objective of this architecture is to support transparent proxying for the purpose of generality and short-term
applicability and does not allow maintaining existing connections with other hosts. Using a stand-alone NCP, a NCP running on network equipment or a dual proxy approach are therefore the most promising solutions for mobility management, although sleeping devices must take care of updating their location while they move around. This section also presented two architectural solutions based on MIP and VNAT in order to support the device’s mobility in NCP framework. The architectural solutions consider transparent proxying without support from third parties and requires anchors to be present at the home network of mobile devices. The proposed solutions have considered both using tunneling and NATting to forward packets to the mobile devices in foreign networks, and discussed that tunneling may be most effective for NAT/firewall traversal at those remote sites.

6.3 Reducing Energy Waste Through Intelligent Coordination Among Daily Used Devices

This section presents another strategy for reducing energy waste in network devices through intelligent collaboration among them. It is based on fact that a user normally use multiple devices in his daily life and all of them have the capability to run almost similar applications. Thus, why not run applications on one of the user device while allowing his other devices to save energy through smart standby or by stopping applications and their activities. This strategy can bring additional benefits or drawbacks over the main NCP concept which are being addressed in this section. Briefly, the section is organized as follows: Section 6.3.1 presents an overview of the proposed strategy and motivates its importance. Section 6.3.2 addresses possible logics for controlling applications and their presence on different user devices. Section 6.3.3 presents possible architectural solutions for realizing intelligent coordination among user devices. Section 6.3.4 briefly addresses the communication protocol. Finally, Section 6.3.5 presents concluding remarks by address the pros and cons of proposed strategy with respect to the NCP concept.

6.3.1 Overview

Today, we have many different choices for computing device ranging from smartphone to desktop computer. Desktop computers got popularity in the last decades as a computing device that can run many different applications ranging from entertainment to communication and research. At the beginning, the public available computers were equipped with low memory of 64MB and storage of upto 10GB. Today, modern public available computers can have upto 16GB memory and 8TB of storage. In recent years, the smartphone also rapidly evolved from simple calling device to a high computing device. Today, modern smartphones are equipped with upto 2.5GHz quad core processors, upto 2GB of memory and 128GB of storage. The smartphones are penetrating at very rapid rate in the world and their specifications are improving day by day. Today, the smartphones can run the modern operating systems that were initially available for desktop computers such as Microsoft Windows 8 and Ubuntu 13.10. In simple words, the modern smartphone can do everything that a desktop computer or laptop can do.

The energy efficiency is primarily important not just for fixed but also for mobile network devices. A good example of daily used mobile device is a smartphone which became an important part of our daily life. Most of the energy consumed by smartphone is due to the Internet based applications which not just utilize data connection but also utilize smartphone resources such as CPU, memory and hardware components such as always connected WiFi or 3G/4G or some other built-in sensors. This effect is quite apparent as an Internet-connected smartphone consumes battery 3 to 4 times faster than when it is not using Internet. A good example of daily used fixed network device is a desktop computer. The desktop computers are usually left powered-up 24/7 in the offices or at homes just for the sake of maintaining network connectivity. A user needs network connectivity either for remote access or for Internet-based applications. The remote access objective can be easily achieved by a NCP running on the gateway which simply wakes-up the device when
a new connection request is received for it on a given protocol and port (e.g., TCP:22 for SSH, TCP:3389 or UDP:3389 for remote desktop protocol). However, maintaining the applications presence (especially closed-source) is quite complex and challenging job for the NCP.

An intelligent coordination among user devices to control the applications running on them can provide significant benefits in terms of reducing network energy waste due to high power devices and improving the battery life of mobile devices. In this strategy, the devices coordinate intelligently to decide which one has to run the applications at a given moment. For example, when the smartphone is running applications, the desktop computer can sleep during that time. If the desktop computer is running applications, the smartphone can improve its battery life by reducing the utilization of its hardware components and switching OFF the applications. This strategy will also address some of the basic issues of the NCP concept. The NCP concept is basically proposed to allow unattended devices to sleep while impersonating the presence of applications running on them. In fact, the NCP concept is very useful in reducing network energy waste but now the smartphones which became an important part of our daily life can also play their role in achieving this objective. People are always equipped with an Internet connected smartphone regardless of work hours or outside work hours. Thus, why not propose an energy saving strategy in which one user device maintains presence for his other devices and so on. This approach can be easily realized in the current Internet standards and can also easily address the basic issues and challenges of the NCP concept.

The most important challenging task in NCP concept is the proxying of ever increasing number of applications on behalf of sleeping devices. The NCP maintains the applications presence by sending or responding to periodic application specific heartbeat messages. This requires the application source code to understand the nature of heartbeat messages which is only possible for open source applications [29]. Further, if the NCP is implemented at the local level, the developers need to have the understanding of different programming languages. There is currently no useful technique available that can be used for proxying of proprietary closed source applications such as skype, viber, MSN messenger, VoIP clients etc. However, the intelligent coordination strategy among devices can solve this issue by not impersonating applications but running them on any active user device e.g., smartphone, laptop, desktop PC, tablets etc. Thus, all the system needs is to keep track of all user devices, their priorities and operational status and run applications on only and only one device that is currently under active use. The second important challenge in the NCP concept is keeping/preserving TCP connections open on behalf of sleeping client devices and resume the same connection back when they wake up. Preserving the TCP connections is very important as they are associated with different applications. Currently this objective is achieved through complex techniques such as TCP connection splitting, TCP connection migrations, end-to-end TCP connection management etc [16, 18]. Each technique has its own pros and cons in terms of scalability and performance. However, in intelligent coordination among devices proposal there is no need to preserve open TCP connections. The third challenge in the NCP concept is keeping track of client devices for correct functioning. If the NCP client is a mobile device, it may move from one network to another network during the sleep mode. Complex techniques can be adopted to keep the NCP updated about its clients mobility [70]. However, the intelligent coordination strategy don’t require to keep track of its client devices positions. The client devices will periodically check with the system if they need to run applications or not. Fourth main issue in the NCP concept is its coverage which is limited only to local network [49, 72]. To make easy the proxying of proprietary closed source applications through collaboration with the application developers, it is important to take the NCP out of LAN boundaries and standardize it. The issues related to IP addresses collision will appear if the NCP covers for more than one local network (different local networks may use same NAT and private IP address space). However, this issue can be partially addressed by making the NCP NAT-aware and using a Universally Unique IDentifier (UUID) for each device which enables the NCP to distinguish it from others. However, still some issues will exists especially related to NAT and Firewall e.g., waking up a sleeping client device. This process becomes more and more complicated if the NCP clients are two, three or more levels behind the NAT (NAT behind NAT). Thus, two way communication between the NCP and its client devices will be a challenging issue due to NAT and Firewall. Luckily, the proposed system on intelligent coordination strategy don’t impersonate IP addresses but just keep track of all devices
and their operational status related to a particular user. Further, one way communication (always initiated from inside the local network) is enough in the proposed system and hence, there is no need to create any security hole in the local network.

Apparently, the intelligent coordination strategy among user devices looks quite appealing and is safe from some basic issues that arise in the NCP concept. It has also some drawbacks. First, this strategy requires the mobile device to be always connected to the Internet so that the applications presence move along with the user whenever necessary. However, most of the users of smartphones may not have 3G/4G data connection. Secondly, the expected energy savings will be always lower than the NCP concept which runs applications on a user device only in case of an update. However, applications will be always running on one of the user device in the intelligent coordination strategy resulting in reduced energy savings. Third, the intelligent coordination strategy cannot meet a user need for remote access to his computer in home or office. Thus, a very basic NCP running on a gateway device will be also required aiding the intelligent coordination approach to achieve this objective. This basic NCP will be only responsible to perform some very simple practically realizable networking tasks (PING, DHCP, ARP and WoC rules only) without worrying about proxying complex applications and protocols.

6.3.2 Devices & Applications Control Logics

The intelligent coordinating system autonomously adapts to the environment and control applications running on different user devices. Some possible control logics for executing applications on a specific user device are the following:

Based on Device Priority

The proposed system may control applications based on the device priority. E.g., a desktop computer has higher priority than laptop, laptop has higher priority than tablet and tablet has higher priority than smartphone. The device priority may be based on weather the device is battery powered or not. When the higher priority device switch ON, the proposed system should run applications on it and stop on all other low priority devices.

Based on Pre-specified Time Periods

The proposed system may control applications based on the pre-specified time periods for each device. E.g., a user can instruct the system to run applications on office computer from 8:00am till 6:00pm, on smartphone from 6:00pm till 8:00pm and on home computer from 8:00pm till 10:00pm and don’t run applications on any of his device from 10:00pm till 8:00am (during user sleeping time).

Mixed User Configurations

The proposed system may control applications on different user devices based on the mixed configurations. A user may want to run some of his applications on computer while some on smartphone based on the application importance at the moment e.g., never run facebook messenger on computer as browser is sufficient or never run skype on smartphone as he want not to be disturb outside work hours etc. For example, a user has 5 different applications A, B, C, D and E. He want his home computer to run all of them, office computer to run only applications A, B and E, and smartphone to run only applications C and D.

6.3.3 Possible Architectural Solutions

The proposed system can have two different possible architectural solutions based on weather the user devices communicate directly with one another or indirectly through a central coordinating unit: (i) centralized architecture and (ii) Ad-Hoc architecture.
Centralized Architecture

The centralized design uses a central coordinating unit with which all user devices communicate. This Central Application Coordinating Unit (CACU) is a global entity running anywhere in the world and manages user accounts and their registered devices. It guarantees to run applications on either a smartphone or tablet or laptop or desktop PC or any other device currently under active use. The user first needs to register all of his devices and applications with the CACU. Fig. 6.6 shows a generic CACU scenario for a single user having different devices. This scenario assumes that the user has three devices: one desktop computer at the office, one laptop at the home and one smartphone which is always with the user where ever he moves. All of these three devices communicate with the CACU which instructs them to run or not to run the applications. E.g., when the user is at the office, the CACU will inform the office computer to run all the applications (such as skype, viber, VoIP clients and other) while informing home laptop and smartphone to stop the applications. During this time, the home computer/laptop can sleep and can be woken-up by very simple NCP running on the HG for the remote access connection requests. Also the smartphone during this time will have lower utilization of memory, CPU, 3G/4G and other hardware resources due to stopping all the applications. The smartphone and home laptop may periodically check with the CACU if they need to resume all the applications. When the user leaves office, his office computer can enter into sleep mode and a very simple NCP in office network will manage its availability for remote access. Meanwhile, the CACU will inform smartphone to run all the applications. This way, applications will run on only one user device based on his location and the device being used at the moment.

![Diagram](image_url)

**Figure 6.6.** The generic CACU scenario. The user has three different devices: a laptop at home, a desktop PC in office and a smartphone. The CACU adopts to the user configurations and priorities and run applications on only and only one device at the moment.

Fig. 6.7 depicts the generic architectural solution. The CACU is a global entity to offer its services for the entire world. Thus, the CACU server may be managing millions of users around the world and each user can have multiple devices. Fig. 6.7 assumes a total of N users who created account at the CACU server. During account registration with the CACU server, the user can mention all of the devices that he will be using and provide configurations and priorities for the devices. The user in his account will also mention the applications that he will be using on his devices. Now based on the configurations and priorities the CACU server will run applications on only one user device at the moment. Fig. 6.7 also depicts different user devices that communicate with the CACU server. Each device runs an CACU client. The CACU client enables the device user to create an account with the CACU server or to add his device to an existing account. The CACU client also allows to specify or change account configurations at the CACU server. Finally, the CACU client includes an Application Controller who is responsible to run or stop applications.
on the device based on the instructions received from CACU server.

![Generic CACU Architecture Diagram]

**Figure 6.7.** The generic CACU architecture.

**Ad-Hoc Architecture**

In the Ad-Hoc scenario, the devices communicate directly with one another without relying on any central coordinating unit (i.e., without CACU unit). This design is more suitable for local networks that easily allows two-way communication among devices without NAT issues or compromising on Firewall. The user will have satisfactory concerns about the privacy and security of data as no third-party is involved in the scenario. Further, many communication paradigms (e.g., UPnP, multicast DNS) are available which allow devices to autonomously discover and seamlessly communicate with one another.

Fig. 6.8 shows the generic Ad-Hoc scenario for a single user having three different devices at both home and office. The communication among devices always take place inside the local networks while outside local networks, the smartphone is responsible to maintain the presence of applications over 3G/4G data connection. In the Ad-Hoc scenario, it is important for all devices to be present in the same network so to be able to communicate. The desktop computers and laptops always connect with the local network however, the smartphone needs to switch from 3G/4G data connection to WiFi whenever it enters in the local network. Fortunately, all modern smartphones have readily attained this requirement. Due to comparatively low power consumption of WiFi over 3G/4G, the smartphones automatically switches to WiFi network whenever an active WiFi access point is discovered. Thus, the process is automatic and seamless without requiring configuration changes or input from the user.

As obvious from Fig. 6.8, a device needs to keep track of all available user devices in the local network, their priorities or user specified configurations. Thus, the devices mutually decide which one has to run the applications. The smartphone is responsible to run applications only when no other active device is available over WiFi connection or when it is using 3G/4G connection. The desktop computers or laptops also need to keep track of the power state changes from the kernel and immediately informs the other available devices in the local network. Thus, once a high priority computer enters into sleep state, the other available devices again mutually decide about the next device to run the applications. Once again a very simple and basic NCP can be used to wake-up the desktop computer or laptop in case of remote access connection requests.

Fig. 6.9 depicts the generic architectural design. It is obvious that the system needs just one type of software that will run on all of the user devices. Each device needs to keep record of other available devices
and their priorities or user specified configurations. A communication protocol is required for each device to discover and communicate with one another. Finally, the Applications Controller is responsible to run or stop applications based on the mutual decision among devices.

### 6.3.4 Communication Interface

The proposed system also requires a communication interface that enables the devices to dynamically discover and communicate with one another. Thus, the devices can dynamically control one another based on the applications control logics. The choice of communication protocol can be based on weather the target scenario is centralized or Ad-Hoc. In centralized scenario, the communication interface should enable the devices to securely communicate with the CACU located anywhere in the world. In Ad-Hoc scenario, the communication can be more private and secure as it takes place inside the local network. However, the standard UPnP architecture (addressed in Chapter 4 & Chapter 5) can also be used for communication in this proposed system.

### 6.3.5 Concluding Remarks

The proposed system has different Pros and Cons compared to the original NCP concept (which is responsible to also proxy applications presence). Some of the Pros include:

1. This system does not proxy applications, thus will not face any issues related to proprietary closed source applications which exists in the original NCP concept.
2. This system has no applications privacy and security concerns. The original NCP concept requires important application secrets or privacy data in order to proxy the presence of a specific user account.

3. This system does not maintain open TCP connections on behalf of sleeping devices which is a challenging task for the NCP.

4. No need for device wake-up or its associated issues from remote locations which is important for the Internet-wide NCP concept.

5. No need to keep track of devices mobility and their changing IP addresses which is important in the NCP concept for incorporating mobility support for mobile devices.

Some Cons of the proposed system include:

1. In terms of reducing the entire network energy waste, this system benefits may be hardly equal to the NCP concept for fixed network devices. However, in terms of improving the battery life of smartphones, the benefits may be lower than the NCP concept.

2. The proposed system solves the application proxying issues which exist in the original NCP concept. However, it cannot address the wake-up of sleeping computers on remote access connection requests. Thus, a very light and simple basic NCP is required to achieve this objective in the proposed system.
Chapter 7

Software Framework & Implementations

This chapter presents in details the implementation efforts to practically realize the NCP prototype. Flexibility, portability and extensibility are the main objectives of the design and implementation; thus, the software framework is conceived as a number of separate blocks each dedicated to perform independently a specific task. At the core of the design there is a generic proxy application, which behaves like a platform to build several different services and provides a set of common functionalities to all of them. It mainly features an abstraction of devices and hides any details about their position, their capability and communication protocols. Around the main proxy platform, other components can be developed for building services and implementing communication protocols. The software framework is very flexible and could be extended to account for more energy-saving services other then the NCP. For example, it may integrate traffic aggregation features, splitting functionalities over several devices, managing mobility of covered devices and so on. However, the scope of this chapter is limited to the implementation of the NCP service only.

This chapter presents the most ambitious framework developed for managing low-power devices. It meets all basic requirements for NCP operation, even those on communication interfaces. This chapter particularly addresses a generic proxy software framework and development of one of its service, the NCP. Beyond, proxy software framework and the NCP service, it also presents the application for client devices. Briefly, this chapter is organized as follows: Section 7.1 presents in detail the proxy software framework and addresses its main functional blocks. Section 7.2 presents the implementation of NCP service in the basic proxy framework. Section 7.3 addresses software for the NCP client devices. Section 7.4 presents the implementation efforts for the UPnP based communication protocol. Finally, Section 7.5 lists the required tools and programming language used to realize the NCP framework.

7.1 Basic Proxy Architecture

The basic architecture of the proxy is shown in Fig. 7.1. It consists of five main components: Protocols, Dispatcher, Devices, TimeScheduler and Services. This structure allows flexible and continuous building of new functionalities into the same software architecture; this way, new protocols and new services can be added without major changes to the main architecture. A number of other components are required around these basic blocks and will be defined in the following.

The main task of the proxy is providing a set of services to client devices. There may be multiple Services running concurrently and providing a rich set of features. Remote devices need to exchange information with each service they wish to subscribe. To this purpose, a set of different Protocols can be used for communication between remote devices and local services. Thus, each Service is not statically bound
to any specific protocol and devices speaking different ‘languages’ can access a service at the same time. The Dispatcher and the local representation of Devices provide the logical abstraction needed to separate Protocols and Services. They manage an agnostic representation of data and information inside the proxy, not tailored to any specific Protocol. The Dispatcher delivers messages received from remote devices by the specific local Protocol to the Service in charge of handling it; messages use an internal representation instead of any protocol-specific format. Devices hold data about real devices and take care of sending messages from Services to remote devices.

The TimeScheduler component provides a generic framework to schedule unsolicited tasks at a given time instant. While the Dispatcher is mostly useful to trigger operations when some external event occurs (such as reception of a message), the TimeScheduler is needed when an event must be scheduled in advance or must run periodically. Fig. 7.1 also shows the main logical flow of information. Red arrows show the path for incoming messages, i.e., messages sent by remote devices to the Proxy. Blue arrows show the path for outgoing messages, i.e., messages sent by the Proxy to remote devices. The bidirectional blue arrow in the bottom right corner suggests some data about devices is stored locally, and thus they can be accessed directly without the need for sending messages. Such local data may be updated by different mechanisms (publish-subscribe, notification, polling).

Zooming out the level of details, the proxy architecture looks like in Fig. 7.2. Around the basic architecture described above, there are other important components to interface the proxy with the outside. On the right side of Fig. 7.2, there is the Configuration Database. It holds application-wide configuration settings, which are made available to all other components (Discovery, Protocols, Services) by means of the Proxy-Configuration. On the middle of Fig. 7.2, there is the ProxyManager. This object controls the whole proxy operation, providing the user with the ability to manage the proxy behavior in real-time. It gives access to current information about registered Devices, subscribed events in the Dispatcher, running modules (Protocols, Discoveries and Services). Different UserInterfaces are provided to access the ProxyManager both locally (i.e., by console) or over the network (both with TCP and UDP protocols), as shown in the left side of Fig. 7.2. Once started, the proxy sets up the basic components (e.g., the Logger, the ProxyManager) and runs the ProxyManager’s entry point, which is a loop waiting for user commands. Obviously, depending on the configuration settings, most of proxy’s modules can be launched automatically.

### 7.1.1 Communication with External Devices

The communication part of the proxy architecture is indeed made of two components: Discovery and Protocols. Discovery allows finding devices in an automatic way. There may be different discovery mechanisms: UPnP, SDP (Bluetooth), SLP, etc. Some of them will be tailored to local environments (LAN, Home Area...
Network (HAN), Wireless LAN (WLAN)), whilst others will scale to the whole Internet. *Discovery* is in charge of finding new devices on the connected links, and registering them into the system. The registration consists of the creation of a new *Device*, with protocol and addressing information to be used for successive communication with that device. The insertion of new devices automatically sends notifications to the whole system about them.

The *Discovery* should also announce proxy’s capability by means of the semantic of the *Protocols* it manages. The second component is the real *Protocol*. The main purpose of *Protocol* is the translation between system messages and protocol-specific syntax and semantics. The implementation of protocols allows communication with different types of devices on different media. Examples of protocols that may be used are UPnP and Hyper Text Transfer Protocol (HTTP).

### 7.1.2 Internal Communication

The *Dispatcher* is the main core of the application. It handles messages and delivers them to the proper service. The purpose of the *Dispatcher* is decoupling the communication protocol and the implementation of the proxy’s services. The *Dispatcher* may deal with message priority and inter-services issues (like multicasting the same information to different services that need it). Handling of message priority is not available in the current implementation. Fig. 7.3 shows the logic behind the *Dispatcher* concept. Basically, the function of the *Dispatcher* is running a set of given functions when some events occur. This requires prior registration of the functions and associated events (the pink table in Fig. 7.3); for example, in Fig. 7.3 two entities register their functions f(.) and g(.) for the same event E. Once the events occur, they are delivered and queued in the *Dispatcher* (the yellow queue); the *Dispatcher* processes them one-by-one by running the corresponding functions previously registered (f(.) and g(.) in the example of Fig. 7.3). The same event E is passed as argument to the functions, as it may embed information.

The subscription to a generic event returns an identifier, which should be locally stored by the caller and used for subsequent unsubscriptions. In the proxy architecture, the *Dispatcher* is used to delivers the messages received from remote devices to the local *Service* instances. Obviously, each *Service* must subscribe messages it is interested in.

### 7.1.3 Tasks Scheduler

In a complex software framework, tasks run in response to different triggers. The *Dispatcher* provides the basic infrastructure to respond to events, triggered by other software components or by external devices. However, many times tasks should be run at given time instant, even periodically. The *TimeScheduler* has
been designed to support this need in a flexible way for all other components and Services within the proxy framework.

The basic architecture for the TimeScheduler is shown in Fig. 7.4. It is internally organized as a sorted queue of waiting tasks, each with its own due time. Once the schedule time is reached, the task is executed by calling a predefined method bound to the same event structure. Two types of event can be scheduled: ‘one-shot’ and ‘periodic’. One-shot event are scheduled at an absolute time instant and are run only once; periodic tasks are scheduled at an absolute time instant too, but they automatically re-schedule themselves again once they have been executed. The execution time and the period are hence the two main parameters that define the behavior of a timed event.

To provide the wider flexibility of operation, the TimeScheduler allows both scheduling and withdrawing events, as well as checking is an even is already scheduled and counting its number of occurrences. This way, an activity could be scheduled at a future time instant (for example, renewing some registration or polling for data) and then, if there is no more need to carry out that activity, it will be withdrawn to avoid sideway effects. Withdrawing is also necessary to stop periodic tasks, which otherwise reschedule themselves indefinitely. To speed up operation, the TimeScheduler does not check whether the events it is scheduling were already scheduled previously; that means multiple occurrences of the same task may be scheduled at the same time. Scheduling an event may fails if that event was previously withdrawn: once an event has been withdrawn, it cannot be scheduled again later. Instead, a new event object must be created (if necessary, the validity of an event can be checked).

7.1.4 Abstraction of Devices

The Device structure is representative of physical devices that access proxy’s functions. Each Device is supposed to implement a number of different functionalities; this affects the operations the proxy can do on the device (mainly, which information is available and which requests/messages can be sent). Obviously, the functionalities of the device are independent of the services implemented in the proxy, but they could be fundamental for the realization of specific functions (for example, the ability to be woken up is necessary to use the NCP service). To keep the approach as general as possible, different kinds of functionalities in the proxy may be managed through different protocols; further, not all Devices are expected to have the
same set of functions. To allow a better separation among different logical functionalities, they have been separated into different DeviceModules, as shown in Fig. 7.5. Each DeviceModule consists of a number of state variables (if any). Such approach should simplify the development of code to manage specific functionalities, giving the developer a restricted view on data structures he really needs.

Specific DeviceModules are allocated in the device structure only when needed, namely for devices that implements those functions. Each DeviceModule includes its own communication Channel. State variables used within DeviceModules describe the evolution of a process. They are representative of the current device’s state and are continuously updated by suitable notification methods. Protocols are in charge of managing the continuous update of state variables. Depending on the specific communication semantic, they may subscribe for change with remote devices, or they may continuously poll remote device to check for any change. State variables allow external entities to subscribe for variable changes. Similarly to the approach followed in the Dispatcher, the subscription returns an identifier that can be used for successive unsubscriptions.

Figure 7.5. The Device structure.

7.1.5 Internal Messages

Messages are exchanged within the proxy and between the proxy and the devices. Messages carry information and data about specific services and features. Three main kinds of messages have been identified:

1. Notifications: these are unidirectional and unsolicited messages, i.e., they do not require a response. Notifications are mainly used to update the value of state variables, to inform about the occurrence of specific events, and so on.

2. Requests: these messages ask for some processing to the entity they are addressed to. As Notifications, they are unsolicited, but they require an acknowledgement from the recipient.

3. Responses: there are solicited messages, as they only come as an acknowledgement of the processing required by Requests.

Notifications are stand-alone messages, while Requests and Responses should always come in couple. The proxy, on the one hand, and the Protocols, on the other hand, are required to generate a Response after the processing of a Request. Starting from the above base types, each Service defines its own Message set.
7.1.6 Abstraction of Services

*Services* are the abstraction of generic functionalities provided by the proxy. Each *Service* provides a set of operations targeting a specific feature. For example, there may be a NCP service, a Basic Power Management Proxy service, a Home Automation Management service, and so on. However, we address here only the implementation of NCP service.

7.2 The NCP Service

Previous section presented a very flexible proxy software framework that can offer multiple services concurrently. However, the scope of this chapter is limited only to the implementation of NCP service. The NCP service allows hosts to move into low power states while maintaining their network presence. The *NCPService* is not a transparent proxy implementation, as it requires hosts to subscribe for the service. This choice enables the NCP proxy to lie anywhere in the network, without forcing it to be on the path of packets to/from local hosts.

The main components of the *NCPService* are actions and filters. Actions are tasks the NCP accomplishes on behalf of sleeping hosts; they include classification rules for matching packets that will trigger an action. Filters are classification engines that take current active actions and look for packets matching the pattern criteria; once a match is found, the corresponding action is invoked.

7.2.1 Basic NCP Service Architecture

The main data structures within the *NCPService* are *NCPActions* and *PacketFilters* as depicted in Fig. 7.6. When the *NCPService* is started, it registers with the *Dispatcher* for four kinds of messages (see the left side of Fig. 7.6): *NCPActionRequest*, *RemoveDeviceModule*, *RemoveDevice*, *UpdateVariable*. The first message is explicitly defined for the NCP service, while the others are generic within the proxy framework. *NCPActionRequests* should be sent by *Protocols* whenever a device requests a NCP service; these messages provide bi-directional communication and thus the NCP always answers with a *NCPActionResponse* including the result of the operation (success/failure) with an optional description and the identifier assigned to the *NCPAction*, which can be used for successive reference. *RemoveDeviceModule*, *RemoveDevice* and *UpdateVariable* are Notifications, thus no answer is sent by the *NCPService*.

Since, the basic task of the *NCPService* is to work on behalf of suspended devices. This implies the need to know the current power state of each device that has previously registered actions with the *NCPService*; such devices must therefore have a *PowerSavingModule*. When an *NCPActionRequest* from a device is received, the *NCPService* registers for notifications with the *PowerSavingModule* at the *Device*'s structure. Every time the power state of the device changes, the *NCPService* is notified (see right side of Fig. 7.6) and can activate/deactivate the *NCPActions* registered by that device. The activation implies loading all *NCPActions* registered by that device into the set of *PacketFilters*; the latter are then responsible to call the correct action whenever a match is found. Deactivation implies unloading all *NCPActions* from the set of *PacketFilters*. The NCP proxy takes the place of sleeping hosts by acting on their behalf; this requires traffic addressed to such hosts to be diverted towards the proxy. To this aim, the *NCPService* automatically adds a special action at the time the first *NCPAction* is registered by a device. The *ARPReplyAction* answers to local ARP requests for the IP address of the device and provides the local MAC address of the proxy. Thus, all subsequent network traffic addressed to the device will be delivered to the proxy. To force other devices’ ARP caches to update to the new situation, a Gratuitous ARP is sent when the device goes to sleep. When the proxy serves Internet hosts (i.e., not lying on the local network), another mechanism will be provided as an alternative to ARP to hijack packets addressed to sleeping devices.

At the bottom of Fig. 7.6, *PacketFilters* are shown. *PacketFilters* inspect packets traversing the proxy and look for matches against their internal patterns. The *PacketFilter* class is quite flexible and general.
to allow different approaches. Indeed, running PacketFilters implies running separate threads and this may lead too much overhead when a large number of devices are present. Briefly, a PacketFilter may be run to look for packets matching a single action, a set of actions or all actions. If one specific action is associated to each PacketFilter, once the PacketFilter has found a match for a packet, the rule to apply is known immediately, as there is a single NCPAction associated with that filter. On the other hand, if multiple NCPActions are associated with the same filter, once the filter reports a match, a look up must be carried out to find which NCPAction is associated to that packet. As already discussed, single-action filters involve more running threads, and their number increases linearly with both the number of devices and registered actions. In the current implementation, the choice was to create a PacketFilter for each type of NCPActions (ARP reply, PING reply, WoC). This should limit the number of threads; the search for the correct NCPAction must scan the whole list and must seek for matching every rule of the same type as the filter.

### 7.2.2 NCP Actions

Maintaining network presence implies a number of different actions, that can be dynamically requested to the service. Basically, three main kinds of activities can be identified:

1. **Network Presence**: the NCP service answers to network routine protocols on behalf of sleeping devices, just to let other entities believe the device is still present and functioning in the network. Such protocols include ARP, PING, DHCP.

2. **Heart-beating**: the NCP service generates protocol and application specific messages on behalf of sleeping devices to keep connections alive.

3. **Wake-up**: in many cases, the NCP cannot operate on behalf of sleeping devices; indeed, it is just required to awake devices when they receive specific network traffic, mainly identified by the transport protocol, the transport ports and/or the packet content.
7.2.3 NCP Filters

The PacketFilter interface defines an abstract entity that inspects packets traversing the proxy. PacketFilters run on multiple interfaces and look for packets matching rules that are implicitly defined by the NCPAction they serve. The PacketFilter interface allows starting and stopping the filter, updating the current rule set, adding and removing interfaces and getting information (filter name, filtering string). Implementation of the PacketFilter interface can exploit different technologies for packet filtering. Packet filtering concerns handling lots of packets and thus is expected to be one of the main issues affecting performance of the NCP service and, consequently, of the proxy infrastructure. Currently, a PcapPacketFilter implementation is provided for PacketFilter based on Pcap libraries. The PcapPacketFilter monitors multiple interfaces; it can run a different thread for each interface or a common thread for all of them. The current implementation tries to hold down the overhead by running the filtering thread only when there are active filters loaded. PcapPacketFilter is also responsible of calling the NCPAction corresponding to a matched packet (multiple NCPActions are associated to the same filter).

7.3 NCP Client’s Application

The NCP client’s application is responsible for providing an effective interface for all applications and services running on the host to register their proxying requests at the NCP. The client application also enables the NCP to get access to the LP service of the client UPnP device module. This will make possible for the NCP to know about the device power state and schedule the device sleeping during the specified periods.

![Figure 7.7. Basic NCP client architecture.](image-url)
The generic architecture of the application for the NCP clients is shown in Fig. 7.7. Its main components consist of HostManager, control point, controlled device, kernel module and the user interface. These components are briefly explained below.

1. **UserInterface**: It provides basic application control options for the user. It also provides interface for the user to register required actions at the NCP service.

2. **HostManager**: It contains the information about the device current status and allows the device to send/receive the actions requests. It is responsible for managing both, the device CP and CD.

3. **Control Point**: The CP enables the device to register actions at the NCP service. These actions define the conditions for the device wake up and also register basic network/application level presence requests for the device during sleeping period.

4. **Controlled Device**: It implements the LP service and contains information about the device power status. The service consists of several actions to manage the device power status. The control server handles basic service management and service/actions registration requests. The CD through eventing keep the NCP updated about the device state table.

5. **Kernel Module**: The kernel module is responsible to keep the device’s CD state table updated about the device current power status. It continuously checks if the OS kernel receives any kind of power state change request. It modifies the corresponding state variables value in the controlled device which immediately informs the NCP about the change.

Different software architectures were used to interface the kernel module with the OS. On Linux, kernel module runs in a separate thread, catches the power state change events received by the kernel and informs the client application through UDP packets. On Windows, the power state changes are detected by using the built-in device power state API provided by the OS.

### 7.4 UPnP Communication Module

The communication module provides flexible and reliable communication between the NCP and its client devices. We selected UPnP protocol as the well-suited approach for the design of NCP due to its easy, flexible, seamless and autonomous configuration and networking features. To implement a UPnP based communication module in the NCP framework, we formalized a reference template for the NCP service and wrapped it into an Internet gateway device which is already standardized by the UPnP forum. The basic internal structure of the UPnP-based communication module is shown in Fig. 7.8. This structure mostly focuses on the discovery, control and eventing part of the UPnP protocol. Both the client and the NCP implement this communication module. The two important components of the module are Event Handler and Event Generator. Both received and generated events related to the Event Handler and the Event Generator, respectively can be classified into two types: CP events and CD or simply device events. Some basic types of received and generated events related to CP and device are shown in Fig. 7.8.

When the UPnP-based communication module is initiated, the CP gets the list of available UPnP devices in the network through the discovery phase. The SubscriptionManager of the CP checks with the DeviceListManager if a specific device is already registered. The SubscriptionManager generates a subscription request event with the required service details if the device is not already registered. This event is received by the SubscriptionRequestHandler which evaluates the request and checks if the required service is implemented by the CD. In case of valid request, it will generate a subscription granted response event that is to be received by the SubscriptionManager. The SubscriptionManager instructs the DeviceListManager for adding the new device. The DeviceListManager through the UPnP description phase will download the device XML description file and status variables value. The DeviceListManager removes a device from
Figure 7.8. Basic structure of UPnP communication module.

The registered devices list when the SubscriptionManager receives a device ByeBye advertisement. The DeviceListManager is also responsible for monitoring the subscription period expiry of the device and contacts SubscriptionManager if the subscription renewal is required. Some main functionalities of the DeviceListManager are shown in the Fig. 7.8.

The ActionGenerator block is responsible for generating the actions to be invoked by the registered device. Each action contains the device and service details, action name and the state variables, if any. This action request event is received by the ActionHandler of the controlled device. The ActionHandler evaluates the request to check if it implements the required action within the specified service. In case of valid action request, the ActionHandler invokes the specific action. The ActionHandler also keeps the state table updated as the action may result in some changes of the state variables value. The StateManager is responsible to immediately inform the CP about the changes in the state variables value. The StateManager may also receive a request from the CP to return the value of a specific state variable. The SubscriptionRequestHandler block is also responsible to broadcast the device unsubscription request when the device leaves the network. The SubscriptionManager upon receiving unsubscription/bye-bye request instructs the DeviceListManager to remove the specific device from the registered devices list.

### 7.5 Implementation Language and Libraries

The NCP and its client device softwares were developed in linux operating system on a standard PC. However, the client device software is also capable to run on Microsoft Windows OS. C++ is the main programming language used with the support of the boost libraries. The NCP software also uses PCAP libraries in order to sniff the packets that are intended for the sleeping devices. The UPnP based communication
module was developed using open source portable UPnP libraries that is also compatible with UPnP device architecture specification version 1.0 and provide support for different operating systems including Linux, BSD, Solaris, embedded Linux and windows operating systems.
Chapter 8

Performance Evaluation

The chapter provides detailed evaluation of our developed NCP framework. We mainly performed experiments to test if the NCP concept works in various realistic scenarios and analyzed its impact on the network operations. We performed large number of experiments in the worst case scenarios to determine the critical issues and performance limiting factors.

Briefly, this chapter is organized as follows: Section 8.1 provides detailed evaluation of our UPnP-based communication framework. It mainly evaluates the communication overhead during different phases such as client registration/de-registration, action registration, power state notification etc. Section 8.2 analyzes the fact that devices consume more power during power state transitions and determine the minimum required standby period in order to achieve positive energy savings. Section 8.3 evaluates the NCP operation effectiveness in two different realistic scenarios: (i) when a sleeping device is accessed by another device located in the same network and (ii) when a sleeping device is accessed by another device located outside its network. Section 8.4 performs basic NCP evaluation particularly evaluating latencies during registration and operation of different NCP behavioral rule. Section 8.5 evaluates the NCP performance on different low power hardwares. Particularly, it presents analysis of latencies during rules registration, NCP memory requirements, latencies in activation and setup of filtering engines and performance of filtering engines under high computational and network traffic loads. Section 8.6 presents the design optimization to improve the performance for the Internet-wide NCP scenario. Here main efforts are done to improve NCP scalability and reduce latencies in setup of filtering engines. Finally, Section 8.7 estimates the energy savings achievable with the NCP concept.

8.1 UPnP Communication Overhead

This section analyzes the UPnP communication protocol in terms of amount of overhead it introduces to transmit a small amount of data. It mainly analyzes (i) the overhead as the number of packets being exchanged during different communication phases to transmit a piece of information and (ii) the overhead due to formatting of the real information before being transmitted. In the UPnP communication paradigm, messages are exchanged to advertise presence in the network, broadcast discovery messages at periodic intervals, download the device XML description file during registration with CP, periodic device subscription renewal messages etc. The knowledge of UPnP communication overhead may have less importance for local NCP scenarios (communication inside the local network) due to high speed links. Further, devices have enough processing power to quickly assemble and disassemble the real information in packets. However, communication overhead could be an important factor especially when it is over the low-bandwidth links which may reduces the speed of communication (e.g., in UPnP remote access scenario for Internet-wide NCP). The UPnP remote access already proposed techniques to transmit fewer possible packets between
two networks (over Internet) with the help of RAS entity; however, still it is important to know how much overhead is introduced due to formatting of real information in the packets before being transmitted over the Internet (which may have low band-width links).

To analyze the UPnP communication overhead, we performed an experiment that lasted 712 seconds (almost 12 minutes). In-depth analysis of packets exchanged during operation was carried out on the network dump taken by the Wireshark network sniffer. Several events occurred during this period: the NCP service and the client device announced themselves and discovered mutually, the client device registered with the NCP, it registered a WakeOnConnection action, it transitioned into sleep and woke up again and finally it de-registered from the NCP. The overhead in terms of number of packets and bytes exchanged is shown in Fig. 8.1 with different numbered events. This overhead includes all packets being exchanged due to the UPnP communication paradigm (including the underline protocol specific control and signaling packets such as TCP ACK). A brief description of the events marked on Fig. 8.1 is given below:

1. Start of the client application. During this event, both the CP and the CD of the client advertised their presence by broadcasting a certain number of Simple Service Discovery Protocol (SSDP) messages.

2. Start of the NCP application and client registration with it. The NCP CP and CD broadcasted their presence. Number of packets and bytes transferred are quite large as the control point at both ends downloaded the device description file (XML) from the URL provided during the SSDP discovery phase. Furthermore, for multiple received SSDP discovery messages, CPs on both ends downloaded device description file in order to know if the device is already registered or not.

3. Steady state condition. This phase only contains the UPnP heart beats or simply the advertisement/discovery messages. Both the NCP and client device continuously renew (before expiry period) their device subscriptions and subscriptions to any state variables they are interested in.

4. Action registration. The client registered a WakeOnConnection action at the NCP.

5. Client power state change notification to the NCP.

6. The sleeping period of the client device can be observed roughly between 300 to 380 seconds in Fig. 8.1.

7. The client device wake-up. The client device informed the NCP about the power state change and the NCP stopped covering for that device. Both the NCP and the client exchanged notifications of changes in state variables.

8. Client device de-registration with the NCP on stopping of the UPnP communication module at the NCP. This was the bye-bye message sent by the NCP to the client CP. The client CP received multiple bye-bye messages and each time it downloaded the XML description file of NCP device to check if the device was in the registered devices list or not.

9. Client bye-bye message broadcasted on the client application exit.

Table 8.1 shows rough statistical results about the network dump with the help of Wireshark. We can note that about 140 KB of data were exchanged for just the few events listed above. The breakdown of number of packets and bytes exchanged for different events are shown graphically in Fig. 8.2. It is clear that most of the packets were exchanged during the steady state condition while the average packet size was the smallest. This is mainly due to the fact that such packets are simply advertisement or discovery packets or renewal of the devices registration period which are quite small in size. The average packet size during the discovery and description phases is the largest. It is mainly because the CP at both ends downloaded the device description XML file to discover the device details. The description file contains all information about the device whose XML formatting can make the packet size quite bigger. The average packet size
Table 8.1. Statistical results of UPnP overhead from the network dump.

<table>
<thead>
<tr>
<th>Statistical Results</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Duration [minutes]</td>
<td>11:52</td>
</tr>
<tr>
<td>Total UPnP based packets transmitted</td>
<td>949</td>
</tr>
<tr>
<td>Average number of packets/sec</td>
<td>1.333</td>
</tr>
<tr>
<td>Average packet size [Bytes]</td>
<td>145.06</td>
</tr>
<tr>
<td>Total Bytes transmitted during 712 sec</td>
<td>131245</td>
</tr>
<tr>
<td>Average Bytes/second</td>
<td>184.33</td>
</tr>
</tbody>
</table>

during device un-subscription is also quite high because this phase once again involves the downloading of XML description file to check if the device is already registered. During the action registration and power state notification phases, a small piece of information is transmitted resulting in comparatively smaller average packet size.

Fig. 8.3(a) and Fig. 8.3(b) show the overhead as the percentage of packets and bytes exchanged during different events, respectively. Fig. 8.3(b) shows that 52% of bytes were exchanged during the steady state period whereas discovery and description phase contribute to 26%. Similarly, Fig. 8.3(a) shows that most of the packets were exchanged during steady state condition (68%) and the discovery phase contributed to 14%. It can also be observed that the overhead during action registration and power state notification was quite low compared to the steady state and the discovery phases due to the same reasons addressed before.

We also analyzed in our experiments the UPnP overhead in terms of real information or data, in terms of formatting the real information in the XML format, additional overhead due to headers and finally total
overhead due to all the packets exchanged in different events. The real information represents the actual size of data necessary to be carried in the packet’s payload such as device identification (e.g., UUID), IP and MAC addresses, device and service identifiers and description files, state table or variables etc. The real information in then formatted into XML format in the UPnP communication paradigm before being transmitted. Every event in UPnP communication paradigm includes a number of packets being exchanged e.g., TCP connection establishment, packets to transfer real data, acknowledgments, TCP connection termination etc. In Table 8.2, we once again break down the UPnP communication overhead into different phases: client device advertisements, client device registration, action registration, power state notification and client device de-registration. It can be observed that the overhead during the client device registration is quite high which is due to the fact that both devices download each others device and service description files including the complete list of state variables. XML formatting of such large information introduces significant overhead in terms of packet size. However, the overhead due to state variable change notification or simply the power state notification is quite small due to small piece of information formatted in XML before transmission.

Table 8.2. UPnP overhead in the communication paradigm for NCP software.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Device advertisements</td>
<td>310</td>
<td>1442</td>
<td>1575</td>
<td>2456</td>
</tr>
<tr>
<td>Client registration</td>
<td>717</td>
<td>4933</td>
<td>5241</td>
<td>8761</td>
</tr>
<tr>
<td>Action registration</td>
<td>133</td>
<td>1165</td>
<td>1304</td>
<td>1940</td>
</tr>
<tr>
<td>State variable update</td>
<td>71</td>
<td>478</td>
<td>540</td>
<td>1348</td>
</tr>
<tr>
<td>Client de-registration</td>
<td>522</td>
<td>2566</td>
<td>3149</td>
<td>4912</td>
</tr>
</tbody>
</table>
8.2 Power Consumption & State Transition Delays

Some previous studies point out that the time required to switch from active state to low power sleep state or to resume from sleep state to active state is not negligible; thus, low power sleeping states should only be used if the device is expected to have long inactivity periods [25]. Some studies also highlighted that the power consumed during states switching may raise up, thus throwing doubt about the potential energy wasting due to frequent power state changes. This section presents experiments to this aim and addresses the limitations due to frequent power state transitions on the usage of NCP.

To assess the potential pitfalls and limitation of the solution based on the NCP concept, we evaluated the effective power saving that can be achieved using the standby mechanisms present in the most part of modern PCs. We set up a simple experimental testbed and carried out some measurements considering two main aspects:

1. Time required to switch between low power sleep and active states.
2. The power consumed during the state transitions.

We carried out measurements on different devices to point out a preliminary evaluation of how much different hardware platforms could impact energy saving and the NCP effectiveness. Briefly, this section is organized as follows: Section 8.2.1 addresses the implications of delays in power state transitions. Section 8.2.2 addresses the implications of very frequent power state transitions and derive a mathematical bound for minimum required standby period to achieve positive energy savings. Section 8.2.3 investigates if the switching delays, minimum required standby period for positive savings and power consumption of a device during idle and standby states are different under different operating systems.

8.2.1 Power State Switching Delays

This subsection analyzes the implications of delays in waking up a sleeping device when some packets are received by the NCP which require client device’s resources. We performed several trials to measure the time required for state transitions. We observed that the devices running Microsoft Windows 7 took about 6 - 14 seconds to wake-up whilst, the same devices running Linux Ubuntu were a bit faster (4 - 10 seconds), but yet with the same large variance.

For more accurate measurements of the state transition times, we also need to consider other factors such as types of routines the CPU is executing, number of accesses to disk when the device is waking-up etc. Our preliminary observations suggested that some packets intended for sleeping device may be lost due to delay in waking up from sleep state. If the packets belong to new connections, the remote peer may receive device unavailable message due to packet loss. In general, loosing the first packet of a connection attempt may imply the failure of the connection; in this situation the waking up the sleeping device would be useless. Many applications have re-transmission mechanism and they attempt several times before failing; in this case, the sleeping device wake-up latency is affected by timeouts used among different attempts (i.e., time between two consecutive requests).

To overcome the limitations due to state transition delays, we incorporated the buffering feature in the NCP software. The buffering is used to cache incoming packets in order to give enough time for the sleeping devices to wake-up (the NCP receives wake-up acknowledgment from the client devices) and then transfers the cached packets to them without causing any packet loss. An an example, consider the SSH remote access application in Linux. It retransmits the request by doubling the timeout value at each attempt; i.e., the timeout at time \( t + 1 \) is computed as \( T_{t+1} = 2 \cdot T_t \), with \( T_0 = 3 \text{ sec} \); in the worst case, the request could be deferred up to few hundreds of seconds. For example, if the request comes at time \( t = 0 \), and the device needs 9.5 seconds to complete its wake-up, the request will only be correctly received by the destination after \( 3 + 6 + 12 = 21 \) seconds, against less than 10 seconds for request to receive with the buffering technique. In this example, buffering allows halving the latency of connection setup.
8.2.2 Minimal Required Standby Period to Achieve Savings

As stated before, frequent power state transitions may result in potential energy wastage due to power consumed during state switching. In this subsection, we performed experiment to analyze the implications of frequent power state transitions. Further, we derived a mathematical bound expressing the minimal required sleep period in order to achieve positive energy balance.

To analyze the implications of frequent power state transitions, it is necessary to observe the power drawn by devices during state switching. In order to simplify the electrical set up for power measurement, we used laptops, whose voltage and current can be easily measured between the power supply and the motherboard. Measuring power consumption of laptop is not a trivial task that can be accomplished by a simple wattmeter for AC supply, as it is known power supplies for such equipment skew the shape of the current. To take accurate values about energy consumption we thus explicitly measure both DC voltage and current after the power supplies of laptops. We used a breadboard with the ad-hoc circuit shown in Fig. 8.4; a DAQ (Data AcQuisition) allows to analyze and to store the correct values of voltage and current. Current absorption is calculated from the voltage on a CSR of 0.05Ω; a voltage divider was used to limit the voltage across the resistor under its threshold.

![Electrical circuit for measuring voltage and current feeding the device.](image)

We measured the voltage at the ingress of a notebook. It can be observed in Fig. 8.5(a) that the voltage values are almost constant and equal to the nominal value. The value of power measurements may be affected due to current drained by the device. It can be observed in Fig. 8.5(a) that the variation of the voltage value during state transitions is less than 1V against the nominal value of 19V (5-6%). The current drained by the device during idle and standby transitions is shown in Fig. 8.5(b). We can expect large spikes in the power consumption values during the state transitions. The power drawn by the device during state transitions is shown in Fig. 8.5(c). It can be observed that the power drawn during the sleep/standby state is much less than the idle state. However, it can also be observed that the power drawn by the device has large spikes during the state transitions (almost double the idle power consumption). If the device goes to standby state for very very short periods, it is possible that the overall power consumption might exceed the power consumption during idle state if the device don’t sleep at all. Thus, the devices should avoid going to sleep state when the idle period is very short because the power consumed during two transitions may exceed the savings from short sleep period.

In this subsection, we also derived a relationship between the switching times and power drawn in order to find the minimal idle period that can provide positive energy savings. To compute the minimum standby period required for positive energy savings, we considered a bound that sum of the power consumed during sleep period and during state transitions should be less than the power consumed if the device stays in idle state for this whole period (without going to sleep state).

\[
\int_{T_{t1}} P_{t1} + \int_{T_s} P_s + \int_{T_{t2}} P_{t2} < \int_{T_s} P_a
\]

where \(T_{t1}\) and \(T_{t2}\) are the switching times from idle to sleep state and vice versa, respectively. \(T_s\) is
time spent in sleep state while $T_a$ is the accumulated time required to go to sleep and come back to active state ($T_a = T_{t1} + T_s + T_{t2}$). $P_{t1}$ and $P_{t2}$ are the power drawn during state transition from idle to sleep state and vice versa, respectively. $P_s$ is the power drawn during sleep state while $P_a$ is the power drawn during idle state. As we observed before that the power consumption during the sleep and idle states are almost constant, thus we can re-write the above equation as follows:

$$\int_{T_{t1}}^{T_{t2}} P_{t1} + \bar{P}_s T_s + \int_{T_{t2}} \bar{P}_a T_a$$

where $\bar{P}_s$ and $\bar{P}_a$ are average values of the power drawn during sleep and idle states, respectively. Expanding $T_a$, and rewriting equation in more compact form using energy notations as follows:

$$E_{t1} + \bar{P}_s T_s + E_{t2} < \bar{P}_a (T_{t1} + T_s + T_{t2})$$

Now the final bound condition for minimum sleep time ($T_s$) to achieve possible energy savings can be written as follows:

$$T_s > \max \left\{ \frac{E_{t1} + E_{t2} - \bar{P}_a (T_{t1} + T_{t2})}{\bar{P}_a - \bar{P}_s}, 0 \right\}$$

Taking into account the data showed in Fig. 8.5(c), we can compute the length of the switching times ($T_{t1} = 7.5$ seconds and $T_{t2} = 4.6$ seconds), the energy wasted during those periods ($E_{t1} = 203.1$ J and $E_{t2} = 120.4$ J), the mean power consumption in standby ($\bar{P}_s = 0.78$ W) and in idle ($\bar{P}_a = 23.6$ W). With the above figures, the minimum time to spend in standby to have a positive energy savings is $T_s = 1.65$ seconds.

### 8.2.3 Comparison of Different OSs and Hardwares

This subsection briefly compares switching delays, power consumption and minimum required standby period for positive energy savings for several different hardware and software platforms. The main objective is to check if the switching times and power consumption change on a device for different operating systems. The choice of the different configurations was made according to the quick availability of devices in the lab; a more thorough investigation should include most widespread versions of OSs and representative samples of hardware platforms. The set of devices and OS configurations used are: a) Dell Inspiron 910 with Windows XP Professional, b) Dell Latitude D531 with Xubuntu 11.04, c) Dell Latitude D531 with Windows 7 Professional, d) Toshiba Satellite A205 with Ubuntu 11.10, e) Toshiba Satellite A205 with Windows 7 Home.

From a set of experiments using different configurations, we found that the power drawn during idle and state transitions and switching delays are rather different for different OSs. Power drawn in standby is the same, and this is easy to guess because in that state no operations from the OS are executed and the current drawn only depends upon the hardware. Our observations as meaningful set of numerical figures extracted
under different configurations are manipulated in Table 8.3. Although we considered a very simple and limited set of trials, we can see there may be large differences between hardware and software platforms. In particular, we can note the conflicting behavior for transient times on the Toshiba Satellite A205 laptop when different OSs are used (last two lines in Table 8.3). Further, it can also be observed that the minimum required standby period for positive energy savings is different for different devices and also depends on the operating system used.

Table 8.3. Switching time and performance analysis on different hardwares and operating systems. Times are measured in seconds \([s]\), power in watt \([W]\), and energy in joules \([J]\).

<table>
<thead>
<tr>
<th>Model</th>
<th>OS</th>
<th>(P_a)</th>
<th>(P_t)</th>
<th>(T_{t1})</th>
<th>(T_{t2})</th>
<th>(E_{t1})</th>
<th>(E_{t2})</th>
<th>(T_s)</th>
<th>(P_{t1})</th>
<th>(P_{t2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dell Inspiron 910</td>
<td>Windows XP</td>
<td>7.76</td>
<td>0.47</td>
<td>15.1</td>
<td>10.5</td>
<td>120.3</td>
<td>90.8</td>
<td>1.71</td>
<td>7.97</td>
<td>8.65</td>
</tr>
<tr>
<td>Dell Latitude D531</td>
<td>Xubuntu 11.04</td>
<td>23.17</td>
<td>1.22</td>
<td>5.5</td>
<td>5.0</td>
<td>103.7</td>
<td>102.3</td>
<td>0</td>
<td>18.85</td>
<td>20.46</td>
</tr>
<tr>
<td>Dell Latitude D531</td>
<td>Windows 7</td>
<td>17.79</td>
<td>1.22</td>
<td>12.0</td>
<td>5.5</td>
<td>151.3</td>
<td>128.6</td>
<td>0</td>
<td>12.61</td>
<td>23.40</td>
</tr>
<tr>
<td>Toshiba Satellite A205</td>
<td>Ubuntu 11.10</td>
<td>23.61</td>
<td>0.78</td>
<td>7.5</td>
<td>4.6</td>
<td>203.1</td>
<td>120.3</td>
<td>1.65</td>
<td>27.08</td>
<td>26.15</td>
</tr>
<tr>
<td>Toshiba Satellite A205</td>
<td>Windows 7</td>
<td>18.30</td>
<td>0.79</td>
<td>4.5</td>
<td>8.0</td>
<td>85.3</td>
<td>185.4</td>
<td>2.40</td>
<td>18.96</td>
<td>23.18</td>
</tr>
</tbody>
</table>

8.3 Performance in Realistic Scenarios

Since the NCP is expected to impersonate presence of sleeping devices, it is important to test its functionalities in different realistic scenarios. Deploying the NCP locally is representative of many use-cases: homes, offices and enterprises. Obviously, the more the number of sleeping devices, the more efficiency comes from having the NCP running on a stand-alone device or network equipment, thus we address our analysis to this direction. We explicitly considered two working scenarios to evaluate the NCP performance in terms of packet loss, latency, switching times etc. The scenarios are briefly described below.

8.3.1 Scenario 1: Sleeping Devices Presence for Local Devices

In this scenario, both the NCP client and a third party host lie in the same LAN and are connected to the same interface of a home gateway/router through a switch/hub. This scenario is depicted in Fig. 8.6; the NCP software is running on the gateway. The main purpose of this scenario is to study whether the NCP behaves correctly while both the sleeping device and its remote peer(s) are on the same side of the gateway and the communication path between them does not cross the NCP (thus, traffic diversion is needed during sleep). This situation is quite typical of office and enterprise environments, where many resources are hosted and shared within the organization and are accessed occasionally by internal users (file folders, printers, databases, etc.).

8.3.2 Scenario 2: Sleeping Devices Presence for Remote Devices

In this scenario, the NCP clients are in the LAN and the NCP runs on the home gateway/router, while third-party hosts are outside in the Internet. The communication between NCP clients and third-party hosts will always cross the NCP, thus traffic diversion is not needed in this case. However, a slow link could be present on the Internet side. In the testbed set up, the NCP client and the third-party host are connected to different Ethernet interfaces of the gateway, without a connection through the Internet. This scenario is depicted in Fig. 8.7. The main objective of this scenario is to check the NCP behavior with peers that are outside the LAN. This is the typical scenario when a user is trying to have remote access to his computer in office from home and vice versa.

In both scenarios, the interface between the NCP and its client devices is based on the UPnP communication framework: devices can register which actions the NCP must undertake while they are asleep by
a set of actions provided by the NCP service; power state transitions are notified through the UPnP notification mechanism for state variables.

Our demonstrated concluded the correct behavior of the NCP in both of the scenarios: the NCP answers ARP and PING requests, and it wakes up the device when a new connection request arrives to a given transport port. We also evaluated the performance in terms of packet losses and latency when the NCP takes over the connection for sleeping devices. The current implementation builds traffic diversion in LANs by broadcasting gratuitous ARP packets, which bind the sleeping client’s IP address to the NCP’s MAC when the NCP takes over the connection, and bind the sleeping client’s IP address back to the client’s MAC when that client wakes up. The ARP spoofing for a client takes a bit of time to complete and during this period some packets addressed to that device may be lost. The effect of the delay in taking over the connection was roughly evaluated by keep sending PING request messages to a sleeping device while it...
switches between sleep and active states and vice versa. We can see in Table 8.4 that no losses are observed in both wake-to-sleep transition (WTS) and sleep-to-wake transition (STW) in both scenarios. The losses are zero even increasing the PING frequency to 5 messages per second (higher frequency than this is not observed as it is usually considered as an attack by the OS). When the peer is in the LAN, packet diversion relies on updating the ARP caches of the peer and network equipment after receiving power state notification from client device. The diversion is activated in the shortest possible time span (before the client device actually enters into sleep state) by the NCP after receiving the power state notification, thus no losses are observed. In the scenario 2, all traffic passes through the NCP, which is always updated about the current power state of covered devices: it can immediately starts responding to PING requests as soon as the client transition to sleep mode without waiting for any ARP table update. Similar considerations hold for packet duplication. When the client device goes to sleep, it notifies the NCP before suspending, and when it wakes up it becomes operational before sending out the notification to the NCP, thus for a while both of them could answer to incoming packets. In scenario 1, ICMP packets are sent either to the client device or to the NCP, depending on the content of the ARP caches; in scenario 2 packets cross the NCP but are not dropped in the current implementation, and thus also arrives at their real destination. Packet duplication should not be a big matter; however, it could be easily solved by dropping packets at the NCP while the client device is asleep. Finally, no ICMP errors were seen during the PING (unreachable destination or no route to host could be expected if the NCP does not work correctly).

<table>
<thead>
<tr>
<th>Number of lost PING responses</th>
<th>Number of duplicated PING responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>WTS STW</td>
<td>WTS STW</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>0 0</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Most applications wait for a limited amount of time when they try to set up a connection; if they do not get a response, the connection is aborted. Sleeping devices should wake up and answer incoming requests before such timeouts occur. The latency for waking up a sleeping device includes the time to send a WOL packet and the time taken by the device to resume from the low-power state and to notify the NCP of its new condition. It was evaluated for the WoC rule, by attempting a connection at a transport port number that a suitable rule was previously registered for. Numerical results for the above measurements are shown in Table 8.5; figures shown are average values over 3 trials. The latency introduced by the NCP is measured as the time elapsed between the packet that triggers a WoC rule and the WOL packet sent to the device. The whole Wake-on-Connection procedure includes this latency and the time the device takes to switch to full power mode. From a user perspective, the most meaningful performance indicator is the application response time. It measures the time elapsed between the first packet sent by the application on the remote device and the first response sent back by the sleeping device to that application. It can be observed in Table 8.5 that the penalty due to NCP operation is negligible with respect to the time the hardware takes to change the power state. However, the need for power saving by standby states is expected to foster better performance from the hardware in the future, thus lowering the impact of the WoC operation.

### 8.4 Basic NCP Evaluation

One of the important performance factor for the NCP operations is achieving the goal of transparency with respect to third parties. Since, the NCP impersonates the presence of sleeping devices, hence the remote peers should not experience degradation in the quality of service or additional latencies during the NCP operations. In this section, we presented the performance of different NCP behavioral rules focusing
mainly on the resources requirements, registration latencies and the latencies experienced by third-parties during NCP operations.

8.4.1 Registration of Behavioral Rules

The client devices register different behavioral rules with the NCP over UPnP communication module. We observed the time required for the registration of each behavioral rule. Table 8.6 presents the values averaged over 10 trials for each behavioral rule. We split the results into NCP processing and NCP+UPnP processing latencies. The NCP processing latency indicates the processing power of the NCP software. However, the UPnP latency depends on the communication medium. Further, this latency may also depend on the type of communication protocol used (UPnP in our case). The communication protocol latency includes calling local internal API at the client devices, formatting of the messages, transmission over the communication medium to NCP and extraction of the messages at NCP. It can be observed in Table 8.6 that the values are quite stable for different types of behavioral rules. Further, it can be observed that the NCP processing latency is negligible and most of the latency comes from the communication protocol (however, it is also quite small, around 100 ms).

Table 8.6. Delays introduced during behavioral rules registrations. All figures are in milliseconds.

<table>
<thead>
<tr>
<th>Registration (UPnP+NCP processing)</th>
<th>NCP processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Average Max Std dev</td>
<td>Min Average Max Std dev</td>
</tr>
<tr>
<td>PING 105.914 106.190 106.488 0.180</td>
<td>0.912 0.955 1.031 0.043</td>
</tr>
<tr>
<td>DHCP 105.168 105.337 105.584 0.149</td>
<td>0.832 0.859 0.906 0.026</td>
</tr>
<tr>
<td>WoC 105.250 105.336 105.396 0.046</td>
<td>0.812 0.844 0.912 0.031</td>
</tr>
<tr>
<td>TCP-KA 102.665 104.924 105.642 1.193</td>
<td>0.580 0.797 0.888 0.117</td>
</tr>
<tr>
<td>HrtBt 102.740 105.189 105.862 0.921</td>
<td>0.833 0.905 1.274 0.131</td>
</tr>
<tr>
<td>All traffic 102.665 105.394 106.488 0.735</td>
<td>0.580 0.866 1.274 0.092</td>
</tr>
</tbody>
</table>

8.4.2 Evaluation of Behavioral Rules

We evaluated the performance of different behavioral rules offered by the NCP. We mainly analyzed the correctness of behavioral rules, NCP processing latencies and latencies experienced by end users.

Ping Rule

In Table 8.7, we performed analysis for PING rule by varying the PING request interval and the number of previously registered client devices at the NCP. We measured the Round Trip Time (RTT) to assess the latency introduced due to NCP operations and observed the packet loss under different conditions. The first two rows in Table 8.7 are reported just for the sake of observations when the client device is active.
and responding to PING request packets. The NCP starts responding to PING request packets as soon as the client device goes to sleep state. It can be observed that the RTT is comparatively larger during the client device sleeping state which is obviously the latency introduced by NCP for sniffing and processing PING request packets. However, the latency is quite small enough and does not cause any adverse effects or packet loss even by increasing the frequency of PING request packets.

Table 8.7. Performance of ICMP PING rule without and with the intervention of the NCP. Different trials were collected by changing the time interval of the PING request packets and the number of devices served by the NCP. Measures include statistics about the Round Trip Time and the packet loss ratio. Results for each trial were averaged over 100 samples.

<table>
<thead>
<tr>
<th>No. of Clients</th>
<th>ICMP Packet Interval</th>
<th>Round Trip Time (ms)</th>
<th>Packet Loss (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Average</td>
</tr>
<tr>
<td>Active Host (w/o NCP)</td>
<td>- 1 sec</td>
<td>0.243</td>
<td>0.380</td>
</tr>
<tr>
<td>Active Host (w/o NCP)</td>
<td>- 500 ms</td>
<td>0.142</td>
<td>0.215</td>
</tr>
<tr>
<td>Sleeping Host (w NCP)</td>
<td>1 1 sec</td>
<td>1.134</td>
<td>1.407</td>
</tr>
<tr>
<td>Sleeping Host (w NCP)</td>
<td>1 500 ms</td>
<td>1.171</td>
<td>1.388</td>
</tr>
<tr>
<td>Sleeping Host (w NCP)</td>
<td>1 200 ms</td>
<td>1.181</td>
<td>1.408</td>
</tr>
<tr>
<td>Sleeping Host (w NCP)</td>
<td>10 500 ms</td>
<td>1.151</td>
<td>1.399</td>
</tr>
<tr>
<td>Sleeping Host (w NCP)</td>
<td>10 200 ms</td>
<td>1.157</td>
<td>1.391</td>
</tr>
</tbody>
</table>

Wake-On-Connection Rule

In Table 8.8, we presented the performance for WoC rule in order to assess the latency experienced by users when they are trying to access a service on the sleeping device. Here we assumed that the user is trying remote access to the sleeping client device, thus it is important for the client device to wake-up. We performed tests for the SSH remote access application. Our analysis considers the responsiveness of the NCP to process the SSH request packets and the time the first packet from the target sleeping device gets back to the SSH client. In Table 8.8, it can be observed that the NCP latency is almost negligible (just few milliseconds), while most part of the latency comes from waking up the sleeping device and get back response from it (about 5 seconds). However, the latencies are still quite small and does not cause any misbehavior.

Table 8.8. Analysis of the Wake-On-Connection Rule. An SSH client was used to initiate new requests towards a sleeping SSH server (protocol TCP, port 22). The analysis considers the time taken by the NCP to send out the first WOL packet to the sleeping device and the latency to receive the answer to the TCP SYN packet by the SSH client. Several trials was performed, changing the number of devices registered at the NCP.

<table>
<thead>
<tr>
<th>No. of Clients</th>
<th>Statistics Over 10 Trials</th>
<th>Avg. No. of Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Average</td>
</tr>
<tr>
<td>WOL Request</td>
<td>1</td>
<td>1.549 ms</td>
</tr>
<tr>
<td>10</td>
<td>1.594 ms</td>
<td>1.885 ms</td>
</tr>
<tr>
<td>Answer to TCP SYN Request</td>
<td>1</td>
<td>3.67 s</td>
</tr>
<tr>
<td>10</td>
<td>3.23 s</td>
<td>5.13 s</td>
</tr>
</tbody>
</table>

Heartbeatting and TCP Migration

The NCP can manage both, solicited and unsolicited heartbeating on behalf of applications running on the sleeping client devices. Further, the NCP is capable to perform heartbeating over UDP as well as
TCP connections. For this rule, the latency is not a big concern as the heartbeat interval for most of the applications lie within the range of hundreds of seconds or few minutes. However, heartbeating over TCP requires seamless and transparent migration of TCP session between the NCP and client device. Thus, we measured the latency to freeze a given TCP session, transfer its state to the NCP over UPnP communication module and resume back there. This operation must complete before client device actually enters into sleep state. Table 8.9 reports the observed latencies. It can be observed that the latencies are quite small and the average values lie around 100 ms.

<table>
<thead>
<tr>
<th>Table 8.9. Analysis of TCP migration feature. The latency measured includes the time required to freeze a given TCP session and transfer the session to the NCP over the UPnP communication module.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min</strong></td>
</tr>
<tr>
<td>1 NCP client registered</td>
</tr>
<tr>
<td>10 NCP clients registered</td>
</tr>
</tbody>
</table>

8.4.3 CPU and Memory Requirements

One of the important performance limited factor for a software is the resources requirements such as CPU and memory. The available resources on the device hosting the NCP service decides the NCP scalability in terms of proxying presence for a number of sleeping devices. The scalability of NCP is indirectly linked with the achievable energy savings. Thus, we performed experiments to analyze the resources required for the NCP operations.

We performed evaluation under synthetic scenario by considering a number of virtual or fake client devices. We could not perform experiments on real devices due to unavailability of large number of client devices. However, the performance considering virtual client devices will be the same if the experiments were performed on real devices. Each virtual device has its own networking parameters (IP & MAC addresses) and sends request to the NCP for proxying presence on its behalf. Each virtual device registers a set of behavioral rules and notifies its power state transitions to the NCP. Indeed, the synthetic scenario is the best option to test the NCP performance under extreme or worst-case scenarios.

The NCP service is expected to run on devices with low power processors which are usually equipped with few megabytes of memory only (e.g., network switch/router). Thus, we performed the first evaluation on the memory required for the NCP operations when it registers large number of client devices (each with a set of behavioral rules). Fig. 8.8 depicts the memory used by the NCP software versus increasing number of client devices. We considered different set of behavioral rules registered by client devices. For simplicity, we considered five different sets of behavioral rules that could be registered by devices in the realistic scenarios. The basic sets of rules include: Network Presence (1 PING rule w/o DHCP), Network Presence (1 PING rule w DHCP), Reachability (1 PING, 1 DHCP & 1 WoC rule), Mid-Weighted Hosts (1 PING, 1 DHCP, 5 WoC, 5 TCP-KA & 5 Heartbeating rules) and High-Weighed Hosts (1 PING, 1 DHCP, 10 WoC, 10 TCP-KA & 10 Heartbeating rules). It can be observed that the memory usage increases with the increasing number of registered client devices. Further, the memory usage also depends on the number of rules registered by each client device (more memory required if large number of behavioral rules requested from the NCP). It should be noted that the registered client devices are all in the active state. We observed that CPU usage by the NCP software is almost negligible when it register client devices in active state.

We performed the second evaluation by considering the devices in the sleeping state and the NCP starts proxying their presence. We considered the ICMP PING rule, the simplest to be evaluated in our synthetic scenario. We observed the CPU and memory usage of the NCP software when it responds to the variable number of PING request packets on behalf of sleeping client devices. In our experiments, we considered 350 sleeping devices registered with the NCP which is then flooded with PING packets intended for the sleeping devices. In Fig. 8.9(a) & Fig. 8.9(b), we observed the memory and instantaneous CPU
requirements while slowly increasing the number of packets processed by the NCP. It can be observed that the memory requirements increases when the traffic load on the NCP increases. However, the increment is quite small and the NCP requires only 58 MB of memory at a pretty high traffic load for the local environment (1500 packets per second). It can be observed in Fig. 8.9(b) that the percentage instantaneous CPU usage of the NCP software also increases with respect to the traffic load. The instantaneous CPU usage reaches upto 23% under pretty high traffic load of 1500 packets per second.

8.4.4 Latency in Behavioral Rules Activation

When a client device enters into sleep state, the NCP immediately activates its set of registered behavioral rules and starts proxying its presence. This is a critical factor as any delay in activation of behavioral rules may result in packets loss or presence loss for the remote peers. The activation of rules mainly implies setup of filtering engines to sniff the packets intended for sleeping client devices which are then processed by the NCP. Our implementations are based on the well known packet capturing libraries called ‘PCAP libraries’ in order to sniff packets intended for sleeping client devices as well as for sending and receiving raw data. In Fig. 8.10, we measured the latency in activating behavioral rules when the client device transitions into sleep state. It can be observed that the TCP KeepAlive rule has the highest activation latency compared to the other behavioral rules. Indeed, this latency comes from inferring the TCP sequence numbers during
activation. However, in our synthetic scenario with large number of client devices, the active TCP sessions don’t really exist for each rule and the NCP gives up after the timeout of 500 ms. Thus, our evaluation in Fig. 8.10 represents the worst case scenario.

![Graph showing latency taken by the activation for each behavioral rule.](image)

Figure 8.10. Breakdown of latency taken by the activation for each behavioral rule.

8.5 Performance Analysis on Different Hardwares

We have evaluated the NCP performance on different low power hardwares. The main objective of such evaluations is to test the correct functioning of the NCP service in home or small office environments where the number of client devices (desktop PCs, laptops, etc) are quite limited. For maximum possible energy savings, the NCP service should offer high scalability while itself running on a very low power device. The low power devices are usually constrained in terms of available memory and CPU, thus it is quite challenging for the NCP to cover for very large number of client devices and behavioral rules. The NCP should be capable to support reasonable number of client devices where each device registers a set of behavioral rules. We have carried out our analysis on three different low power hardwares including Lantiq experimental home gateway, Raspberry Pi pocket PC and a very low power PC called Jetway. Fig. 8.11 shows the pictorial view of these hardwares along with their specifications including available memory, CPU and power consumptions in idle and full load conditions. The NCP deployment on Lantiq designates the NCP service being offered by the home gateway. While Jetway and Raspberry represent the standalone NCP deployment scenario with negligible power consumption.

Once again to evaluate the NCP performance on these low power hardwares, we considered the virtual or fake client devices which register a set of behavioral rules with the NCP. Each virtual client device has its own networking parameters (such as IP and MAC addresses) and notifies their power state transitions to the NCP. Thus, the performance of the synthetic scenario will be the same as for the real client devices.

8.5.1 Latency in Behavioral Rules Registration

It is important for the NCP to register behavioral rules from client devices as quickly as possible. The NCP is expected to cover for many client devices simultaneously and randomly receives the behavioral rules registration requests. In order to maintain transparencies with respect to third parties, the NCP should
have shortest possible latency in registration and activation of behavioral rules. The client devices uses a suitable communication protocol to register behavioral rules with the NCP. In our case the communication protocol is based on the UPnP architecture. Thus, we characterized the latency into NCP latency and UPnP or communication protocol latency. The NCP latency corresponds to the time when a behavioral rule registration request is received by the NCP to the time when it is successfully registered by the NCP. The NCP returns action ID and session ID after successful rule registration which can be used by client device for de-registration of that specific rule whenever required. Thus, the UPnP latency corresponds to the time required to return action identifiers (action ID, session ID) to the client device. The UPnP latency also depends on the processing power of the NCP hosting device in formatting (XML formatting), generating and transmitting UPnP messages and receiving back response from client device. Thus, the processing power of client device and the type of communication medium also contribute to the UPnP latency.

**Lantiq**

Fig. 8.12 depicts the latencies in behavioral rules registration for Lantiq experimental home gateway. We observed the NCP processing and NCP + UPnP processing latencies for each type of behavioral rule with respect to increasing number of previously registered client devices. The values in Fig. 8.12 were plotted for each new client device and its set of behavioral rules. It can be observed that the NCP latency in behavioral rules registration is quite small (average around 35 ms). The latency increases slightly with increase in the number of previously registered client devices, however, the increase is quite negligible. The same considerations also hold for the NCP + UPnP latency which particularly depends on the processing power of NCP hosting device and the type of communication medium. It can be observed from Fig. 8.12 that the UPnP latency alone is roughly equal to 20 ms.

**Raspberry**

Fig. 8.13 depicts the latencies in behavioral rules registration for Raspberry Pi pocket PC. We observed the NCP processing and NCP + UPnP processing latencies for each type of behavioral rule with respect to increasing number of previously registered client devices. It can be observed in Fig. 8.13 that the NCP latency in behavioral rules registration is quite small (average around 37 ms). The latency increases slightly with increase in the number of previously registered client devices, however, the increase is quite negligible. It can also be observed from Fig. 8.13 that the UPnP latency alone is roughly equal to 25 ms. Comparing with Fig. 8.12, Raspberry Pi latencies are roughly similar to Lantiq experimental home gateway for behavioral rules registration. Both are very low power devices with limited processing capabilities, thus
the values fluctuate over a wider range. The expected results should be quite stable with very low fluctuation if the NCP service is deployed on a power device with huge processing capabilities.

**Jetway**

Fig. 8.14 depicts the latencies in behavioral rules registration for a very low power Jetway PC. We observed the NCP processing and NCP + UPnP processing latencies for each type of behavioral rule with respect to increasing number of previously registered client devices. It can be observed in Fig. 8.14 that the NCP latency in behavioral rules registration is quite small (average around 9 ms). The latency increases slightly with increase in the number of previously registered client devices, however, the increase is quite negligible. It can also be observed from Fig. 8.14 that the UPnP latency alone is roughly equal to 5 ms. Comparing with Lantiq home gateway and Raspberry Pi pocket PC, Jetway has more processing capabilities. Thus, the latencies are much smaller than the Lantiq home gateway and Raspberry Pi pocket PC for behavioral rules registration. Due to more processing power, the latencies fluctuate over a very small margin (roughly 2 ms) on Jetway PC compared to Lantiq and Raspberry Pi (roughly 5 ms). Thus, it is proved that the latencies will be quite small and stable with very low fluctuation if the NCP service is deployed on a power device with huge processing capabilities.
8.5.2 Memory Requirements

As stated before, the memory available on the NCP service hosting device is the performance and scalability limiting factor. Whereas, the achievable savings from the NCP concept depend on the scalability (number of client device supported). Thus, we carried our experiments by deploying the NCP service on different types of low power hardwares and analyzed the memory requirements with different number of registered client devices and different set of registered behavioral rules.

Our NCP supports two different ways of setting up PCAP filtering engine. The first approach performs as much filtering as possible by specifying client device parameters in the PCAP string. As an example, the PCAP filter string consists of IP addresses of all client devices along with the indication to sniff only ICMP traffic. The filter string is based on ORing the information from all registered rules from all client devices. This approach performs well but as the filter string becomes long with the increasing number of registered client devices and behavioral rules, the latency in setting up Berkeley Packet Filter (BPF) increases and at a certain point the latency becomes unacceptable for the NCP operations. This is the major drawback which limits the NCP scalability. Another filtering method supported by our NCP framework is ‘Lightweight’ (Lw-PCAP) mode where the filtering string does not depend on the specific parameters of the behavioral rules but just a set of generic indications. For example, the filtering string for the PING rule will remain fixed. Thus, the Lw-PCAP filtering mode of NCP reduces much the latency in activating the filter once the client device enters into sleep state. However, large number of packets will be sniffed with Lw-PCAP where many packets does not belong to covered NCP client devices. Thus, it increases load on the NCP software in discarding and selecting/responding to right packets.

For the analysis of memory requirements, we considered different set of behavioral rules registered by client devices. For simplicity, we considered four different sets of behavioral rules that could be registered by client devices in the realistic scenarios. The basic sets of rules include: Network Presence (1 PING rule w/o DHCP), Network Presence (1 PING rule w DHCP), Reachability (1 PING, 1 DHCP & 1 WoC rule) and Each Rule Once (1 PING, 1 DHCP, 1 WoC, 1 TCP-KA & 1 Heartbeating rules). We performed the experiments on all three different types of hardwares under consideration and analyzed the memory usage by the NCP software under both, using standard PCAP filtering and using LwPCAP filtering.

Lantiq

The most suitable place for the NCP service is the home gateway which is an always powered-up device and will not cause any incremental power consumption. We deployment the NCP service on the Lantiq experimental home gateway and performed the memory analysis which is a critical factor limiting the
NCP scalability. Fig. 8.15 depicts the memory usage of the NCP service on Lantiq home gateway with increasing number of registered client devices where each client device registers a set of behavioral rules. We performed the analysis for both, the client devices in active state and sleeping state using the PCAP and LwPCAP filtering engines. It can be observed that the memory usage increases with the increasing number of registered client devices. Further, the memory usage also depends on the number of rules registered by each client device (more memory required if large number of behavioral rules requested from the NCP). It can also be observed in Fig. 8.15 that client devices in active state requires less memory than client devices in sleeping state (when PCAP & LwPCAP filtering is activated). Further, the increase in the memory usage is very small when the client devices are in active state. For different configurations in Fig. 8.15, it can be observed that the LwPCAP filtering engine has more memory usage than the standard PCAP filtering engine. However, the increase in memory usage for LwPCAP is very small compared to standard PCAP filtering when the client devices are in sleeping state.

Figure 8.15. Memory usage vs increasing number of registered NCP client devices on Lantiq experimental home gateway.

Raspberry

Although, Raspberry Pi is a very small low power PC but can provide more scalability than Lantiq home gateway due to more available memory. Fig. 8.16 depicts the memory usage of the NCP service on Raspberry Pi pocket PC with increasing number of registered client devices where each client device registers a set of behavioral rules. It can be observed that the memory usage increases with the increasing number of registered client devices. Further, the memory usage also depends on the number of rules registered by each client device (more memory required if large number of behavioral rules requested from the NCP).
Likewise Lantiq home gateway, it can also be observed in Fig. 8.16 that client devices in active state requires less memory than client devices in sleeping state (when PCAP & LwPCAP filtering is activated). The memory observations are somehow similar to the Lantiq home gateway. The increase in memory usage for LwPCAP is very small compared to standard PCAP filtering when the client devices are in sleeping state. However, unlike Lantiq home gateway, it can be observed in Fig. 8.16 that the LwPCAP filtering engine has less memory usage than the standard PCAP filtering engine.

Jetway

Jetway low power PC has much more available memory than Raspberry Pi pocket PC and Lantiq home gateway, thus is capable to provide very high scalability. Fig. 8.17 depicts the memory usage of the NCP service on Jetway low power PC with increasing number of registered client devices where each client device registers a set of behavioral rules. It can be observed that the memory usage increases with the increasing number of registered client devices. Further, the memory usage also depends on the number of rules registered by each client device (more memory required if large number of behavioral rules requested from the NCP). Likewise Lantiq home gateway & Raspberry Pi, it can also be observed in Fig. 8.17 that client devices in active state requires less memory than client devices in sleeping state (when PCAP & LwPCAP filtering is activated). Thus, the memory observations are somehow similar to the Lantiq home gateway and Raspberry Pi. The increase in memory usage for LwPCAP is very small compared to standard PCAP filtering when the client devices are in sleeping state. However, unlike Lantiq home gateway, it can be observed in Fig. 8.17 that the LwPCAP filtering engine has less memory usage than the standard PCAP filtering engine.
8.5.3 NCP Processing Latencies

The processing power of the device hosting the NCP service can impact the whole NCP operations. This also impact the latency experienced by remote users when NCP responds to packets on behalf of sleeping client devices. More the processing power of NCP hosting device, lesser will be the latencies to get back response from NCP. We expect that the NCP processing latencies may also depend on the number of client devices registered. Thus, we performed the analysis for PING and WoC rules on different hardwares with increasing number of registered client devices.

We measured the latencies for PING messages by using the well-know PING application. It can be observed in Fig. 8.18 that the latencies increase slightly with the number of proxied client devices. The values in Fig. 8.18 are averaged over 1000 packets. The latencies are however, one order of magnitude larger than the normal values in the LAN (few hundreds milliseconds). The reason to this is ascribable to PCAP libraries which return packets after a short timeout.

We also measured the latencies for WoC rule in Fig. 8.19 with increasing number of registered client devices. Here latencies indicate the NCP processing time in sending WOL packet after a new incoming connection to sleeping client device is seen. For this purposed, we registered a WoC rule for a given transport protocol and port number (we conducted experiments using SSH) and then we attempt to establish connection on that port. Fig. 8.19 represents the latencies averaged over 100 trials. Once again, we observed a slight increase in the latencies with increasing number of registered client devices. For WoC rule, the latency experienced by users is the sum of NCP processing latency in sending WOL packet (Fig. 8.19) and time required for the sleeping device to wake-up. The device wake-up latencies depend on the device type and OS and is irrelevant with the NCP processing power. For most of the latest PCs, the wake-up latencies
are around 3-5 seconds. The wake-up latencies are not negligible but can be tolerated by users without causing any abnormal behavior on network operations.

8.5.4 Latency in Device Registration & Setup of Filtering Engine

The NCP should start covering for its client devices as soon as they enter into sleep state. This is very critical factor especially when large number of client devices and behavioral rules are registered which can delay the NCP operations. Fig. 8.20 represents the time taken by the NCP to start covering on behalf of sleeping clients versus increasing number of previously registered client devices. For test purpose, each client device registers 1 PING, 1 WoC, 1 TCP KeepAlive, 1 Heartbeating rule. Here two different latency factors are involved: (i) time to activate behavioral rules as shown in Fig. 8.20(a) (i.e., performing any
preliminary operations depending on the rule type) and (ii) latency in setting up filtering engine as shown in Fig. 8.20(b) (i.e., setting up PCAP filters to sniff packets intended for sleeping client devices).

![Graph](image)

(a) Activation of all behavioral rules

(b) Latency Setting up filtering engine

Figure 8.20. Latency when NCP start proxying for a new client device versus increasing number of registered client devices. Every client device registers same set of behavioral rules (1 Ping, 1 WoC, 1 TCP-KA and 1 Heartbeating).

It can be observed in Fig. 8.20 that the latency in setting up the filtering engine is the most critical and increases to several seconds when the number of registered sleeping client devices increases. However, it also depends on the processing power of the NCP hosting device (small latencies are observed on devices with high processing power). It is important to note that the latencies are strictly related to the PCAP libraries which are not conceived for handling very long filtering strings (the length of filtering string increases with increase in the number of registered client device and behavioral rules). Also, standard PCAP filtering is not suitable for frequent changes in the filter parameters (changes every time a client device suspends or wakes up). Thus, we evaluated the latencies in setting up filtering engine (critical factor) on different hardwares for both PCAP and LwPCAP filtering methods in the following.

**Lantiq**

Once again we considered the same sets of behavioral rules to measure the latencies in client devices registration and setup of filtering engines. The sets of behavioral rules under consideration include: Network Presence (1 PING rule w/o DHCP), Network Presence (1 PING rule w DHCP), Reachability (1 PING, 1 DHCP & 1 WoC rule) and Each Rule Once (1 PING, 1 DHCP, 1 WoC, 1 TCP-KA & 1 Heartbeating rules). Fig. 8.21 represents the latencies in client devices registration and setup of filtering engines. The latencies in Fig. 8.21 only include the registration latencies when the client devices are awake. However, the latencies also accumulate the time required to setup filtering engines for sleeping client devices. Once again, we performed experiments under both, using standard PCAP filtering and using LwPCAP filtering.

Fig. 8.21 depicts latencies on the Lantiq home gateway, the most suitable place for the NCP service in home/small office environment. It can be observed from Fig. 8.21 that the latencies increase with the increase in the number of previously registered client devices. Further, the latencies also depend on the number of rules registered by each client device (large latencies observed with large number of behavioral rules requested from the NCP). It can also be observed from Fig. 8.21 that the registration latencies for awake clients are much lower than the latencies in setting up filtering engines for sleeping clients (when PCAP & LwPCAP filtering is activated). However, for different configurations in Fig. 8.21, LwPCAP has much lower latencies in activating filtering engines on behalf of sleeping client devices than the standard PCAP filtering.
Since the Raspberry Pi pocket PC is also a low processing device, we expect higher latencies values. Fig. 8.22 depicts latencies on the Raspberry Pi for both, awake clients and sleeping clients under PCAP and LwPCAP filtering methods. The performance observations are similar to the Lantiq home gateway. The latencies increases with the number of previously registered client devices and number of behavioral rules/client device (large latencies observed with large number of behavioral rules requested from the NCP). Likewise, Lantiq home gateway, the LwPCAP has much lower observed latencies than the standard PCAP filtering method.

Jetway

The Jetway low power PC has higher processing power than the Lantiq home gateway and Raspberry Pi, thus we expect much lower latencies. Fig. 8.23 depicts latencies on the Jetway PC for both, awake clients and sleeping clients under PCAP and LwPCAP filtering methods. The performance observations are similar to the Lantiq home gateway and Raspberry Pi. The latencies increases with the number of previously registered client devices and number of behavioral rules/client device (large latencies observed with large number of behavioral rules requested from the NCP). Likewise, Lantiq home gateway and Raspberry Pi, the LwPCAP has much lower observed latencies than the standard PCAP filtering method. However, it is proved from Fig. 8.23 that the latencies in activation of filtering engines depend on the processing power of the device hosting the NCP service. The Jetway PC has very low latencies due to more processing power.
than Lantiq and Raspberry Pi.

### 8.5.5 Performance of Filtering Engines under High Network Traffic

As explained before, our NCP software supports two different types of filtering methods, PCAP and LwPCAP. The main limitation of PCAP is the long latency in activation of filtering engine when a large number of client devices and behavioral rules are registered. However, LwPCAP has low latency but introduces high load on the NCP to distinguish the packets intended for covered client devices. The performance of the filtering engines ultimately decides about the effectiveness and scalability of the NCP.

We performed evaluation of the filtering engines under worst case scenarios by introducing heavy noise traffic load to check its impact on the NCP capabilities. For this purpose, we flooded the NCP with very high noise traffic with the aim to stress the filtering engine. We performed experiments on the PING rule under high network traffic. We measured the time taken by the NCP to generate response to PING requests on behalf of sleeping client device while it is flooded with the high ICMP PING or UDP noise traffic addressed for other devices. With ICMP noise traffic, we are stressing the same filter for which we are performing measurements whereas we are stressing other filters with UDP noise traffic. We performed measurements for both basic PCAP and LwPCAP under various sizes of the buffers. Small buffer size may get filled under high traffic and results in packet loss. Large buffer size requires more memory and increases processing latencies but will be less prone to errors.
Packet Loss

Fig. 8.24 and Fig. 8.25 show the packet loss measured on 100 PING request packets intended to the sleeping client device under varying ICMP and UDP noise traffic, respectively. It can be observed that with the increase in noise traffic, the percentage packet loss also increases. It can be observed that the Lw-PCAP mode performs better than basic PCAP mode just on the Lantiq home gateway with low packet loss under high noise traffic. However, packet loss is higher for LwPCAP mode other hardwares (Raspberry and Jetway). However, the packet loss also depends on the device processing capabilities. The Raspberry Pi is showing better performance than the Lantiq home gateway. To check its behavior under critical conditions we changed the noise scale from 500-5000 packets per second (pps) to 1000-10000 pps. Unlike for Lantiq board, this time the basic PCAP filtering behaves better than Lw-PCAP filtering; starts loosing packets since 3000 pps which is a quite heavy load for this kind of device. The Lw-PCAP mode losses more packets as this mode does not exploit the optimization of BPF. The Jetway is equipped with an Atom processor which has more power than the ARM processor of the Raspberry Pi and the Lantiq boards. To stress the NCP software, we further increased the noise load to the range 10000-80000 pps. Thus, the best performance can be observed for Jetway. The Jetway shows similar performance observations as for Raspberry Pi; the Lw-PCAP mode has more packet losses as it must handle all noise traffic by itself. For Lw-PCAP mode, the packet losses can be greatly reduced by properly setting the PCAP buffer size. Different curves (in Fig. 8.24 and Fig. 8.25) refer to different buffer sizes for PCAP operation, in units of 65536 bytes. The packet losses also depend on the size of the PCAP buffer. The smaller size buffers fill in before the previous capture packets are processed and results in more losses.
Figure 8.24. Packet loss measured over 100 ICMP PING request packets intended for sleeping client devices with ICMP PING noise traffic.

Figure 8.25. Packet loss measured over 100 ICMP PING request packets intended for sleeping client devices with UDP noise traffic.

Filtering Engine Latency

Fig. 8.26 and Fig. 8.27 show the latency measured on 100 PING request packets intended to the sleeping client device under varying ICMP and UDP noise traffic, respectively. Similar to packet loss, it can be observed that with the increase in noise traffic, the filter latency also increases. The Lantiq board is showing higher latency compared to Raspberry Pi and Jetway as it has the small processing power and memory. The Lw-PCAP causes more packet losses on Lantiq home gateway, however can offer very low latencies; NCP with Lw-PCAP mode can answer packets with latencies that are very close to those without using the NCP. However, the performance of Lw-PCAP for Raspberry Pi and Jetway is once again different than Lantiq board; the latencies with Lw-PCAP mode are higher than standard PCAP filtering.

Figure 8.26. Latency measured and averaged over 100 ICMP PING request packets intended for the sleeping client devices with ICMP PING noise traffic.
Figure 8.27. Latency measured and averaged over 100 ICMP PING request packets intended for the sleeping client devices with UDP noise traffic.

CPU & Memory Requirements

Fig. 8.28 depicts the memory usage by NCP software with different sizes of PCAP buffer. It can be observed that the memory utilization is linearly proportional with the size of PCAP buffer (perhaps Lantiq may have different memory allocation strategy in its kernel). Fig. 8.29 shows the CPU utilization when the device is running with no NCP or with different configurations of NCP. Once again, the percentage CPU utilization is higher for devices with low processing capabilities. It can be observed that for Jetway, the CPU load can raise over 100% as the processor has four cores (maximum utilization can reach up to 400% as the sum from all four cores). Further, it can be observed that the CPU load is very high with Lw-PCAP mode due to less optimization in filtering. This is the main reason for high loss percentage and large latencies.

Figure 8.28. Virtual Memory Size reserved for NCP versus different sizes of PCAP buffer.

Figure 8.29. Percentage CPU usage versus increasing traffic load on NCP. ‘No NCP’ means that the NCP service is not running and the client device is awake and receiving packets. The other two curves represent percentage CPU usage when the NCP is running basic and lightweight PCAP.
8.5.6 High Computational Load

We performed the NCP evaluation under high computational load to analyze the performance constraints due to processing power of the device hosting the NCP service. For this purpose, we made selection of the behavioral rule that requires more processing. PING messages are basically used for debugging purpose and occasionally sent. WoC is also expected to be less frequent. DHCP lease expiration times are usually set quite large by the DHCP server. The TCP keep-alives are trigger by the remote peers and the default interval is usually 7200 seconds. However, the load due to application specific heartbeat messages is most difficult to estimate. Most of the applications have heartbeat interval of few minutes or even less. Thus, the NCP heartbeating rule expects far higher traffic load than the other behavioral rules.

In our experiments, we considered unsolicited heartbeating in order to also assess our scheduler functionality. Depending on the processing power of the device hosting the NCP service, we registered specific number of the client devices. Each client device registers a PING rule and 10 heartbeating rules each with the period of 1 second (very unrealistic interval but chosen small enough to generate high load). The inter-arrival times between the consecutive heartbeat messages should ideally be equal to 1 second (Reference Interval), however we observed higher values the NCP hosting device has not enough processing power to manage heartbeating for client devices and behavioral rules. Fig. 8.30 presents the mean inter-arrival times averaged over 300 measurements.

![Figure 8.30. Inter-arrival times. Heartbeating messages were scheduled once for each second (reference interval), and 10 Heartbeating rules were registered for each client device.](image)

Fig. 8.30 shows the Lantiq board can serve up to 90 devices (with 10 1-seconds Heartbeating each) without any delay; with 100 devices the CPU comes close to saturation (see Fig. 8.31) and the scheduled Heartbeating cannot be served in time. Also note the generation becomes more unstable when the limit is reached, as demonstrated by the larger variance. The same holds for the Raspberry Pi, which has the same kind of processor. Instead, the higher processing power of the Jetway device allows to cover for 600 devices before any significant delay is experienced; this time as well, delays occur when the CPU utilization reaches its limit.

![Figure 8.31. CPU usage. Heartbeating messages were scheduled once for each second (reference interval), and 10 Heartbeating rules were registered for each device.](image)
8.5.7 Interference Among Behavioral Rules

The main objective of this analysis is to check if heavy load for one behavioral rule impact the behavioral for other behavioral rules. For this purpose, we generated heavy load for the heartbeating rule and measured its impact on the ARP rule. Once again, we performed analysis under variable number of previously registered client devices on different hardwares where each client device registers 10 heartbeating rules with the interval of 1 second. Fig. 8.32 depicts the mean inter-arrival time and standard deviation for the ARP responses when the ARP requests were generated at the rate of 20 packets/second (50 ms gap between packets).

In our experiments, we observed that the standard deviation becomes larger with the number of previously registered client devices. However, the mean value remains the same as the rate at which the ARP request packets were generated. Thus, the high load of heartbeating rule marginally affected the behavior of the ARP rule. This is because the processing of each type of behavioral rule is done in a different thread. In short, this is a great feature that overwhelming of one behavioral rule does not lead to affect performance of whole system.

![Graphs](a) Lantiq Home Gateway  
(b) Raspberry Pi  
(c) Jetway

Figure 8.32. Inter-arrival times for the ARP responses. ARP requests were generated in parallel to heavy load Heartbeating (messages were scheduled once for each second, 10 Heartbeating rules for each client device) at frequency of 20 packets/second.

8.6 Considerations for Internet-wide NCP

The Internet-wide NCP is expected to cover for very large number of client devices. Thus, it is necessary to overcome the limitation of local NCP experienced before. The local NCP covers for less number of devices due to which it is deployed on a low power network entity with limited available resources such as memory and CPU. However, due to very large number of client devices, the Internet-wide NCP should be deployed on a very powerful device with huge memory and processing power. The power consumption of such device will not be a concern as it will spread over very large number of devices globally causing negligible incremental power consumption. It is important to note that most of the limitations of local NCP came from the limited available resources on low power devices. These limitations will automatically overcome in case of Internet-wide NCP being deployed on a power device or using some kind of distributed processing mechanism.

8.6.1 Performance Optimization

The operations of NCP is highly based on the PCAP libraries. As we noticed the limitations of basic PCAP filtering which is not originally designed for long filtering strings and/or frequent changes in the filtering string. Thus, one obvious optimization is the LwPCAP which offers very low latency even when covering for very large number of client devices and behavioral rules.
With basic PCAP, the latency to activate behavioral rules for a sleeping client device increases exponentially with the number of previously covered client devices. Fig. 8.33 depicts the latencies in activating behavioral rules and setting up filtering engine for two different devices: one Raspberry Pi pocket PC (ARMv6 700 MHz and 512 MB RAM) and other desktop PC (Apple iMAC 9.1 equipped with Core 2 Duo CPU 2.93 GHz and 4 GB RAM). It can be observed that the time to activate behavioral rules is not a critical concern as it is very low (less than a second). The main performance concern is the time required to setup the filtering engine for a new sleeping device when already covering for very large number of client devices. However, the filtering engine setup latency gets even more worse when a more meaningful set of behavioral rules are considered (as in Fig. 8.34).

![Latency in activation of behavioral rules](image)

**Figure 8.33.** Latency in activation of behavioral rules. Each client device registers: 1 PING, 1 WoC, 1 TCP-KA and 1 Heartbeating.

![Latency in activation of behavioral rules](image)

**Figure 8.34.** Latency in activation of behavioral rules. Each client device registers: 1 PING, 10 WoC, 10 TCP-KA and 5 Heartbeating.

In order to optimize the performance of NCP to support high scalability in the Internet scenario, we reconsidered some changes in our software design. Initially, PCAP libraries seemed to be the optimal and effective tool for packet filtering. However, the latencies in setting up filtering engine becomes unacceptable when large number of client devices and behavioral rules are registered. Thus, PCAP should only be used for general purpose filtering without specifying behavioral rules parameters (the LwPCAP filtering method explain before). The raw packets which are sent out from the normal networking stack of the OS should not rely on PCAP libraries as the time to set them up reaches to several seconds when large number of client devices and behavioral rules are covered. Thus, we switched to OS-specific raw sockets which results in
loosing code portability but in return can provide gain in efficiency. The activation of behavioral rules may also wait sometimes for the I/O operations (e.g., packets exchanged with remote peers). Thus, the latency due to I/O will be considerably reduced if the rules activation are carried out in parallel.

Figure 8.35. Latency in activation of behavioral rules after design optimization. Each client device registers: 1 PING, 1 WoC, 1 TCP-KA and 1 Heartbeating.

The design optimization provided great improvements in the performance as apparent from the Fig. 8.35. Now it takes less than 1 second for the NCP to start covering on behalf of client device even when already covering for large number of client devices and behavioral rules. We performed this analysis under the worst case scenario with 500 ms timeout for the TCP-KA rule. Fig. 8.36 presents performance in a more realistic set of behavioral rules. Thus, design optimization provided great improvements in the performance. The latency is now almost constant even with large number of TCP-KA rules.

Figure 8.36. Latency in activation of behavioral rules after design optimization. Each client device registers: 1 PING, 10 WoC, 10 TCP-KA and 5 Heartbeating.

### 8.6.2 Performance Analysis on High Processing Device

Along with the filtering engine, the NCP performance is also dependent on the device hosting the NCP service. A more powerful device is expected to offer much improved performance and scalability.
Latency in Behavioral Rules Registration

In Section 8.5.1, we presented the latencies in the registration of behavioral rules on different low power hardwares for local environment. The latencies in the registration of behavioral rules were quite small even on Lantiq home gateway and may not cause any adverse effects on the NCP performance. However, these latencies can be further reduced if more powerful device is used to host the NCP service.

Fig. 8.37 depicts the latencies in behavioral rules registration on a standard desktop PC (Intel Core i7, CPU: 2.2 GHz, RAM: 4 GB). We observed the NCP processing and NCP + UPnP processing latencies for each type of behavioral rule with respect to increasing number of previously registered client devices. Compared to Lantiq home gateway (average NCP latency of 35 ms and UPnP latency of 20 ms), the NCP processing latency on desktop PC reduced to 1.5 ms and UPnP latency on average to 0.6 ms. With such low latency values, the NCP can successfully process rules registration concurrently from large number of client devices.

Figure 8.37. Latency introduced in registration of different behavioral rules vs increasing number of registered client devices on a standard PC.

Memory Requirements

The memory available on the device hosting the NCP service is an important factor determining the scalability and indirectly the achievable savings. In Section 8.5.2, we presented the memory requirements for the NCP on different low power hardwares. However, for the desktop PC, we have 4 GB of available memory which is more than enough to support thousands of client devices. For the analysis of memory requirements, we considered different set of behavioral rules registered by client devices. For simplicity, we considered four different sets of behavioral rules: Network Presence (1 PING rule w/o DHCP), Network Presence (1 PING rule w DHCP), Reachability (1 PING, 1 DHCP & 1 WoC rule) and Each Rule Once (1 PING, 1 DHCP, 1 WoC, 1 TCP-KA & 1 Heartbeating rules). Fig. 8.38 depicts the memory usage of the NCP service on standard PC with increasing number of registered client devices where each client device registers a set of behavioral rules. Likewise, low power hardwares, the memory usage increases with the increasing number of registered client devices. Further, the memory usage also depends on the number of rules registered by each client device (more memory required if large number of behavioral rules requested from the NCP). However, unlike Lantiq home gateway, the LwPCAP has less memory requirements than the basic PCAP filtering method (similar to Raspberry Pi and Jetway PC).
Launched Device Registration & Setup of Filtering Engine

In Section 8.5.4, we presented the latencies in client device registration and setup of filtering engine for different low power hardwares. The latency in setting up the filtering engine is the most critical for the NCP operations and for maintaining the transparency with respect to third parties. However, it depends on the processing power of the NCP hosting device (small latencies are observed on devices with high processing power).

Once again we considered the same sets of behavioral rules to measure the latencies in client devices registration and setup of filtering engines. The sets of behavioral rules under consideration include: Network Presence (1 PING rule w/o DHCP), Network Presence (1 PING rule w DHCP), Reachability (1 PING, 1 DHCP & 1 WoC rule) and Each Rule Once (1 PING, 1 DHCP, 1 WoC, 1 TCP-KA & 1 Heartbeating rules). Since the desktop PC is more powerful than the low power hardwares tested before, we are expecting lower latencies. Fig. 8.39 depicts latencies on the desktop PC for both, awake clients and sleeping clients under PCAP and LwPCAP filtering methods. The performance observations are similar to the low power hardwares tested before. However, the latencies are much lower on desktop PC due to its high processing power.

8.7 Expected Energy Savings

The real energy savings depend on how long the network devices can sleep and what type of traffic can be managed by the NCP on their behalf. Estimating the time a device can sleep is more complicated and
requires detailed survey on user behavior. The NCP behavior for different kinds of network traffic also
determines how long its client devices can sleep. To have an estimation of possible energy savings, authors
in [7] provided detailed analysis of network traffic in office and home environments. The authors have
divided proxying into 4 categories; (i) Proxy-1: wake-up the device on any packet except ignorable traffic
(HSRP, PIM, HSPR, DHCP etc), (ii) Proxy-2: wake-up the device on any packet except with those requiring
simple mechanical response (ARP, ICMP, IGMP etc), (iii) Proxy-3: proxy a small set of applications such
as telnet, ssh, vnc, file sharing and NetBIOS, and (iv) Proxy-4: proxy all kind of traffic except schedule
tasks such as regular network backups, anti-virus or software updates. Based on these proxies types, the
authors have estimated possible sleep duration as percentage of the idle periods for both home and office
environments [7]. Using the basic information from [7], we calculated the maximum sleep duration per year
as in Table 8.10 considering that a typical desktop PC remains idle for 60% of the time (14.4 hours/day).

The energy savings achievable with the NCP concept can be expressed as follows:

\[ E_{\text{Savings}} = \sum_{i=0}^{N} (E_{i, \text{No--NCP}} - E_{i, \text{With--NCP}}) - E_{\text{Hardware}} \]  

(8.1)

where \( E_{\text{Savings}} \) is the achievable energy savings with the NCP concept, \( E_{i, \text{No--NCP}} \) is the energy
consumed when the client device \( i \) remains always powered up, \( E_{i, \text{With--NCP}} \) is the energy consumed when
client device \( i \) can sleep when idle, \( E_{\text{Hardware}} \) represents the energy consumed by the device hosting the
NCP service and \( N \) represents the total number of devices benefitting from the NCP service (its maximum
value depends on the hardware hosting the NCP service). \( E_{i, \text{With--NCP}} \) is the sum of the energy consumed
Table 8.10. Maximum sleep duration for a client device in home and office environments.

<table>
<thead>
<tr>
<th></th>
<th>Office Environment</th>
<th>Home Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proxy-1</td>
<td>Proxy-2</td>
</tr>
<tr>
<td>Sleep (percentage of idle time)</td>
<td>20%</td>
<td>48%</td>
</tr>
<tr>
<td>Sleep/day (hours)</td>
<td>2.88</td>
<td>6.91</td>
</tr>
<tr>
<td>Sleep/year (hours)</td>
<td>1051.2</td>
<td>2522.15</td>
</tr>
</tbody>
</table>

by the device in active state \(E_{i,\text{Awake}}\) and sleeping state \(E_{i,\text{Sleep}}\). The values of \(E_{i,\text{Awake}}\) and \(E_{i,\text{Sleep}}\) depend on the time the device \(i\) stays in active or sleep state. The above equation can be rewritten as follows:

\[
E_{\text{Savings}} = \sum_{i=0}^{N} (E_{i,\text{No-NCP}} - E_{i,\text{Awake}} - E_{i,\text{Sleep}}) - E_{\text{Hardware}}
\]  

(8.2)

The equation can be expanded using power notation as follows:

\[
E_{\text{Savings}} = \sum_{i=0}^{N} (P_{i,\text{Awake}}t_{24h} - P_{i,\text{Awake}}t_{i,\text{Use}} - P_{i,\text{Sleep}}t_{i,\text{Sleep}}) - P_{\text{Hardware}}t_{24h}
\]  

(8.3)

where \(P_{i,\text{Awake}}\) and \(P_{i,\text{Sleep}}\) represent the power consumption of the client device \(i\) in awake and sleep states, respectively. \(P_{\text{Hardware}}\) represents the power consumption of the device hosting the NCP service which remains powered-up 24/7 to make the NCP service always available. \(t_{i,\text{Use}}\) and \(t_{i,\text{Sleep}}\) represent the time client device \(i\) stays in awake and sleep states, respectively. \(t_{24h}\) means the device stays powered-up 24/7. Since \(t_{24h} = t_{i,\text{Use}} + t_{i,\text{Sleep}}\), the equation can be rewritten as:

\[
E_{\text{Savings}} = \sum_{i=0}^{N} (P_{i,\text{Awake}}t_{i,\text{Sleep}} - P_{i,\text{Sleep}}t_{i,\text{Sleep}}) - P_{\text{Hardware}}t_{24h}
\]  

(8.4)

The final expression for achievable energy savings can be written as:

\[
E_{\text{Savings}} = \sum_{i=0}^{N} (P_{i,\text{Awake}} - P_{i,\text{Sleep}})t_{i,\text{Sleep}} - P_{\text{Hardware}}t_{24h}
\]  

(8.5)

It can be observed from equation 8.5 that expected energy savings depend on the number of client devices, power consumption of the client devices, power consumption of the device hosting the NCP service and the sleep duration of the client devices. In Fig. 8.40, we plotted the expected energy savings for NCP service being offered by different hardwares assuming 60 W as average power consumption for a typical desktop PC when awake and 5 W when sleeping. We considered the full load power consumption of the devices hosting the NCP service for the worst-case analysis. It is important to note that Fig. 8.40 shows results when using standard PCAP filtering in NCP software. From our previous experiments, we observed that the setup of the filtering engine is the most critical factor when using standard PCAP filtering (than other performance factors such as memory) and limits the NCP scalability (see Fig. 8.20(b)). However, setup of filtering engine also indirectly depends on the processing power of the device hosting the NCP service. Considering 1 second as the threshold to avoid any packet loss and to successfully impersonate a client device presence, the NCP is hardly scalable upto 52 devices on Jetway while 36 and 39 on Lantiq and Raspberry, respectively. The scalability will be high if a more powerful device is used for hosting the NCP service. Remember, this scalability limitation is due to standard PCAP filtering. Using Lw-PCAP filtering feature of the NCP software, scalability can be easily raised to over thousand client devices. Lw-PCAP
filtering feature has no strict scalability limiting factor (almost constant filter setup latencies, negligible memory requirements with increasing client devices etc) however, may result in some packet loss when the traffic or filtering load is too high.

![Graphs showing energy savings for different proxies](image)

Figure 8.40. Expected energy savings for a typical desktop PC in home and office environments using different types of proxies. Note: The analysis is presented for different hardwares using standard PCAP filtering.

Fig. 8.40 depicts the maximum possible energy savings achievable from a single NCP service on different hardwares using standard PCAP filtering (based on their scalability). It is important to note that Jetway only provides positive energy savings if it is covering for more than 4 client devices using Proxy-1 and more than 1 client device using Proxy-2 in office environment. However, Jetway must cover for more than 1 client device (using Proxy-1 & Proxy-2) in home environment for positive energy savings. The main reason behind this is the comparatively high power consumption of Jetway (than Lantiq and Raspberry) and very small sleep duration of client devices under Proxy-1 and Proxy-2 scenarios. Further, this limit also depends on the awake and sleep power consumption of the client devices. Thus, the NCP must cover for larger number of client devices if it is being hosted on a high power device (power consumption of NCP hosting device will spread over all client devices causing negligible incremental power consumption). Considering 22 cent/kWh as the average cost electricity in Europe, a single NCP service can provide maximum savings of 2823, 1978 and 2149 Euro/year in office environment on Jetway, Lantiq and Raspberry, respectively. The savings are comparatively higher in home environment and a single NCP service can provide maximum savings of 3149, 2204 and 2393 Euro/year on Jetway, Lantiq and Raspberry, respectively. The savings might look small, however can constitute to Billions of Euro annual savings if the NCP service is used worldwide covering for millions of devices.
Chapter 9

Conclusions

Reducing energy waste in ICT devices is a primary research topic for the network engineers due to rising cost of electricity, limited available natural resources, continuous growth and development of new ICT devices and environmental concerns. In this dissertation, we have addressed an important research topic called ‘Network Connectivity Proxy’ which is an optimal strategy to reduce idle energy waste in ICT devices. The NCP allows networked devices to sleep when idle without loosing their network standings/presence. In short, the NCP maintains link layer, network layer, transport layer and application layer presence on behalf of sleeping devices.

9.1 Main Contributions

The NCP is quite useful approach that can provide Billions of Euro annual savings. In this dissertation, we have presented the design and implementation of our NCP prototype and addressed its supported features and capabilities. We evaluated the performance of our developed NCP prototype with large number of experimental tests and evaluated its effectiveness in the realistic scenarios. We also presented a collection of our contributions to state of the art such as design of a flexible and reliable communication protocol, expanding of NCP coverage beyond LAN boundaries, NCP support for mobile devices etc. In particular, we summarize our main contributions in the following:

• Due to very large number of available network based applications and protocols (which are continuously increasing), it is tedious job to implement their proxying capability in the NCP. It is important for the NCP to provide generalized set of behavioral rules and features which can be requested by heterogeneous devices. Thus, we developed quite generalized set of behavioral rules supported by our NCP prototype which can be enough to provide desired proxying capability for any application or protocol. Further, we proposed a template based approach to generate/respond to periodic application specific presence messages on behalf of sleeping devices. Our proposed approach do not require application source code and is also suitable for the proprietary closed-source applications. However, this feature of our NCP prototype is future oriented and requires application developers to incorporate in their applications a new feature for filling the heartbeat message template at NCP(if they want their clients to take benefit from the NCP service). We also implemented ‘TCP Migration’ feature in our NCP software to preserve open TCP connections on behalf of sleeping client devices. The TCP migration feature transfers a TCP session from client device to NCP and vice versa and resumes there in complete transparency with respect to their peers.

• The second interesting quality of our developed NCP prototype is its deployment flexibility in the network. Our NCP prototype is very flexible and can work in any deployment scenario i.e., placing
the NCP service on on-board NIC, switch/router or on a standalone PC. This flexibility enables us to deploy the NCP service on any device that meets our needs (e.g., based on the required scalability). On-board NIC and switch/router are the optimal locations for the NCP service if it is expected to cover for fewer devices e.g., home/small office environment. However, the NCP service can be deployed on a powerful PC if high scalability is desirable e.g., medium or large size organizations.

- We developed a very flexible and reliable communication protocol for information exchange between the NCP and client devices (such as power state notifications, rules registration/de-registration, device registration/de-registration etc). We proposed UPnP architecture as the best suitable option for the NCP framework to avoid any kind of configuration issues. Our developed UPnP-based communication protocol supports interesting features such as auto-discovery, zero-configuration and seamless communication between the NCP and client devices. Our another motivation for developing communication protocol based on UPnP architecture is that it can be supported by heterogeneous devices; thus enabling deployment flexibility for the NCP service as addressed before. Also, a recent remote-access specification of the UPnP architecture allows communication between devices located in two different networks over the Internet (thus, making it suitable also for the Internet NCP approach addressed in this dissertation).

- We proposed different architectural designs to enable NCP support for mobile devices. Its true that mobile devices are usually portable and consume much less power; however, the NCP concept can help to significantly improve their battery life. Although our current NCP implementation works very well for fixed and mobile devices located in the same local network; however, our NCP prototype lacks the implementation of our proposed mobility management architectures (e.g., when a mobile device goes out of the local network). In this dissertation, we mainly proposed different possible architectural designs for supporting client devices mobility in the NCP framework and addressed key issues and challenges that can arise under different deployment considerations of the NCP service.

- We expanded the original NCP coverage beyond LAN boundaries which can analyze the real potential of NCP in terms of energy savings by covering for very large number of client devices globally. A single global and powerful NCP instance located anywhere in the Internet can make easier the implementation of complex tasks and boosts up the energy savings by also shutting down the unused access links and the packets forwarding equipments whenever possible. However, Internet NCP faces several critical issues and challenges including NAT/Firewall issues and traffic diversion in public infrastructure. In this dissertation, we mainly highlighted different possible issues and challenges and proposed different architectural designs for the Internet-wide NCP and traffic diversion strategies in the public infrastructure. We enhanced our original NCP framework for local networks by improving its scalability and drastically reducing the NCP processing time when a client device goes to sleep mode. The implementation of traffic diversion strategies is out of scope of this dissertation and is our possible future work in this direction.

- The final important contribution of this dissertation is the extensive performance evaluation of our developed NCP prototype in different realistic scenarios. The NCP performance especially the scalability depends on the device hosting the NCP service. Thus, we performed large number of experiments and evaluated the effectiveness of NCP service on different low power hardware entities (Lantiq home gateway, Raspberry Pi pocket PC and Jetway low power PC). We performed our experiments under extreme or worst conditions and analyzed different issues and critical factors limiting the NCP scalability on specific hardwares. Further, we analyzed the effectiveness of NCP filtering engine under heavy traffic load and computational load and proposed and tested different enhancements to improve the overall performance.
9.2 Future Work

In this dissertation, we have researched the NCP concept in quite different ways to reduce the idle energy waste in network devices. However, there are still some open issues and challenges that need to be addressed. Some possible future work on the NCP concept include the following.

- Although we have addressed the NCP support for mobile devices in Chapter ?? and highlighted different issues and challenges in supporting mobility for mobile devices. We proposed different architectural designs for supporting device mobility and addressed the benefits and limitations of each. However, our current NCP prototype lacks the implementation of our proposed mobility management architectures. Thus, future work will focus on the implementation of suitable mobility management architecture and proper assessment for its effectiveness in the realistic scenarios.

- We proposed different architectural designs to expand the NCP coverage beyond LAN boundaries in Chapter ?? . However, our current NCP prototype lacks the implementation of our different proposed traffic diversion strategies in the public infrastructure. Thus, future work will focus on the implementation of proposed traffic diversion mechanisms and evaluation of their effectiveness and impact on the NCP operation. Since, user privacies may be exchanged over public Internet between LNCP and INCP; it is important to incorporate proper security measures in the communication protocol as well as the security of INCP.

- It is important to note that the heartbeating rule of our NCP prototype is future oriented and requires application developers to incorporate a new feature in their applications for filling heartbeat message template at the NCP. However, we successfully tested this NCP behavioral rule for our developed IM application; the future applications need to incorporate support for this feature. Further, future work can focus proposing and implementing some further behavioral rules for the NCP to manage complex applications on behalf of sleeping client devices such as sharing or downloading files over P2P etc.

- Our developed NCP framework is cooperative which requires also a piece of software on the client devices for communication with the NCP. Invisible NCP offers transparency with respect to client devices however faces several critical issues such as predicting device power state, predicting device presence in the network, predicting what applications and protocols to proxy etc. Future work may focus on the development of invisible NCP by somehow effectively addressing its key limitations.
Appendix A

UPnP Architecture

A.1 Overview

The UPnP is an emerging standard that operates on the top of several networking protocols to provide interesting features such as auto-discovery, zero-configuration and seamless communication between the network devices. It uses distributed soft states for service advertisements, automatic and transparent networking between devices without requiring any infrastructure. Each UPnP device carries a description file written in XML syntax. The description file is the formal definition of the logical device expressed in the standard format/template including device manufacturer name, model name, model number, serial number, and URLs for controlling, eventing, and device presentation. The UPnP device may be offering one or more services which exposes some actions and models the state of a physical device with a number of state variables. Each service carries a service description file written in XML syntax which contains the formal definition of the functionalities offered by that service.

The UPnP technology allows network devices to periodically announce their presence in the network, discover the presence of other devices, get the description of available devices and services and remotely control them, subscribe to different events and present the devices and services descriptions in human readable format. Such features of the UPnP architecture make use of the several different protocols: (i) SSDP to search/advertise device presence, (ii) General Event Notification Architecture (GENA) protocol for notification about device updates and (iii) Simple Object Access Protocol (SOAP) which is used to send actions/commands to a UPnP device. The SOAP and GENA protocols usually operate on the top of HTTP protocol while SSDP uses HTTPU (an extension of the HTTP protocol which uses UDP as the transport protocol instead of TCP).

The UDA specified two main roles for the UPnP devices: CD and CP. The CD is the physical entity that offers one or more services. The CP runs on another network device that sends commands to specific service of the CD and invokes particular actions.

A.1.1 UPnP Control Point

Fig. A.1 depicts the basic structure of UPnP CP. The Device Manager manages the list of all discovered devices over the network and retrieve their device and service description files. The activities of Device Manager are based on the SSDP protocol on the top of UDP protocol. The CP also contains Action Manager that can invoke different actions on the CD related to different services. Finally, the Event Manager is responsible for subscription of different events such as changes in the state variables related to a particular action. The Event Manager is also responsible to periodically renew the device and service descriptions before their registration expires. Subscription renewal is particular important to know about the absence of CD if it accidentally leaves the network due to cable disconnection or other reasons.
A.1.2 UPnP Controlled Device

Fig. A.2 depicts the basic structure of UPnP CD. The CD may be offering one or more services. The Description Manager is responsible for managing the device description as well as the description of all services it is offering. The device and services descriptions are expressed in XML files. The Description Manager is also responsible for periodic advertisement of the CD in the network and provide relevant description files to the CP during the registration process. Another part of CD is Services Manager that manages the implementation of different services and their supported actions. The Services Manager receives commands from the CP after which it executes the particular action in the specified service. The Services Manager is also responsible to keep the CP updated about the CD status if any of its state variable value has changed.

A.2 Semantics of the UPnP Networking

The basic semantics of the networking between UPnP enabled devices is depicted in Fig. A.3. The UPnP semantics can be briefly described in the following:
• **Addressing:** When connected to the network, both UPnP enabled devices get an IP address to be able to communicate with one another. A UPnP device may obtain IP address through DHCP, auto-IP or use user pre-configured IP address.

• **Discovery:** Discovery of UPnP devices happen immediately after addressing. The UPnP enabled CD advertises its services presence in the network using SSDP. Similarly, a UPnP enabled CP also uses SSDP in order to find services of interest offered by the CD.

• **Description:** Description is followed by discovery. During description phase, the UPnP enabled CP retrieves the service and CD descriptions using the control URL obtained during discovery phase of the CD. The device and service descriptions are expressed in XML format that contains complete information about the specific CD and service.

• **Control:** During this phase, the UPnP enabled CP invokes different actions in the services offered by the CD using SOAP protocol. The control messages are also expressed in the XML format.

• **Eventing:** During this phase, the UPnP enabled CD returns the results of actions performed during the control phase. These action may cause changes in the values of state variables which are notified to the CP. Further, services also use eventing messages to publish updates about state variables that are not caused due to control phase. Eventing messages are also expressed in XML format.

• **Presentation:** The CP retrieves the status/information of the CD using the presentation URL which may allow users to control and/or just view the CD status.

![Figure A.3. Semantics of the UPnP networking.](image)
Appendix B

Mobile IP

Mobile IP [73,74] was proposed to manage mobility for nomadic hosts. It identifies three main architectural elements: Home Agent, Foreign Agent and Mobile Nodes (see Fig. B.1). MNs and the HA are assigned IP addresses with a common prefix, which is called Home Network; the HA always stays in the home site, whilst MNs are free to move around but keep the same Home Addresses. When MNs are at home, their addresses are topologically congruent with the site, and nothing happens different from the standard Internet behavior; however, when they are away, their addresses do not belong to the Foreign Network they are visiting, packets cannot be routed there and this is where the MIP protocol comes into play.

Figure B.1. MIP architecture. Packets intended for MNs are caught by their HA and forwarded through the tunnel when they are away from home. The picture shows reverse tunneling of packets sent by MNs.

The Home Agent is used as a fixed anchor to reach mobile nodes wherever they lie; it always knows their current position. The HA and the FA send out mobility advertisements announcing their presence and their network prefix; by looking at these messages a MN can easily and quickly infer whether it is at home or in a foreign network. In the second case, it registers its position with the HA; the registration includes a Care-of Address. The CoA is an IP address belonging to the current network; it could be the address of the FA (FA-CoA) or a temporary address acquired by the MN either by DHCP or other auto configuration mechanisms. A tunnel is set up between the HA and the device holding the CoA; this tunnel is used to deliver packets to the MN.
Packets addressed to MNs are routed towards their HN. When a MN is at home, packets are directly received; when the MN is away, the HA catches its packets by answering ARP requests addressed to the MN’s address with its own MAC address. Packets are then forwarded by the HA to the current CoA through the tunnel (see Fig. B.1). When a FA-CoA is used, the FA delivers packets to the MN on the local site despite its address falls outside the prefix of the network. In any case, no changes are seen by applications and on-going transport sessions while the device is moving around.

Packets from MNs in the Foreign Networks could be sent directly. However, many routers have anti-spoofing filters that discard outgoing packets whose source address does not belong to the network’s prefix. Moreover, this creates an asymmetry in the paths to and from MNs; this effect is known as triangular routing (from the geometrically shape given by the forwarding between CN and the HA, between the HA and the MN and between the MN and the CN) and could have many side effects. Alternatively, packets can be sent through the same tunnel between the HA and CoA (reverse tunneling), which solves many issues related to triangular routing.

The presence of the FA speeds up the registration procedure and leads to faster hand overs. They are not present in MIPv6 [74] because their functionalities could be included in IPv6 routers. Further, MIPv6 includes route optimization, which allows direct tunneling of packets between the CN and the MN (most IPv6 nodes are expected to be mobility-aware). A number of extensions have been proposed for low latency, fast hand overs, hierarchical operation of MIP; it not worth including them in this short review mainly aiming at introducing MIP for supporting mobile low-power nodes.
Appendix C

Virtual Network Address Translation

The VNAT [69] is a proposal for managing mobility through translation between network identifiers. In VNAT, three elements are added to the standard stack: Virtualization, Translation and Migration (shown in Fig. C.1).

Virtualization translates addresses used by applications into virtual identifiers. Virtual identifiers never change during a session and thus can be used for permanent and unique host identification. Translation translates virtual identifiers into real IP addresses and port numbers congruent with the underlying infrastructure. Real identifiers changes while a device moves. Operation of the Translation is basically the same like standard NAT, hence the name of this proposal. Migration is a cross-layer component that synchronizes information and operation of Virtualization and Translation. It includes all signaling required to update Translation after a migration. An example of protocol operation is shown in Fig. C.2.

Real identifiers are used at the network layer, which are topologically congruent with the current point of attachment; these addresses are expected to change while the host moves around. The transport layer will use virtual identifiers, and will never be affected by migration issues. A typical question is: how could a host be aware of virtual identifiers used by remote peers? And another one is: what kind of addresses should applications use and what would be the relationship with virtual identifiers? In practice, a good solution would be to choose virtual identifiers as the real addresses used when each session is started. This
would bring two benefits: I) there is no need for any signaling for getting the peer’s virtual identifier and ii) translation overhead is only due after a migration occurs.

VNAT can be used in two ways: end-to-end or with a proxy supporting one peer. However, the native paradigm for this scheme fits better the first approach. VNAT also has some basic limitations as follows:

- in principle, it is not applicable to connectionless protocols; applications should make use of the same real address, to enable correct operation of Virtualization, but this behavior is not consistent with the current Internet architecture;

- simultaneous mobility of both peer is not supported; connections are lost in this case;

- more than one host could use the same address to connect to the same peer; this could happen when an host migrate, because its applications keeps on using the address locally, but the address may be assigned to another host in that subnet.
Bibliography


Bibliography


