Neighbor Discovery Proxy-Gateway for 6LoWPAN-based Wireless Sensor Networks

Design, Implementation, analysis, and evaluation

LUIS MAQUEDA ARA

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

Neighbor Discovery Proxy-Gateway for 6LoWPAN-based Wireless Sensor Networks

Design, Implementation, analysis, and evaluation

Luis Maqueda Ara
lc.maqueda@gmail.com

December 21, 2011

Supervisor: Gerald Q. Maguire Jr.

School of Information and Communication Technology
KTH Royal Institute of Technology

Stockholm, Sweden

Abstract

The IETF 6LoWPAN working group has defined a number of optimizations to adapt the traditional IPv6 Neighbor Discovery protocol to *non-transitive* wireless links. While these optimizations result in a more efficient use of the resources of hosts **within** a 6LoWPAN network, they introduce a number of impediments for communication **between** nodes in traditional IPv6 networks and nodes in 6LoWPAN networks. This document describes how to overcome these obstacles by providing the necessary proxy mechanisms, leading to a transparent, seamless, and cost-effective integration of 6LoWPAN nodes into existing IPv6 network infrastructures. In particular, this document details the requirements, specification, and implementation of an embedded device responsible for such integration: a 6LoWPAN Neighbor Discovery Proxy-Gateway (6LP-GW).

Moreover, this report demonstrates that integrating 6LoWPAN nodes into existing IPv6 networks by means of a 6LP-GW as described here is both feasible and convenient in most situations. This convenience can be observed from both the network and the end-user perspectives:

From the network's point of view, the solution proposed here integrates 6LoWPAN **into** an existing IPv6 network. Hence, 6LoPWAN nodes and traditional IPv6 devices can coexist within the same IPv6 subnet, sharing the same network prefix. Furthermore, enabling such integration and coexistence is **simple** and **inexpensive** in contrast to other solutions. The main reason for this simplicity is that the 6LP-GW is completely **transparent** from both the network layer and the neighbor discovery protocol's perspective: while each type of node still takes advantage of its own **specific** neighbor discovery protocol's features, all of them share the same IPv6 subnet and no node in the network is able to determine the nature of its neighbors (simply on the basis of the neighbor discovery protocol).

Each of the above advantages leads to an immediate benefit from the end-users' perspective: the integration of the 6LoWPAN network into the existing infrastructure, frees the user from having to acquire an expensive (and so far rare) border router. Instead, the end-user simply buys a 6LP-GW which, as previously mentioned, is inexpensive compared to the former; the 6LP-GW **broadens** the existing IPv6 router's functionality (in contrast to a 6LBR which would replace it). In addition, it is important to mention that using a 6LP-GW could not be simpler; once attached to an IPv6 router's LAN port, **no further intervention** is required.

As result, the solution proposed here undoubtedly eases and speeds up the **deployment** process of 6LoWPAN, enabling immediate use by even the most inexperienced user.

Sammanfattning

IETF 6LoWPAN arbetsgruppen har definierat ett antal optimeringar för att anpassa traditionella IPv6-protokollet granneupptäck till icke-transitiva trådlösa länkar. Medan dessa optimeringar resultera i en mer effektiv användning av resurserna värddatorer inom ett 6LoWPAN nätverk, införa de ett antal hinder för kommunikation mellan traditionella IPv6-nätverk och 6LoWPAN nätverk. Denna avhandling beskriver hur man kan övervinna dessa hinder genom att tillhandahålla nödvändiga proxy mekanismer som leder till ett öppet, sömlös, och kostnadseffektiv integration av 6LoWPAN noder i redan befintliga IPv6-nätverk infrastruktur. I synnerhet denna avhandling beskrivs de krav, specifikation och implementering av inbäddade enheter som ansvarar för dessa integration: 6LoWPAN granneupptäck proxy-gateway (6LP-GW).

Dessutom visar denna avhandling att integrera 6LoWPAN noder i befintliga IPv6-nät genom en 6LP-GW som beskrivs här är både möjligt och praktiskt i de flesta situationer. Denna bekvämlighet kan observeras från både nätverket och slutanvändarens perspektiv. Från nätverket synvinkel föreslog lösningen här integrerar 6LoWPAN i ett befintligt IPv6-nätverk. Därför kan 6LoPWAN noder och traditionella IPv6-enheter som samexisterar inom samma IPv6-subnät, att dela samma nätverk prefix. Dessutom möjliggör en sådan integration och samexistens är enkelt och billigt i motsats till andra lösningar. Den främsta orsaken till denna enkelhet är att 6LP-GW är helt transparent både från nätverkslagret och granneupptäckprotokoll perspektiv: medan varje nod fortfarande drar nytta av sin egen specifika granneupptäckprotokoll funktioner, alla har samma IPv6-subnät och ingen nod i nätverket har möjlighet att avgöra vilken typ av sina grannar.

Var och en av ovanstående fördelar leder till en omedelbar nytta av slutanvändarnas perspektiv: integrationen av 6LoWPAN nätet i den befintliga infrastrukturen, frigör användaren från att skaffa en dyr (och hittills sällsynt) gränsen router. Istället köper slutanvändaren helt enkelt en 6LP-GW som, vilket tidigare nämnts, är billig jämfört med tidigare, den 6LP-GW breddar den befintliga IPv6-router funktionalitet (i motsats till en 6LBR som skulle ersätta det). Dessutom är det viktigt att nämna att med en 6LP-GW kan inte vara enklare, när ansluten till en IPv6-router LAN-port krävs ingen ytterligare åtgärd behövs.

Som resultat av detta föreslog lösningen här underlättar onekligen och påskyndar distributionsprocessen för 6LoWPAN, vilket möjliggör omedelbar användning av även de mest oerfarna användare.

Acknowledgements

First and foremost I wish to thank my supervisor and examiner at KTH, professor Gerald Q. (Chip) Maguire Jr., not only for his invaluable guidance and advice throughout all this thesis project and his thorough and detailed reviews of my work, but also for having encouraged me to discover (and enjoy) a new field of study and a new vision of engineering in general.

I wish to thank also all the people that I have got to know and that have stood by my side during my three years of stay in Stockholm. Some of them have been helpful from the academic perspective, some from the personal standpoint, but each of them has partially contributed to shape the today's me and thus all of them will remain my firm friends for the rest of my life. Among these people, I wish to specially thank Elena Marquez, for her support, her help, her affection, and her unconditional love.

Special thanks to my beloved family, who despite the distance, has always stood by me, supporting and encouraging me to walk the line and keep working.

In addition, I wish to mention and thank all the people who has contributed to my education as engineer, meaning with this all the professors at my home school of engineering, Centro Politécnico Superior (CPS) of Zaragoza, deserving such mention, whose effort, willing, vocation and talent has made me love my profession.

Last, but not least, I wish to thank Nicolas Mechin in particular, and Sen.se in general, for their guidance and economic support, and for having broadened the perspective of this thesis from an academic project to a commercial product.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

This document requires readers to be familiar with terms and concepts described in RFC 4861 [34], RFC 4862 [47], RFC 4919 [30], RFC 4944 [33], draft-ietf-6lowpan-nd [43], and RFC 6282 [27]. For clarity we summarize some of these terms and give a short description of them before presenting them in next sections.

6CO	6LoWPAN Context option (I-D.ietf-6lowpan-nd)
6LBR	6LoWPAN Border Router (I-D.ietf-6lowpan-nd)
6LH	6LoWPAN Host, in contrast with a 6R (I-D.ietf-6lowpan-nd)
6LoWPAN-ND	Neighbor Discovery optimization for LLNs (I-D.ietf-6lowpan-nd)
6LR	6LoWPAN Router. 6LRs have only one interface and therefore they are not Border Routers
6R	6LoWPAN Router, either a 6LR or 6LBR
ABRO	Authoritative Border Router option (I-D.ietf-6lowpan-nd)
ACK	Acknowledgement
AES	Advanced Encryption Standard
API	Application Programming Interface
ARO	Address Registration option (I-D.ietf-6lowpan-nd)
ARP	Address Resolution Protocol (RFC 826)
BOOTP	Bootstrap Protocol (RFC 1531)
CRC	Cyclic Redundancy Check
DAC	Duplicate Address Confirmation message (I-D.ietf-6lowpan-nd)
DAD	Duplicate Address Detection (RFC 4861)
DAR	Duplicate Address Request message (I-D.ietf-6lowpan-nd)
DHCP	Dynamic Host Configuration Protocol (RFC 2131)
DNS	Domain Name Server
DTLS	Datagram Transport Layer Security
ECN	Explicit Congestion Notification (A Protocol for Packet Network Interconnection)

EUI-64	IEEE's 64-bit Extended Unique Identifier (EUI-64)
FCF	Frame Control Field (IEEE 802.15.4)
FCS	Frame Check Sequence (IEEE 802.15.4)
FFD	Full-Function Device (IEEE 802.15.4)
GTS	Guaranteed Time Slot
HAL	Hardware Abstraction Layer
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IHL	Internet Header Length (A Protocol for Packet Network Interconnection)
IID	Interface Identifier (RFC 4291)
IKE	Internet Key Exchange (RFC 2409)
IPHC	IPv6 Header compression (RFC 6282)
IPv4	Internet Protocol version 4 (RFC 791)
IPv6	Internet Protocol version 6 (RFC 2460)
IPv6-ND	Neighbor Discovery for IPv6 protocol (RFC 4861)
LLN	Low-power and Lossy Network
LoWPAN	Low power Wireless Personal Area Network
LRU	Least Recently Used
LR-WPAN	Low-Rate Wireless Personal Area Network (IEEE 802.15.4)
MAC	Medium Access Control (IEEE 802.3)
MPDU	MAC Protocol Data Unit (IEEE 802.15.4)
MTU	Maximum Transfer Unit
NA	Neighbor Advertisement message (RFC 4861)
NC	Neighbor Cache (RFC 4861)

NCD	Non-Constrained Device. Any device not having strong restrictions in terms of availability of resources (for example, a personal computer)
NCE	Neighbor Cache Entry (RFC 4861)
ND	Neighbor Discovery protocol, either 6LoWPAN-ND or IPv6-ND
NS	Neighbor Solicitation message (RFC 4861)
NUD	Neighbor Unreachability Detection (RFC 4861)
OSI	International Standards Organization's Open System Interconnect
PAN	Personal Area Network
PIO	Prefix Information option (RFC 4861)
RA	Router Advertisement message (RFC 4861)
RFC	Request For Comments
RFD	Reduced-Function Device (IEEE 802.15.4)
RFID	Radio-Frequency Identification
Router	Either a RR or 6R
RPL	IPv6 Routing Protocol for Low power and Lossy Networks
RR	Regular IPv6 router, in contrast with a 6R
RS	Router Solicitation message (RFC 4861)
RSSI	Received Signal Strength Indication
RSTP	Rapid Spanning-Tree Protocol (IEEE 802.1D)
RX	Reception
SEND	Secure Neighbor Discovery (RFC 3971)
SFD	Start of Frame Delimiter
SLLAO	Source Link-Layer Address option (RFC 4861)

SPI	Serial Peripheral Interface
STP	Spanning-Tree Protocol (IEEE 802.1D)
TLLAO	Target Link-Layer Address option (RFC 4861)
TX	Transmission

Chapter 1

Introduction

1.1 General introduction to the area

There is an increasing interest in using wireless communication with sensors and actuators in homes, office buildings, factories, and even outdoors. Moreover, there is a desire to incorporate these devices as part of the Internet — so that these devices could be accessed from anywhere.

From this perspective, embedding a TCP/IP stack into these sensing and acting devices seems an attractive idea, which is reinforced by the new features IPv6 provides (such as the large address space [26] and address autoconfiguration [47]). However, the TCP/IP protocol suite was not originally intended for such devices; its requirements for the underlying link layers are generally too strong to be carried out by resource-constrained devices, while certain network layer features are too complex and resource consuming.

For these reasons the IETF defined 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks): an adaptation layer which intermediates between the network and the link layers to provide all the services that the network layer requires but the link layer can not provide. In particular, RFC 4919 [30] states the problems and goals for the transmission of IPv6 packets over IEEE 802.15.4 [4] media. These problems and goals are addressed in RFC 4944 [33], "Transmission of IPv6 Packets over IEEE 802.15.4 Networks". This latter specification, together with the fact that most resource-intensive tasks present in the TCP/IP stack (mainly related to TCP) are usually not required in the applications under consideration for 6LoWPAN, makes it possible and efficient to implement the relevant subset of IPv6 in resource-constrained devices.

However, despite 6LoPWAN having successfully accomplished the tasks it was intended for, an important feature of IPv6 – specifically the Neighbor Discovery protocol [34] neither properly fits the characteristics of the underlying link layer nor meets the power-saving needs of devices that form 6LoWPAN networks. The reasons for this are:

- The extensive use of multicast in traditional IPv6 Neighbor Discovery. Multicast traffic is expensive in terms of overall power consumption since it requires every node in the network to receive and process a packet, even though such a packet is likely to be of no use for the node and hence ends up being discarded in most cases. In addition, it is important to note that a Low power Wireless Personal Area Network (LoWPAN)

is a set of domains within which broadcasting works, but there is no LoWPAN-wide multicast. Thus the usual assumption that a network segment (and hence a subnet prefix) is equivalent to a broadcast domain is not necessarily valid in the case of a LoWPAN.

- Incorrect assumptions about link properties. LoWPANs may consist of **non-transitive** links, i.e., wireless links with undetermined connectivity properties, as defined in RFC 5889 [8]. This is due to the fact that most nodes in LoWPANs want to sleep most of the time (to reduce power consumption) and are likely to be moved around (or even out of) their environment, leading to lossy links in which nodes may not always be reachable. In turn, Neighbor Discovery for IPv6 assumes that every node in the network is always listening to the medium and hence reachable.

- 6LoWPAN needs support for specific features. These features, despite not being part of the 6LoWPAN specification in RFC 4944, constitute a significant factor in terms of efficiency and power savings. Due to the presence of Neighbor Discovery in IPv6 and the traditional functions it performs, it seems reasonable to extend Neighbor Discovery so that it fulfils the needs 6LoWPAN devices.

The result is that traditional IPv6 Neighbor Discovery does not fulfil the requirements of 6LoWPAN networks, some of its features are unsuitable and/or inefficient, and some others do not work. For these reasons, the IETF 6LoWPAN working group defined a number of improvements/changes to RFC 4861 [34] and RFC 4862 [34] in "Neighbor Discovery Optimization for Low-power and Lossy Networks", I-D.ietf-6lowpan-nd [43] in order to adapt the Neighbor Discovery protocol for 6LoWPAN networks.

1.2 Longer problem statement

The optimizations defined in I-D.ietf-6lowpan-nd achieve a more efficient use of resources, and hence, a reduction in power consumption. However, these changes to traditional Neighbor Discovery are so significant that the "Neighbor Discovery Optimization for Low-power and Lossy Network" could be considered to constitute, in fact, a different Neighbor Discovery protocol (6LoWPAN-ND from now on) and, what is more, these changes cause both protocols to be **incompatible**. Section 2.9 describes the main features and behaviour of each protocol as well as the key differences between them.

Since the Neighbor Discovery protocol has link-local scope, the aforementioned incompatibility implies the side effect that 6LoWPAN devices using 6LoWPAN-ND **can not share the same network link** (hence, IPv6 subnet) as regular IPv6 devices using traditional Neighbor Discovery for IPv6. Thus, while RFC 4944 enables the use of IPv6 over low-power, lossy, asymmetric, and non-transitive networks, draft-ietf-6lowpan-nd

introduces some constraints regarding its deployment, turning the *existence* of a 6LoWPAN Border Router and a new IPv6 subnet into a *requirement*, for even the simplest 6LoWPAN deployment.

1.3 Goals

The ultimate goal of this thesis project is to investigate if integrating 6LoWPAN wireless links into existing IPv6 networks is possible. This integration should overcome the physical and logical communication barriers derived from the differences between the two network links *without* requiring a single change in the IPv6 network's infrastructure.

Therefore, this thesis covers all the processes from the initial literature study, to the final evaluation of results, passing through the necessary steps of description, formal specification, and implementation of an embedded Neighbor Discovery Proxy-Gateway for 6LoWPAN-based wireless sensor networks (6LP-GW). Such a device shall enable the integration of a 6LoWPAN network into an existing IPv6 network in a low-cost, homogeneous, transparent, and seamless manner, without losing any of the advantages derived from the use of the Neighbor Discovery protocol optimization for low power, lossy networks (LLNs).

Additionally, such a 6LP-GW shall furnish proper mechanisms to allow subsequent further integration of network applications, ranging from network monitoring and management tools to tunnelling services which would enable the use of 6LoWPAN devices in non-IPv6 environments. Such mechanisms consist of the integration of full IPv6 and IPv4 communication stacks, each with its own protocol-specific autoconfiguration/initial IP address configuration and address resolution operations.

A side goal of this thesis project, is the implementation and integration of the "Neighbor Discovery Optimization for Low-power and Lossy Networks", specified in I-D.ietf-6lowpan-nd, into the Contiki Operating System. This was a useful means to demonstrate the correct operation of the 6LP-GW, as no implementations of this relatively new protocol were available when this thesis project was first proposed. While it was not strictly necessary to do it this implementation using Contiki, it was very convenient to be able to implement and evaluate our solution while still taking advantage of an existing open-source, well-known, and tested 6LoWPAN implementation.

1.4 Required background knowledge

Throughout this report, the reader will find countless references to different communication protocols. Naturally, familiarity with these protocols will ease comprehension of the report. For completeness, Chapter 2 provides an overview of these protocols, with special attention to 6LoWPAN and Neighbor

Discovery (in both its flavours), so that any reader with basic knowledge about computer networks can successfully read this report. In addition, although a comprehensive list of acronyms and specific terms is provided for clarity on pages xv - xviii, familiarity with common computing concepts is assumed.

Needless to say, an experienced reader may choose to skip part or all of the Chapter 2 to directly go to the method proposed in Chapter 3. However, even such an advanced reader may find it useful to review the description of the two different Neighbor Discovery protocols along with the differences between them (Section 2.9, starting on page 19), as a thorough understanding of these protocols is crucial for a complete understanding of both the proxy operations described in this document and the motivations behind them.

1.5 Structure of this thesis

This thesis is divided into six chapters which follow a logical sequential order. These chapters are in turn grouped in pairs where the first pair of chapters is an introduction pair, the second pair explains the work that has been done, and the last pair analyses the results of the proposed solution.

The first chapter, “Introduction”, describes the area within which the problem addressed in this thesis lays, states the problem, and defines the goals to be achieved by this thesis. In addition, it explains what knowledge is required for the correct understanding of this thesis and describes the structure of this thesis. Chapter 2, “Background”, provides a general overview of most of the protocols, concepts, and previous work related to or relevant to the subsequent chapters. Chapter 3, “Method”, analyses the requirements of our application and provides a detailed specification for the operation of a Neighbor Discovery Proxy-Gateway for 6LoWPAN-based wireless sensor networks. The following chapter (Chapter 4), “Applying the Method”, describes the most relevant aspects of the actual implementation of the device. The fifth chapter, “Analysis”, presents tests that have been performed with the implemented the device in order to evaluate both its correctness and performance. The last chapter (Chapter 6), “Conclusions”, analyses the results obtained in Chapter 5 and summarizes the conclusions reached as result of the work performed during this thesis project.

Chapter 2

Background

2.1 The Internet Protocol Suite

The Internet Protocol Suite is a set of communication protocols grouped for the first time in RFC 1122 [11] and RFC 1123 [10]. It is often referred to as TCP/IP protocol stack due the division into abstraction layers of the communications suite. In this stack the information flows in both directions, but in such a way that each layer communicates only with the layer immediately above or beneath, by encapsulating the data on the way down and de-encapsulating it on the way up. In order for this communication to occur, each layer requires the layer underneath to meet certain requirements, and has likewise to fulfil the requirements of the layer placed immediately above.

According to RFC 1122, the Internet Protocol Suite is divided into four abstraction layers: Link Layer, Internet Layer, Transport Layer, and Application Layer. However, due to the usual mapping of the TCP/IP stack onto the International Standards Organization's Open System Interconnect (OSI) model, it is also common to refer to the Physical Layer as a hardware layer at the lowest part of the Link Layer. Figure 2.1, illustrates the TCP/IP protocol stack, including the Physical Layer within its lowest level.

Link Layer Groups the different protocols that operate only between adjacent nodes in the same link segment.

Internet Layer Is the set of protocols in charge of delivering packets from the originating host to the destination, traversing different networks if necessary. Due to the mapping onto the OSI model, it is also commonly referred to as "Network layer".

Transport Layer Provides convenient services such as connection-oriented data-stream support, reliable end-to-end communication, flow and congestion control, and host-level multiplexing.

Application Layer The set protocols involved in the process-to-process communication.

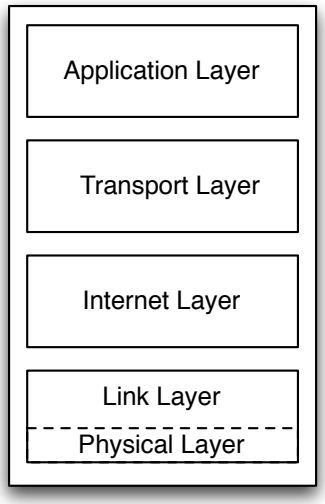

Figure 2.1: The TCP/IP stack, including the Physical Layer (dashed)

2.2 IEEE 802.3

IEEE 802.3 [5] is a IEEE working group and a set of standards rather than a single standard. There are several versions and amendments, with IEEE 802.3-2008 being the latest revision. IEEE 802.3-2008 defines the physical layer and data link layer's media access control (MAC) of a wired Ethernet.

As for the physical layer, this family of standards supports several types of media, such as different types of coaxial cable, shielded and unshielded twisted pair, and Fiber-Optics. The supported transmission data rates range from 10 Mbit/s to 100 Gbit/s. Some media support half or full-duplex transmission.

The MAC protocol specified in IEEE standard 802.3 is Carrier Sense Multiple Access with Collision Detection (CSMA/CD). This MAC protocol was utilized in the experimental Ethernet developed at Xerox Palo Alto Research Center. However, new implementations operating in full-duplex mode no longer utilize CSMA/CD —since in full-duplex mode for a point-to-point link there is no probability of collisions. This MAC layer consists of the channel-access portion of the link layer used by Ethernet, but does not define a logical link control protocol (generally implementations use the IEEE 802.2 logical link layer). Consequently, the standard defines the mapping between IEEE 802.3 MAC service interface primitives. As result, Ethernet's data link-layer protocol can be encapsulated within the MAC Client Data field of IEEE 802.3 packets (the common set of service interface primitives enables bridging between IEEE 802 MAC/PHY protocols). Figure 2.2 illustrates this Ethernet data link-layer into IEEE 802.3 MAC Client Data field encapsulation.

Note that when used with IEEE 802.2 there is an additional header before the Length/Type field.

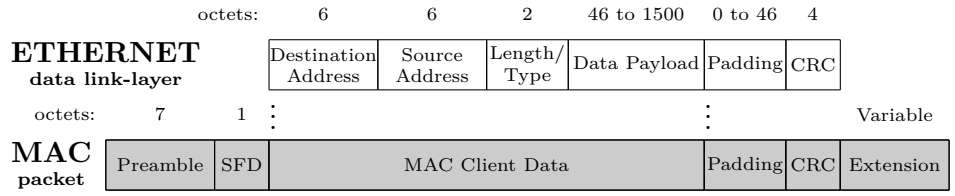

Figure 2.2: Ethernet data link layer protocol encapsulated into a the MAC Client Data field of a IEEE 802.3 MAC packet

Preamble Used for synchronization between sender and receiver.

SFD Start of Frame Delimiter. Indicates the end of the preamble and the start of the packet data. It has constant value of 0xAB (171_{10}).

Destination Address 48-bit IEEE 802.3 MAC address of the destination of the frame.

Source Address 48-bit IEEE 802.3 MAC address of the originator of the frame.

Type/Length This field can have two different meanings. If its value is greater than 1500, then it indicates the type of upper-layer packet being transported. If the field value is less than or equal to 1500, it indicates the length of the payload.

Data Payload The data being transmitted.

Padding Optional Padding. This is required if the total Ethernet frame length is less than 64 bytes.

CRC Cyclic redundancy check for integrity verification.

Extension Optional field included only in half-duplex operation when the frame is shorter than the CSMA/CD slot time.

2.3 IEEE 802.15.4

The IEEE 802.15.4 standard [4] specifies the physical and media access control (MAC) for low-rate wireless personal area networks (LR-WPANs). Although LR-WPANs fall within the wireless personal area networks (WPANs) family of standards, they may extend the personal operating space; an LR-WPAN is a simple, low-cost wireless communication network optimized for use in applications with limited power and limited throughput requirements. LR-WPANs aim for low power consumption and low cost, whilst maintaining a reliable data transfer, short-range communication link, and simple and flexible protocol.

2.3.1 IEEE 802.15.4 Topologies

The IEEE 802.15.4 standard defines two different device types: full-function devices (FFDs) and reduced-function devices (RFDs). FFDs can participate in the Personal Area Network (PAN) as a PAN coordinator, as a coordinator, or as a device. Even though a network may consist of just RFDs, the presence of at least one FFD acting as a PAN coordinator is recommended.

An LR-WPAN may operate in either peer-to-peer or star topologies. In a star topology, all the communication between devices must pass through the central node, which is the PAN coordinator. The PAN coordinator is thus responsible for initiating, routing, and terminating the communication in the network. On the other hand, in a peer-to-peer network, communication between any two nodes is possible as long they are in range; this topology offers greater flexibility, allowing all sorts of mesh formations, but at the cost of increased node power consumption. Peer-to-peer topologies *require* a PAN coordinator; however they are also likely to require a suitable routing protocol in case multihop is needed (i.e. if two nodes are not in range). This routing protocol should be provided by the upper layers and hence is beyond the scope of the IEEE 802.15.4 standard.

2.3.2 IEEE 802.15.4 Physical Layer

Since its release in 2003, different amendments have been defined adding new possible physical layers and/or extending the capabilities of the previously defined ones. At the time of writing this document (June 2011), the different unlicensed frequency bands and modulations, together with the supported data rates defined by the IEEE 802.15.4 physical layer are shown in Table 2.1.

2.3.3 IEEE 802.15.4 Medium Access Control (MAC)

The MAC layer is responsible for the following tasks:

- Beacon management
- PAN association and disassociation.
- Employing the CSMA-CA mechanism for channel access.
- Handling and maintaining the Guaranteed Time Slot (GTS) mechanism.
- Frame validation
- Acknowledged frame delivery
- Supporting device security.

Table 2.1: IEEE 802.15.4 physical layers, sorted by release date

Physical layer (MHz)	**Frequency Band (MHz)**	**Modulation**	**Bit rate (kb/s)**	**Description**
868/915	868 – 868.6	BPSK	20	BPSK: Binary phase-shift keying
	902 – 928		40	
868/915	868 – 868.6	ASK	250	ASK: Amplitude-shift keying
	902 – 928		250	
868/915	868 – 868.6	O-QPSK	100	O-QPSK: Offset quadrature phase-shift keying
	902 – 928		250	
2450	2,400 – 2,483.5	O-QPSK	250	
UWB	250 – 750	BPM-BPSK	851, 110, 6,810 and 27,240	BPM: Burst phase modulation UWB: Ultra-wide band
	3,244 – 4,742			
	5,944 – 10,234			
2,450 (CSS)	2,400 – 2,483.5	DQPSK	1,000	CSS: Chirp spread spectrum DQPSK: differential quadrature phase-shift keying
			250	
780	779 – 787	O-QPSK	250	
780	779 – 787	MPSK	250	MPSK: M-order phase-shift keying
950	950 – 956	GFSK	100	GFSK: Gaussian frequency-shift keying
950	950 – 956	BPSK	20	

In star-topologies, the IEEE 802.15.4 MAC layer provides a beacon-based synchronization mechanism for data transmission and reception between devices and the PAN coordinator, which permits nodes to only listen to the channel at regular intervals, allowing for power saving. In peer-to-peer topologies, however, this synchronization mechanism is not provided by the standard and, if required by specific applications, needs to be implement at upper layers.

The MAC layer defines four different types of frames: beacon frames, acknowledgement frames, MAC command frames, and data frames. Beacon frames are used in the synchronization mechanism. Acknowledgement frames, whose use is optional, are used to acknowledge transmissions. MAC command frames carry protocol commands, such as "Association request", or "Data request". Finally, data frames are used for all transfers of data. Figure 2.3 illustrates the structure of a data frame.

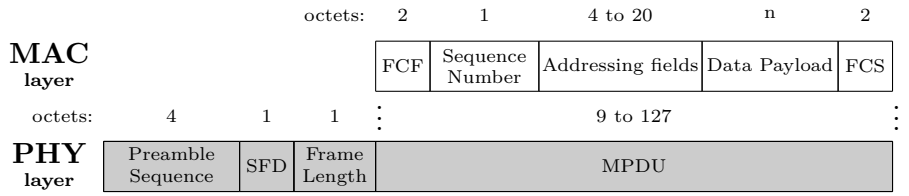

Figure 2.3: IEEE 802.15.4 data frame

Preamble Sequence Used to obtain chip and symbol synchronization with an incoming message. It is composed of 32 binary zeros.

SFD Start of Frame Delimiter. Indicates the end of the preamble and the start of the packet data. It has constant value of 0xE5 (229_{10}).

Frame Length Length of the MAC protocol data unit (MPDU).

FCF Frame Control Field. Contains information defining the frame type, addressing modes, and other control flags.

Sequence Number Used to match acknowledgement frames to data or MAC command frames

Addressing Fields The IEEE 802.15.4 standard supports short (16 bit) and long (64 bit) address. In addition, if the source and destination PAN identifiers are the same, one of them can be elided. Hence, this field containing the source and destination addresses as well as the source

	and destination PAN identifier, has variable length, and it is to be interpreted according to the FCF.
Data Payload	The data being transmitted.
FCS	Frame Check Sequence for data integrity verification.

2.4 Internet Protocol

The Internet Protocol (IP) is the principal Internet Layer protocol. It is a connectionless, best-effort, unreliable internetworking protocol which provides the necessary functions to deliver a packet from a source to a destination (both identified by fixed length addresses) over a system composed of an arbitrary number of networks. It also provides mechanisms for packet fragmentation and reassembly, if necessary.

The Internet Protocol was first defined by Vint Cerf and Robert Kahn in an IEEE journal paper entitled "A Protocol for Packet Network Interconnection" [12]. The protocol was later revised and updated up to its fourth version (IPv4), which is defined in RFC 791 [39], and became the first widely deployed version of IP.

2.4.1 IPv4

Internet Protocol version 4 (IPv4) is defined in RFC 791 [39] (replacing its previous definition in RFC 760 [38]). It uses 32-bit addresses, which limits the total number of IPv4 addresses to 2^{32}. Tts header has variable length (due to the options field), as illustrated in Figure 2.4 and described below. These two features (address length are variable length), together with the need for Flow Labelling capability constitute the main shortcomings/limitations of the protocol, and hence the reasons that have made necessary the definition of its next version (version 6). These features are explained in more detail in Section 2.4.2.

Version	Internet Protocol version. It has a value of 4 for IPv4.
IHL	Internet Header Length in multiples of 4 bytes. It is required since the header may contain a variable number of options.
Type of Service	The Type of Service (ToS) field provides an indication of the parameters of the quality of service desired. It is used to specify the treatment of the datagram during its transmission. RFC 2474 [35] redefines this field as the "Differentiated Services field" (DS field)

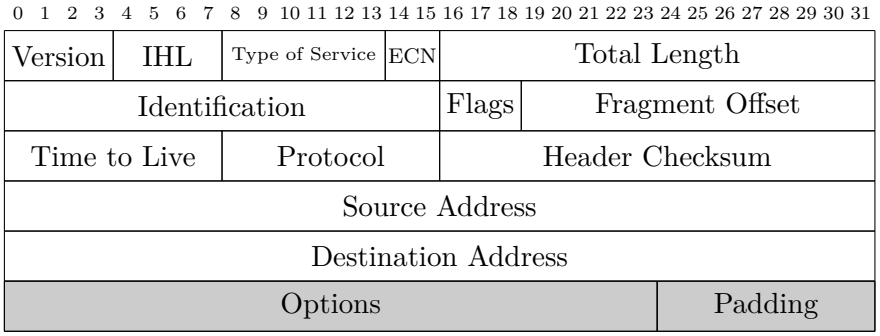

Figure 2.4: IPv4 datagram header. Light grey coloured fields are optional.

due to the limited practical use of the Type of Service field and the need for a new field by new real-time protocols.

ECN Explicit Congestion Notification (formerly part of ToS).

Total Length The total length of the packet, including the variable-length header. This field is needed to calculate the payload length, and imposes a maximum total packet length of $2^{16} - 1 = 65{,}535$ bytes.

Identification Numeric identifier used to uniquely identify a set of fragments belonging to the same packet.

Flags Used for fragmentation control, indicating whether a fragment is the last fragment or not of a packet, or if fragmentation is allowed for a packet.

Fragment Offset Specifies the offset of a fragment relative to the beginning of the original packet. This field is required for packet reassembly.

Time to Live Sets a maximum packet lifetime, to prevent packets from persisting in the network due to, for example, routing loops.

Protocol Indicates the protocol of the packet encapsulated by the IP header and transported in the IP payload.

Header Checksum 16-bit checksum field, used for header error-checking.

Source Address 32-bit IP address of the source of the datagram

Destination Address 32-bit IP address of the destination of the datagram

Options Optional field. It can contain a list of different options, but it must always be terminated with an "End of Options" option.

Padding Since the number of options is variable and the length of each option is also variable, and the header length field (IHL) is expressed in 32-bit multiples, padding is needed to ensure that the header contains an integral number of 32-bit words.

2.4.2 IPv6

The IP protocol version 6 (IPv6) is defined in RFC 2460 [15] (replacing its previous definition in RFC 1883 [14]). It was defined in 1998 in order to succeed IPv4, with the goal of overcoming a number of IPv4 shortcomings, especially, for dealing with the anticipated IPv4 address exhaustion.

The primary changes from IPv4 to IPv6 are an increased address space, which is 128 bits (allowing for up to 2^{128} — about 3.4×10^{38}) different IPv6 addresses), a simplified header format, with includes a fixed header-length and improved support for extension headers and options (allowing for more efficient packet forwarding and greater flexibility for introducing new options), flow labelling capability (with which the sender is allowed to request special handling by routers), and authentication and privacy capabilities. The IPv6 header format is described below and illustrated in figure 2.5.

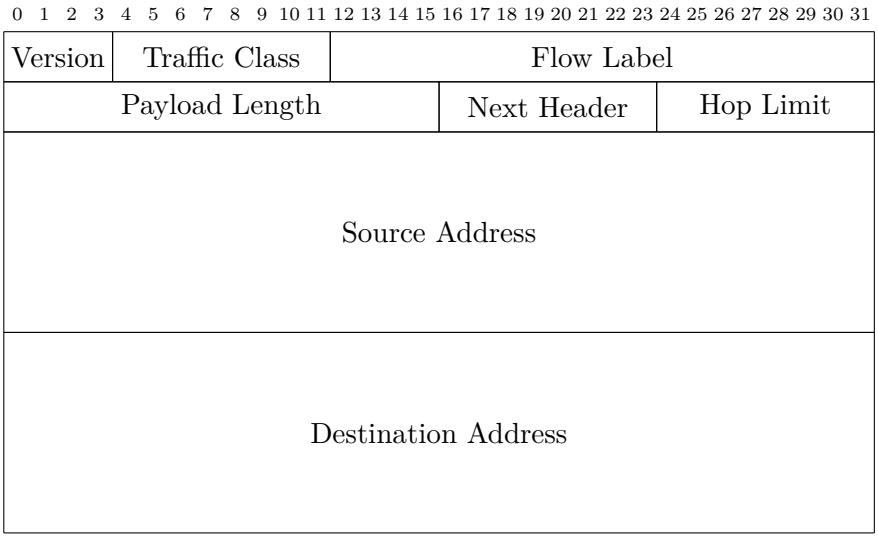

Figure 2.5: IPv6 datagram header

Version Internet Protocol version. It has a value of 6 for IPv6.

Traffic Class	Identifies different priorities.
Flow Label	Used by the source to label sequences of packets for which it requires special handling by routers, such as a non-default quality of service or "real-time" service. This field replaces the "Type of Service" field in IPv4.
Payload Length	Length of the IPv6 packet payload, not including the length of the header (40-bytes fixed length). Note that extension headers, if present, are considered part of the payload.
Next Header	Identifies the type of header immediately following the IPv6 header. This next header may indicate any upper-layer protocol or an IPv6 extension header.
Hop Limit	Packet lifetime. Used to prevents packets from indefinitely persisting in the network. It is specified as a number of hops (in contrast to the "Time to Live" IPv4 field, which is specified in seconds, requiring nodes to perform difficult time computations), which is decremented at every node where the packet is forwarded.
Source Address	128-bit IP address of the source of the datagram
Destination Address	128-bit IP address of the destination of the datagram

2.5 The Internet of the Things

The Internet of Things [32] is a paradigm which aims to provide everyday objects with a unique address, enabling their integration into the Internet. These objects are expected to provide contextual information and/or perform certain actions, according to their own interpretation of context and/or the orders received from remote hosts. This fact makes IPv6 (especially, its extremely large address space) a perfectly suited protocol for its use in the identification and communication between these objects and the rest of the Internet.

It is important to note that these objects do **not** require special capabilities: since a unique bar code or unique identifier in a radio-frequency identification (RFID) tag is sufficient to provide a unique identifier to an object, enabling every object to be identified and hence, integrated into the the Internet. However, the more processing capabilities the object has, the wider its communication capabilities will be; while some of these objects may have a read-only RFID tag (which may inform others of the object's location

when passing RFID readers located in known places), other "things" might implement a fully-compliant IPv6 protocol stack, becoming "first-class Internet citizens" capable of sending and receiving information to and from the Internet, just as any other network attached computer might. Between these two extremes, a vast range of possibilities exists, in which each object is required to implement *only* the minimum features necessary for its specific application.

Therefore, the Internet of Things concept provides for a large set of applications such as home automation, security, monitoring, smart metering, and management among others, providing the means to transform their environments into a smart, context-aware entity with the ability to sense and act. Consequently, these applications target many different markets, from individuals or families to industry.

2.6 6LoWPAN

6LoWPAN is an intermediate layer that allows the transport of IPv6 (see Section 2.4.2) packets over IEEE 802.15.4 (see Section 2.3) frames. Although the term 6LoWPAN stands for IPv6 over Low-power Wireless Personal Area Networks, it may extend the personal operating space, similar to LR-WPANs. RFC 4919 [30] describes an overview, assumptions, problem statement, and goals, while RFC 4944 [33] defines the standard itself. Figure 2.6 depicts how an IPv6 packet is encapsulated into a IEEE 802.15.4 frame using the 6LoWPAN adaptation layer.

The IPv6 standard defines certain requirements for the link-layers over which it is to be transported. However, the IEEE 802.15.4 MAC layer does not fulfil these requirements in certain points. Hence, the 6LoWPAN specification defines not only the frame format for the transmission of IPv6 packets over IEEE 802.15.4, but also the mechanisms to obtain a unique IPv6 address from either, 16-bit or 64-bit IEEE 802.15.4 MAC addresses (using Stateless Address Autoconfiguration—defined in RFC 4862 [47]), and to overcome the limitations of IEEE 802.15.4 [4].

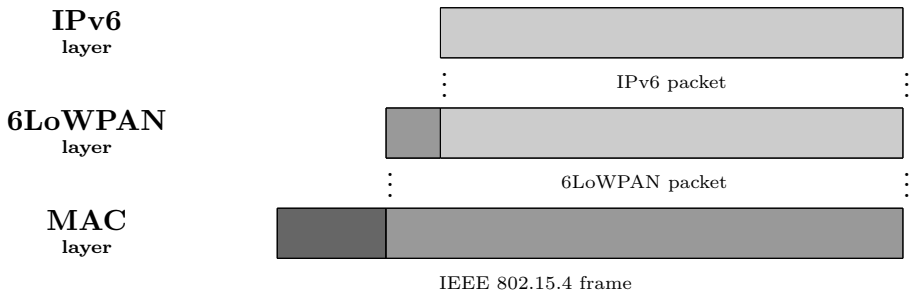

Figure 2.6: 6LoWPAN Intermediate layer

2.6.1 6LoWPAN Motivations

The minimum Maximum Transfer Unit (MTU) required for a link-layer transporting IPv6 packets is, as defined in RFC 2460, 1280 octets. This is far beyond the maximum IEEE 802.15.4 frame size, which is 127 octets. Of these 127 octets, the maximum MAC header size is 25 octets and, if IEEE 802.15.4 link-layer security is enabled, it may use up to 21 additional octets. This leaves only 81 octets available for IPv6 transport. As the IPv6 header length is 40 bytes, only 41 bytes are available for transport layers and so on.

In order to meet the IPv6 minimum MTU requirements, 6LoWPAN defines a fragmentation and reassembly mechanism that allows splitting IPv6 packets at the 6LoWPAN adaptation layer into smaller fragments that can be handled by the link-layer, with this process being transparent to the Internet layer. However, applications using 6LoWPAN are not expected to use large packets, hence, in order to avoid fragmentation as much as possible, 6LoWPAN defines an IPv6 header compression mechanism.

2.6.2 6LoWPAN Packet Compression

As previously mentioned, in order to minimize the necessity of fragmentation, 6LoWPAN implements IPv6 and next header compression (IPv6 Extension headers, UDP, etc.). Although RFC 4944 defines a powerful stateless compression mechanism, it can only reach its maximum effectiveness in link-local packet transmissions. This approach is inefficient for most practical cases, where 6LoWPAN devices communicate with devices external to the 6LoWPAN using routable addresses, hence the header compression mechanism defined in RFC 4944 is, at the time of this writing, in the process of being updated by IPHC, a widely accepted new and optimized context-based compression, defined in RFC 6282 [27]. In addition, there are some approaches to transport-layer header compression (still works in progress), such as the specification in I-D.ietf-bormann-6lowpan-ghc [9], which describes mechanisms for generic header compression.

The IPHC compression mechanism described in RFC 6282 permits compressing the 40-byte IPv6 header down to 2 octets for link-local communications and to 3 octets for non-link-local transmissions, which corresponds to compression rates of 95% and 92.5% respectively. Note that these 2 or 3 bytes include the 6LoWPAN dispatch bit field (1 octet), which would be included even if no compression at all is used (in fact, the IPHC compression mechanism makes use of the 5 rightmost bits of the 6LoWPAN dispatch bit field for compression rather than for identification). Figure 2.7 illustrates the structure of the 6LoWPAN IPHC header (as bit fields).

Dispatch — 6LoWPAN Dispatch value for IPHC compression, has a constant value of 011_2.

Figure 2.7: 6LoWPN IPHC base header

TF Traffic Class and Flow Label; this field being 11_2 means that both have a value of 0 and are elided.

NH Specifies whether the next header is carried in-line or compressed.

HLIM Hop-limit. If different than zero, the hop-limit is elided.

CID If set, an 8-bit field containing context identifiers follows this header.

SAC Indicates whether the source address compression is stateless or stateful.

SAM Indicates the source address compression mode.

M If set, destination address is multicast.

DAC Indicates whether the destination address compression is stateless or stateful.

DAM Indicates the destination address compression mode.

As previously mentioned, this compression mechanism relies on shared context, which means that mechanisms to maintain and disseminate such contexts must be provided. Although RFC 6282 does not specify which information a context is composed of, or how maintenance and dissemination of contexts are performed, these features have been further defined in I-D.ietf-6lowpan-nd [43].

2.6.3 6LoWPAN Fragmentation

In order to comply with the IPv6 specification [15], 6LoWPAN provides support for packet fragmentation. When an entire IPv6 datagram does not fit within a single IEEE 802.15.4 MAC frame, such datagrams should be split into fragments. Each of these fragments needs to be encapsulated into a 6LoWPAN packet adding a fragmentation header which specifies an IPv6 packet identifier (so each fragment can be associated with its corresponding IPv6 datagram), an offset, and the total IPv6 packet length. This fragmentation header adds a 4

octet overhead for the first fragment, and 5 for the second and subsequent fragments. Therefore, in order to minimize this overhead, fragmentation should be avoided as mush as possible.

2.7 Address Resolution Protocol

The Address Resolution protocol (ARP) is defined in RFC 826 [37]. It is used for resolution of network layer addresses into link-layer addresses. Despite having been implemented in several combinations of networks and link-layers, the most popular cases are IPv4 over IEEE 802.3 and IPv4 over IEEE 802.11 (WiFi). ARP uses ARP tables and a request-response mechanism in order to perform its task. Its main function is fairly simple:

1. Node A (having link-layer address a and IP address x) wants to send a packet to node B, whose link-layer address is b and IP address is y.
2. A searches its ARP table for B's link-layer address.
3. If found, then the packet can be delivered directly, so A jumps to step 8; if not, then the sender sends an ARP request packet to the link-layer broadcast address, asking "Who has IP address y?"
4. B receives the ARP request (since it is a broadcast packet) and identifies its IP address (y) in the packet's target address field.
5. B stores the pair $<x, a>$ in its ARP table and responds with an ARP response packet to A including its link-layer address (b).
6. A receives B's ARP response and stores the pair $<y, b>$ in its ARP table.
7. Now, both, receiver and sender know each other's link-layer address. A goes back to step 1.
8. A sends its packet to B.

ARP is replaced by the Neighbor Discovery protocol in the IPv6 network layer (see Section 2.9).

2.8 Dynamic Host Configuration Protocol

The Dynamic Host Configuration Protocol (DHCP) is used to provide automatic configuration in IPv4 networks. It was first defined in RFC 1531 [17] as part of the Bootstrap Protocol (BOOTP), and later, redefined as a protocol itself in RFC 2131 [18].

DHCP uses UDP as its transport layer protocol. Its operation involves interchanging 4 packets between the host being configured (running a DHCP client) and the DHCP server: DHCP Discovery, DHCP Offer, DHCP Request, and DHCP acknowledgement. As the outcome of this message exchange, the host can acquire a variable set of network parameters, the most important being its IPv4 address, subnet mask, and router's IPv4 address. This address assignment is for a limited lease time specified in the DHCP acknowledgement. When half the time has elapsed, the host initiates the DHCP renewal process by sending a new DHCP request to renew its lease.

DHCP's successor for IPv6 networks is DHCPv6, specified in RFC 3319 [42]. This can be used as stateful counterpart to RFC 4862 "IPv6 Stateless Address Autoconfiguration" [47] (see Section 2.9.1), either separately or concurrently with it.

2.9 Neighbor Discovery

The following three sections describe the "Neighbor Discovery for IPv6" protocol, defined in RFC 4861 [34] and its modified version, "Neighbor Discovery Optimization for Low-power and Lossy Networks", which, at the time of this writing, is specified by the work-in-progress Internet Draft I-D.ietf-6lowpan-nd [43]. Both protocols, the original and its modified version, are compared in the third section (Section 2.9.3).

2.9.1 Neighbor Discovery for IPv6

The Neighbor Discovery protocol for IPv6 (IPv6-ND) specified in RFC 4861 [34] provides the mechanisms required to accomplish the following tasks: Router Discovery, Prefix Discovery, Parameter Discovery, Address Autoconfiguration (by the means specified in RFC 4862 [47] and RFC 3319 [42]), Address resolution, Next-hop determination, Neighbor Unreachability Detection, Duplicate Address Detection, and Redirect. Section 2.9.1.1 details the messages defined by the IPv6-ND protocol and section 2.9.1.2 briefly describes the protocol's operation.

2.9.1.1 IPv6 Neighbor Discovery messages

In order to achieve its goals, RFC 4861 defines a number of new ICMPv6 [13] messages: Router Solicitation (RS), Router Advertisement (RA), Neighbor Solicitation (NS), Neighbor Advertisement (NA), and Redirect. Table 2.2 depicts these message types. A description of the options noted in this table is given in sections 2.9.1.2 and 2.9.2.2.

RS Routers send Router Advertisement (RA) messages periodically. In order to prompt routers to send a RA immediately, hosts may send a RS message, enabling faster interface initialization.

RA Routers advertise their presence together with a number of network parameters via RA messages. These messages can be sent either periodically or "on demand", as a response to a RS message. The parameters these messages announce include, among others, prefix information, maximum transfer unit, and hop limit.

NS A Node may send NS messages to a neighbor in order to determine its link-layer address (Address Resolution) or to verify its reachability (Neighbor Unreachability Detection). In addition, NS messages are used in the stateless address autoconfiguration process for Duplicate Address Detection.

NA NA messages are sent in response to NS messages. They typically carry link-layer address information in order to complete the address resolution procedure. In addition, a node may multicast unsolicited NA messages if it detects that its link-layer address has changed. This provides a fast (and unreliable) way to spread the new link-layer address to neighboring nodes.

Redirect Since routers are aware of the network's configuration, they can send Redirect messages in order to inform hosts of a better next-hop towards their destination (or to inform that the destination is in fact a neighbor).

2.9.1.2 IPv6-ND Protocol Overview

During bootstrapping, hosts need to (1) discover routers and network information, and (2) configure/autoconfigure their IPv6 interfaces. To accomplish the first, they send up to MAX_RTR_SOLICITATIONS RS messages to the all-routers multicast address [25]. The response from the routers should be a RA carrying the expected information (prefix and network parameters). To achieve the second, a host uses either a manually configured IPv6 address for each interface or generates a link-local IPv6 address as specified in RFC 4862 [47], section 5.3. Global IPv6 addresses are configured as specified in section 5.5 of RFC 4862 using the information received in the Prefix Information Option (PIO) of RAs.

In addition, DAD must be performed for every address prior to assigning this address to an interface. DAD consists of sending up to *DupAddrDetectTransmits* NS messages that carry the address that the node is checking for duplicates in the Target Address field. The IPv6 source

Table 2.2: IPv6-ND message types.

Message type	**Possible options**	**IPv6-ND message purpose**	**Defined in**
Neighbor Solicitation (NS)	SLLAO	Address Resolution, Neighbor Unreachability Detection, Duplicate Address Detection	RFC 4861, section 4.3
Neighbor Advertisement (NA)	TLLAO	Response to NS, New information propagation	RFC 4861, section 4.4
Router Solicitation (RS)	SLLAO	Prompt routers to generate Router Advertisement messages	RFC 4861, section 4.1
Router Advertisement (RA)	SLLAO, MTU, PIO	Prefix and link-parameter dissemination	RFC 4861, section 4.2
Redirect	TLLAO, Redirect Header	Inform a host of a better first-hop node on the path to a destination	RFC 4861, section 4.5

address of the NS is the unspecified address and the destination address is the Solicited-node multicast address [25] of the target. If there is no answer within a certain period of time (*RetransTimer* milliseconds), then depending on the value of *DupAddrDetectTransmits*, another NS is sent or the address is assumed to be unique, i.e., that no other node is using the same IPv6 address. Both constants (*RetransTimer* and *DupAddrDetectTransmits*) are defined in RFC 4861 and RFC 4862 respectively, with default values of 1,000 milliseconds and 1 respectively).

After the node's interfaces are configured, when a node wants to send a packet to a neighbor, it first sends a NS message to the Solicited-node multicast address in order to resolve the target's link-layer address. The link-layer address will be specified in the NA sent in response and thus, direct communication will be possible. In order to ensure that a neighbor is still reachable (or to refresh the information in the Neighbor Cache) a node can also unicast NS messages.

2.9.2 Neighbor Discovery Optimization for LLNs

The Neighbor Discovery Optimization for Low power, Lossy Networks (6LoWPAN-ND) is a set of modifications of IPv6-ND rather than a new protocol. These modifications were introduced in I-D.ietf-6lowpan-nd [43]. The main goal of these modifications is to optimize Neighbor Discovery for power-constrained devices that may utilize non-transitive links. A secondary goal is to provide support for certain features required in 6LoWPAN networks, such as the context propagation feature to enable the use of context-based IPv6 Header Compression [27].

2.9.2.1 6LoWPAN Neighbor Discovery messages

The message types in 6LoWPAN-ND remain the same, with the following exceptions:

- 3 new options are introduced: Address Registration option (ARO), 6LoWPAN Context Information option (6CO), and Authoritative Border Router option (ABRO). ARO is mandatory, while 6CO and ABRO are optional.
- 2 new optional message types are introduced: Duplicate Address Request (DAR) and Duplicate Address Confirmation (DAC).
- Redirect messages are not used in route-over topologies.

Table 2.3 shows the message types and options used in 6LoWPAN-ND.

2.9.2.2 6LoWPAN-ND Protocol Overview

During interface initialization, hosts send up to MAX_RTR_SOLICITATIONS (3 by default) RS messages in order to discover routers. In response, routers send RAs which may include, in addition to other options such as the Prefix Information option (PIO), one or more 6LoWPAN Context options (6COs), and an Authoritative Border Router option (ABRO). Note that both RS and RA must also carry a Source Link-Layer Address option (SLLAO) [43].

If IEEE's 64-bit Extended Unique Identifier (EUI-64) based addresses are used, DAD is not required. Otherwise, DAD is performed via the new Address Registration feature and, optionally, using the new multihop DAD by means of DAR and DAC messages.

The Address Registration feature is performed by sending NS and NA messages carrying the new ARO option, i.e., a host sends a unicast NS with the ARO option to the router(s). The ARO contains the EUI-64 of the sending interface, in order to uniquely identify it, and a registration lifetime.

A router that receives such a NS message, tries to register the address in its Neighbor Cache (NC). If there is no other node using the same IPv6

Table 2.3: 6LoWPAN-ND message types. New message types and options in red.

Message type	Possible options	6LoWPAN-ND message purpose	RFC 4861[34]	I-D.ietf-6lowpan-nd [43]
Neighbor Solicitation (NS)	SLLAO, ARO	Address Registration, Neighbor Unreachability Detection	§4.3	§4.1
Neighbor Advertisement (NA)	TLLAO, ARO	Response to NS, New information propagation	§4.4	§4.1
Router Solicitation (RS)	SLLAO	Prompt routers to generate Router Advertisement messages	§4.1	-
Router Advertisement (RA)	SLLAO, MTU, PIO, 6CO, ABRO	Prefix, context and link-parameter dissemination	§4.2	§4.2; §4.3
Duplicate Address Request (DAR)	-	Perform multihop duplicate address detection	§4.5	§4.4
Duplicate Address Confirmation (DAC)	-	Response to DAR	§4.5	§4.4

address in the NC, then the registration succeeds and the router creates a new Neighbor Cache Entry (NCE) which will remain valid until the lifetime expires. In contrast, if this IPv6 address was in use by another node (with a different EUI-64) or if there is no space in the NC for the new entry, then the registration fails. In both cases a NA containing the same ARO option will be

sent in response, along with the corresponding status value. The status value informs the node trying to register its address of the result of its registration attempt.

If the optional multihop DAD feature is implemented, then this last step may take a bit longer as a 6LoWPAN Router (6LR) that receives a NS including an ARO from an IPv6 source address not in its NC, will send a DAR message to the 6LoWPAN Border Router (6LBR). If the 6LR receives a positive DAC in response to its DAR, it will send back a NA with the corresponding ARO and status value to the node that originated the NS.

Hosts (and 6LRs) need to periodically refresh their NCEs in the routers by re-sending a NS with an ARO specifying a new lifetime. Note that this registration attempt also confirms that the destination routers are still reachable and therefore is used also for Neighbor Unreachability Detection (NUD).

In 6LoWPAN-ND, hosts do **not** perform address resolution. Thus, when a host wants to send a packet, this packet is **always** sent via a router. The router will determine whether the destination node is reachable or not, according to its NC. This means that the router is the only direct neighbor for each host.

2.9.3 IPv6-ND vs. 6LoWPAN-ND

This section describes the differences in the IPv6-ND operations that the 6LoWPAN-ND optimizations introduce and highlights the differences with the greatest impact in terms of compatibility between the two protocols.

2.9.3.1 Differences

The optimizations introduced by I-D.ietf-6lowpan-nd [43] are enumerated below and the differences with respect to IPv6-ND are described in Table 2.4.

1. Host-initiated interactions to allow for sleeping hosts.
2. Elimination of multicast-based address resolution for hosts.
3. A host address registration feature using a new option in unicast Neighbor Solicitation and Neighbor Advertisement messages.
4. A new Neighbor Discovery option to distribute 6LoWPAN header compression context to hosts.
5. Optional multihop distribution of prefix and 6LoWPAN header compression context.
6. Optional multihop duplicate address detection using two new ICMPv6 message types.

Table 2.4: 6LoWPAN-ND optimizations and differences regarding IPv6-ND.

#	6LoWPAN-ND optimization	IPv6-ND behavior	6LoWPAN-ND behavior
1	Host-initiated interactions to allow for sleeping hosts	Routers send periodic RAs and hosts send multicast NS between them	RAs sent mainly in response to RSs; NSs are not sent between hosts
2	Elimination of multicast-based address resolution for hosts	Hosts multicast NSs to perform address resolution	All communication between hosts is via routers
3	New host address registration feature using a new option in unicast NSs and NAs	DAD, NUD, and Address Resolution performed as specified in [34] and [47]	The address registration feature provides support for DAD, NUD, and Address Resolution.
4	New Neighbor Discovery option to distribute context information	Not used	Context information is disseminated by 6CO options in RAs
5	Optional multihop distribution of prefix and context information	Not used	Enhances the distribution of prefix and enables dissemination of context information
6	Optional multihop duplicate address detection	DAD performed as specified in [47]	Provides support for non-EUI-64 based addresses in route-over topologies

2.9.3.2 Incompatibilities

Some of the differences mentioned in section 2.9.3.1 require special attention due to the incompatibilities they introduce. An attempt to internetwork 6LoWPAN and IPv6 networks (each of them running their corresponding ND protocol) without handling these incompatibilities in a proper way would result, at best, in a misuse of the ND protocol while, in most cases, this communication would be impossible.

Of these differences, #1, #2, and #3 are the most significant. In IPv6-ND address resolution is performed by (multicasting) NS and NA messages. In contrast, address resolution is not performed by 6LoWPAN hosts (although routers may do address resolution). 6LoWPAN hosts do not even join the

Solicited-node multicast address; instead, they register with routers by means of the address registration feature (#3). When a host needs to send a packet, the packet is delivered via a 6LoWPAN Router (6R) that is aware of the target's link-layer address due to this node's current registration.

Moreover, non-constrained devices (NCDs) trying to deliver a packet to a neighboring 6LoWPAN host would first multicast a NS message to the destination's Solicited-Node multicast address for Address Resolution. As 6LoWPAN hosts do not join the Solicited-Node multicast group, they will not be listening to this address, hence the NS will not be answered, therefore the NCD would assume that the destination is unreachable.

Furthermore, DAD is also performed in IPv6-ND by sending a Neighbor Solicitation message to the Solicited-node multicast address (as described in Section 2.9.1). However, no host in the 6LoWPAN network will respond to a message sent to this multicast address and thus, a NCD would never detect whether another 6LoWPAN host has the same IPv6 address it is trying to use.

In contrast, I-D.ietf-6lowpan-nd states that either DAD is not needed (if EUI-64 based IPv6 addresses are used) or multihop DAD is used (if there are non-EUI-64 based addresses in a route-over topology). Note that this assumes that the EUI-64 based IPv6 address will be unique. Section 3.2 examines some of the other assumptions that I-D.ietf-6lowpan-nd makes.

2.10 What have others already done?

This section describes a number of related works. This related work is divided into to neighbor discovery proxies and background material about Contiki (as this open source software will be used in the developments reported in this thesis).

2.10.1 Neighbor Discovery proxies

The concept of Neighbor Discovery proxies is not original; previous work has described some specifications of ND proxies. The proxy operations described in this thesis were highly influenced by two specific documents: "Neighbor Discovery Proxies" (RFC 4389) [46] and "6LoWPAN Backbone Router" (draft-thubert-6lowpan-backbone-router) [48].

RFC 4389 describes the required proxy operations for some special cases when bridging different types of media requires network-layer support. Although our case also involves bridging two different types of media and such bridging requires network-layer support (mainly link-layer address translation in ND messages), the reasons that make the use of a proxy necessary in our scenario are significantly different that those described in RFC 4389. Specifically, the use of a proxy is necessary in our case because each interface of the 6LP-GW is attached to a network segment that utilizes a different

ND protocol and, more importantly, these protocols are incompatible. In contrast, the Neighbor Discovery Proxies specified in RFC 4389 considers the same Neighbor Discovery protocol on all interfaces.

The Internet Draft "6LoWPAN Backbone Router" describes a situation very similar to ours. In particular, the scenario depicted in this internet draft consists of several small 6LoWPAN LLNs connected to a transit link (Ethernet) by means of a backbone router per LLN. Such backbone routers perform Neighbor Discovery proxying between the 6LoWPAN networks and the Ethernet transit link. Figure 2.8 illustrates the scenario presented in draft-thubert-6lowpan-backbone-router.

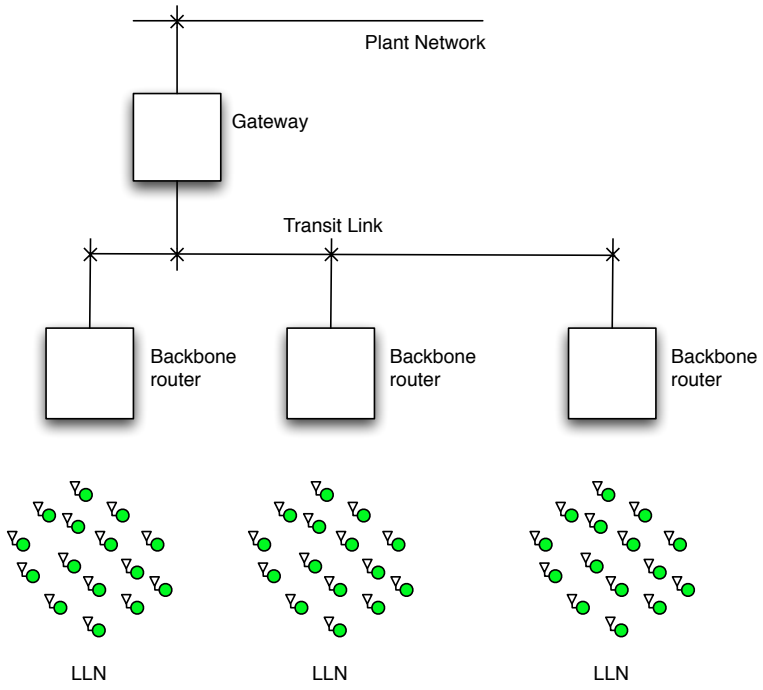

Figure 2.8: Backbone Routers scenario

While some of the proxy mechanisms described in "6LoWPAN Backbone Router" are similar to the ones described in this thesis (see Section 3.2), others differ significantly or are out of the scope of one or the other document.

The most important difference between the 6LoWPAN backbone router and the 6LP-GW is that the former performs network-layer routing between its two interfaces, whilst the latter neither routes packets nor performs internet-layer forwarding, but rather operates as a bridge (although it requires network-layer support in certain situations). Moreover, while a 6LoWPAN backbone router defines an IPv6 subnet, the 6LP-GW does not, but rather it is the port of the IPv6 router (which the 6LP-GW connects to) that defines the subnet.

Figure 2.9 illustrates this difference.

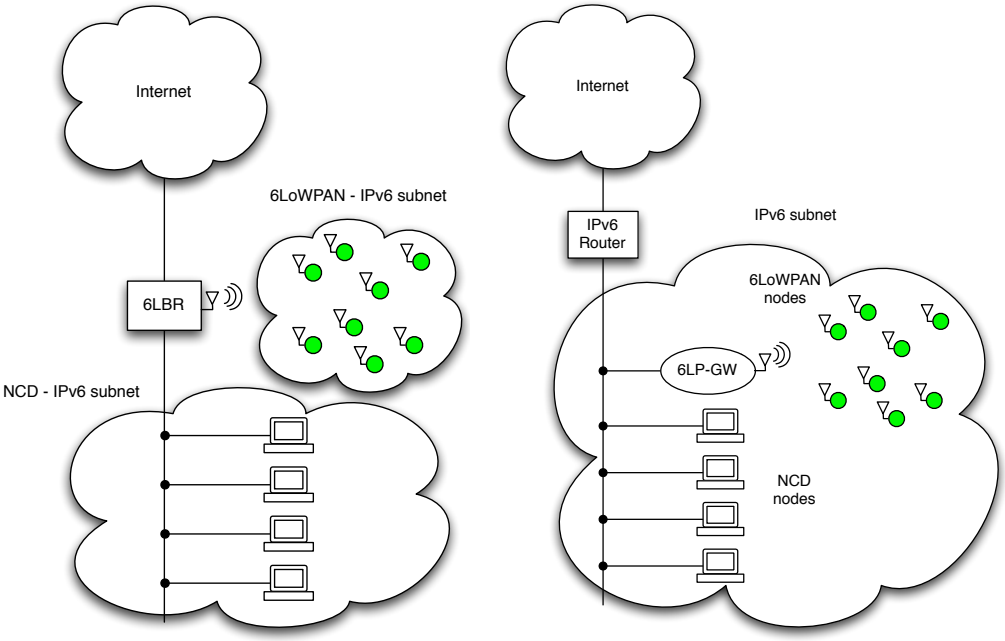

Figure 2.9: 6LBR vs. 6LP-GW. The figure illustrates the differences in the topology derived from the use of a 6LBR (left) or a 6LP-GW (right). Note that the use of a 6LBR necessarily causes the creation of a new IPv6 subnet, while all the nodes in the network in which the 6LP-GW operate share the same network prefix.

Furthermore, 6LoWPAN backbone routers are required to operate as a "distributed database of all the LLN nodes", while the 6LP-GW maintains a strictly local database of the LLNs within the 6LoWPAN it is connected to. The reason for this is that there is no requirement that the 6LP-GW acts as a mobility anchor.

These differences have their roots in the different problems each device addresses; while a "6LoWPAN Backbone Router" aims to provide scalable support for mobility of 6LNs between LLNs without requiring them to register with each LLN's 6LBR, the 6LP-GW proposed in this thesis aims to integrate 6LoWPAN LLNs **into** existing IPv6 infrastructures without requiring a single change in the existing network infrastructure. This allows the end-user to literally "attach" a 6LoWPAN LLN to her/his existing IPv6 router. In contrast the "6LoWPAN Backbone Router" approach cannot be used in some settings, for example most home IPv6 routers preclude intra-LAN routing.

In conclusion, while the solution proposed in "6LoWPAN Backbone Router" targets mobility and scalability in large installations (such as in office

buildings and/or industrial plants) without considering the problem of home routers, the implementation proposed here (using the 6LP-GW) aims for rapid deployment of 6LoWPAN networks at low cost, which involves allowing re-utilization of existing home routers, but leaves mobility issues out of scope. Section 6.2, however, provides guidelines for enabling the same mobility support as provided by draft-thubert-6lowpan-backbone-router while still not requiring intra-LAN routing (and hence allowing the use of home routers). It should also be noted that 6LP-GWs can also be attached to multiport Ethernet switches, enabling their use in settings with such switches. A Power over Ethernet version of the Ethernet interface of the current hardware prototype platform in which the 6LP-GW is implemented (Hogaza board v1.2 [50]) is being designed in a companion Bachelor's thesis project [22].

2.10.2 The Contiki Operating System

Contiki is an open source lightweight platform-independent operating system for embedded platforms written in the C programming language. Although it lacks certain features expected to be present in any operating system (for example, it does not provide hardware management functions), it provides a powerful and richly-featured framework for embedded system development.

Among its main features, Contiki provides a memory-efficient abstraction mechanism for multitasking-like process development called *protothreads* [21] controlled by a simple event-driven kernel, different libraries for memory allocation and management, message-passing-based interprocess communication, and fully-compliant and lightweight IPv4 and IPv6 communication stacks, including a 6LoWPAN implementation.

2.10.2.1 Contiki's protothreads

Protothreads [21] are a programming abstraction that provides a function-level conditional blocking wait statement: `PT_WAIT_UNTIL()`. Conceptually, `PT_WAIT_UNTIL()` blocks the executing function —protothread— until a condition (passed as parameter) evaluates to true. This relatively simple abstraction allows programmers to structure their entire application as a set of independent processes, rather than a large monolithic one, thus improving scalability, maintainability, and manageability of the code. Protothreads are intended to simplify event driven programming, which usually consist of a state machine. Such a state machine is traditionally implemented by a large infinite loop with a conditional switch statement inside it. Figures 2.10 and 2.11 (on pages 31 and 32 respectively) illustrate two different implementations of state machines, comparing the traditional loop-switch and protothreads approaches.

Protothreads are built on top of an underlying mechanism called local continuations. Local continuations can be seen simply as a means to store

the state of a protothread. Local continuations can perform two operations: *set* and *resume*. A *set* operation stores the current position at which it is invoked, whereas a *resume* operation causes the program to *jump* to the point previously stored in the local continuation (the point at which the *set* operation was invoked).

Hence, we can consider protothreads as C functions which utilize local continuations to alter their normal execution flow. When the execution reaches a `PT_WAIT_UNTIL()` statement, a *set* operation is performed. If the condition passed as parameter of the `PT_WAIT_UNTIL()` is met, then the function continues executing; otherwise, the control is returned to the caller function (i.e., the program *jumps* to the end of the function). The return value which is produced in this case, informs the caller function that the protothread has not finished, but rather that it is waiting for something. The next time the protothread is invoked, a *resume* operation is performed, causing the program to *jump* to the previously stored position and to re-evaluate *condition*. Note that the use of protothreads requires the presence of a scheduler function, which can be as simple as an infinite loop in which all the *protothreaded* functions are called sequentially. In addition to the `PT_WAIT_UNTIL()` statement, [21] also defines other useful local-continuation-based mechanisms, such as the `PT_YIELD()`. The `PT_YIELD()` statement performs a single unconditional blocking wait, which causes the protothread to unconditionally return the control to the caller. The next time the protothread is executed, it will continue its execution from the point where `PT_YIELD()` was invoked.

Regarding the C implementation, protothreads are implemented by means of macros that expand to a set of instructions which perform the operations of local continuations (*set* and *resume*) and, optionally, evaluate conditions [21]. Local continuations can be implemented in two different ways: using the *Labels as Values* GCC compiler extension (which allows storing labels in variables) together with `goto` statements, or relying on standard C switch statements together with the standard `__LINE__` macro (which expands to the number of the line at which `__LINE__` is used). The former leads to slightly better results in terms of code size and speed, but it depends on the availability of the GCC compiler for a certain architecture, while the latter can be utilized together with any standard C compiler.

2.10.2.2 Contiki's kernel

In order to explain how the Contiki kernel works, we introduce the Contiki *process* concept. Contiki's protothreads are wrapped within the *process* structure (C `struct`). This *process* structure stores a protothread's context information, such as the *process* name, a pointer to the *protothreaded* function, the protothread itself (containing its local continuation), the *process* state, and

```
state : {GREEN, AMBER, RED}

void semaphore() {
 state = GREEN  // set initial state
 light(GREEN)
 timer_set(GREEN_TIMER)

 while (1) {
  switch(state) {
  case (GREEN):
   if (timer_expired(GREEN_TIMER) ||
     pedestrian_button_pressed()) {
    state = AMBER
    light(AMBER)
    timer_set(AMBER_TIMER)
   }
  case (AMBER):
   if (timer_expired(AMBER_TIMER)) {
    state = RED
    light(RED)
    timer_set(RED_TIMER)
   }
  case (RED):
   if (timer_expired(RED_TIMER)) {
    state = GREEN
    light(GREEN)
    timer_set(GREEN_TIMER)
   }
  }
 }
}
```

State machine implemented using a traditional loop-switch mechanism.

```
pt_semaphore:
PT_BEGIN
 while (1) {
  light(GREEN)
  timer_set(GREEN_TIMER)
  PT_WAIT_UNTIL(timer_expired(GREEN_TIMER) ||
    pedestrian_button_pressed())

  light(AMBER)
  timer_set(AMBER_TIMER)
  PT_WAIT_UNTIL(timer_expired(AMBER_TIMER))

  light(RED)
  timer_set(RED_TIMER)
  PT_WAIT_UNTIL(timer_expired(RED_TIMER))
 }
PT_END
```

State machine implemented using the protothreads abstraction mechanism.

Figure 2.10: Example of the same state machine implemented using traditional loop-switch statements (left) and Contiki protothreads (right), both in pseudocode. This state machine represents the operation that controls the state transitions in a hypothetical street semaphore provided with a button that can be pressed by pedestrians to trigger a transition from green-light state to amber first and, eventually red-light state (for the cars that are flowing perpendicular to the pedestrian.

a boolean variable which indicates whether the *process* has requested a poll from the kernel.

This kernel is event-driven, which means that the *process* to be called next is chosen depending on whether a *process* has any pending event. This approach implies that, a *process* whose *protothread* has performed a `PT_WAIT_UNTIL(condition)` or `PT_YIELD()` statement, will not continue its execution unless another *process* posts an event addressed to it or if it has **actively** requested to be polled by the kernel **before** executing the blocking statement. This approach does not take advantage of the variety of returning values implemented by the protothreads mechanism [21] and requires the programmer to ensure that no *process* will enter a permanent sleep state, but it ensures that *processes* are not unnecessarily checked repeatedly if they

```
state : {ON, OFF}

main loop:
 state = OFF  // set initial state

 while (1) {
  switch(state) {
  case (OFF):
   if (button_pressed()) {
    timer_set(LIGHT_TIMER)
    light_on()
    state = ON
   }
  case (ON):
   if (timer_expired(LIGHT_TIMER) ||
     button_pressed()) {
    light_off()
    state = OFF
   }
  }
 }
```

State machine implemented using a traditional loop-switch mechanism.

```
button_pt:
PT_BEGIN
 while (1) {
  PT_WAIT_UNTIL(button_pressed())
  post_event(light_pt, BUTTON_PRESSED)
 }
PT_END

timer_pt:
PT_BEGIN
 while (1) {
  PT_WAIT_UNTIL(event == TIMER_START)
  timer_set(LIGHT_TIMER)
  PT_WAIT_UNTIL(timer_expired(LIGHT_TIMER))
  post_event(LIGHT_TIMER_EXPIRED)
 }
PT_END

light_pt:
PT_BEGIN
 light_off()
 while (1) {
  PT_WAIT_UNTIL(event == BUTTON_PRESSED)
  light_on()
  post_event(timer_pt, TIMER_START)
  PT_WAIT_UNTIL(event == BUTTON_PRESSED ||
    event == LIGHT_TIMER_EXPIRED))
  light_off()
 }
PT_END
```

State machine implemented using the protothreads abstraction mechanism. This approach implements 3 different protothreads, each of them taking care of a different task.

Figure 2.11: Example of the same state machine implemented using traditional loop-switch statements (left) and Contiki protothreads (right), both in pseudocode. This state machine represents an light switch that remains turned off until a button is pressed. When the button is pressed, it remains turned on until a timer expires or the button is pressed again. In this case, the version utilizing protothreads may seem *a priori* more complex than the traditional approach. However, it clearly separates each functional block into a different protothread, which improves modularity and scalability of the application and hence, simplifies the addition of new states/events. it is important to note that in this case there are only 2 events and 2 states, but the complexity of the code would increase exponentially while the number of states/events increases linearly (the exponential increase occurs because the number of different cases can potentially be equal to the Cartesian product of events and states).

have nothing to do.

Additionally, Contiki provides a rich set of timer libraries, which in spite of not being strictly part of the system kernel, play an important role in most applications. Among these libraries, the *Event timer library* module is

tightly coupled to the kernel itself. This library provides an especially powerful resource which enables *processes* to set timers. However, this module has the particularity that it initializes a running *process* itself, the *etimer* process, which will post an event to the *process* that set the timer when this specific timer expires. This mechanism provides a useful and safe way for *processes* to set a timer and yield the control to other *processes*, with the certainty that they will continue their execution after the timer expires.

Chapter 3

Method

This chapter, constitutes one of the main contributions of this thesis. It includes the specification of the proxy operations required to integrate 6LoPWAN links into IPv6 networks.

Regarding this operational specification, the 6LP-GW's behaviour is detailed in a per-message-type basis: this describes the actions that will be triggered in the 6LP-GW for each of the different ND messages that may arrive at each of its two interfaces, providing both a comprehensive explanation and the justification for the specific behaviour that is proposed.

3.1 Assumptions and application scenario

The following two sections describe both the assumptions upon which the specification and further implementation of the proxy operations described here are based, and the application scenario of this thesis project.

3.1.1 Assumptions

I-D.ietf-6lowpan-nd states that the Neighbor Discovery protocol optimizations that it introduces are compatible with both Mesh-under and Route-over topologies. Although the proxy operation described here does not affect this compatibility, the remainder of this report will assume a route-over topology.

The link-layer scenario considered here takes account only of IEEE 802.15.4 and IEEE 802.3 links. Nevertheless, the techniques described in this document may also be applicable to other types of media as long as IPv6-ND is used in one segment while 6LoWPAN-ND is used in the other, and may even be used for interconnecting more than two link-layer media, but these considerations are out of the scope of the present document. Thus for the remainder of this document we will explicitly only consider a two port device, with one interface being IEEE 802.3 and the other interface being IEEE 802.15.4.

The terms "MAC address" or "link-layer address" will be used indiscriminately in the present document, referring to both 64-bit (IEEE 802.15.4) and 48-bit (IEEE 802.3) MAC addresses (making no distinction between them) since there is an IEEE defined direct mapping from 48 bit MAC addresses to 64 bit addresses [6].

Due to the nature of the 6LoWPAN Proxy-Gateway (6LP-GW), the availability of a forwarding mechanism between the two interfaces is required. While the choice of this forwarding mechanism need **not** impose a specific network topology (Route-over or Mesh-under) as long as it is applied properly,

a layer-3 forwarding mechanisms would increase both implementation and network complexity. Therefore, the procedures described in this document make use of a layer-2 forwarding mechanism in order not to add any complexity regarding issues which are of low interest for the purposes of this thesis project.

6LoWPAN supports different compression mechanisms which may be used or not. If compression is used, the 6LP-GW will be responsible for compressing and decompressing packets as required. In order to maximize the utility (thus minimizing power consumption) of both 6LoWPAN and 6LoWPAN-ND, the compression mechanism implemented in the 6LP-GW is that described in RFC 6282 [27].

6LoWPAN fragmentation and reassembly are currently **not** supported by this implementation. The reasons for this are the limited scope of this thesis project (with its focus on sensor nodes) and the unbalanced trade-off between usefulness and increased complexity in terms of processing and code size that would be derived from its implementation. However, if a commercial application were to implement the procedures described here, this 6LoWPAN feature should be included in order to comply with the IPv6 standard's requirements. Section 6.2 describes how this feature should be implemented.

Some of the new features 6LoWPAN-ND introduces are defined in I-D.ietf-6lowpan-nd as optional. For the same reasons as explained above, the implementation described here treats only the 6LoWPAN Context Option feature, considered of high interest for experimental and practical purposes. The treatment of the remaining optional features (ABRO option, multihop DAD, etc.) is out of scope of this thesis project.

The 6LP-GW is required to keep track of the IPv6 and MAC address(es) of the RR(s) in order to properly forward packets directed to the RR(s) or generate packets on behalf of the RR(s). Regarding the number of RRs, this document assumes that there is only one RR in the network: how to carry out RR-address management for networks with more than one RR is outside the scope of this thesis. As for the number of addresses of the RR, only its link-layer address will be used for the purposes of the proxy operations since, as specified in RFC 4861, that is the only valid address from which RA messages can be sent.

6LoWPAN nodes may form their IPv6 addresses using their IEEE 802.15.4 long (64 bits) or short addresses (16 bits). When IPv6 addresses derived from IEEE 802.15.4 short addresses are present in the network, 6LoWPAN-ND mandates performing duplicate address detection. For practical purposes, we assume that only EUI-64-derived IPv6 addresses will be present in the 6LoWPAN network.

3.1.2 Application scenario

The 6LP-GW is intended to sit between an IPv6 router (referred to in the remainder of this document as RR) and a 6LoWPAN network. The

6LP-GW extends the RR's functionality by adding an IEEE 802.15.4 interface in addition to the RR's existing interfaces (typically IEEE 803.3 and IEEE 802.11). This added functionality conceptually turns the RR into a 6LBR. This 6LP-GW enables 6LoWPAN devices to be part of the same network as any other non-constrained device, without requiring any further special treatment. Therefore the 6LP-GW must perform all the operations necessary to enable ordinary IPv6 hosts to communicate with 6LoWPAN hosts and *vice versa*. Figure 3.1 illustrates the 6LP-GW in this application scenario.

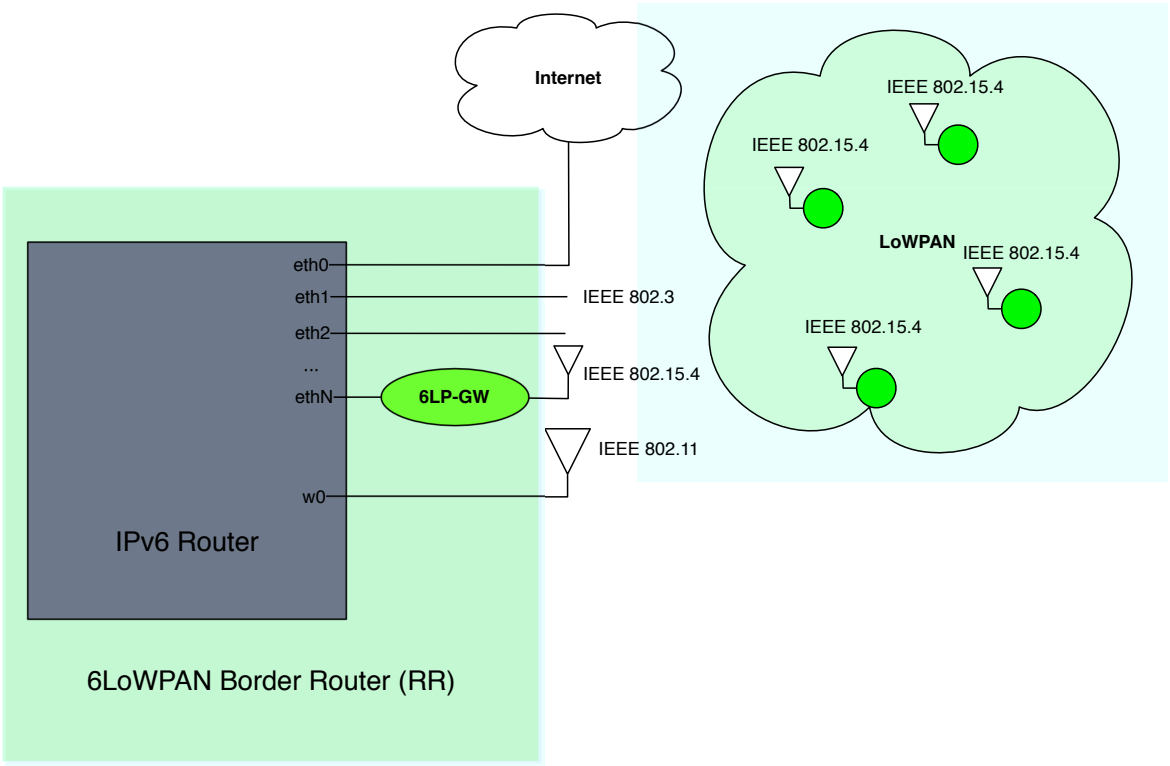

Figure 3.1: The 6LP-GW internetworks the LoWPAN and an IPv6 router.

3.2 Proxy Operation Specification

Due to the significant innovation of the approach proposed in this document concerning Neighbor Discovery proxy operation between two **different** Neighbor Discovery protocols, and under-IP internetworking of 6LoWPAN networks, an Internet Draft (I-D.ietf-maqueda-6lowpan-pgw [31]) describing these procedures was written and submitted to the IETF. However, this thesis provides a full description of such procedures for completeness.

3.2.1 Proxy Operation overview

The 6LP-GW together with the RR will be seen from the 6LoWPAN side as a 6LBR. In contrast, from the IPv6 side, it is impossible to distinguish between NCDs and 6LoWPAN nodes. Regarding forwarding, the 6LP-GW has to both keep track of which node is in each segment and apply the required link-layer address translation between IEEE 802.15.4 and Ethernet addresses. Note that the 6LP-GW is invisible for any of the link segments it internetworks, meaning that it does not need to have either an IPv6 address or a MAC address for the purpose of acting as a proxy or gateway. However, the device considered here has both IPv6 and MAC addresses associated with its Ethernet interface. These addresses could be used for management and/or monitoring of the device (including loading new software into it, node association, etc.) or the network (number of nodes, traffic, etc.), but this functionality lies outside the scope of this thesis.

In addition, some ND options carry link-layer addresses (mainly SLLAO and TLLAO). Packets containing these option require extra processing in order to translate from 48-bit MAC addresses into 64-bit MAC addresses and vice-versa, depending on which segment these packets originate from.

Moreover, 6LoWPAN features such as decompression and compression are performed by the 6LP-GW on incoming and outgoing packets when appropriate. Therefore, to enable the operations described here, every packet reaching the 6LP-GW, regardless of the segment where it originated, shall be processed as required, applying the corresponding compression/decompression operations.

Note also that the proxy mechanisms described below consider only the case of packets traversing from one segment to the other. The 6LP-GW does not forward unicast packets directed to the same segment they came from.

In all cases, validity checks of the incoming ND messages will be performed as specified in the corresponding ND specification. The specific way to process a ND message will depend on which segment it originates from and will be explain in later sections.

3.2.2 Conceptual datastructures and Initialization

In addition to the data structures required for the forwarding mechanism, the 6LP-GW needs to maintain a Neighbor Cache (NC) just as if it were a 6LR (or 6LBR). The maintenance procedures for this cache extend those described in I-D.ietf-6lowpan-nd [43]. This means that the 6LP-GW has to create/refresh entries when receiving Neighbor Solicitation messages (NS) and it must also remove Neighbor Cache Entries (NCEs) when their registration lifetime expires. Receiving an ARO with zero lifetime will cause the 6LP-GW to immediately delete the corresponding NCE.

In addition to the information expected to be contained in every NCE, this specification requires the inclusion of the following elements: an ARO-pending flag, an awaiting-RA flag, and a Duplicate Address Detection (DAD) timer. The meanings of these variables will be explained later in this section. Furthermore, as context-based header compression is used, the 6LP-GW also needs to perform context information maintenance and dissemination just as if it were a 6LBR. At bootstrapping, the 6LP-GW initializes all the data structures needed to create and maintain both a NC and the Context information table.

3.2.3 Packet Forwarding

The 6LP-GW's main purpose is to **forward** IPv6 packets originated in one segment to the other segment. Since its operation is transparent from both segments' standpoint, the 6LP-GW has to promiscuously listen to the physical media. Therefore, it needs a suitable forwarding mechanism in order to determine whether a packet's destination lies on a different segment than the one where it originated (and, hence, must be forwarded) or its destination is on the same segment it was sent from (so the 6LP-GW simply drops the packet).

The mechanism chosen for this operation is a very simple approach to bridging [1]: The 6LP-GW maintains a bridge table in which each entry is a pair <MAC address, interface>. For every incoming packet, it checks whether the source MAC address is stored in the bridge table; if it is not, the pair <source MAC address, incoming interface> is added to the bridge table. Next, it checks the destination MAC address: if it is a multicast address, or a unicast address not having a matching entry in the bridge table, it is forwarded to every interface, but the incoming interface; if it is a unicast address and there is a corresponding entry in the bridge table, then the packet is forwarded to the interface associated with it. For maintenance of this table, a least recently used (LRU) policy is applied in order to replace old entries by new ones when the table is full. This behaviour is illustrated in Figure 3.2.

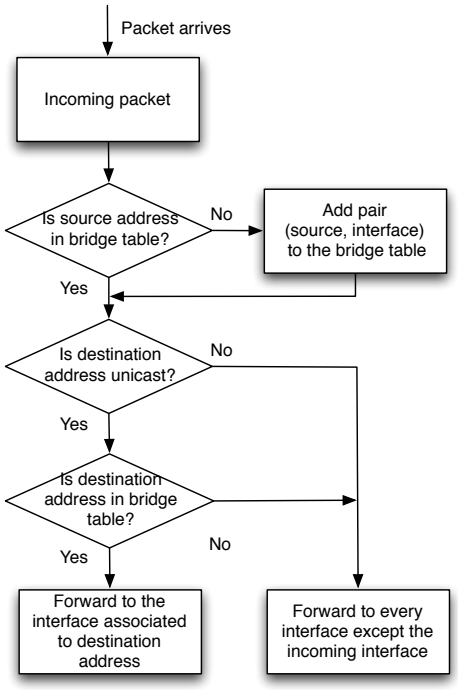

Figure 3.2: The 6LP-GW Forwarding mechanism

In addition, an appropriate MAC translation mechanism has to be applied when required, since IEEE 802.15.4 MAC addresses are 64 bits long while IEEE 802.3 MAC addresses are 48 bits long. The IEEE standard document "Guidelines for 64-bit Global Identifier (EUI-64™) Registration Authority" [6] defines a simple and convenient EUI-48 to EUI-64 mapping which perfectly addresses this application's needs. This mechanism consists of inserting the constant value $0xFFFE_{16}$ between the company identifier (i.e., the 3 left-most bytes of the EUI-48) and the manufacturer-selected extension identifier (i.e., the 3 right most bytes of the EUI-48). Figure 3.3 illustrates how an EUI-48 is encapsulated into an EUI-64 MAC address.

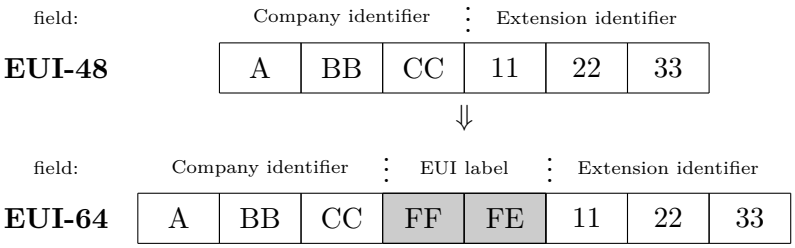

Figure 3.3: EUI-48 Encapsulated in EUI-64

As for the particular case of the 6LP-GW, every MAC address is considered to be 64-bits long. Ethernet MAC addresses are converted into their corresponding 64-bit MAC address as soon as an Ethernet packet arrives at the 6LP-GW and, only in the very final step of sending a packet out from the 6LP-GW, it is checked whether the segment where the packet is to be sent operates with 48 or 64-bit addresses. On the other hand, all the IEEE 802.15.4 MAC addresses used in our experimental deployment have their fifth and fourth least significant bytes being 0xFFFE. Although this approach simplifies the link-layer address translation procedure, it must be noted that the IEEE registration authority forbids this practice (64-bit values of the form ccccccFFFEeeeeee are never assigned). An approach suitable for a commercial product could use a simple table in order to keep track of the IEEE 802.15.4 addresses in the network and remove/insert the fifth and fourth least significant bytes of addresses in the outgoing/incoming packets directed to or arriving from the Ethernet segment. However, this and other approaches are outside the scope of this thesis.

Applying the link-layer address translation mechanism described above means that further processing (bridging, proxy operation, host operation, etc.) does not require taking into account the link-layer address length nor its origin; all addresses are 64-bit long and all of them have their fifth and fourth least significant bytes being 0xFFFE.

3.2.4 Proxy operation

This section describes in detail the ND-proxy's conceptual operation performed by the 6LP-GW. All the operations described in this section are applied only to ND packets arriving at the 6LP-GW; non-ND packets will be forwarded as described in Section 3.2.3.

3.2.4.1 Processing Neighbor Solicitation Messages

The Neighbor Solicitation messages that reach the 6LP-GW may have been originated for different purposes. The appropriate way to process them depends on this purpose and it will differ depending on their origin and their structure. Figure 3.4 shows the different types of Neighbor Solicitation messages that may arrive at the 6LP-GW.

Neighbor Solicitation originating in IEEE 802.15.4 segment

As Figure 3.4 shows, we distinguish three different types of possible NS messages that can arrive from the IEEE 802.15.4 segment: multicast NS and unicast NS (with/without ARO).

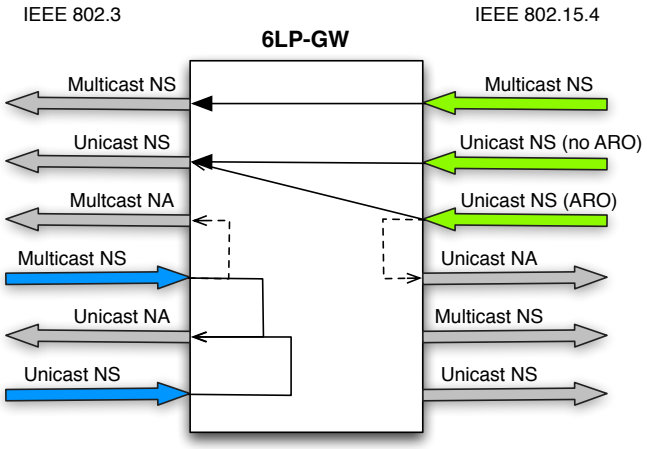

Figure 3.4: NS message processing. Incoming NS messages are connected by arrows to the Neighbor Discovery messages that may be generated in response. The dashed arrows represent conditional responses.

Multicast NS

A multicast NS originated in the IEEE 802.15.4 can only have the purpose of performing address resolution. As per Section 2.9.2 6LoWPAN hosts (6Hs) do not perform address resolution, but 6LoWPAN Routers (6Rs) may optionally do so. However, these multicast NS messages have the sole purpose of discovering the link-layer address of other 6Rs. As no 6R is present in the IEEE 802.3 segment apart from the one that is formed by the RR together with the 6LP-GW (as previously said, they are together seen from the IEEE 802.15.4 segment as a 6LBR), hence the 6LP-GW will proceed as follows: upon reception of a multicast NS message originating in the IEEE 802.15.4 segment, the 6LP-GW will examime its target address; if it matches the RR's IPv6 address, the packet will be forwarded unchanged (apart from the appropriate MAC translation); otherwise the packet will be discarded.

Unicast NS not containing an ARO option

As defined in RFC 4861, unicast NS without ARO messages are sent as probes to test for reachability. 6Hs do not maintain Neighbor Cache Entries (NCEs) for other hosts, but only for 6Rs. Therefore it is unlikely that any 6H sends a unicast NS to any node other than a 6R. However, we should keep in mind that the 6LP-GW together with the RR will be seen from the 6LoWPAN link as a 6LBR and thus, this 6LBR has to respond to such NS messages. Regarding 6Rs, nothing in I-D.ietf-6lowpan-nd precludes 6R's from sending this type of message to any other node in the network. Therefore, a unicast NS message

not containing an ARO option must be forwarded to the IEEE 802.3 interface unchanged (apart from the appropriate MAC translation).

Unicast NS containing an ARO option

Unicast NS messages containing an ARO option are sent fot two purposes: (1) as part of the 6LoWPAN-ND registration procedure and (2) to perform NUD (to determine the reachability of the router to which they are sent). As these messages are only sent to 6Rs, and the 6LP-GW together with the RR is seen as a 6LBR, it is likely that the 6LP-GW will receive such messages having as their destination IPv6 address the RR's IPv6 address. Therefore, the 6LP-GW performs the normal operations of a 6R when receiving this type of message directed to the RR, i.e., the NS message must be processed as specified in section 6.5 of I-D.ietf-6lowpan-nd in terms of validity and the NC maintenance procedure, but with some differences as will be explained below. Should the IPv6 destination address of a NS message including an ARO **not** match the RR's IPv6 address, then the packet will be discarded. If, for some reason, the RR's IPv6 address is unknown when the NS arrives, then the packet will also be discarded.

On the other hand, RFC 4861 requires every node in the IPv6 network to perform duplicate address detection (DAD). Therefore, performing DAD on behalf of 6LoWPAN nodes that are to be integrated into the IPv6 link is necessary in order to comply with the specification. On the other hand, 6LoWPAN-ND only requires performing DAD when *non-EUI-64-based* IPv6 addresses are being used in the network. As previously stated in Section 3.1.1, the 6LoWPAN nodes present in our scenario will *only* make use of EUI-64-based IPv6 addresses (either "real" EUI-64 or EUI-48 encapsulated into EUI-64), hence nodes in the IEEE 802.3 segment need **not** perform DAD in the IEEE 802.15.4 segment.

For the above reasons, the 6LP-GW must perform not only the registration procedure, but also DAD (in the IPv6-ND way on the IEEE 802.3 interface) and NUD when receiving a unicast NS with an ARO option. Both operations (DAD and NUD) are performed on behalf of the 6LoWPAN node that is trying to register its address.

In order to perform DAD, the 6LP-GW must send a NS, formatted as explained in Section 2.9.1.2, to the Solicited-node multicast address corresponding to the source address of the incoming NS. For NUD, the NS message originated in the IEEE 802.15.4 segment should be forwarded to the IEEE 802.3 segment so a subsequent NA response will confirm the reachability of the router.

Unfortunately, DAD is an expensive process as it takes a long time to wait for messages that are not going to receive responses [51] and it can not be performed in parallel with NUD due to the risk of duplicate addresses. As waiting for both to complete sequentially may delay the autoconfiguration

process excessively, we choose to perform DAD only upon registration and then, NUD upon re-registration. Figure 3.5 describes the complete address registration procedure and Section 3.2.4.1 details how DAD is performed.

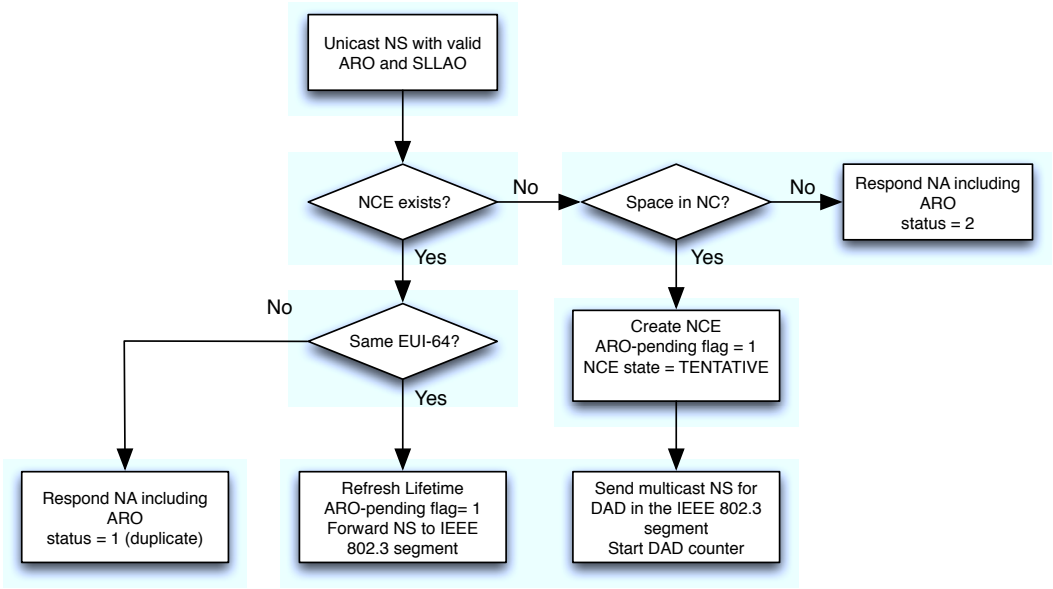

Figure 3.5: NS with ARO processing diagram.

Considering all of the above, upon receipt of a valid NS message destined for the RR and containing valid ARO and SLLAO options, the 6LP-GW shall behave as described below (see Figure 3.5).

The 6LP-GW searches its NC for a NCE with same IPv6 address as the IPv6 source address of the incoming NS message; if no matching NCE is found, then the 6LP-GW creates a new NCE for the node being registered. If there is no space left in the NC, then the registration fails and the 6LP-GW generates a NA including an ARO with status = 2, as specified in section 6.5.2 of I-D.ietf-6lowpan-nd [43]. If there is space available in the NC, then a new entry is created with a state value of TENTATIVE and its ARO-pending flag is set to 1. In this final case a NA is **not** generated in response, but rather the 6LP-GW performs Duplicate Address Detection (DAD) on the IEEE 802.3 segment *on behalf* of the node that is issuing its registration. This procedure is performed similar to the procedure described in section 5.4 of RFC 4862. The DAD process is detailed in Section 3.2.4.1.

If there is a matching NCE whose EUI-64 value differs from the EUI-64 present in the ARO, then the address is a duplicate and the 6LP-GW must generate and send a NA message including an ARO with status = 1 (duplicate), as specified in section 6.5.2 of I-D.ietf-6lowpan-nd. If, instead, the EUI-64 is the same as present in the ARO, then this is the case of a

re-registration and therefore, the ARO-pending flag must be set to 1, the registration lifetime must be refreshed with the contents of the ARO option, and the received NS message is forwarded to the IEEE 802.3 segment in order to perform NUD (note that the NA message produced in response will need to be intercepted later as the RR is not able to handle ARO options – see Section 3.2.4.2).

Note that in certain situations of the above procedure, the 6LP-GW responds to NSs on behalf of the RR. Therefore, for every such packet being generated in the 6LP-GW on behalf of the RR, the Router flag must be 1 and the IPv6 source address must be the IPv6 address of the RR *attached* to the 6LP-GW. This address already should be known due to the previous RS and RA exchange.

It is also important to note that TENTATIVE entries should be timed out TENTATIVE_NCE_LIFETIME seconds after their creation in order to leave space in the NC for other hosts, as specified in I-D.ietf-6lowpan-nd [43].

Performing DAD on behalf of IEEE 802.15.4 nodes

DAD is performed as specified in RFC 4862 and we assume the existence of the variables RetransTimer and DupAddrDetectTransmits, defined in RFC 4861 and RFC 4862 respectively.

The 6LP-GW must maintain a DAD timer for each NCE in the NC. A DAD timer will be started when the corresponding NS is sent. If no NA is received in response after *RetransTimer* milliseconds, then the 6LP-PGW will either send another NS or end the DAD process, depending on the value of *DupAddrDetectTransmits*.

If the DAD process completes successfully, then the 6LP-GW changes the state of the corresponding NCE to REGISTERED, and the ARO-pending flag to 0. In addition, the information contained in the NCE is used to generate and send a NA message including an ARO option with status = 0 (success) to the node that originated the registration.

If DAD fails, then a similar NA including an ARO option with status = 1 (duplicate) must be generated and sent to the node (in the IEEE 802.15.4 segment) that originated the registration. This message must be sent as specified in section 6.5.2 of I-D-ietf-6lowpan-nd (i.e., to the link-local IPv6 address formed from the Interface Identifier (IID) derived from the EUI-64 in the NCE, due to a possible risk of link-layer address collision). After sending the message, the NCE can be deleted.

Neighbor Solicitation originating in IEEE 802.3 segment

As stated in RFC 4861 and RFC 4862 and as illustrated in Figure 3.4, both unicast and multicast NS messages originating in the IEEE 802.3 segment

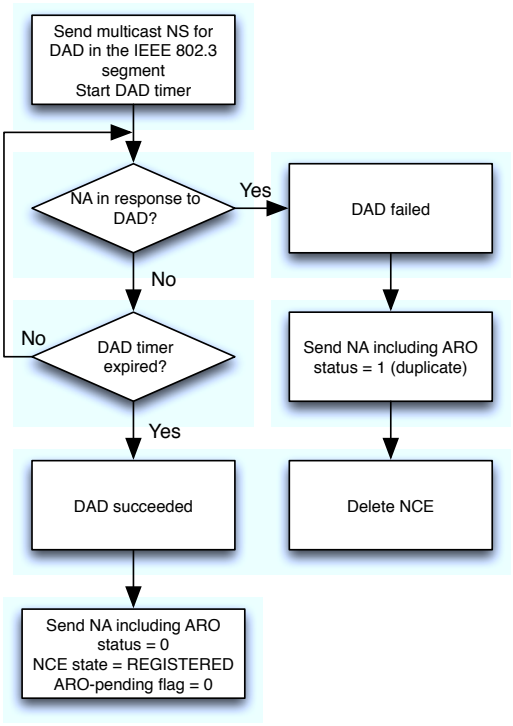

Figure 3.6: DAD performed on behalf of 6LoWPAN nodes. The diagram illustrates the process assuming DupAddrDetectTransmits = 1.

may arrive at the 6LP-GW. These messages can be sent with three different purposes: Address Resolution, NUD, and DAD.

For Address Resolution and DAD, the NS messages are sent to the Solicited-node *multicast* address of the recipient while, for NUD, they are unicast.

As previously mentioned, 6Hs would respond to the unicast NS messages, but they do not join the Solicited-node multicast address and, therefore, they will not respond to these multicast NS messages. In contrast, 6Rs must join the Solicited-node multicast address and thus they must respond to both unicast and multicast NS messages.

We should note here that the 6LP-GW is aware of every node that is currently reachable in the IEEE 802.15.4 segment due to the 6LoWPAN-ND registration process.

Unicast NS

Unicast NS messages are sent for reachability detection (NUD). These messages could be forwarded unchanged (except for the appropriate MAC translation) to the IEEE 802.15.4 segment in order that the target nodes could

respond to the NS with a NA. However, as 6LoWPAN nodes are registered with the 6LP-GW, the information contained in its NC is *a priori* sufficient to generate the response, thus wireless nodes save energy as they neither need to receive nor send the NS and NA messages, respectively (see the dashed lines coming out of the blue unicast NS arrow in Figure 3.4 on page 42). It is important to note that 6LoWPAN nodes are only required to register non-link-local addresses with routers. Thus, when receiving a unicast NS, the 6LP-GW will behave as follows:

- If the incoming NS's target address is not a link-local address, the 6LP-GW will search its NC for a matching entry in the REGISTERED state. If found, a NA message shall be generated and sent in response to the NS as specified in section 7.2.4 of RFC 4861, using the matching NCE's address as the source address of the message.
- If the incoming NS's destination address is a link-local address, the 6LP-GW will generate a link-local (EUI-64-based) IPv6 address for every different EUI-64 (contained in REGISTERED NCEs) stored in its NC. If one of these EUI-64 generated addresses matches the target of the NS, then the 6LP-GW will respond with a NA to the NS as specified in section 7.2.4 of RFC 4861, using the generated link-local address as the source address of the NA.

In all cases, according to section 5.4.3 of RFC 4862, the 6LP-GW will **not** generate a response if the matching address is in the TENTATIVE state. On the other hand, when sending out NAs on behalf of 6LNs, the following considerations must be taken into account:

- The Router flag must be set to the corresponding NCE isRouter flag value.
- The Solicited flag must be set to 1, since the NA is responding to a NS message.
- The Override flag must be set to 1, as recommended for this case in Section 4.3 of RFC 4861.

Multicast NS

Multicast NS messages are sent to perform either Address Resolution or DAD. These messages invoke responses from 6Rs, but not by 6Hs (since as noted earlier 6Hs do not join the Solicted-node multicast group). The 6LP-GW may be aware of which entries in its NC correspond to 6Rs due to previously intercepted RA or NA (having its Router flag set) messages and thus, the 6LP-GW could choose to forward these multicast NS messages only to these

6Rs. However, this approach is unreliable since it requires that all 6R nodes have previously sent at least one such NA or RA message.

Therefore, this implementation proceeds using a similar approach as for unicast NS messages; for every multicast NS message arriving on the IEEE 802.3 interface, the 6LP-GW generates an appropriate NA in response as follows:

Unless the IPv6 source address of the incoming NS is the unspecified address, the packet is sent for Address Resolution. In this case, the 6LP-GW operates **exactly** as described in Section 3.2.4.1 on page 46, with the following exception:

The NA generated in response (if any) **must** include a TLLAO option containing the link-layer address of the node in the NCE, as mandated in section 4.4 of RFC 4861 for NAs responding to multicast NSs (the 6LP-GW is sending this message *on behalf* of a node receiving a multicast NS message).

Note that RFC 4861 does not preclude the inclusion of the TLLAO option in NA messages responding to unicast NSs; it simply indicates that it is unnecessary.

On the other hand, if the source address of the NS is the unspecified address, then the NS message has been sent to perform DAD. In this case, the 6LP-GW must also search in its NC for a NCE having the same IPv6 address as the target address of the NS message. If a matching entry is found, then the sender of the NS is trying to configure a duplicate address and the 6LP-GW behaviour depends on the NCE's state: If the NCE's state is REGISTERED, then the 6LP-GW shall behave as described in section 7.2.3 of RFC 4861. This means responding with a NA to the all-nodes multicast address (**only** in the IEEE 802.3 segment, since that is the segment affected by the DAD in progress and, thereby, multicasting such a NA in the IEEE 802.15.4 would only be a waste of energy).

Conversely, if the NCE state is TENTATIVE, then the DAD timer for that NCE must be stopped assuming that the registration process for the corresponding NCE failed. The 6LP-GW must report the registration failure to the corresponding 6LN by generating and sending a NA including an ARO option with status value = 1. As specified in RFC 4862, the 6LP-GW shall perform no actions on the IEEE 802.3 segment.

3.2.4.2 Processing Neighbor Advertisement messages

As specified in RFC 4861, 2 types of NA messages may be generated:

- Unicast solicited NA generated in response to NS messages.
- Multicast unsolicited NA generated.

Figure 3.7 illustrates the different types of NA messages that can arrive at the 6LP-GW.

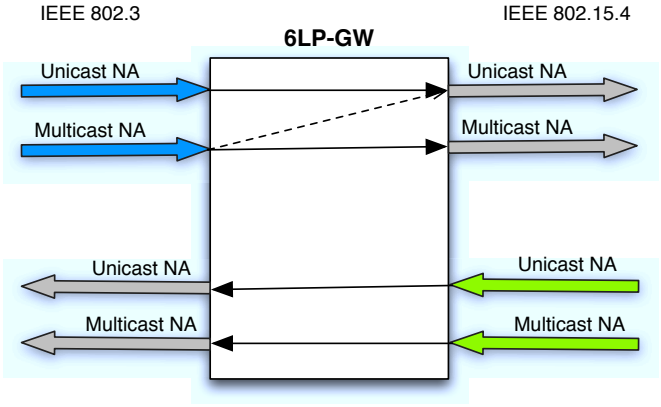

Figure 3.7: NA message processing. Each type of incoming NA is connected by arrows to the corresponding output this NA will produce.

Neighbor Advertisement originating in IEEE 802.15.4 segment

As explained in Section 3.2.4.1, this implementation opts for having the 6LP-GW respond to NS messages on behalf of 6LNs. Therefore, it is unlikely that any 6LN sends unicast solicited NA messages in response to NSs coming from the IEEE 802.3 segment (naturally, this communication is possible between 6LNs, as long as at least one of them is a 6R, but as direct communication is possible in that case, this is not a matter of importance for the 6LP-GW). Hence, if an unicast solicited NA arrives at the 6LP-GW's IEEE 802.15.4 interface, then it will be interpreted as an error and the packet will be discarded.

On the other hand, however, 6LNs could still send multicast unsolicited NA messages for quick information propagation. According to RFC 4861, if a node determines that its link-layer address has changed, it may multicast a multicast unsolicited NA message. It seems unlikely that a 6LN experiences a need for sending such messages, but as I-D.ietf-6lowpan-nd does not preclude it, we assume that a 6LN may do so. Thus, unsolicited multicast NA messages arriving to the 6LP-GW from the IEEE 802.15.4 segment, shall be forwarded unchanged (except for the appropriate MAC translation) to the IEEE 802.3 segment.

Neighbor Advertisement originating in IEEE 802.3 segment

Likewise in the IEEE 802.15.4 segment, the NA messages originating in IEEE 802.3 segment that can reach the 6LP-GW may be solicited and unicast, or unsolicited and multicast, each of them having originated for different reasons, as explained below.

Unicast NA

A unicast NA message could only originate as a response to a unicast NS message sent from the IEEE 802.15.4 segment. The original NS message could be either a probe sent for reachability confirmation (NUD) or part of the registration process.

In all cases, when receiving a solicited unicast NA from the IEEE 802.3 segment, the 6LP-GW searches its NC for a NCE matching the destination (or target) address of the NA. If a matching entry in the REGISTERED state is found, then the NA will be forwarded to the IEEE 802.15.4 segment. Otherwise, such an NA should be considered an error and shall be discarded. However, depending on the value of the ARO-pending flag in the matching NCE, the 6LP-GW may be required to perform an additional task: if the ARO-pending flag of the matching NCE is set to 1, that means that the incoming NA is not the response to a NS sent for NUD, but the final part of the 6LoWPAN-ND registration process. In this case, the ARO-pending flag must be set to 0 and an ARO option containing a status value of 0 and its EUI-64 field filled with the value in the NCE must be appended to the incoming NA before forwarding it to the corresponding 6LN in the IEEE 802.15.4 segment. In all cases, forwarding will require the appropriate MAC translation.

Multicast NA

Unsolicited Multicast NA messages may arrive at the 6LP-GW on the IEEE 802.3 interface either for quick information propagation if the link-layer address of the sender changed or as a response to a NS previously sent for DAD (meaning that there is a duplicate).

If the target address in the NA message corresponds to a NCE whose state is TENTATIVE, that means DAD failed for that 6LN. In this case, the 6LP-GW must notify the registering 6LN its registration failure by generating a new NA (on behalf of the RR it is attached to), with an ARO option containing a status value of 1 (duplicate). This packet shall be sent on the IEEE 802.15.4 interface according to I-D.ietf-6lowpan-nd section 6.5.2 (i.e., to the link-local IPv6 address of the node, that can be generated from the EUI-64 stored in the NCE).

If there is no TENTATIVE NCE whose IPv6 address matches the target, then the packet is forwarded unchanged (except for the appropriate MAC translation), since it may have been generated for quick information propagation.

3.2.4.3 Processing Router Solicitation messages

This section describes the processing of RS messages. The different types of RSs that may arrive at the 6LP-GW are shown in Figure 3.8.

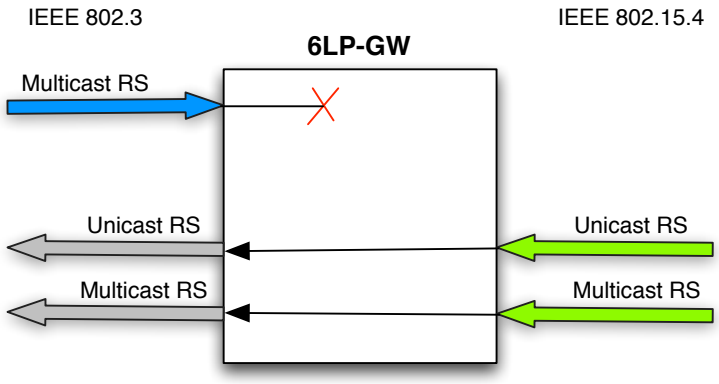

Figure 3.8: RS message processing. The different types of incoming RS messages are connected by arrows to the outgoing messages they will produce.

In IPv6-ND, RS messages are sent only during bootstrapping and they are mainly multicast; the IPv6 source address in RAs may be the unspecified address and the inclusion of a SLLAO is not mandated by in RFC 4861. In contrast, i-D.ietf-6lowpan-nd specifies that routers are not required to send periodic RA messages, therefore, hosts will send RS messages more frequently (instead of doing so only during bootstrapping) in order to obtain and maintain their prefixes, addresses, and contexts; these RS will be unicast unless the link-layer address of the router is not known (i.e., for example, when bootstrapping).

Router Solicitation originating in IEEE 802.15.4 segment

Upon receiving a RS message originating in the IEEE 802.15.4 segment, the 6LP-GW will first check that the source address is not the unspecified address, and that a SLLAO option is present in the message. If any of these conditions is not met, the RS will be silently discarded. If both conditions are met, then the 6LP-GW will search its NC for a NCE corresponding to the source IPv6 address of the incoming message. If no matching NCE is found, the 6LP-GW will create it. In any case, the awaiting-RA of such NCE will be set to 1 and the RS message will be forwarded unchanged (except for the appropriate MAC translation) to the IEEE 802.3 segment. If the creation of a NCE is necessary but the NC is full (and hence the NCE cannot be created), then the RS message shall be discarded. The treatment of RA messages that will arrive responding to these RSs is explained in Section 3.2.4.4.

Router Solicitation originating in IEEE 802.3 segment

6LRs will never be the first hop for any node in the IEEE 802.3 segment since the RR will always be in between. In order to save energy and processing time, this implementation chooses to mask the router nature of 6LRs so that they are seen as simple hosts from the IEEE 802.3 point of view. To do so, the 6LP-GW will silently discard any RS message coming from the IEEE 802.3 segment.

3.2.4.4 Processing Router Advertisement messages

There are fundamental differences between IPv6-ND and 6LoWPAN-ND regarding the sending and processing of RA messages. Such differences lead to the processing of such messages as detailed below and illustrated in Figure 3.9.

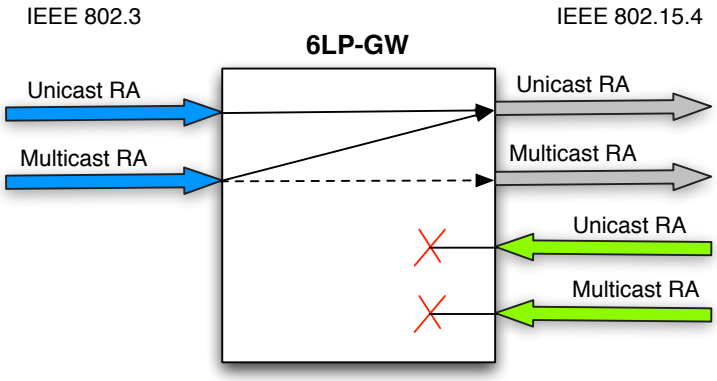

Figure 3.9: RA message processing. The different types of incoming RA messages connected to the outgoing messages they may produce.

Router Advertisement originating in IEEE 802.15.4 segment

As in Section 3.2.4.3, RA messages originated in the IEEE 802.15.4 segment are of no importance for NCDs, since no host in the IEEE 802.3 segment can make use of a 6R as its next-hop, nor will the RR route packets into its LAN subnet. Therefore, such RA messages originated in the IEEE 802.15.4 segment will be silently discarded.

Router Advertisement originating in IEEE 802.3 segment

It seems natural that RA messages reaching the IEEE 802.3 interface should end up being forwarded to the IEEE 802.15.4 interface. However, due to the major importance of RA messages in the autoconfiguration process, and the significant differences between its processing in the two ND protocols for

which the 6LP-GW provides proxy operations, its treatment by the 6LP-GW requires certain considerations:

- First, one of the main purposes of I-D.ietf-6lowpan-nd is to reduce the highly costly multicast traffic in 6LoWPAN networks. Hence, the Internet Draft states that, in contrast to RFC 4861, 6Rs need not send periodic multicast RAs. In addition, 6LoWPAN-ND mandates the inclusion of the SLLAO option in RS messages (which is not required by RFC 4861) so that 6Rs can respond with unicast-addressed RAs, instead of multicasting them. Furthermore, I-D.ietf-6lowpan-nd requires that 6Rs always respond to the **unicast** address of the originator of the RS (this is possible due to SLLAO option included in RSs). Moreover, such RAs always include an SLLAO option so that the 6LN soliciting the RA obtains the link-layer address of the 6R in the same step (instead of needing further address resolution). Conversely, RFC 4861 suggests that routers may omit the SLLAO option in RAs, and that RAs may be sent to the all-nodes multicast address as a response to a RS.

- As for next-hop determination, I-D.ietf-6lowpan-nd states that all prefixes but the link-local (FE80::) are assumed to be off-link in 6LoWPAN-ND (in contrast to the on-link definition in RFC 4861, updated by RFC 5942 [45]). Hence, 6Hs will ignore any Prefix Information option (PIO) option whose 'L' (on-link) flag is set. However, RRs usually advertise global prefixes with the 'L' flag set in the PIO option within the local network. Considering that the PIO option is the only way hosts have to acquire a global address (if no other mechanisms such as DHCPv6 [19] are used for address autoconfiguration) and that the same prefix which is considered to be on-link in the IEEE 802.3 segment is assumed to be off-link in the IEEE 802.15.4 segment, the 6LP-GW must always clear this 'L' flag in the PIO option for every packet originating in the IEEE 802.3 segment and directed to the IEEE 802.15.4 segment.

- Regarding the optional context management and dissemination, the 6LP-GW must perform the operations of a 6LBR: it must create, maintain, and disseminate such contexts using 6CO options included in RA messages. This section only describes the dissemination of such contexts while, Section 3.2.4.4 details **how** such contexts are created and maintained.

- Finally, as seen in previous sections, there are several situations where the 6LP-GW needs to send messages on behalf of the RR. As RAs originating in the IEEE 802.3 segment are necessarily sent by the RR, the 6LP-GW needs to save both, the source MAC address and the source IPv6 address of such RA messages for subsequent use. As specified in

RFC 4861, RA messages are always sent from the link-local (FE80::) address, which simplifies the task of managing the RR's addresses. However, there could be more than one RR present in the network. This would require extra management of the RR's address, but such management is out of scope of this thesis project.

Having provided all the above considerations, the processing of RA messages originating in the IEEE 802.3 occurs as follows:

Upon arrival of a RA message, the 6LP-GW will retrieve both the RR's IPv6 and MAC addresses from the RA's IPv6 and Ethernet headers respectively, then store then for further use. Next, it will check the ICMPv6 options or the RA as described here:

- If no SLLAO option is present in the RA, the 6LP-GW will append the corresponding SLLAO according to the previously retrieved RR's MAC address.
- If a PIO option is included, the 6LP-GW will clear its 'L' (on-link) flag, if it is set.
- For every context in use in the context table, the 6LP-GW shall append a 6CO option.

Note that inclusion of a new SLLAO and/or 6CO option(s), as well as the modification of the 'L' flag of the PIO option calls for recomputation of ICMPv6 checksum, as described in section 2.3 of RFC 4443 [13].

Finally, in order to minimize the amount of unnecessary multicast traffic in the IEEE 802.15.4 segment, the 6LP-GW will forward the RA as follows:

If the packet's destination address is unicast, then the 6LP-GW will examine its NC searching for this address. If a matching NCE having its awaiting-RA flag set is found (regardless of its state), then the RA will be forwarded to its destination and the 6LP-GW will clear the awaiting-RA flag in the corresponding NCE. If no matching NCE is found, or if the matching NCE has its awaiting-RA flag set to zero, the RA will be silently discarded.

In contrast, if the RA's destination address is the all nodes multicast address, then, **for every** NCE having its awaiting-RA flag set to 1, the 6LP-GW will replace the RA's destination (multicast) IPv6 and MAC addresses by these belonging to the NCE and send the packet out to through the IEEE 802.15.4 interface, clearing the awaiting-RA flag afterwards. If no NCE having its awaiting-RA flag set is found, the packet shall be silently discarded.

Additionally, the 6LP-GW uses a boolean variable indicating whether there is **any** NCE with its awaiting-RA flag set or not throughout the whole NC, which discharges the 6LP-GW from performing all the previous tasks (except for the RR's addresses retrieval) if the RA to be processed is meant to be discarded.

Context management and dissemination

As previously said, the 6LP-GW must take responsibility for all the 6LoWPAN-ND-related tasks assigned to a 6LBR. Despite context management and dissemination being an optional feature, it falls among the 6LBR tasks which the 6LP-GW implements. This section explains how context management and dissemination is performed in this implementation.

Although the approach taken here may seem simplistic, it is sufficient to demonstrate the ability of the 6LP-GW to successfully handle 6LoWPAN contexts. Section 3.2.4.4 explains how contexts are disseminated all over the 6LoWPAN network using RA messages. Thus, the only remaining aspects regarding this issue are context creation and management.

As for context creation, the current implementation of the 6LP-GW only considers PIO-based context creation. This means that, when receiving a RA containing a PIO option, the 6LP-GW will search its context table for a context having the same prefix as contained in the PIO option. If no matching context is found, that will result in a new context. Newly created contexts must be assigned a numeric context identifier ranging from 0 to 15. Since use of context identifier 0 saves 1 octet in the IPHC header (see Section 2.6.2), this is the first context identifier that will be assigned to a context. Subsequent contexts, if any, shall use subsequent context identifiers. The reason to create a context from the network prefix is simple: it would be present in every packet involved in the communication with nodes external to the local network. This simple approach could be improved by utilizing more advanced context creation techniques, but such techniques are outside the scope of this thesis project. Section 6.2, however, provides some advice about this topic.

Regarding context maintenance, this implementation mainly follows the proceedings described in I-D.ietf-6lowpan-nd, with some specific extensions, but always compliant to the Internet Draft. New contexts are created in *uncompress-only* state, so that new contexts can arrive to every node in the network before anyone uses them for compression. After a certain (manually configurable) time, the context moves to its *normal* state, in which it can be used for compression and uncompression. In this state, every prefix/es contained in PIO options of RAs arriving to the 6LP-GW, will refresh the lifetime/s of the corresponding entry/entries in the context table.

Should a context lifetime expire, then this context will move to the *expired* state. In the *expired* state, contexts are announced again as *uncompress-only*, so that nodes receiving 6COs with these contexts update their context tables and stop using them for compression. If a RA containing a prefix in its PIO that corresponds to a *expired* context arrives at the 6LP-GW, the context's lifetime refreshes and its state reverts to *normal* again. Otherwise, after a period of twice *Default Router Lifetime* seconds (announced in RA messages) the context is deleted. When a new context is created or when a context's state changes, then the next RA arriving to the 6LP-GW from the IEEE 802.3

will be forwarded to the all-nodes multicast address [25] in the IEEE.802.15.4 segment, even if the original RA's destination address was not the all-nodes multicast address.

3.2.4.5 Processing a Redirect

Redirect messages are sent by routers to inform hosts of a better next-hop. They can be sent when a router receives a packet destined to some host which could be reached at less cost by choosing another router as next-hop, or directly (if the destination host is known to be in same network (link-local) as the sender.

Redirect originating in IEEE 802.15.4 segment

According to I-D.ietf-6lowpan-nd, redirects are not used by 6LoWPAN-ND in route-over topologies, (although they may be used in mesh-under topologies). As the topology under consideration in this thesis project is assumed to be a route-over topology, these messages, if any, will be discarded as they are of no use. We should note here that, since the use of Redirects is not mandatory in other topologies either, the approach is still valid even if no assumptions regarding topology were made.

Redirect originating in IEEE 802.3 segment

As noted earlier, Redirect messages can be sent by a router to inform a sending node of a better next-hop to the destination. This better next-hop may be another router on the path to the destination, or the destination itself, if it happens to be a neighbor. However, the mechanisms to determine the "best" next-hop differ in the two ND protocols under consideration. According to RFC 4861, the originator of a unicast packet performs a longest prefix match to determine whether the destination is on-link or not. I-D.ietf-6lowpan-nd simplifies this process as follows: if the destination address is a link-local address (FE80::), then the destination is on link. Otherwise, the destination is off-link. In both cases, if the destination is determined to be off-link, the packet is sent via a router (selected as specified in RFC 4861, section 6.3.6).

As a reader may infer, when a 6LN sends a packet to the global address of a neighboring NCD on the IEEE 802.3 segment, the 6LN's next-hop determination algorithm will determine that the destination is off-link (even if both, sender and destination share the same prefix). Thus, the packet will be sent from the 6LN to the NCD via the RR. The RR, according to RFC 4861, will determine that the source and destination of such a packet are neighbors and hence, besides forwarding it to its destination, it might well send a Redirect message to inform the originator (the 6LN) that the destination could be reached directly in one single hop. Since that is not true

in our particular case, and in addition, 6LNs are unable to process Redirects, these messages shall also be discarded.

3.2.4.6 Non-proxy Features

ND Option Filtering

RFC 4861 states that any node that happens to receive an unrecognised option in a ND message, should simply ignore such an option and continue processing the next one. This fact allows the 6LP-GW to forward packets from one segment to the other not caring about possible options that could be misinterpreted or cause the whole packet to be discarded. However, in terms of power consumption, every single byte transferred counts. This power consumption affects every node involved in the communication (both senders and recipients) being particularly critical in battery-powered nodes.

For this reason, this implementation will filter out some ND options, contained mainly in RA messages, that could be considered irrelevant for 6LoWPAN networks. Examples of options that can be filtered out are the Recursive Domain Name Server (DNS) Option (defined in RFC 6106 [28]) or the Flags Expansion Option (defined in RFC 5175 [23]). Note that SLLAO, MTU, and PIO options should not be filtered out. The 6LP-GW implementation described here will remove all the options in RA messages originating in the IEEE 802.3 segment that are to be forwarded to the IEEE 802.15.4, except for SLLAO, MTU, and PIO options.

It is also possible to filter out irrelevant options of messages originating in the IEEE 802.15.4 segment and directed to the IEEE 802.3 interface, such as the ARO, 6CO, and ABRO. However, this filtering is of minor interest since it would occur in an Ethernet link where the devices involved are likely to be plugged into the power mains.

Chapter 4

Applying the Method

This chapter describes the implementation details of the application specified in the previous chapter (Chapter 3). Since Contiki was utilized in the implementation being described, Section 4.1 provides a comprehensive overview about what was already done and what is ready to use out of the box, what was already done but required certain modifications, and what was not done at all. Moreover, this chapter provides an overview of the whole application, followed by a thorough explanation of each of the different functional modules that comprise it.

4.1 What Contiki's provides and does not provide

This section describes the parts of the 6LP-GW application that are part of the Contiki core and distinguishes these from those parts that have been completely or partially developed as part of this thesis project.

4.1.1 What Contiki provides

As explained in Chapter 2, Section 2.10.2,

Contiki furnishes a richly-featured development toolbox. In addition to the kernel and the protothreads implementation (described in Section 2.10.2), Contiki provides full IPv4 and IPv6 stacks, a standard-compliant 6LoWPAN implementation, and a large set of libraries. Of these elements, we make use of the IPv4 and IPv6 stacks, the Transport Layer application interface, 6LoWPAN, and several other modules and libraries. How we have used this existing code is described below.

IPv4 stack

The IPv4 stack runs as any other Contiki protothread and supports both TCP and UDP protocols. It includes ICMP (part of which is used), ARP, and DHCP implementations. While UDP, TCP, and ICMP are implemented as part of the IPv4 module, DHCP and ARP are implemented as separate modules. DHCP is implemented as a protothread and in order to use it together with the rest of the Contiki code, it needs to be wrapped within a Contiki *process* (see Section 2.10.2.2). In contrast, ARP is implemented as a set of functions, which must be called manually when performing certain Ethernet-related operations.

IPv6 stack

The IPv6 stack also runs as a Contiki protothread and supports both TCP and UDP. In addition, it includes complete ICMPv6 and Neighbor Discovery for IPv6 (RFC 4861 [34]) implementations.

Transport Layer application interface

Contiki defines a lightweight socket-like application programming interface (API) for application-level communication. Like the Unix socket API, this API allows the creation of TCP and UDP *connections* which maintain the association that usually defines a socket, i.e., source address and local port and, in case of TCP, destination address and remote port. This API also supports most common Unix socket operations, such as creating/eliminating *connections* (`socket()`), listening for incoming connection requests (TCP `listen()`), sending connection requests (TCP `connect()`), binding *connections* to ports (`bind()`), and sending and receiving packets (`send()` and `recv()` in case of TCP, and `sendto()` and `recvfrom()` in case of UDP).

6LoWPAN

The 6LoWPAN implementation currently supports only 64-bit "long" addresses. It supports fragmentation and different compression mechanisms. The supported compression mechanisms are stateless HC1 compression (defined in RFC 4944 [33]), and stateful IPHC compression (defined in RFC 6282 [27]), which obsoletes the former. However, the IPHC compression implementation has some limitations which will be described in Section 4.1.3. In addition, Contiki also includes several IEEE 802.15.4 MAC layers, which provide support for encapsulating 6LoWPAN packets into IEEE 802.15.4 frames among other features.

Other modules and libraries

- Contiki provides several libraries for time measurement. The most basic one is the *Timer library*, which provides simple and lightweight function for timer management and provides the base on top of which other timer libraries are built. This library implements the following functions:

`timer_set`	Sets a timer to a time interval.
`timer_reset`	Resets a timer. The former expiration time of the timer becomes its new starting point, which allows the timer to remain stable over time.

`timer_restart`	Restarts a timer. The new starting point of the timer is the current time (in contrast to `timer_reset`).
`timer_expired`	Evaluates whether a timer has already expired or not.
`timer_remaining`	Returns the time remaining until expiration of a timer.

Together with the *Timer library* which measures the time in *system tics*, Contiki provides the *Seconds timer library*, which in contrast to the *Timer library*, implements timers having a second as their timing unit. However, neither the Timer library nor the *Seconds timer library* provide any mechanism to inform the *process* setting the timer about any timers' expiration. Therefore, any *process* using these timers has to actively poll the timer by means of the `timer_expired` function or its equivalent in the *Seconds timer library*, `stimer_expired`, in order to determine whether a timer has expired or not.

For this reason, Contiki provides the *Event timer library*. This library, as mentioned Section 2.10.2.2, implements an active *process* (the etimer *process*) that periodically checks all the *Event timers* and posts an event (`PROCESS_EVENT_TIMER`) to the *process* who set the timer when such timer expires.

A similar utility is provided by the *Callback timer library*. This library also runs an active *process* (the ctimer *process*) that periodically checks the *Callback timers*. Similar to the *Event timer library*, the *Callback timer library* allows setting timers that do not need to be actively polled, but with the difference that the callback timer library is independent of the *process* that specifies the callback timer. This means that instead of posting an event to the *process* who set a timer, the *Callback timer library* will call a callback function (provided when the timer was set) when the timer expires. The *Callback timer library* is useful for certain situations where utilizing a *process* to verify a timer would simply be overkill.

Finally, the last timer library provided by Contiki is the *Real-time task scheduling library*. Unlike the rest of the timers, this library does not rely on the *Timer library* nor is it dependent upon any process to check for timer expirations. Instead, it relies on the hardware-specific real time module (if any) present in wide variety of micro-controllers. This library, similar to the *Callback timer library* allows scheduling a task (function) to be executed at a specified time in the future.

We should note, however, that the implementation described in this thesis does not make use of the *Callback timer library* nor the *Real-time task scheduling library*; they are described here only for completeness.

4.1.2 What Contiki requires

This section explains the elements required by Contiki in general, while Section 4.2.1 explains these elements in detail for the implementation described in this thesis.

Since Contiki is platform-independent software, no platform-specific code is provided. Instead, a number of platform-specific functions, constants, and data-type definitions must be provided so that Contiki can make use of the platform's resources. This set of platform-specific code includes elements that can be considered drivers for the hardware components needed by the run-time system, which together constitute a"Contiki port" (as it is called by the by the Contiki community).

On the other hand, no implementation would be complete without (at least) one application. The Contiki's protothreads library together with the transport layer application interface provide a rich set of functions that allow implementation of any kind of applications. However, since strictly speaking this does not pose a Contiki requirement, we will leave the implementation of applications for the moment.

Figure 4.1 illustrates a typical Contiki application highlighting its platform-specific requirements.

Contiki Clock library

The key element of a Contiki port is the *Clock library*, which is used by the timer libraries. This code must thus be initialized before any other Contiki module that uses timers. This module is highly platform-specific and needs to be implemented specifically for each Contiki port. Contiki provides a set of function declarations whose definition needs to be implemented for this module to work. In addition, the module requires the definition of the constant `CLOCK_CONF_SECOND`, which specifies the duration of a second in terms of system ticks, and the definition of the `clock_time_t` data type, which holds values that are based upon the number of system ticks since system start up.

The *Clock library* elements that need to be provided are:

`CLOCK_CONF_SECOND` This constant represents one (1) second measured in system ticks. The meaning of a system tick is explained below.

`clock_time_t` This data type definition sets the type of the variable that will hold the number of system ticks since system start up. Note that in order to be able to compare two

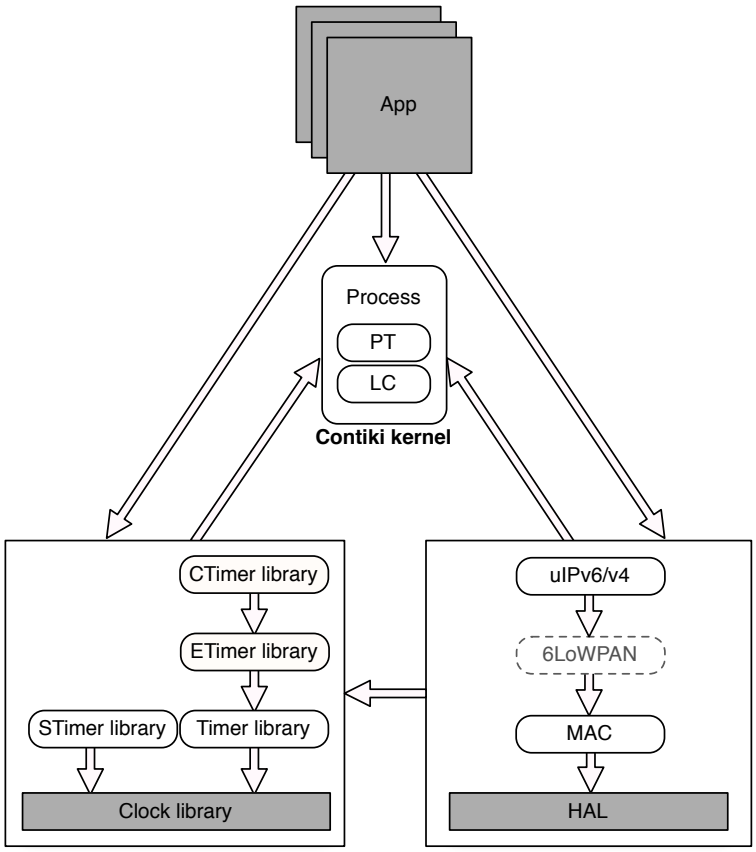

Figure 4.1: A typical Contiki-based application. Grey boxes represent implementation-specific modules. The dashed box containing the text "6LoWPAN" mean that the 6LoWPAN layer may be present or not. Arrows show the dependency direction. As the figure illustrates, some timer libraries and the uIP module makes use of the Contiki protothreads library; applications on top of the stack make use of the Contiki protothreads library along with the uIP transport layer application interface and (optionally) timers; and both the Contiki timer library and the Contiki's uIP implementation rely on hardware-specific modules.

times when there has been a wrap around of this variable in between the two samples, the maximum interval within which two times can be compared is restricted to `MAX_VALUE_OF(clock_time_t) / 2`. Thus, a too-short data type would cause the maximum possible interval between two times being compared to be too small. In addition, a too-big data type (depending on the

micro processor architecture) could cause an expensive addition operation to be executed very frequently and will also consume more RAM. Therefore, this data type must be carefully selected between all the unsigned data types provided by the C compiler.

`clock_init()` Initializes the clock module. This function consists in most cases (when allowed by the micro-controller) on setting a timer interrupt which will be executed every certain number of micro-controller processor cycles. As the micro-controller's clock frequency is a well known value, this number of cycles constitutes the basic time unit of the *Clock library*, i.e., a system tick.

As for the interrupt routine it could just increment a static variable (for convenience, we will call it `system_ticks` in the rest of this section) every time it is invoked. However, for certain implementations it might be convenient to add (among others) the following elements to this interrupt service routine:

- A seconds counter which would be incremented by one every time `system_ticks` modulus `CLOCK_CONF_SECOND` equals zero. The reason for this is explained below.
- If the *Event timer library* is being used, the etimer *process* needs to be polled repeatedly so it can perform its periodic timer expiration check-and-notification. It could be constantly polled in a infinite loop in the main application body, but that would cause many unnecessary invocations, some of which would happen within the same system tick, which is completely useless and wasteful of resources. The *Event timer library* provides two functions that can be used to poll the etimer process only when required. These functions are `etimer_pending()`, which returns a value other than zero when there is any *Event timer* pending and `etimer_request_poll()` which request a poll from the Contiki kernel on behalf of the etimer process. Thus, if the *Event timer library* is being used, it might be useful to perform a conditional call to the `etimer_request_poll()` function if `etimer_pending()` returns a value different than zero **within** the timer interrupt service routine.

Needless to say, this interrupt must be executed as stably over time as possible to avoid clock jitter. Note that the execution of the interrupt service routine poses a delay in the execution. This delay does not represent a problem as long as it is constant (i.e., the execution of the interrupt service routine always takes the same time). However, adding conditional elements to this code can lead to jitter, which calls for some other compensatory mechanisms to avoid potential clock drifts.

`clock_time()` Returns the number of system ticks since the system started. Usually this count is held in the previously mentioned variable `system_tics` which is incremented in the timer interrupt routine set in `clock_init()`. Note that the variable holding the system ticks is likely to wrap around (depending on the length of the `clock_time_t` data type).

`clock_seconds()` Returns the number of seconds elapsed since system start up. Provided that the number of ticks per second is well known (as noted earlier, this depends on the CPU frequency and the timer interrupt setting), this function can be easily implemented in two different ways: returning `system_ticks` / `CLOCK_CONF_SECOND`, or returning the value of a variable which is also increased in the interrupt routine every time `system_ticks` modulo `CLOCK_CONF_SECOND` equals zero (as explained above). The former approach will save the amount of RAM corresponding to the variable holding the number of seconds, and avoid some of the CPU time that would be used to calculate the modulus operation every time the timer interrupt is executed, at the expense of performing an extra division on the `clock_seconds()` function (which will be executed much less frequently than the timer interrupt). Additionally, the latter method offers a longer interval before the number of seconds wraps around.

Communication drivers

The Contiki communication stack is built by piling its different components one on top of each other. These components are independent from each other except that each of them must implement a number of functions required by the immediately upper component in the stack. Then, the functions each component implements are grouped into a C `struct` called *driver*, forming an object similar to that used in object-oriented programming. Finally, all these drivers together are grouped by means of a set of C macros into the network stack ("netstack") module. Figure 4.2 illustrates how all these drivers fit into the netstack module.

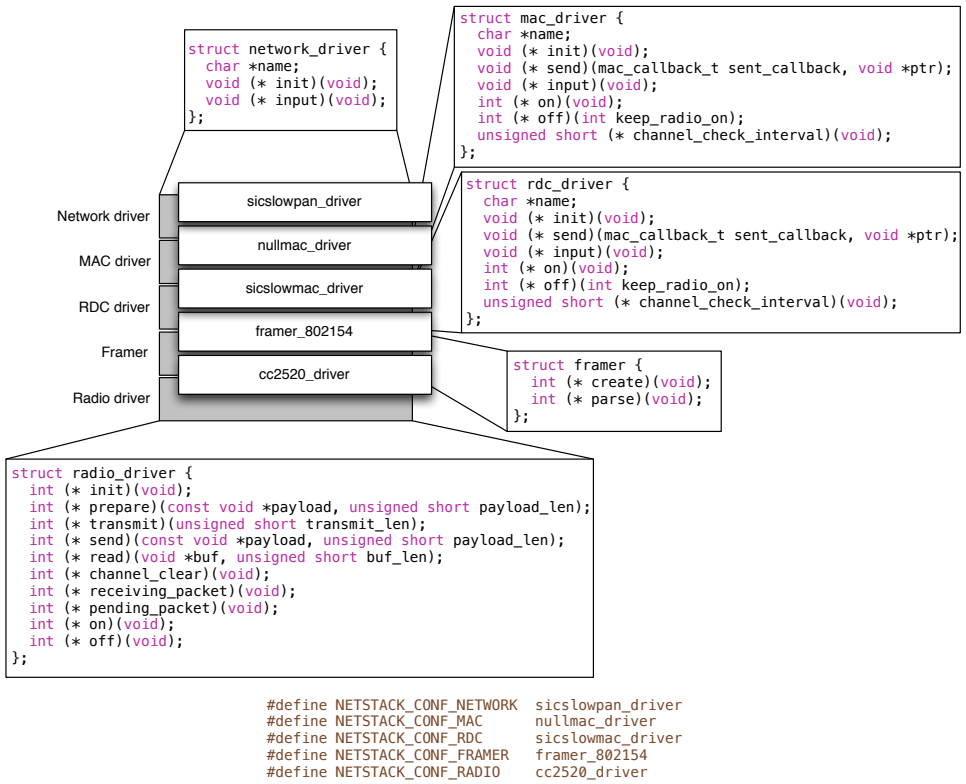

Figure 4.2: The Contiki network stack. As the figure illustrates, different components may fit in each slot as long as they implement the functions defined by the corresponding driver "slot".

In addition, these components are pluggable, thus any of them can be replaced by another component as long as it implements the interface functions of the *driver*. Contiki itself provides several alternatives for some of these components so that different implementations can make use of the most appropriate ones. Additionally, developers are free to replace any of them

by their own components.

However, Contiki can not provide the lowest component in the stack, as this is the platform-specific communication hardware *driver*. In addition to the *Clock library*, this is another situation in which a Contiki module requires hardware drivers. In our case, there are two different components that must be plugged into the communication stack's lower-layer slot in order to transmit and receiver frames: the Ethernet controller for the device's Ethernet interface (Microchip's ENC28J60) and the 2.4 GHz IEEE 802.15.4 radio frequency transceiver controller for its IEEE 802.15.4 radio interface (Texas Instruments' CC2520). As we will see in section 4.1.3, the fact that two different components should be placed into the same stack's *slot* poses another challenge for our application.

Both drivers require functions to initialize the corresponding device, and to send and receive a frame. Both drivers implement a Contiki *process* that needs to be polled by the Contiki kernel to check whether there is an incoming packet pending to be read and, if so, pass it to the stack for processing. The sending of a frame, in contrast, is performed by the IPv4 or IPv6 *process* when required via the driver communication interface, thus it is not necessary to wake the driver *process*.

The specifics of the implementation of each of these drivers will be explained in detail in Section 4.2.

Other hardware drivers

In addition to the previously mentioned hardware components, the board on which the 6LP-GW is implemented has three LEDs and two buttons whose drivers, despite not being used in the final version of the code, have been implemented for debugging purposes as part of this thesis project. Since the implementation of these drivers is not required for a basic Contiki port and, in turn, it is tightly coupled to our specific implementation, a detailed explanation of this implementation will be given in Section 4.2.1.

4.1.3 What Contiki does not provide

The previous sections have explained what Contiki provides and what Contiki needs to work. This section focus on the features that comprise the 6LP-GW requirements but that are not provided by Contiki. These are in addition to the required hardware drivers, and the code that implements the proxy-gateway itself.

As previously mentioned, Contiki includes communication utilities, such as IPv4 and IPv6 stacks, both featuring related protocol implementations (specifically 6LoWPAN, ND, ARP, and DHCP), and provides the necessary tools for application-layer communication. However, the constrained nature of the target devices Contiki is intended for, and the recentness of the

6LoWPAN-ND protocol lead to a number of gaps that we must fill in order to achieve our goals:

- The IPv6 implementation provided by Contiki only supports the ND protocol for IPv6, as specified in RFC 4861. However, it does not include the ND optimizations for LLNs as defined in draft-ietf-6lowpan-nd. As providing proxy operation between ND for IPv6 and the optimized version of ND for LLNs is one of the two main purposes of the 6LP-GW (the other main goal is the gateway operation), this imposes a major challenge for this thesis project. In order to achieve our goals and test the results, we required both implementing most of draft-ietf-6lowpan-nd's requirements for a router in the 6LP-GW and also implementing the *host behaviour* defined in the Internet Draft for external 6LoWPAN Hosts (6LHs).

- Contiki only allows the use of one (1) interface. However, due to the gateway nature of the 6LP-GW this presents an obvious problem: the 6LP-GW requires two interfaces: Ethernet and IEEE 802.15.4. Therefore, this called for significant modifications in the Contiki IP communication stack's structure.

- Contiki provides implementations of both IPv4 and IPv6 stacks. However, it does not allow their concurrent utilisation. Instead, implementations need to choose between IPv4 or IPv6. Since one of our requirements is to achieve a fully-featured internet device allowing further and effortless incorporation of any IP-version application, the use of a dual stack constitutes another challenge that must be dealt with.

- The 6LoWPAN implementation included in Contiki supports the stateful IPv6 header compression specified in RFC 6282. However, the current implementation of such compression mechanism has certain limitations. These limitations have their main root in the fact that no context dissemination mechanism had been specified until the release of draft-ietf-6lowpan-nd. As previously said, Contiki does not implement the draft-ietf-6lowpan-nd specification and therefore it provides no support for context management and dissemination. Instead, in the current Contiki implementation, contexts must be manually, statically included at compilation time in every node which is to share such contexts. Another limitation is the fact that the supported contexts are limited to a fixed length of 64 bits. Since we implement the context management and dissemination mechanisms defined in draft-ietf-6lowpan-nd in the 6LP-GW, our implementation chooses to use such mechanisms to their best advantage, by enabling support for up to 128-bit context lengths (which allows fully elision of any global IPv6

address), that are dynamically added and removed from the 6LoWPAN network as required.

- The fact that our application requires two interfaces, each of them requiring its corresponding hardware driver, together with the added dual stack and the 6LP-GW pseudo-layer laying in between them, introduces several mismatches between our needs and the base Contiki's network stack (*netstack*) structure (remember Figure 4.2 in Section 4.1.2). Thus, implementing our own netstack drivers and trying to fit them into the current netstack structure may be inefficient or simply not work. Hence, our application not only must implement the required network stack modules which compose the network stack, but we also need to extend Contiki's netstack structure to meet our requirements.

4.2 Application Overview

The application implemented in this thesis is divided into several parts, each of them performing a different task. The easiest way to understand the application running on the 6LP-GW is, thus, to divide it into the functional units it is comprised of. Figure 4.3 depicts a diagram of the application, including the communication flows between the different functional units.

Each of these functional units usually corresponds to a separate layer in our particular dual stack, which makes easy to understand the overall application. Since the whole application runs on top of the Contiki kernel, some of these functional units are implemented as Contiki processes, while others are simply built as libraries providing useful resources to other parts of the application.

It is important to remark that the application is integrated by using two well differentiated components:

1. The 6LP-GW, this is, the logic in charge of performing the proxy and forwarding operations, as described in Chapter 3.

2. The internet host "residing" in the device along with the 6LP-GW (which we will refer to as local-host for the remainder of this thesis), enables integration of additional application components. Note that this local-host is an "Ethernet host", which means that it does not use 6LoWPAN nor 6LoWPAN-ND, but rather uses the standard IPv4 and IPv6 stacks.

These two components are independent of each other. This means that the 6LP-GW treats the local-host as if it were an external component attached to another (virtual) Ethernet interface. Thus the 6LP-GW performs the same proxy and forwarding mechanisms for it as for any other node connected to the Ethernet link. Similarly, the local-host operates oblivious to the presence

of the 6LP-GW; thus the local-host will see any other 6LN or NCD in the network as regular Ethernet hosts. Note however that this relationship between the local-host and the 6LP-GW only applies to the local-host's IPv6 communication; all incoming and outgoing local-host's IPv4 traffic simply bypasses the 6LP-GW logic (as shown with the rightmost arrow linking the units labelled "IEEE 802.3 MAC" and "ARP").

The following sections describe in detail each of the functional units in a bottom-up fashion.

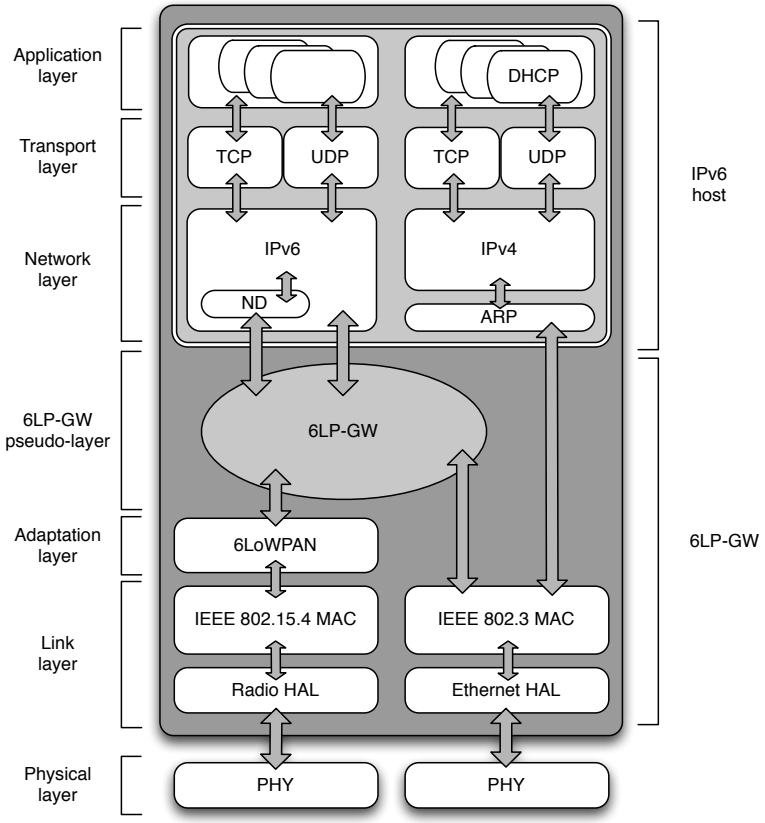

Figure 4.3: The 6LP-GW application diagram. The area surrounded by a white border in the upper part constitutes the local-host logic; the oval area in the center part of the diagram represents the 6LP-GW logic in; and the lower part of the diagram illustrates the lower level layers through which packets are delivered or sent to its corresponding destination. Note that the Ethernet-related functional units are common to both the 6LP-GW and the local-host.

4.2.1 Hardware Abstraction Layer

The hardware abstraction layer (HAL) is the logic that "abstracts" the hardware specific details, thus hiding these details from the rest of the application. This implementation follows the commonly-used black box approach, which consists of providing the necessary interface functions required by the immediately upper-layer logic while keeping the latter unaware of implementation details. The following sections describe the operation of each of the drivers implemented as part of this hardware abstraction layer. These descriptions provide a comprehensive explanation of the operation performed by the specific module under discussion, rather than a detailed and less instrumental explanation of the source code. A reader interested in implementation-specific issues is referred to Appendix C for further details.

4.2.1.1 The Clock library implementation

As previously mentioned (see Section 4.1.2), any Contiki implementation must provide a Clock library with the following elements:

- `CLOCK_CONF_SECOND`
- `clock_time_t`
- `clock_init()`
- `clock_time()`
- `clock_seconds()`

In order to explain and justify the choices made regarding this driver, it is important to know some details about the specific micro-controller being used in our implementation. This micro-controller is a Texas Instruments family-5 ultra low-power MSP430 (MSP430F5435). This micro-controller is based on a 16-bit RISC processor. The MSP430F5435 has three different clock signals: MCLK, SMCLK, and ACLK. Each of this clock signals can be sourced from different external crystals or from the micro-controller's internal oscillator. In our case, MCLK and SMCLK are sourced from an external high-frequency crystal (32 MHz), while ACLK is sourced from an external 32,768 Hz crystal. Each of the MSP430's functional units requiring a clock signal can be sourced from any of the above clock signals, which can be used directly or divided (usually) by 2, 4, or 8. The divisors can vary however depending on the specific functional unit.

In the clock library, we use the MSP430's Timer A module in order to configure the micro-controller's timer interrupt. The Timer A module is sourced by clock signal ACLK divided by 8. Since ACLK is sourced from a 32,768 Hz crystal, this makes our particular configuration of timer A run at $32,768/8 = 4,096$ Hz.

The Timer A module is configured to execute the timer interrupt routine every 256 ACLK/8 cycles. This value needs to be chosen carefully, considering the trade-off between timer granularity and the frequency at which the timer interrupt routine is executed. In other words, too high a value would cause our minimum possible timer period to be too long, while too low a value would cause the interrupt routine to be executed too frequently, stealing processor cycles from the application or from other micro-controller modules. The interval value of 256 allows for a minimum timer of $(1/4,096)/256 = 1/16$ seconds, this is 62.5 milliseconds (which defines the duration of a system tick). This value seems to be appropriate for our application as it provides a balance between interrupt frequency and use of processor cycles. Note that, since the CPU is driven by MCLK, which operates at 32/2 MHz, i.e., a CPU clock frequency of 16 MHz, executing the interrupt routine once every 1/16 seconds means that this interrupt routine is executed once every 1,000,000 CPU cycles.

Given this discussion an astute reader may have already guessed that our `CLOCK_CONF_SECOND` is 16. Since this is the number of times the interrupt routine has to be executed to measure one second.

Regarding the definition of the `clock_time_t`, the chosen data type is `unsigned long`, which is a 32-bit long data type in our specific architecture. The reason for this choice instead of a 16-bit data type (note that the MSP430 has a 16-bit CPU) is that, in order to compare two timestamps (which is what timers need to do), the maximum distance between two timestamps that can be compared is half the maximum value allowed by the data type. Thus, a 16-bit data type would allow for a maximum timer of $65,535/2 = 32,767$ system ticks. This number of system ticks expressed in seconds is 32,767/16 = 2,047 seconds (or 34 minutes) which may be too small a range for certain timer requirements. In addition, the MSP430 microprocessor is powerful enough to perform a 32-bit addition without its overall performance being negatively affected.

Given all of the above choices, defining the rest of the functions of the Clock library is relatively straightforward: `clock_init()` initializes the Timer A module selecting ACLK divided by 8 as its source and sets a timer interrupt to be executed every 256 ACLK/8 cycles; the timer interrupt is implemented as follows:

```
interrupt void timer_interrupt(void) {

    system_ticks++;

    if (0 == (ticks % CLOCK_SECOND)) {
        seconds++;
    }
    /* If there are Event timers pending,
        notify the event timer module */
    if (etimer_pending()) {
        etimer_request_poll();
    }
}
```

Where `system_ticks` and `seconds` are static variables of type `clock_time_t`. Thus, `clock_time()` returns the value of variable `system_ticks` while `clock_seconds()` returns the value of `seconds`. The final part of the function, as explained in Section 4.1.2, provides support for the Event timer library, requesting a poll on its behalf to the Contiki kernel every time a system tick occurs if there is any pending Event timer.

4.2.1.2 Ethernet Controller Driver

The Ethernet controller driver is split into two layers: the lower one provides the actual hardware abstraction layer while the upper layer implements a Contiki process which requests polls from the Contiki kernel whenever the Ethernet controller signals that there is an incoming packet.

As for the lower layer, the Ethernet micro-controller used is a Microchip's ENC28J60. This micro-controller implements a SPI interface and an instruction set which allows the MSP430 to interact with it. Thus, all the communication between the MSP430 and the ENC28J60 (including Ethernet packets being received or sent out) occurs through this SPI interface and using the ENC28J60's instruction set.

The hardware abstraction layer needs to provide the upper layer with a set of functions to initialize the ENC28J60, and to send and read Ethernet packets. In addition it provides a function to check whether there is any packet pending to be read and an interrupt routine that requests a poll on behalf of its upper-layer's process from the Contiki kernel when a packet arrives (Note that only processes can be polled from the Contiki kernel).

As mentioned above, the upper-layer Ethernet driver implements a running Contiki process that, when polled, checks whether there is an incoming Ethernet packet pending to be read, and, if so, reads it and forwards it to the upper layer (in this case the Ethernet MAC layer). In addition, this upper-layer driver encapsulates lower-layer functions into slightly more complex functions that can be invoked from upper layers. Among this additional functions, this Ethernet driver supports turning on/off the whole Ethernet

operation and subtracts the Ethernet's CRC length from the packet's payload length.

4.2.1.3 Radio Transceiver Controller Driver

While the Ethernet and the radio drivers have many aspects in common, the radio driver has some extra requirements. These requirements are due to the use of a specific IEEE 802.15.4 MAC layer provided by Contiki (*Contikimac*) which implements a radio duty cycling mechanism that allows for power saving by periodically turning the radio receiver off at specific times [20]. Although the radio driver implemented in the 6LP-GW as part of this thesis work fulfils these requirements by providing all the required interface functions (hence, it could be used in conjunction with the *Contikimac* MAC layer), the IEEE 802.15.4 MAC implementation utilised in our application is an earlier version (included in the Contiki 2.4 version) called *Sicslowmac* which does not perform the radio duty cycling mechanism, but keeps the radio receiver on. The reason for this is that the radio duty cycling mechanism saves power by turning off the radio, thus increasing the probability of missing frames during the time it is powered off. Since the 6LP-GW is assumed to be powered by the power mains, we choose not to use this radio receiver duty cycle mechanism.

4.2.1.4 Other Drivers

As previously mentioned, the 6LP-GW implementation developed as part of this thesis project includes several components that, although not required for the purpose of our study, were useful for debugging purposes. Two such components support the board's two buttons and three LEDs. Details about the implementation of these two components are detailed in this this section.

Buttons

The buttons driver can be seen as if it were divided into two different abstraction layers: the lower layer simply checks whether a button is pressed or not, the upper layer performs some operations at the Contiki operating system level.

Regarding the lower layer, two functions, one for each button, return a value other than zero if the corresponding button is pressed and zero otherwise. This is performed simply by checking the value of the pin to which each button is directly connected.

As for the upper layer, the implementation provides a function that permits Contiki *processes* to register themselves to use the buttons. This way, when a button is pressed, all the registered *processes* are notified about it. This notification includes information regarding which of the two buttons has been pressed. By this simple mechanism we enable Contiki *processes* to perform

blocking waits that depend upon on the state of the buttons, without requiring active waits that would block the entire system.

LEDs

The LEDs driver is even simpler than the buttons driver; it simply provides three functions (in addition to the required initialisation function) that permit setting a certain LED on, off, or perform a toggle operation on it depending on a parameter that specifies its color (this identifies the specific LED since the three different LEDs are red, green, and yellow).

4.2.2 MAC layer

The MAC layer of this implementation performs the operations required for the encapsulation/decapsulation and transmission/reception of IP-layer packets over the physical media. The 6LP-GW internetworks two different media types, each of them having different MAC requirements. These two different MAC layers are described in the following paragraphs.

4.2.2.1 IEEE 802.3 MAC layer

Before proceeding to describing the behaviour of this module, it is important to clarify the different types of IP packets that may traverse it. Our implementation has three different sources (and their three corresponding destinations) of IP packets, which will require different treatment. The 6LP-GW will use the IEEE 802.3 MAC layer in order to receive or send packets over the IEEE 802.3 media. Additionally, the local-host implemented in the same device will also use this same functionality. Moreover, this host has a dual stack, which means that either IPv4 and IPv6 packets will be sent or received over through this MAC layer.

The IEEE 802.3 MAC layer implemented operates differently depending on the IP version of the layer-3 packet. In the case of IPv4 packets, this module is responsible for performing link-layer address resolution (which, unlike the case of IPv6, is not performed at the IP layer). For this purpose, an implementation of the ARP protocol provided by Contiki has been used. In particular, the MAC layer will operate as follows regarding IPv4 traffic:

- For outgoing, non-multicast, IPv4 traffic, it will generate the entire link-layer (Ethernet) header depending on the execution of the ARP protocol. If ARP is able to determine the link-layer address of the destination based on the destination IPv4 address, this address will be placed in the destination address field of the Ethernet header. If not, the ARP code will replace the outgoing packet by an ARP request for that address. In either case, the rest of the Ethernet header fields (source link-layer

address and the type/length field) will be filled with the local node's link-layer address and the corresponding ethertype, which may be either the corresponding value for IPv4, or for ARP, depending on whether ARP succeeded into resolving the destination link-layer address or not.

- For incoming, non-multicast IPv4 traffic, the ARP algorithm will update its cache with the pair <IPv4 source address, source link-layer address> if no entry with these values was already present.

- Periodically, the MAC layer will perform maintenance operations on the ARP cache. In order to do so, the IEEE 802.3 MAC layer is implemented as a Contiki *process* which utilizes a periodic event timer.

In contrast, IPv6 traffic does not require address resolution. Indeed, the task of address resolution is performed by the ND protocol and therefore it occurs at the IP layer. If a packet's IPv6 source or destination address corresponds to the local host, its ND Address Resolution algorithm will handle the details regarding link-layer address resolution appropriately. If the IPv6 packet has been generated by or is destined to the 6LP-GW, then the 6LP-GW module itself will take appropriate care of the link-layer addresses.

In all cases, the IEEE 802.3 MAC implementation will multiplex and forward incoming traffic to the required upper-layer module, which may be either the IPv4 or IPv6 stacks of the local node, or the 6LP-GW module.

4.2.2.2 IEEE 802.15.4 MAC layer

Regarding the IEEE 802.15.4 MAC layer, few changes have been made to the code provided by Contiki. Contiki provides several MAC layers, each of them implementing different features. The one utilised by this implementation is called "sicslowmac" and it performs the basic operations regarding the IEEE 802.15.4 MAC layer. For outgoing packets, it generates the contents of the link-layer header; for incoming packets, it parses the contents of the IEEE 802.15.4 header (discarding any malformed packets) and forwards the packets to the appropriate upper layer, which in this case is always the 6LoWPAN Adaptation layer.

4.2.3 6LoWPAN Adaptation layer

The 6LoWPAN Adaptation layer is located between the IEEE 802.15.4 MAC layer and the 6LP-GW module. As will be explained in Section 4.2.4, the IPv6 layer of the local-host stack does not have direct access to the 6LoWPAN layer.

As we already mentioned, Contiki includes an implementation of the 6LoWPAN adaptation layer. However this implementation lacks certain features regarding the stateful IPv6 header compression feature specified in RFC 6282 [27] that were of interest for this thesis project. In particular,

what is missing in the original code is a dynamic mechanism to add/remove contexts, and code to utilize such dynamically added contexts for stateful compression/decompression. In addition, the 6LoWPAN implementation only allows the use of 64-bit fixed-length contexts, whereas RFC 6282 does not impose this limitation and I-D.ietf-6lowpan-nd [43] permits the dissemination of variable-length contexts of up to 128 bits. Thus, some modifications were made to the original Contiki's 6LoWPAN code in order to permit the utilisation of arbitrarily long, dynamically acquired contexts.

Apart from the changes described above, the rest of the 6LoWPAN code included in the implementation of the 6LP-GW is mostly the same as the original code and performs the following tasks:

- For outgoing traffic, the 6LoWPAN layer compresses the IPv6 header and forwards the packet to the IEEE 802.15.4 MAC layer.
- For incoming traffic, the IPv6 header is uncompressed and the packet is forwarded to the 6LP-GW pseudo layer.

4.2.4 The 6LP-GW pseudo-layer

The 6LP-GW's operation is described in detail in Chapter 3. Its implementation comprises the functions that handle packet forwarding and ND-proxying. This module is divided into three different sub-modules: the Forwarding module, the ND module, and the Proxy module. Each of these performing their tasks following as described below:

The Forwarding module

As its name suggests, the Forwarding module comprises the functions that handle the tasks regarding packet forwarding. Basically, this module is in charge of performing initial processing of incoming traffic, as well as sending out outgoing packets.

Regarding the incoming traffic operations, it performs the basic bridging operations described in Section 3.2.3. In addition it translates any link-layer addresses that might be present in the payload (mainly in ND packets) when necessary. Moreover, incoming packets are passed through a filter that discards all IPv6 traffic which is not UDP or ICMPv6.

Processing of outgoing packets consists of multiplexing these packets to the appropriate interface according to their origin and destination. This process also involves translating link-layer addresses (as explained for incoming packets) when necessary.

Note that the local-host is considered to be attached to a virtual Ethernet interface of the 6LP-GW, and thus this host must also be taken into account for this processing of incoming and outgoing packets. In fact, this is the reason why the 6LoWPAN Adaptation layer does not have direct access to the

IPv6 module: it is the 6LP-GW (in particular, the Forwarding sub-module) who multiplexes all the incoming IPv6 traffic to its corresponding destination, which may be the local-host, a remote host, or none of them (e.g. if the proxy operations determine that such a packet needs to be replaced by another packet).

It is important note also that the process of determining the source and destination link-layer addresses is performed according to the information gathered by the bridging function, which might be modified by the Proxy operation.

The ND module

This ND module provides all the functions and data structures required to perform the tasks regarding 6LoWPAN-ND that correspond to a 6LR, as specified in I-D.ietf-6lowpan-nd. This means that this module provides the facilities to handle and maintain the NC, and the contexts table. Although not being strictly part of the 6LoWPAN-ND specification, this module also performs the operations regarding the DAD mechanism when performed on behalf of 6LHs. The reasons to include these functionality here is that, apart from being related to the ND protocol, the results (either positive or negative) of DAD imply generating 6LoWPAN-ND responses (see Section 3.2.4.1).

The Proxy module

The Proxy module is the main sub-module among the three comprising the 6LP-GW implementation. It implements a Contiki *process* that is in charge of performing ND-proxying, which means that it carries out the operations described in Section 3.2.4, besides controlling the other two sub-modules for the required tasks. Figure 4.4 illustrates the relationship between these three modules.

4.2.5 Network Layer

Regarding the network layer, few changes were made to the IPv4 and IPv6 implementations that come with Contiki. These few modifications are mainly related to the way packets get into or out from the stacks rather than to their actual implementation. The reason for this is that, as mentioned in Section 4.1.3, Contiki does **not** support the use of the IPv4 and IPv6 stacks simultaneously.

In addition, both the stacks reuse certain variables and functions (using the same names) when used separately.When possible, these variables and functions have been made available for both stacks simultaneously, allowing their shared use. This is the case of the variables `uip_buf` and `uip_len`, which hold the buffer for both incoming and outgoing traffic, and the length of the

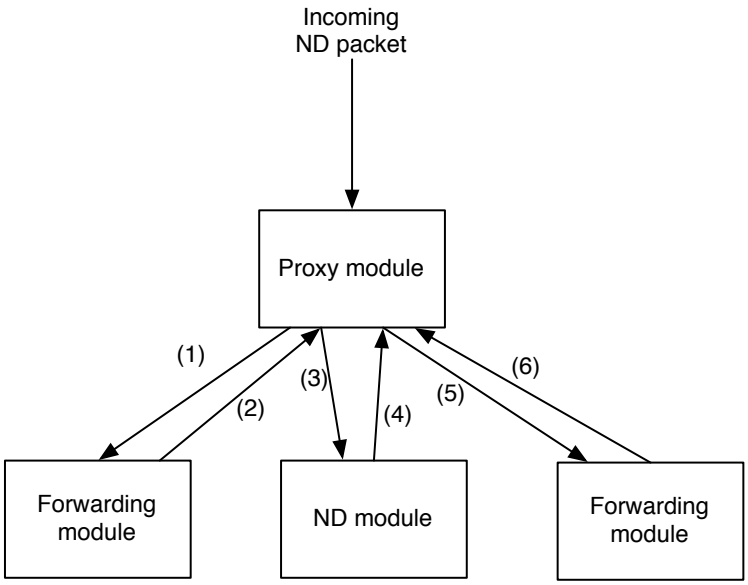

Figure 4.4: The 6LP-GW module architecture. The numbers in the arrows indicate the order in which each module's operations are invoked for the event of an incoming ND packet. The forwarding module appears twice because it is normally required to handle the reception and dispatch of packets.

data contained in it respectively. When this shared use has not been possible, a duplication and renaming has been applied.

The data-flow between both IP stacks and the modules lying immediately above or beneath them in the communications stack is illustrated in Figure 4.3 on page 70.

4.2.6 Application Layer

The only application implemented in the application layer as part of this thesis project is a DHCP client. Contiki provides the core functions required for this DHCP client. It was only necessary to implement a Contiki *process* to handle specific events (arrival of packets or expiration of a timer) and two callback functions. Of these functions, one will be called if the DHCP client succeeds in acquiring an IPv4 address while the other will be invoked in the event of failing to renew the lease of the assigned address with the DHCP server.

Chapter 5

Analysis

This chapter describes the different procedures that have been used to evaluate the application's compliance with its specified requirements in terms of both correctness and efficiency.

5.1 Method for Evaluation

The evaluation carried out in this thesis is comprised of two different tests. In order to verify that the application is **correct** (i.e., given a specific state and a certain input, it generates the expected output), we define a set of use cases (test cases) and then run a test to validate the application for each of them. Note that these tests, however, are only useful to verify the functional requirements, not providing information about non-functional requirements.

On the other hand, the main non-functional requirement of our application is to have **acceptable processing times**, which should be short enough so that the 6LP-GW does not pose any throughput-limiting factor in the network.

5.1.1 Use cases test

This test will cover most possibilities regarding ND packets that may arrive at the 6LP-GW. Additionally, these tests consider the different situations, i.e., the states in which the 6LP-GW may be, different options and option values in these ND packets, and the interface on which these packets were received. In those cases where a certain parameter would have no influence on the results, the value of this parameter may be not taken into account for the specific use case under consideration. For details about the use cases and the expected results, refer to Chapter 3.

This test will **not** consider the case of erroneous or malformed packets. the reasons for not considering these types of packets are the fact that the number of different erroneous packets that could be generated is nearly countless, and that the correctness of the behaviour of the 6LP-GW upon arrival or erroneous packets can be easily inferred by studying the source code. Thus, the proper behaviour of the 6LP-GW under these circumstances (i.e., silently discarding malformed of erroneous packets) shall be assumed.

The use cases under which the 6LP-GW will be tested to operate correctly are the following:

1. Receiving RS messages:

    a) Test: Receiving RS messages on the Ethernet interface.

Expected result: The RS message should be silently discarded and no output should be generated in response.

b) Test: Receiving a RS on the IEEE 802.15.4 interface from the unspecified address.

Expected result: Same as in previous test.

c) Test: Receiving a RS on the IEEE 802.15.4 interface without a SLLAO option.

Expected result: Same as in previous test.

d) Test: Receiving a RS on the IEEE 802.15.4 interface from an address different than the unspecified address which has no matching entry in the NCE and with a valid SLLAO option, if there is no space left in the NCE.

Expected result: Same as in previous test.

e) Test: Receiving a RS on the IEEE 802.15.4 interface from an address different than the unspecified address and with a valid SLLAO option.

Expected results: The RS message should be forwarded with the appropriate link-layer address translation.

2. Receiving RA messages:

a) Test: Receiving a RA on the IEEE 802.15.4 interface.

Expected result: The RA message should be silently discarded and no output should be generated in response.

b) Test: Receiving a RA on the Ethernet interface under the following conditions:

- No 6LH has previously sent a RS.
- No PIO option is present in the RA message.

Expected result: Same as in previous test.

c) Test: Receiving a RA on the Ethernet interface under the following conditions:

- No 6LH has previously sent a RS.
- A valid PIO option which had already been received in previous RA messages is present in the current RA message.

Expected result: Same as in previous test.

d) Test: Receiving a RA on the Ethernet interface under the following conditions:

- At least one 6LH has previously sent a RS.
- A valid PIO option which had already been received in previous RA messages is present in the current RA message.

- The 'L' (on-link) flag of the PIO option is set to 1.
- There is no SLLAO option present in the RA message.

Expected result: The RA message should be unicasted to each the of 6LH that are known to have solicited a RA. The 'L' (on-link) flag of the PIO option should be set to 0 in the outgoing RA messages. The RA message should contain 6CO options corresponding to known (previously announced and whose lifetime has not yet expired) prefixes. In addition the RA message should contain a SLLAO option whose link-layer address matches the RR's link-layer address.

e) Test: Receiving a RA on the Ethernet interface under the following conditions:

- At least one 6LH has previously sent a RS.
- A valid PIO option which had already been received in previous RA messages is present in the current RA message.
- The 'L' (on-link) flag of the PIO option is set to 0.
- There is no SLLAO option present in the RA message.

Expected result: Same as in previous test.

f) Test: Receiving a RA on the Ethernet interface under the following conditions:

- At least one 6LH has previously sent a RS.
- A valid PIO option which had already been received in previous RA messages is present in the current RA message.
- The 'L' (on-link) flag of the PIO option is set to 1.
- There is a valid SLLAO option present in the RA message.

Expected result: Same as in previous test.

g) Test: Receiving a RA on the Ethernet interface under the following conditions:

- At least one 6LH has previously sent a RS.
- A valid PIO option which had already been received in previous RA messages is present in the current RA message.
- The 'L' (on-link) flag of the PIO option is set to 0.
- There is a valid SLLAO option present in the RA message.

Expected result: Same as in previous test.

h) Test: Receiving a RA on the Ethernet interface under the following conditions:

- A valid PIO option which had not been received before in previous RA messages is present in the current RA message.

- The 'L' (on-link) flag of the PIO option is set to 1.
- There is no SLLAO option present in the RA message.

Expected result: A RA message containing a 6CO option whose context matches the prefix announced in the RA (along with other 6COs corresponding to other advertised prefixes, if any) should be multicasted to every node in the IEEE 802.15.4 segment. The 'L' (on-link) flag of the PIO option should be set to 0 in the outgoing RA message. In addition the RA message should contain a SLLAO option whose link-layer address matches the RR's link-layer address.

i) Test: Receiving a RA on the Ethernet interface under the following conditions:

- A valid PIO option which had not been received before in previous RA messages is present in the current RA message.
- The 'L' (on-link) flag of the PIO option is set to 0.
- There is no SLLAO option present in the RA message.

Expected result: Same as in previous test.

j) Test: Receiving a RA on the Ethernet interface under the following conditions:

- A valid PIO option which had not been received before in previous RA messages is present in the current RA message.
- The 'L' (on-link) flag of the PIO option is set to 1.
- There is a valid SLLAO option present in the RA message.

Expected result: Same as in previous test.

k) Test: Receiving a RA on the Ethernet interface under the following conditions:

- A valid PIO option which had not been received before in previous RA messages is present in the current RA message.
- The 'L' (on-link) flag of the PIO option is set to 0.
- There is a valid SLLAO option present in the RA message.

Expected result: Same as in previous test.

3. Receiving NS messages:

a) Test: Receiving a NS on the Ethernet interface from the unspecified address under the following conditions:

- There is no NCE corresponding to a 6LH whose IPv6 address matches the target address contained in the NS message.

Expected result: The NS message should be silently discarded and no output should be generated in response.

b) Test: Receiving a NS on the Ethernet interface from the unspecified address under the following conditions:

- There is a NCE corresponding to a 6LH whose IPv6 address matches the target address contained in the NS message and whose state is REGISTERED.

Expected result: A NA message with the same target address as the incoming NS should be generated and sent to the all-nodes multicast address on the Ethernet segment.

c) Test: Receiving a NS on the Ethernet interface from the unspecified address under the following conditions:

- There is a NCE corresponding to a 6LH whose IPv6 address matches the target address contained in the NS message and whose state is TENTATIVE.

Expected result: A NA message having the RR's IPv6 address as its target address should be generated and sent to the 6LH in TENTATIVE whose IPv6 address matched the target address of the received NS. Such NA should contain an ARO option with status code 1.

d) Test: Receiving a NS on the Ethernet interface from the unspecified address under the following conditions:

- There is a NCE corresponding to a 6LH whose IPv6 address matches the target address contained in the NS message and whose state is GARBAGE-COLLECTIBLE

Expected result: The NS message should be silently discarded and no output should be generated in response.

e) Test: Receiving a NS on the Ethernet interface from an address other than the unspecified address under the following conditions:

- There is no NCE corresponding to a 6LH whose IPv6 address matches the target address contained in the NS message.

Expected result: The NS message should be silently discarded and no output should be generated in response.

f) Test: Receiving a NS on the Ethernet interface from an address other than the unspecified address under the following conditions:

- There is a NCE corresponding to a 6LH whose IPv6 address matches the target address contained in the NS message and whose state is REGISTERED.

Expected result: A NA message with the same target address as the incoming NS should be generated and unicasted in response to the originator of the solicitation.

g) Test: Receiving a multicast NS on the IEEE 802.15.4 interface from the unspecified address under the following conditions:

- The NS message does not include an ARO option.
- The NS message does not include an SLLAO option.

Expected result: The NS message should be forwarded unchanged (except for the appropriate link-layer address translation) to the Ethernet interface.

h) Test: Receiving a NS on the IEEE 802.15.4 interface from the unspecified address under the following conditions:

- The NS message does not include an ARO option.
- The NS message does not include an SLLAO option.

Expected result: Same as in previous test.

i) Test: Receiving a NS on the IEEE 802.15.4 interface from the unspecified address under the following conditions:

- The NS message does not include an ARO option.
- The NS message includes include an SLLAO option.

Expected result: Same as in previous test.

j) Test: Receiving a NS on the IEEE 802.15.4 interface from the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message does not include an SLLAO option.

Expected result: The ARO option in the NS message is ignored and removed prior to forwarding the NS message (with the appropriate link-layer address translation) to the Ethernet interface.

k) Test: Receiving a NS on the IEEE 802.15.4 interface from the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.

Expected result: Same as in previous test.

l) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message does not include an ARO option.
- The NS message does not include an SLLAO option.

Expected result: The NS message should be forwarded unchanged (except for the appropriate link-layer address translation) to the Ethernet interface.

m) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message does not include an SLLAO option.

Expected result: The ARO option in the NS message is ignored and removed prior to forwarding the NS message (with the appropriate link-layer address translation) to the Ethernet interface.

n) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message does not include valid ARO option.
- The NS message includes a valid SLLAO option.

Expected result: The NS message should be forwarded unchanged (except for the appropriate link-layer address translation) to the Ethernet interface.

o) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.
- Either the RR's IPv6 address or the RR's link-layer address are not known by the 6LP-GW.

Expected result: The NS message should be silently discarded and no output should be generated in response.

p) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.
- Both the RR's IPv6 address and the RR's link-layer address are known by the 6LP-GW.
- The destination IPv6 address, target IPv6 address and the destination link-layer address match the corresponding RR's addresses.
- The 6LP-GW does not contain NCE corresponding to the originator of the solicitation.
- There is no space left in the NC

Expected result: A NA whose target is copied from the solicitation and its IPv6 address is the RR's IPv6 address is generated and sent to the originator of the NS. This NA contains an ARO option copied from the solicitation but containing a status value of 2 (NC full).

q) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.
- Both the RR's IPv6 address and the RR's link-layer address are known by the 6LP-GW.
- The destination IPv6 address, target IPv6 address and the destination link-layer address match the corresponding RR's addresses.
- The 6LP-GW does not contain NCE corresponding to the originator of the solicitation.
- There is space left in the NC.

Expected result: A NS having the unspecified address as its source address and the RR's IPv6 address as target is generated and multicasted to the all-nodes multicast address on the Ethernet interface. This NS contains no options.

r) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.
- Both the RR's IPv6 address and the RR's link-layer address are known by the 6LP-GW.
- The destination IPv6 address, target IPv6 address and the destination link-layer address match the corresponding RR's addresses.
- The 6LP-GW contains NCE corresponding to a 6LH with an IPv6 address that matches the source of the solicitation but with a different link-layer address than the one contained in the EUI-64 field of the ARO option.

Expected result: A NA whose target is copied from the solicitation and its IPv6 address is the RR's IPv6 address is generated and sent to the originator of the NS. This NA contains an ARO option copied from the solicitation but containing a status value of 2 (NC full).

s) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.
- Both the RR's IPv6 address and the RR's link-layer address are known by the 6LP-GW.
- The destination IPv6 address, target IPv6 address and the destination link-layer address match the corresponding RR's addresses.
- The 6LP-GW contains NCE corresponding to a 6LH with an IPv6 address that matches the source of the solicitation but with a different link-layer address than the one contained in the EUI-64 field of the ARO option.

Expected result: A NA whose target is copied from the solicitation and its IPv6 address is the RR's IPv6 address is generated and sent to the originator of the NS. This NA contains an ARO option copied from the solicitation but containing a status value of 1 (duplicate).

t) Test: Receiving a NS on the IEEE 802.15.4 interface from an address other than the unspecified address under the following conditions:

- The NS message includes a valid ARO option.
- The NS message includes a valid SLLAO option.
- Both the RR's IPv6 address and the RR's link-layer address are known by the 6LP-GW.
- The destination IPv6 address, target IPv6 address and the destination link-layer address match the corresponding RR's addresses.
- The 6LP-GW contains NCE corresponding to a 6LH with an IPv6 address that matches the source of the solicitation and a the link-layer address matching the one contained in the EUI-64 field of the ARO option.

Expected result: A NA whose target is copied from the solicitation and its IPv6 address is the RR's IPv6 address is generated and sent to the originator of the NS. This NA contains an ARO option copied from the solicitation but containing a status value of 0 (success).

4. Receiving NA messages:

a) Test: Receiving a multicast NA message on the Ethernet interface under the following conditions:

- There is no NCE corresponding to a 6LH that has the same IPv6 address as the one contained in the target of the advertisement.

Expected result: The NA message should be silently discarded and no output should be generated in response.

b) Test: Receiving a multicast NA message on the Ethernet interface under the following conditions:

- There is a NCE corresponding to a 6LH that has the same IPv6 address as the one contained in the target and whose state is GARBAGE-COLLECTIBLE.

Expected result: Same as in above test.

c) Test: Receiving a multicast NA message on the Ethernet interface under the following conditions:

- There is a NCE corresponding to a 6LH that has the same IPv6 address as the one contained in the target and whose state is REGISTERED.

Expected result: The NA is forwarded unchanged (except for the appropriate link-layer address translation) on the IEEE 802.15.4 interface.

d) Test: Receiving a multicast NA message on the Ethernet interface under the following conditions:

- There is a NCE corresponding to a 6LH that has the same IPv6 address as the one contained in the target and whose state is TENTATIVE.

Expected result: A new NA is generated and sent to the link-local address of the 6LH corresponding to the NCE whose address matches the target. Such NA has its source address copied from the RR's address, an SLLAO option containing the RR's link-layer and an ARO option with status code 1 (duplicate).

e) Test: Receiving a unicast NA message on the Ethernet interface under the following conditions:

- The destination of the NA has not a corresponding NCE in the NC.

Expected result: The NA message should be silently discarded and no output should be generated in response.

f) Test: Receiving a unicast NA message on the Ethernet interface under the following conditions:

- The destination of the NA has a corresponding NCE in the NC in TENTATIVE state.

Expected result: Same as in previous test.

g) Test: Receiving a unicast NA message on the Ethernet interface under the following conditions:

- The destination of the NA has a corresponding NCE in the NC in GARBAGE-COLLECTIBLE state.

Expected result: Same as in previous test.

h) Test: Receiving a unicast NA message on the Ethernet interface under the following conditions:

- The destination of the NA has a corresponding NCE in the NC in REGISTERED state.
- The NCE corresponding to the destination of the NA has its ARO-pending flag set to zero.

Expected result: The NA is forwarded unchanged (except for the appropriate link-layer address translation) on the IEEE 802.15.4 interface.

i) Test: Receiving a unicast NA message on the Ethernet interface under the following conditions:

- The destination of the NA has a corresponding NCE in the NC in REGISTERED state.
- The NCE corresponding to the destination of the NA has its ARO-pending flag set to one.

Expected result: The NA is forwarded on the IEEE 802.15.4 interface (applying the appropriate link-layer address translation). Such NA is appended an ARO option whose EUI-64 field contains the link-layer address stored in the NCE and its status code is 0 (success).

5.1.2 Processing time and throughput measurement

In order to measure the 6LP-GW's processing time, we will utilize a common technique that consists of sending two back-to-back packets (with a minimum inter-packet gap) to the 6LP-GW and measuring the time between them being output. The difference in time between these packet outputs represents the processing time of the second packet.

In particular, the 6LP-GW will receive 93 pairs of back-to-back UDP packets, each pair having an increasing payload size. This payload size will range from 1 to 93 bytes, which is the maximum payload size we can allocate on a IEEE 802.15.4 frame, given the overhead imposed by each of the different protocols in our particular scenario. This scenario will consist of a 6LH, a NCD, and the 6LP-GW implemented on the Hogaza v1.2 prototype board [50]. Although all the nodes are within the same subnet, the IPv6

packets will be sent to the global address of these devices. The reasons for using global addresses instead of link-local addresses (considering that they all are within the same subnet) are two: testing the proper handling of the devices' autoconfiguration by the 6LP-GW, and testing and taking into account the stateful compression/uncompression feature, which processing times may affect the measurements taken during the evaluation process. The UDP packets will be sent to/from port numbers within the 0xF0B0 to 0xF0BF range, which are the port numbers allowed to be compressed down from 16 to 4 bits by the RFC 6282 specification.

In addition, in order to alleviate possible measurement flaws, this test will be repeated 100 times for each of the 93 payload lengths. Of these 100 measurements, the highest and the lowest values will be removed, while the rest will be used to calculate the average processing time.

Since the 6LP-GW behaves differently depending on both the type of packet being received and the direction in which packets are forwarded, the two packets comprising the back-to-back pair must be equal and the measurements will be computed for packets originating in a (wireless) 6LH and sent to a neighbouring (wired) NCD through the 6LP-GW and vice-versa.

5.2 Analysis of metric results

5.2.1 Use cases

After testing with each of the use cases described in Section 5.1.1, the outcome was that the result obtained matched the expected result in 100% of the cases. Such results indicate the correctness of the implemented application.

Thus, we can conclude that the tests' results show the proper behaviour of the 6LP-GW for these use cases. However, it is important to note that the set of use cases analysed, although complete enough for the purpose of providing a general evaluation of the behaviour of the implementation, is far from complete in comparison with the total number of **actual** possible cases that may occur in practice. Naturally, exhaustive testing of the total number of possible cases is, in practice, not feasible due to the extremely large amount of them.

5.2.2 Processing time and throughput

The performance tests conducted of the 6LP-GW implementation with payloads ranging from 1 to 93 octets showed that the processing times range between 1.761 ms and 5.047 ms for the case of packets traversing the device from the IEEE 802.15.4 port towards the Ethernet interface, and between 2.851 ms and 6.756 ms for the case of packets traversing the device in the opposite direction. This leads to average processing time of 3.404 ms for the former case and 4.804 ms for the latter. The average time considering both

directions is 4.104 ms. These results, along with the average times can be observed in Figures 5.1 and 5.2

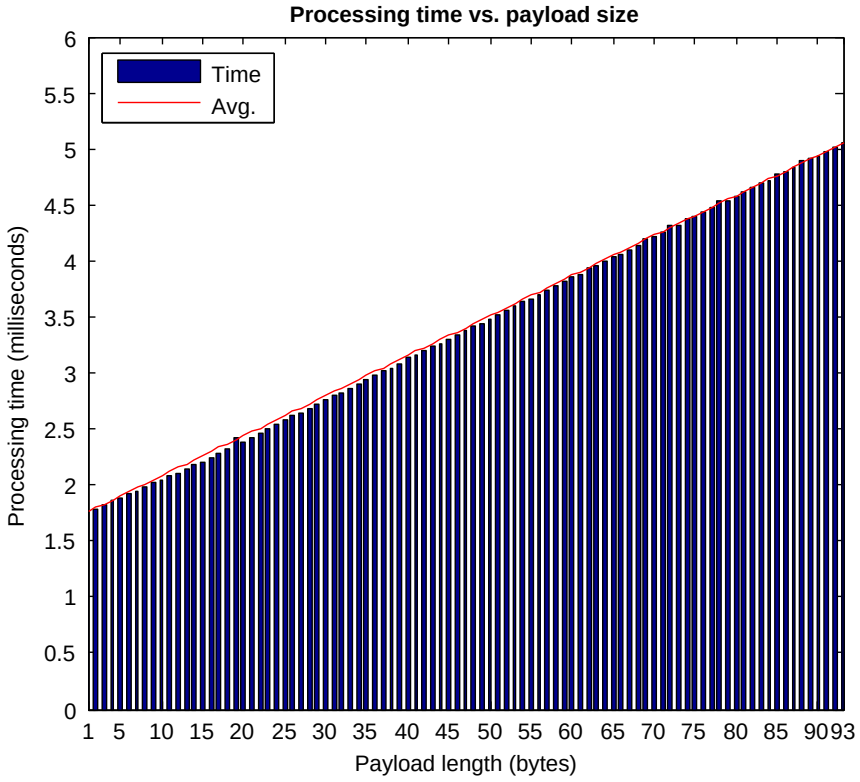

Figure 5.1: Radio to Ethernet performance test. The graph shows the processing time of packets depending on their payload length.

As these results show, the processing times for the case of packets being forwarded from the Ethernet interface to the IEEE 802.15.4 interface are slightly longer (roughly about 1.4 ms) than the times of packets being forwarded in the opposite direction. Figure 5.3 illustrates these differences in the measured times.

A reason for this is the fact that the radio transceiver is always in reception mode, and only switches to transmission mode when there is packet is to be transmitted. The time to switch between reception and transmission is specified in the IEEE 802.15.4 standard [4] (and thus in CC2520 datasheet) as the transmission (TX) turnaround time, which is 192 μs. The time it takes to ramp down the signals to switch back from TX to reception (RX) is only 2 μs, which we can consider irrelevant to our measurements. Thus, in order to transmit two back to back packets, the 6LP-GW behaves as follows:

1. The 6LP-GW switches from RX to TX mode in order to transmit the

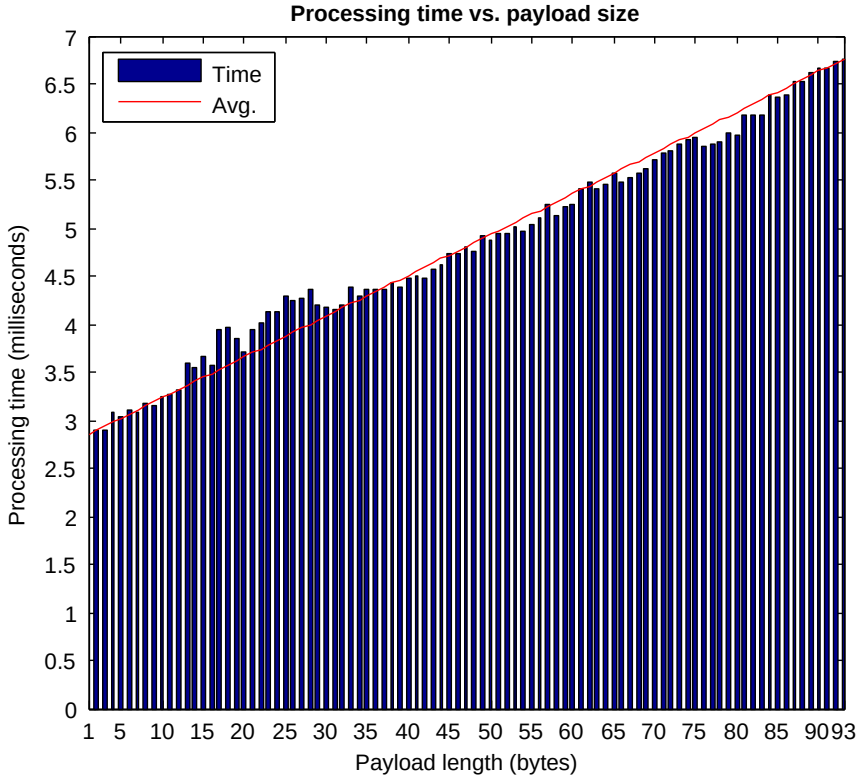

Figure 5.2: Ethernet to radio performance test. The graph shows the processing time of packets depending on their payload length.

first packet → 192 μs

2. After transmitting, the 6LP-GW switches back from TX to RX → 2 μs
3. The 6LP-GW switches again from RX to TX to transmit the second packet → 192 μs

Although this explains only a constant shift in the measured time of 386 μs, there are other factors that affect our transmission timings. For instance, prior to begin a transmission, the transmitting code waits until the received signal strength indication (RSSI) becomes valid, and then, it sends the command to transmit the packet to the transceiver and, more importantly it **waits** until the transmission has completed prior to continue executing further tasks. This *active wait* contrasts with the way that reception is performed, as in reception the complete packet is received without interrupting the microprocessor until the whole packet has actually been received.

Given that all the packets are identical (except for the FCS in the IEEE 802.15.4 header, the UDP checksum in the UDP header, and the

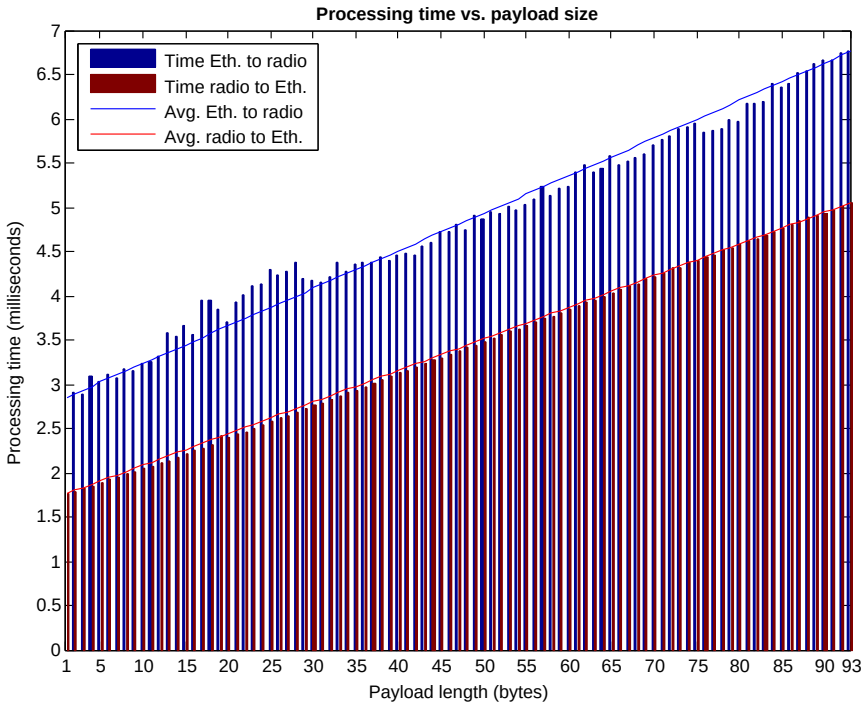

Figure 5.3: Comparison of measured times depending on the forwarding direction

payload, which only varies in its length), we can easily calculate the marginal cost in time of processing an additional byte. Considering that the processing time is not exactly a linear function of the payload length, we calculate the time it takes to process 1 byte as the average of the differences between the processing times of packets having consecutive lengths:

$$time_{byte} = \frac{\sum_{i=2}^{93} (x_i - x_{i-1})}{93}$$

This processing time per additional payload byte is 0.035 ms in the case of the Ethernet-to-radio traffic and 0.042 in the opposite direction. The average processing time per additional byte is hence 0.039 ms.

Given these measurements, we can calculate our processing bit rate as the inverse of the time it takes to process an additional byte divided by 8 bits per byte:

$$bitrate = \frac{1}{\frac{time_{byte}}{8}} = 204.603 \quad kb/s$$

The theoretical highest data rate achievable in ideal conditions by IEEE 802.15.4 transceivers in the 2.4 GHz band is 250 kb/s. Considering

that the conditions under which the performance tests were conducted are far from ideal, we conclude that the results achieved are quite satisfactory.

Chapter 6

Conclusions

This chapter explains the conclusions obtained throughout the design, development, and evaluation described in this thesis and proposes a number of improvements, extensions, or complements that may be of interest in order to continue this work.

6.1 Conclusions

In this section we will state the conclusions and insights gained as result of this thesis project.

6.1.1 Goals

In the beginning of this report, we enumerated the different goals of this thesis project (Chapter 1, Section 1.3). In this section, we will analyse the results obtained regarding each of these goals.

The first and foremost goal of this thesis was to investigate if integrating 6LoWPAN devices into existing IPv6 networks *without* requiring a single modification in the existing network infrastructure was a feasible task. As the results show, this task is not only feasible, but also efficient and cost-effective (given the hardware costs described in [50]).

Use case testing of the 6LP-GW has demonstrated that it behaves properly and efficiently in its two main tasks: internetworking the two different link-layers present in each of the network segments under consideration (IEEE 802.15.4 and IEEE 802.3), and operating as a proxy between the two versions of the ND protocol (IPv6-ND and 6LoWPAN-ND). Both the efficiency in terms of throughput and effective bit rate as well as the proper behaviour of the proxy operations have been tested as described in Chapter 5.

Additionally, the inexpensive hardware components comprising the embedded platform (Hogaza v1.2) together with the software implementation of the 6LP-GW have proven to allow the integration of 6LoWPAN devices in standard networks in a cost-effective and simple manner, requiring no changes in the existing infrastructure nor further intervention beyond plugging in the device implementing the 6LP-GW into a (home) router's Ethernet jack.

A secondary goal was to provide the required resources in the 6LP-GW so that further integration of new features is possible. This task has been carried out by the implementation of a dual-stack-enabled host that resides within the same device, sharing resources with the 6LP-GW and virtually connected to its Ethernet interface. This host utilizes IPv6-ND as its ND protocol,

and behaves as any other NCD in the network except for its constrained capabilities. In order to test the proper behaviour of this host, two different applications have been loaded and run concurrently on it: a DHCP client (using IPv4) and a CoAP [44] server (using IPv6). Although no formal tests have been conducted regarding the operation of these two applications (this testing is outside the scope of this thesis), they have behaved in practice according to their respective expected behaviours. While the incorporation of this virtual local-host in the same constrained device in which the 6LP-GW runs may cause decreased performance of the latter, all the tests carried out and described in Chapter 5 have been performed while this host (with the two applications previously mentioned) was also enabled and running. Given the results obtained in these tests, we can state that the performance of the 6LP-GW is satisfactory enough even if the presence of the local-host decreases the performance of the 6LP-GW.

Finally, a side goal of this thesis was the implementation and integration into the Contiki operating system of the 6LoWPAN-ND protocol for a host, as described in I-D.ietf-6lowpan-nd [43]. As explained in Chapter 1, this was a requirement in order to test the proxy operation of the 6LP-GW, since no implementation of this protocol was available when this thesis was first proposed. The resulting implementation has been shown to work well and has been tested in several platforms such as the MSP430-based Hogaza v1.2 [50], Damper v1.0 [50], and Torrija v1.1 [41], TI's CC2531-based evaluation board CC2531EMK v2.0, and a Cortex-M3-based board. The implementation has been shown to behave properly in all these devices, allowing interoperation between them and the 6LP-GW. However, interoperability between this implementation and another implementation is still untested due the limited number of implementations of the 6LoWPAN-ND protocol currently available.

6.1.2 Insights and suggestions for further work

Throughout this thesis project, comprising initial research, specification, implementation, and analysis, there have been several situations requiring reflection or involving decisions. A retrospective observation of the overall process from its very beginning leads to a number of insights which are worth mentioning as they may be useful advice for future work that might build upon the results of this project.

It is probably needless to mention how important the study of the previous literature is for any task involving a certain level of complexity. The importance of the literature study lays not only in the need for specific knowledge in order to start the implementation; research is likely to reveal that certain tasks have already been completed, hence exploiting this knowledge may enable us to save a deal of time, or make it possible to reject a certain method in favour of another in advance, instead of having to implement all the alternatives. Even if the study of previous work reveals that the whole

project has already been done by someone else, or that it is simply pointless for some reason, this knowledge is valuable.

Far from being an exception, this thesis has required studying, understanding, and nearly memorizing certain documents (mainly RFCs and Internet Drafts). Moreover, the literature study is likely to continue beyond the initial stage of any project, occurring all through the project. It is widely known how fast the field of computer science develops as new methods or technologies are introduced. In the particular case of this thesis, this fact is especially important due to the role that certain Internet Drafts (i.e., specifications that are still under development) have played.

Other examples of the spread of the research throughout the whole project are the importance that a proper understanding of third-party software documentation or specific hardware datasheets has played in this project. In the development of embedded software, efficiency is not only a desirable feature, but in fact the constrained nature of the platform and the scarceness of resources makes efficiency a mandatory requirement. Failing to meet this requirement may cause the exhaustion of a certain resource, leading to the impossibility to complete the project and forcing a huge step back in the development process. Producing efficient and optimal code in an embedded platform requires deep knowledge and understanding of any involved third-party software and/or hardware, which can only be acquired by means of studying the available documentation for each of the components involved.

As for the development process, it is essential to remark on the importance of a careful and thorough plan for the software architecture prior to the actual coding in order to generate a high quality application. As before, this planning acquires even more importance when, as in our case, third-party software is involved in the implementation. As previously mentioned, this requires studying and understanding in detail the third-party software's documentation (which sometimes may not be available or sufficiently complete), for its integration in the software under development. A poorly planned implementation will certainly cause software flaws regarding modularity, scalability, maintainability, and readability in a best-case scenario. This normally leads to a longer development process as a change in a certain software module forces changes in many other places and leads to so-called spaghetti code, which is likely to contain dead code, be inefficient, and a potential source of bugs. In addition, it is important to remark that developing embedded code poses a major challenge regarding the usual processing time versus RAM consumption trade-off which developers are usually forced to face. In this case, the constrained nature of the platform in which the software is to be run tips the balance in favour of RAM in most cases, but still this is something that has to be analysed for each particular case.

Regarding the analysis task, careful planning is also very important. It is fundamental to clearly state what features are to be analysed and what tests should be performed in order to test these features. In addition, as a suggestion

for future work, it is worth mentioning that the analysis is not something that should be left for the final phase of a project: for instance, performing periodic analysis at different stages of a given project may be useful to discard a certain approach in favour of another method of performing a certain task.

6.2 Future work

Due to the breadth of the field of our study, there have been several aspects that have been left out of scope of this thesis. Although sufficiently complete for the purpose of our study, the implemented device has certain deficiencies that may limit its usability in a real environment. In addition, there are several improvements that could be made to the current implementation in order to broaden its functionality or applicability. This section enumerates some of these features that have been left out or work that has been left undone, together with some related work that may be of interest for future thesis projects.

6.2.1 What has been left undone?

Below we describe a number of improvements that might be made to the current implementation of the 6LP-GW.

6.2.1.1 Loop Avoidance

One of the main features that has not been implemented and could cause malfunction of the device in certain environments is a loop-avoidance mechanism. The 6LP-GW performs packet forwarding at layer two (although with some support for upper layers). Thus, the 6LP-GW is subject to the same loop problems that bridges had initially (prior to the invention of the spanning-tree protocol (STP) [1]) if more than one 6LP-GWs are present and in range with each other in the same network segment. The obvious solutions to this issue are either the implementation of the STP protocol (or its optimized version, the rapid spanning-tree protocol (RSTP) [2]), or the implementation of a layer-3 routing protocol such as the Routing Protocol for LLNs (RPL) [49]. However, whatever the choice be, special considerations have to be taken.

In the case of the implementation of a layer-2 loop-avoiding algorithm such as the STP or RTSP, it is fundamental to do it in such a way that no extra multicast traffic is produced in the IEEE 802.15.4 segment. We have already stated the disadvantages of excessive multicast traffic in IEEE 802.15.4 nodes (see Chapter 1, Section 1.1). Indeed, as we have also mentioned, one of the main goals of 6LoWPAN-ND is to reduce multicast traffic. Thus, due to the extensive use of multicast the STP protocol makes, its implementation in the 6LP-GW may seem unsuitable at a first glance. However, all the 6LP-GWs that may suffer from the risk of producing loops in the network will necessarily

be attached to the same network segment (otherwise there would not exist any risk of loops). Therefore, apart from their IEEE 802.15.4 segment, they can also communicate through their Ethernet interface, where they can generate multicast traffic without involving any power consumption in any other 6LNs.

If a layer-3 routing algorithm is to be implemented in order to solve the loop problem there is also an issue to keep in mind: as already mentioned, one of the main advantages of the 6LP-GW approach to internetwork 6LoWPAN and IPv6 networks over other layer-3 methods (mainly routing) is that the 6LP-GW requires no change in the existing network infrastructure. The most general case, at least in home environments, is that the router is unable to perform intra-LAN routing. This means that making the 6LP-GW behave as a router would simply not work: the IPv6 home router will not route any packet directed to a third node through this 6LP-GW. This means that the 6LP-GW needs to keep fooling the home router into "believing" that it is sending the packets *directly* to their destination. Then, the 6LP-GW would perform any routing algorithm (preferably RPL) on its IEEE 802.15.4 interface. Note that in order to do so, the 6LP-GW would also need to collaborate in the creation of the common path tree or any other task which may be responsibility of the routing nodes involved, depending on the specific routing algorithm.

6.2.1.2 6LoWPAN Fragmentation

The 6LoWPAN fragmentation feature defined in RFC 4944 [33] is actually implemented and available (although not exhaustively tested) in the Contiki sources. Thus, enabling this feature would require little modification to the current implementation and might be of interest for certain applications expecting large amounts of data or simply unwilling to limit themselves to a highly constrained maximum payload size.

6.2.1.3 Advanced Context Creation and Management

A proper mechanism for automatic creation of 6LoWPAN contexts is a feature that will definitely be of interest for most applications, as it may allow **every** 6LN in the network to save up to 16 bytes per packet sent (thus, allowing these bytes to be used for application data or saving a substantial amount of power over time due to needing to transmit fewer bits). Currently, the only contexts that the 6LP-GW will create are those derived directly from the prefix(es) announced in router advertisement messages. This means that it will not be possible to apply context-based compression to global destination addresses (unless they happen to be in the local network). Thus, what we propose here is the implementation of an intelligent algorithm capable of creating, maintaining, and deleting contexts depending, for instance, on the frequency with which certain global addresses (or prefixes) are used in either outgoing or incoming packets.

6.2.1.4 Radio Duty Cycling Mechanisms

Lower down to the link layer, there is another feature that has also been left out of the scope of this thesis project and might be of interest for future implementations. This is the use of a radio duty cycling mechanism such as the one described in [20] and included in the Contiki sources as the"contikimac" MAC layer implementation. While a radio duty cycling mechanism may not seem of much interest from the 6LP-GW's point of view due to the fact that is is powered by the power mains, its implementation would still be useful for the rest of the 6LNs in the network. The reason for this is that when a radio duty cycling mechanism is to be used, all the parties involved in the communication need to implement this radio duty cycling mechanism in order to communicate. For instance, the radio duty cycling mechanism implemented in contikimac relies on performing retransmissions of outgoing packets during a certain time slot until an IEEE 802.15.4 acknowledgement (ACK) is received or the time slot ends. This way, if the receiver wakes up at any point during this time slot and remains awake during a period of time long enough to find out that there is a packet for it, it will successfully receive every packet destined to it, even though it is sleeping most of the time. Thus, both transmitter and receiver must behave in the appropriate way at the time of transmitting, and both must at least implement the automatic sending of ACKs following the successful reception of a IEEE 80.15.4 frame. This means that even if we choose not to make the 6LP-GW enter sleep mode (thus allowing a fast transmission of ACKs as soon as it receives any packet, hence freeing the sender node from the need of retransmitting the packet), we need to make the 6LP-GW send packets such that they are retransmitted during a certain time slot until an ACK is received or the slot time elapses, as well as implementing the automatic sending of ACKs. Note that in this case the 6LP-GW would have to send ACKs upon successful reception of packets which are addressed to other nodes.

6.2.1.5 Power over Ethernet

As for the physical layer, a possible improvement may consist of changing the way in which the 6LP-GW is powered. The current implementation requires the use of a 230 V (AC) to 5 V (DC) adapter. Considering the cost of the hardware components required for our purpose, this adapter constitutes the most expensive element of the design (for details, see Appendix D). Thus, a solution allowing the elimination of this component from the hardware design would drastically reduce the cost of the device.

The use of Power over Ethernet [3] might well avoid the need for this adapter and hence the need of a plug for it, thus reducing the total space and installation requirements. Such a feature would enable the installation of several 6LP-GWs in environments in which multiport Ethernet switches

where present, as is the case of many office buildings, or government facilities such as hospitals, universities, or police stations among others. An important advantage of this would be the elimination of the need for intermediate nodes required for multi-hop, with the consequent savings in terms of power (due to the elimination of multi-hop), and money (due to the elimination of intermediate nodes, and for the power savings previously mentioned). However, the requirements for this are two: the implementation of a loop-avoidance mechanism such as those described in Section 6.2.1.1 and the availability of Power over Ethernet in the installation.

6.2.1.6 Security

The last unfulfilled feature, spanning across almost all layers is the issue of security. The current implementation includes no security mechanisms. Since 6LoWPAN devices may control different home appliances or even be in charge of surveillance applications, the incorporation of appropriate security is a sensitive and important issue. There are several options regarding security mechanisms that might be applied to different layers.

At the MAC sublayer, the IEEE 802.15.4 standard provides a set of security suites that comprise a set of symmetric cryptography algorithms (using the Advanced Encryption Standard — AES), and parameters for the application of this algorithm.

As for the IP layer, IPsec [29] is the most common security mechanism for protecting all traffic at the IP layer. It provides integrity, data origin authentication, anti-replay features, and confidentiality by means of cryptographic key exchanges using IKE [24]. In addition to IPsec and also at the IP layer, secure ND (SEND) [7] defines mechanisms to secure the ND protocol against a set of threats defined in [36] by means of defining some new options to carry public key-based signatures (RSA).

Regarding security at the transport layer, Datagram Transport Layer Security (DTLS) [40] is the preferred choice by the IETF community. It consists of an adaptation of the Transport Layer Security (TLS) [16] for datagram-oriented communication. The main purpose of DTLS is to prevent eavesdropping, tampering, or message forgery. To realize these functions DTLS relies on different cryptographic operations for encryption and signing.

Needless to say, whatever the choice of the security mechanism(s) might be, implementations need to bear in mind the constrained nature of the devices involved, which may determine the feasibility of a certain mechanism.

6.2.2 Next obvious things to be done

Apart from the unfulfilled features enumerated in Section 6.2.1, it seems reasonable to analyse the results and conclusions of this thesis now that it has been completed in order to define new projects.

In particular, the author of this thesis wishes to point out to the feasibility of implementing new access points to internetwork different media types with IEEE 802.15.4. For instance, an 802.11 to 802.15.4 access point would serve the same purpose as the 6LP-GW, but without the need to be physically attached to any router. This could certainly lead to ease of use in the presence of Wi-Fi networks (note that many home routers are also 802.11 access points). In addition, such an access point would make it possible to benefit from all the advantages derived from the installation of several 6LP-GWs in certain buildings as mentioned in Section 6.2.1.5, but with different requirements. In this case, these requirements would be that the 6LP-GWs are within the range of a Wi-Fi network and a power mains plug, in contrast to the need for a Power-over-Ethernet-enabled multiport Ethernet switch. Note that the implementation of a loop avoidance mechanism as described in Section 6.2.1.1 remains as a requirement in this case.

References

[1] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Common Specifications Part 3: Media Access Control (MAC) Bridges. *ANSI/IEEE Std 802.1D, 1998 Edition*, pages 58 – 107, 1998.

[2] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Common Specifications Part 3: Media Access Control (MAC) Bridges. *ANSI/IEEE Std 802.1D, 1998 Edition*, pages 58 – 107, 1998.

[3] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Amendment: Data Terminal Equipment (DTE) Power via Media Dependent Interface (MDI). *IEEE Std 802.3af-2003 (Amendment to IEEE Std 802.3-2002, including IEEE Std 802.3ae-2002)*, pages i – 121, 6 2003.

[4] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pages 0_1 –305, 2006.

[5] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Section One. *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)*, pages c1 –597, 26 2008.

[6] Guidelines For 64-bit Global Identifier (EUI-64) Registration Authority. *IEEE Standards Assocation*, April 2010.

[7] J. Arkko, J. Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971, Internet Engineering Task Force, March 2005.

[8] E. Baccelli and M. Townsley. IP Addressing Model in Ad Hoc Networks. RFC 5889, Internet Engineering Task Force, September 2010.

[9] C. Bormann. 6LoWPAN Generic Compression of Headers and Header-like Payloads. Internet-Draft draft-bormann-6lowpan-ghc-03, Internet Engineering Task Force, October 2011. Work in progress.

[10] R. Braden. Requirements for Internet Hosts - Application and Support. RFC 1123, Internet Engineering Task Force, October 1989.

[11] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, Internet Engineering Task Force, October 1989.

[12] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):637 – 648, May 1974.

[13] A. Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443, Internet Engineering Task Force, March 2006.

[14] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 1883, Internet Engineering Task Force, December 1995.

[15] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, Internet Engineering Task Force, December 1998.

[16] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, August 2008.

[17] R. Droms. Dynamic Host Configuration Protocol. RFC 1531, Internet Engineering Task Force, October 1993.

[18] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, Internet Engineering Task Force, March 1997.

[19] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315, Internet Engineering Task Force, July 2003.

[20] Adam Dunkels, Luca Mottola, Nicolas Tsiftes, Fredrik Österlind, Joakim Eriksson, and Niclas Finne. The Announcement Layer: Beacon Coordination for the Sensornet Stack. In *Proceedings of EWSN 2011*, Bonn, Germany, February 2011.

[21] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, nov 2006.

[22] Mikhail Filippov. Power over Ethernet for a 6LoWPAN gateway. *Bachelor's thesis*, 2011. Work in progress (completion expected in December 2011).

[23] B. Haberman and R. Hinden. IPv6 Router Advertisement Flags Option. RFC 5175, Internet Engineering Task Force, March 2008.

[24] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, Internet Engineering Task Force, November 1998.

[25] R. Hinden and S. Deering. Internet Protocol Version 6 (IPv6) Addressing Architecture. RFC 3513, Internet Engineering Task Force, April 2003.

[26] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, Internet Engineering Task Force, February 2006.

[27] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282, Internet Engineering Task Force, September 2011.

[28] J. Jeong, S. Park, L. Beloeil, and S. Madanapalli. IPv6 Router Advertisement Options for DNS Configuration. RFC 6106, Internet Engineering Task Force, November 2010.

[29] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301, Internet Engineering Task Force, December 2005.

[30] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919, Internet Engineering Task Force, August 2007.

[31] L. Maqueda. Guidelines for the Operation of a 6LoWPAN-ND Proxy Gateway. Internet-Draft draft-maqueda-6lowpan-pgw-00, Internet Engineering Task Force, March 2011. Work in progress.

[32] Friedemann Mattern and Christian Floerkemeier. *From the Internet of Computers to the Internet of Things*, volume 6462 of *LNCS*, pages 242–259. Springer, 2010.

[33] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, Internet Engineering Task Force, September 2007.

[34] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, Internet Engineering Task Force, September 2007.

[35] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, Internet Engineering Task Force, December 1998.

[36] P. Nikander, J. Kempf, and E. Nordmark. IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756, Internet Engineering Task Force, May 2004.

[37] D. Plummer. Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 0826, Internet Engineering Task Force, November 1982.

[38] J. Postel. DoD standard Internet Protocol. RFC 0760, Internet Engineering Task Force, January 1980.

[39] J. Postel. Internet Protocol. RFC 0791, Internet Engineering Task Force, September 1981.

[40] E. Rescorla and N. NagendraModadugu. Datagram Transport Layer Security version 1.2. Internet-Draft draft-ietf-tls-rfc4347-bis-06, Internet Engineering Task Force, July 2011. Work in progress.

[41] Sergio A. Floriano Sanchez. Wireless Sensor Network design and implementation in a Home-event Management System. *Master's thesis*, 2011. Work in progress (completion expected in October 2011).

[42] H. Schulzrinne and B. Volz. Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers. RFC 3319, Internet Engineering Task Force, July 2003.

[43] Z. Shelby, S. Chakrabarti, and E. Nordmark. Neighbor Discovery Optimization for Low Power and Lossy Networks(6LoWPAN). Internet-Draft draft-ietf-6lowpan-nd-18, Internet Engineering Task Force, October 2011. Work in progress.

[44] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). Internet-Draft draft-ietf-core-coap-08, Internet Engineering Task Force, October 2011. Work in progress.

[45] H. Singh, W. Beebee, and E. Nordmark. IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes. RFC 5942, Internet Engineering Task Force, July 2010.

[46] D. Thaler, M. Talwar, and C. Patel. Neighbor Discovery Proxies (ND Proxy). RFC 4389, Internet Engineering Task Force, April 2006.

[47] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862, Internet Engineering Task Force, September 2007.

[48] P. Thubert. 6LoWPAN Backbone Router. Internet-Draft draft-thubert-6lowpan-backbone-router-02, Internet Engineering Task Force, June 2010. Work in progress.

[49] T. TimWinter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Internet-Draft draft-ietf-roll-rpl-19, Internet Engineering Task Force, March 2011. Work in progress.

[50] Joaquín Juan Toledo. Wireless Sensor Architecture for a Home Event Management System. *Master's thesis*, 2011. Work in progress (completion expected in December 2011).

[51] Jon-Olov Vatn and Gerald Q. Maguire Jr. The effect of using co-located care-of addresses on macro handover latency. *Fourteenth Nordic Teletraffic Seminar (NTS 14)*, August 1998.

Appendix A: 6LoWPAN-ND Host implementation

Since the 6LP-GW described in this master thesis acts as an intermediary between the ND and 6LoWPAN-ND protocols and between the Ethernet and IEEE 802.15.4 link layers, the availability of a host implementing the 6LoWPAN-ND in the IEEE 802.15.4 segment was necessary in order to verify the correct behaviour of our application.

It has already been mentioned that Contiki does **not** implement the 6LoWPAN-ND protocol. This, together with the lack of available open source implementations of such protocol made practically imperative to develop our own implementation the "host behaviour" of the 6LoWPAN-ND protocol, as defined in draft-ietf-6lowpan-nd [43].

In order to implement the 6LoWPAN-ND protocol, we took the ND implementation present in Contiki as the starting point, applying the necessary modifications as required. As a reminder of the differences between 6LoWPAN-ND and IPv6-ND, we should recall Table 2.4 in Section 2.9.3.1 (page 24).

1. Host initiated interactions to allow for sleeping hosts. As opposed to IPv6-ND that simply removes routers, prefixes, or neighbors when their corresponding lifetimes expire, we need to trigger sending RSs (either multicast or unicast, as required) as specified in section 5.3 of draft-ietf-6lowpan-nd. In order to do so, the code performs periodic checks of prefixes and router lifetimes and triggers sending RSs when necessary. When the code determines that it is necessary to start sending RSs, we first check whether it is possible to unicast them. After the process of sending unicast RSs to a specific router has been initiated, we need to keep a counter with the number of RSs that have been sent to that specific router. This counter is placed in the data structure that holds the default router information (`uip_ds6_defrt_list`) within the default routers list. When this count reaches MAX_RTR_SOLICITATIONS (which defaults to 3), the code switches to sending multicast RSs.

   The interval between sending RSs is calculated by means of a binary exponential back-off, with the maximum interval determined by the constant MAX_RTR_SOLICITATION_INTERVAL (which defaults to 60 seconds).

2. Elimination of multicast-based address resolution for hosts. In order to implement this feature, the function `tcpip_ipv6_output()` has been modified so that all prefixes but the link-local are considered to be off-link. Hence, packets directed to link-local addresses are sent directly

to their destination without performing address resolution. All packets addressed to non link-local destinations are sent via an IPv6 router. Consequently, no NCEs are created nor maintained for nodes that are not routers. This way, we eliminate the need for multicasting NSs for AR as specified in draft-ietf-6lowpan-nd.

3. A host address registration feature using a new option in unicast Neighbor Solicitation and Neighbor Advertisement messages. This feature requires mayor modifications to the code. According to draft-ietf-6lowpan-nd hosts must register addresses with routers. While these registrations can be easily handled by routers (each registered address corresponds to a NCE in the router), this is not the case for hosts. The way Contiki handles addresses, NCEs, and default routers, is by maintaining a table for each (i.e., a list of addresses, a list of default routers and the NC). However, an address can be registered with several routers, and this registration process could be in a different state for each of these routers. This absence of a 1-to-1 mapping between addresses and routers motivated the addition of a new list of registrations that has to be maintained in order to implement this address registration feature. Each registration in this list is held in the C struct `uip_ds6_reg_t` which is defined as follows:

```
/* Structure to handle 6lowpan-nd registrations */
typedef struct uip_ds6_reg {
  u8_t isused;
  u8_t state;
  uip_ds6_addr_t* addr;
  uip_ds6_defrt_t* defrt;
  struct stimer reg_lifetime;
  struct timer registration_timer;
  u8_t reg_count;
} uip_ds6_reg_t;
```

This structure basically links an address to a router, along with some control information such as the registration state and lifetime. The fields `registration_timer` and `reg_count` are used during the registration process in order to measure the interval (in time) between sending NSs for registration and the number of NSs sent to a router during this process. Note that according to draft-ietf-6lowpan-nd, a NS should be retransmitted up to MAX_UNICAST_SOLICIT (i.e., 3) times using a minimum interval of RETRANS_TIMER (1000) milliseconds.

In addition to creating and maintaining registration entries for each address and router, this feature requires also the proper handling of the new ARO option. This option needs to be appended in outgoing NS messages sent for registration and this option must be

processed according to draft-ietf-6lowpan-nd for incoming NAs. To do so, modifications in the Contiki functions `uip_nd6_ns_output()` and `uip_nd6_na_input()` (that handle the sending of NSs and receiving of NAs respectively) were required. In addition, the logic in charge of maintaining the registrations table and determining whether sending NS is needed, or to which router a certain NS has to be sent, is placed mainly in the function `uip_ds6_periodic()` (although some other functions such as `uip_nd6_ra_input()` also contribute to this process).

4. A new Neighbor Discovery option to distribute 6LoWPAN header compression context to hosts. Despite being optional, this feature has been also included in the 6LoWPAN-ND implementation developed as part of this master thesis. This feature requires proper handling of 6CO options present in RAs, creation and maintenance of compression contexts, and, of course, its utilization for compression.

   The C data structure used by Contiki to store compression contexts was insufficient for our requirements. Thus, it was necessary to redefine the C structure that holds the 6LoWPAN contexts in order to fulfil our requirements. This data structure is defined as as follows:

```
/* Structure to store 6lowpan compression contexts */
typedef struct uip_ds6_addr_context {
  uip_ds6_context_state_t state;
  u8_t length;
  u8_t context_id;
  uip_ipaddr_t prefix;
  struct stimer vlifetime;
  /* The router that announced this context */
  uip_ds6_defrt_t* defrt;
  u16_t defrt_lifetime;
} uip_ds6_addr_context_t;
```

   The pointer to the default router (`uip_ds6_defrt_t* defrt`) is used enables us to unicast a RS to the specific router that announced the context if we need to update the context, rather than multicasting such RS to the all-routers multicast address. The field storing the default router lifetime (`u16_t defrt_lifetime`) is required because, according to draft-ietf-6lowpan-nd, if a context valid lifetime expires the context's state must be set to an uncompression-only state for a period of "twice the Default Router Lifetime". After that period, if no 6CO has been received to update that context, then this context should be deleted. Therefore, we need to remember the default router lifetime. Moreover, we can not use the corresponding value in "defrt" because that router may have been deleted by the time we need the default router's lifetime.

   The creation of compression contexts and their addition to the context table is perform by the Contiki function `uip_nd6_ra_input()`. In

this function, incoming RAs are parsed in order to retrieve the ND options present in their payload. If a 6CO option is found, a new compression context is created and added to the context table (unless its lifetime is zero). The maintenance of contexts in the context table is performed in the function `uip_ds6_periodic()`. Finally, the utilisation of compression context for the purpose of compression and uncompression required modifications in the functions `compress_hdr_hc06()` and `uncompress_hdr_hc06()` respectively. In order to make the most of the implementation, we decided to support up to 128 bits long contexts (the maximum context length according to RFC 6282), instead of the 64-bit, fixed-length prefixes that were supported by Contiki. This decision forced significant changes in the way compression and decompression takes place. Moreover, in order to handle the context table for different purposes, proper functions had to be implemented for context addition, lookup (by prefix or context identifier), and deletion (i.e., `uip_ds6_context_add()`, `uip_ds6_context_lookup_by_id()`, `uip_ds6_context_lookup_by_prefix()`, and `uip_ds6_context_rm()` respectively).

The last two different features that are defined as optional are not included in this implementation. These optional (and unimplemented) functions are:

(a) Optional multihop distribution of prefix and 6LoWPAN header compression context.

(b) Optional multihop duplicate address detection using two new ICMPv6 message types.

Appendix B: Traffic captures

The following figures correspond to fragments of traffic captures which illustrate some of the most representative interactions of the 6LP-GW with other internet hosts. In fact, these interactions are between 6LHs and NCDs (hosts and/or routers), with the 6LP-GW acting as an intermediary between them.

These traffic captures were obtained by means of a HP switch having one of its ports configured as monitor, a 6LoWPAN traffic sniffer (built using a spare Hogaza board v1.2 [50]), and Wireshark (http://www.wireshark.org/) as the network protocol analyser. Packets with green foreground are 6LoWPAN packets captured though the 6LoWPAN sniffer whereas packets displayed with light blue foreground are IPv6 packets coming from the Ethernet wire.

6LH Bootstrapping

The following figure illustrates a complete 6LH bootstrapping sequence.

Time	Source	Destination	Protocol	Length	Info
7.486623	fe80::207:62ff:fe81:513	ff02::2	ICMPv6	60	Router Solicitation
7.487105	fe80::207:62ff:fe81:513	ff02::2	ICMPv6	70	Router Solicitation
7.488201	fe80::20c:cfff:fea8:9800	ff02::1	ICMPv6	118	Router Advertisement
7.494510	fe80::20c:cfff:fea8:9800	fe80::207:62ff:fe81:513	ICMPv6	121	Router Advertisement
7.552037	2001:6b0:1:2000:207:62ff:fe81:513	fe80::20c:cfff:fea8:9800	ICMPv6	112	Neighbor Solicitation
7.552632	::	ff02::1:ff81:513	ICMPv6	78	Neighbor Solicitation
8.565903	fe80::20c:cfff:fea8:9800	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	112	Neighbor Advertisement

Figure B.1: 6LH bootstrapping capture

1. The 6LH multicasts a RS to the all-routers multicast address.

2. The 6LP-GW forwards the packet unchanged (except for the appropriate link-layer address translation). Note that the difference between the the RSs' lengths is due to the different link-layers and the presence of the 6LoWPAN headers in the first case.

3. The IPv6 router sends a RA to the all-nodes multicast address in response to the RS. Note that such an RA includes a PIO option with its "L" (on-link) flag set (apart from a SLLAO option). See Figure B.2.

4. The 6LP-GW forwards the packet to the 6LH's unicast address from which the solicitation was sent. In addition the "L" (on-link) flag of the PIO option in the RA is cleared (see figure B.3) and a 6CO option is appended (figure B.4).

5. The 6LH sends a NS including an ARO option for registration to the router originating the previous RA.

6. The 6LP-GW performs DAD on the Ethernet segment on behalf of the 6LH rather than forwarding the solicitation.

7. Once DAD completes successfully, the 6LP-GW sends a NA including an ARO option to the 6LH that originated the solicitation. The status field in the ARO informs the 6LH about the state of the registration (which in this case was successful).

```
▽ ICMPv6 Option (Prefix information : 2001:6b0:1:2000::/64)
    Type: Prefix information (3)
    Length: 4 (32 bytes)
    Prefix Length: 64
  ▽ Flag: 0xc0
      1... .... = On-link flag(L): Set
      .1.. .... = Autonomous address-configuration flag(A): Set
      ..00 0000 = Reserved: 0
    Valid Lifetime: 2592000
    Preferred Lifetime: 604800
    Reserved
    Prefix: 2001:6b0:1:2000:: (2001:6b0:1:2000::)
```

Figure B.2: PIO option in IPv6-ND RA

```
▽ ICMPv6 Option (Prefix information : 2001:6b0:1:2000::/64)
    Type: Prefix information (3)
    Length: 4 (32 bytes)
    Prefix Length: 64
  ▽ Flag: 0x40
      0... .... = On-link flag(L): Not set
      .1.. .... = Autonomous address-configuration flag(A): Set
      ..00 0000 = Reserved: 0
    Valid Lifetime: 2592000
    Preferred Lifetime: 604800
    Reserved
    Prefix: 2001:6b0:1:2000:: (2001:6b0:1:2000::)
```

Figure B.3: PIO option in 6LoWPAN-ND RA

```
▽ ICMPv6 Option (6LoWPAN Context Option 2001:6b0:1:2000::/64)
    Type: 6LoWPAN Context Option (32)
    Length: 2 (16 bytes)
    Context Length: 64
  ▷ Flag: 0x00
    Reserved
    Lifetime: 10
    Context Prefix: 2001:6b0:1:2000:: (2001:6b0:1:2000::)
```

Figure B.4: 6CO option in 6LoWPAN-ND RA

6LH Registration renewal

Figure B.5 illustrates a registration renewal sequence. The text following the figure describes the steps in this sequence.

Time	Source	Destination	Protocol	Length	Info
20.156831	2001:6b0:1:2000:207:62ff:fe81:513	fe80::20c:cfff:fea8:9800	ICMPv6	112	Neighbor Solicitation
20.157740	2001:6b0:1:2000:207:62ff:fe81:513	fe80::20c:cfff:fea8:9800	ICMPv6	102	Neighbor Solicitation
20.159507	fe80::20c:cfff:fea8:9800	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	78	Neighbor Advertisement
20.164519	fe80::20c:cfff:fea8:9800	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	97	Neighbor Advertisement

Figure B.5: 6LH Registration renewal

1. The 6LH sends a NS including an ARO option to the IPv6 router's address.

2. The 6LP-GW forwards the NS to the IPv6 router unchanged (other than the link-layer address translation). Note that the ARO option does not need to be filtered out; it will simply be ignored by the IPv6 router.

3. The IPv6 router responds with a NA to the solicitation

4. The 6LP-GW appends an appropriate ARO option to the NA and forwards the packet to the 6LH that originated the solicitation. Figure B.6 shows this NA including an ARO option (note that this is erroneously tagged as "Address Resolution Option" by Wireshark's dissector).

```
  Type: Neighbor Advertisement (136)
  Code: 0
  Checksum: 0x5043 [correct]
▽ Flags: 0xc0000000
    1... .... .... .... .... .... .... .... = Router: Set
    .1.. .... .... .... .... .... .... .... = Solicited: Set
    ..0. .... .... .... .... .... .... .... = Override: Not set
    ...0 0000 0000 0000 0000 0000 0000 0000 = Reserved: 0
  Target Address: fe80::20c:cfff:fea8:9800 (fe80::20c:cfff:fea8:9800)
▽ ICMPv6 Option (Address Resolution Option : Register 00:07:62:ff:fe:81:05:13 Success)
    Type: Address Resolution Option (131)
    Length: 2 (16 bytes)
    Status: Success (0)
    Reserved
    Registration  Lifetime: 1
    EUI-64: 000762fffe810513
```

Figure B.6: NA including ARO option

NCD performing NUD on a 6LH

The following capture shows the process of a NCD performing NUD on a 6LH.

Time	Source	Destination	Protocol	Length	Info
25.163571	fe80::20c:cfff:fea8:9800	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	86	Neighbor Solicitation
25.163693	2001:6b0:1:2000:207:62ff:fe81:513	fe80::20c:cfff:fea8:9800	ICMPv6	86	Neighbor Advertisement

Figure B.7: NCD performing NUD on a 6LH

1. A NCD (in this particular case, the RR) sends a NS to the 6LH for NUD.
2. The 6LP-GW responds on behalf of the 6LH by sending a NA to the originator of the registration. This is possible due to the Address Registration feature of 6LoWPAN-ND. Note that no IEEE 802.15.4 traffic is generated in this operation.

Pinging a 6LH

The capture shown in Figure B.8 shows an external internet host sending two ICMPv6 Ping packets to a 6LH.

Time	Source	Destination	Protocol	Length	Info
520.853796	2002:82e5:87fe::82e5:87fe	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	94	Echo (ping) request
520.858758	2002:82e5:87fe::82e5:87fe	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	97	Echo (ping) request
520.865042	2001:6b0:1:2000:207:62ff:fe81:513	2002:82e5:87fe::82e5:87fe	ICMPv6	96	Echo (ping) reply
520.865867	2001:6b0:1:2000:207:62ff:fe81:513	2002:82e5:87fe::82e5:87fe	ICMPv6	94	Echo (ping) reply
521.854957	2002:82e5:87fe::82e5:87fe	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	94	Echo (ping) request
521.859927	2002:82e5:87fe::82e5:87fe	2001:6b0:1:2000:207:62ff:fe81:513	ICMPv6	97	Echo (ping) request
521.866219	2001:6b0:1:2000:207:62ff:fe81:513	2002:82e5:87fe::82e5:87fe	ICMPv6	96	Echo (ping) reply
521.867047	2001:6b0:1:2000:207:62ff:fe81:513	2002:82e5:87fe::82e5:87fe	ICMPv6	94	Echo (ping) reply

Figure B.8: NCD performing a Ping of a 6LH

1. The router unwittingly forwards an echo request packet directed to a 6LH to the 6LP-GW.
2. The 6LP-GW forwards the request to the 6LH. No transformations are applied to the packet except for the link-layer address translation.
3. The 6LH receives the echo request and generates an echo reply that is sent to the destination.
4. The 6LP-GW forwards the packet to the router that should subsequently forward it towards its destination.
5. This process is repeated.

Appendix C: Source code

The source code of the application developed as part of this master thesis is publicly available in the following repository:

https://6lp-gw.googlecode.com/svn/6LP-GW_Hogaza_v2.1

Appendix D: Hardware Specification

The hardware implementation of the 6LP-GW utilized in this thesis project has been designed and developed as part of Joaquín Juan Toledo's master's thesis project [50]. It comprises three different boards: the TI's evaluation board CC2591EM 3.0, the Olimex's ENC28J60-H development board, and the Hogaza v1.2.

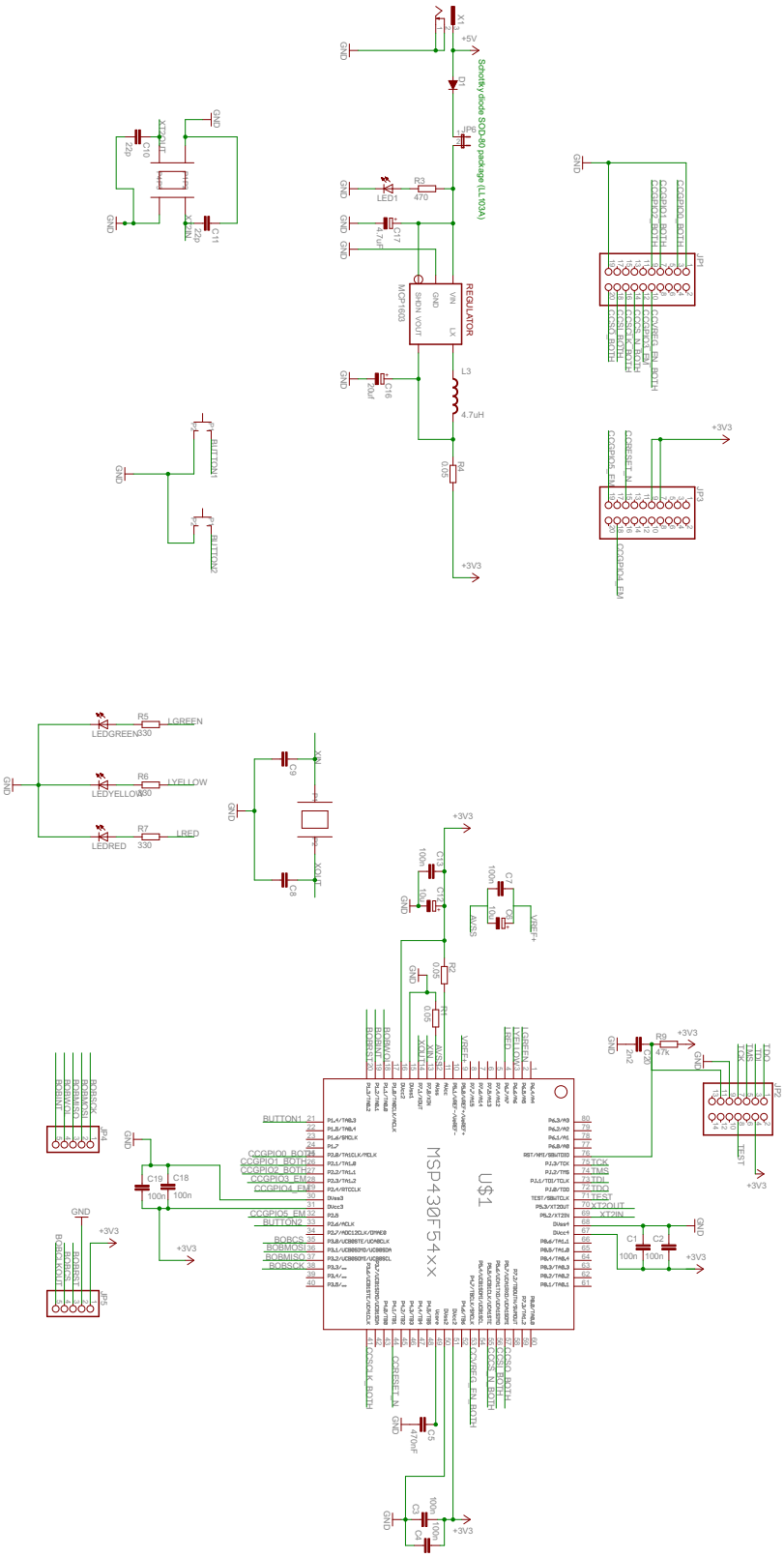

Figure D.1: Hogaza v1.2 Schematics.

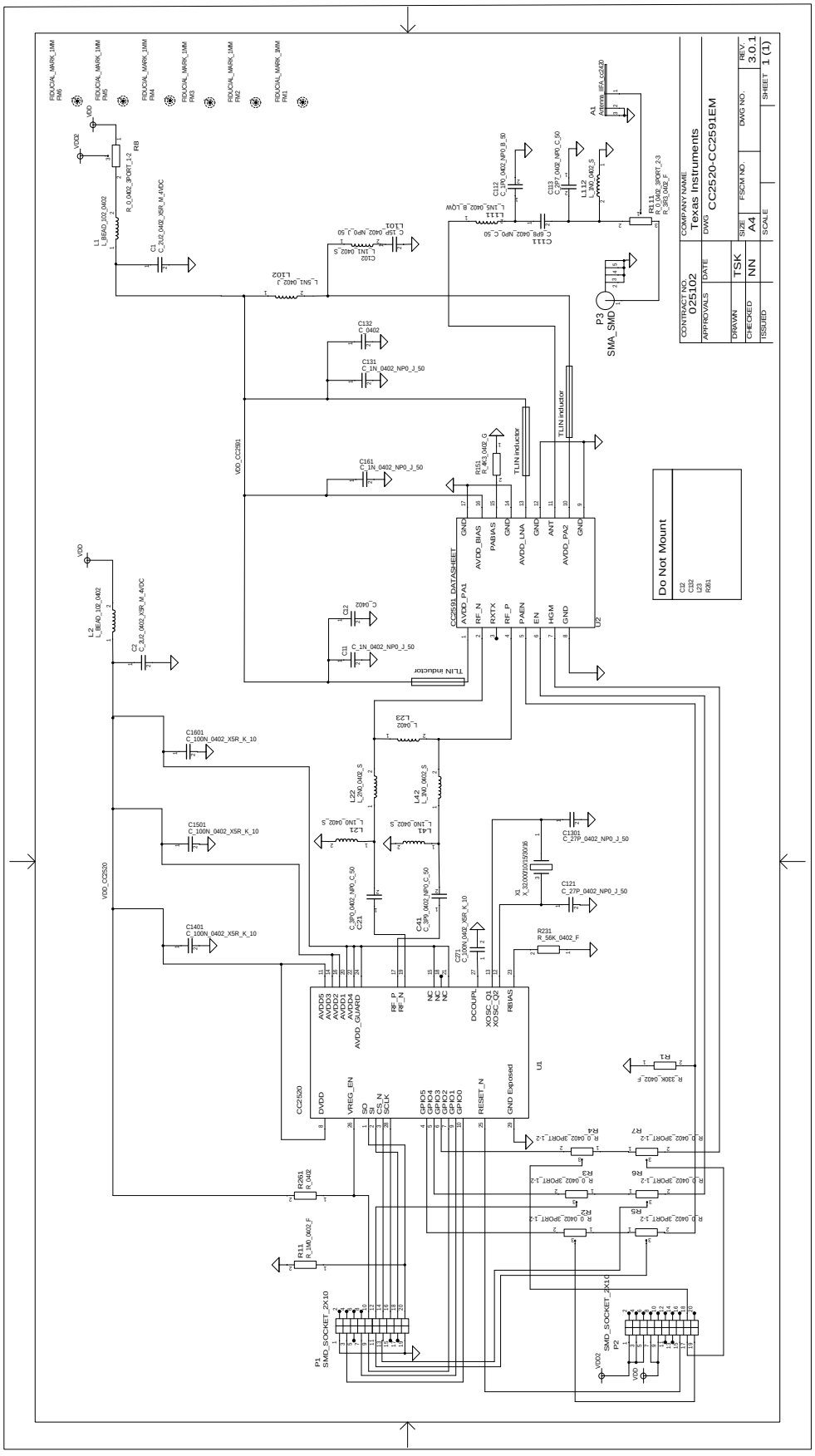

Figure D.2: CC2591EM 3.0 Schematics.

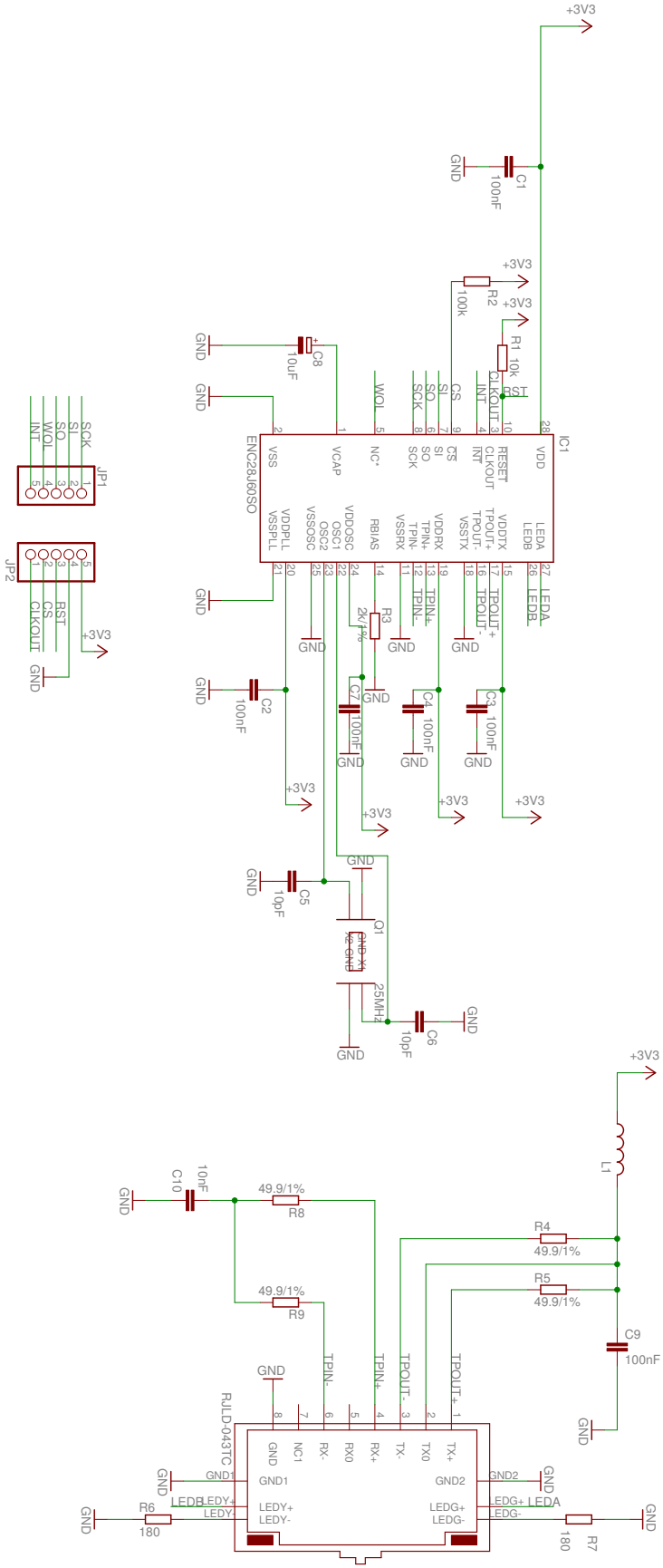

Figure D.3: ENC28J60-H Schematics.

TRITA-ICT-EX-2011:221

www.kth.se