

Proyecto Fin de Carrera

Design and Implementation of Multicast Listener Discovery Protocol on constrained devices

Autor

Daniel Bailo Estallo

Director

Enrique Torres Moreno

INGENIERÍA INFORMÁTICA

2013

Resumen

Para la aplicación y apoyo del uso de IPv6 en 6LoWPANs (*Low-power Wireless Personal Area Networks*), ha habido numerosas investigaciones y se han desarrollado protocolos y mecanismos estandarizados. Sin embargo para la comunicación multicast en estas redes, el tema esta aún bastante abierto a la investigación. La comunicación multicast permite conectar routers con hosts preseleccionados por grupos. La comunicación multicast es muy beneficiosa para aplicaciones con dispositivos con recursos limitados ya que ahorra energía y ancho de banda. A continuación mostramos posibles ejemplos de estas aplicaciones, la iluminación de un edificio organizada por plantas, una red de sensores de temperatura organizados por áreas y un largo número de aplicaciones basadas en la comunicación de un punto a varios puntos preseleccionados.

El grupo de investigación de la universidad de Aalto (Finlandia) llamado MAMMoTH (*Massive Scale Machine-to-Machine Service*) tiene como uno de sus objetivos construir un protocolo multicast para dispositivos con recursos limitados. Para el desarrollo de este protocolo, es necesario un protocolo de encaminamiento multicast y un protocolo de gestión de grupos multicast. Este último, es el protocolo que he desarrollado como “*research assistant*” para mi proyecto final de carrera.

En este proyecto final de carrera, se ha diseñado, implementado y evaluado el protocolo MLD para dispositivos con recursos limitados. MLD permite a un router IPv6 gestionar grupos multicast. No obstante, el uso de MLD en LoWPANs tiene varios problemas como la definición del area local, el tamaño de los paquete y la complejidad del comportamiento del router.

El protocolo ha sido implementado en Contiki, un sistema operativo para desarrollar para el “*Internet of Things*”. Contiki permite conectar sistemas pequeños de poco coste con poca potencia a Internet. Hemos ampliado la pila TCP/IP de Contiki para respaldar MLD.

El protocolo ha sido evaluado y analizado sobre un simulador en diferentes topologías para validar el funcionamiento. Del mismo modo, también se ha verificado que el tamaño del objeto creado no ocupaba más memoria de la disponible en los dispositivos Z1 Zolertia.

Indice

1. Memoria.....	1
1.1. Introducción.....	1
1.2. Desarrollo del trabajo.....	2
1.3. Resultados.....	3
1.4. Conclusión.....	4
2. Anexo. Memoria detallada en inglés	

1. Memoria

Este proyecto final de carrera se ha realizado en Aalto University (Finlandia). Por esta razón, la mayor parte de la memoria esta en inglés. La estructura de la memoria se podría dividir en dos partes, un breve resumen de lo realizado en castellano y la memoria más detallada explicada en inglés. La parte en castellano describe un resumen del proyecto final de carrera (PFC), para continuar con una introducción al tema donde se desarrolla el PFC. La siguiente subsección explica el desarrollo del proyecto. Luego, se comentan los resultados obtenidos y se finaliza con la conclusión. Como anexo, aparece la memoria detallada en inglés que ya hemos nombrado y allí se podrá ver con más detalle su estructura.

1.1. Introducción

El “*Internet of Things*” (IoT) es un novedoso paradigma que esta ganando terreno en el mundo de las comunicaciones inalámbricas. Grandes compañías como Cisco o Ericsson afirman que en el 2020, habrá más de 50 mil millones de “*things*” conectadas a Internet.

El IoT permite la comunicación entre dispositivos, haciendo posible que los dispositivos físicos puedan comunicarse y que la gente pueda saber cuál es su estado y su localización. Con esta idea, se pueden desarrollar muchas aplicaciones para mejorar el día a día de las personas como por ejemplo: monitorizar el estado de salud de una persona, una aplicación para el ahorro energético de los domicilios, una aplicación que ayude a gestionar el stock en un almacén y un largo etcétera.

A pesar del exitoso futuro que se pronostica en este paradigma, hay ciertos problemas en el IoT. Los dispositivos que se interconectan suelen tener recursos limitados, pocos recursos computacionales, memorias reducidas y sistemas débiles de comunicación, no funcionan con los protocolos y mecanismos existentes.

Dado este problema, muchas organizaciones están trabajando con un mismo objetivo, estandarizar tecnologías para el IoT. IPv6 esta siendo respaldado por el IETF (Internet Engineering Task Force) como la tecnología a nivel de red para conectar los dispositivos por su gran estandarización en las comunicaciones. Un grupo del IETF ha diseñado ya un protocolo IPv6 para LoWPANs (Low power Wireless Personal Area Networks) que permite a los dispositivos de poca potencia usar IPv6. El grupo ROLL (Routing Over Low power and Lossy networks) del IETF ha diseñado un protocolo llamado RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Una capa de aplicación esta siendo también desarrollada por otro grupo del IETF, el protocolo CoAP (Constrained Application Protocol).

Como queda visto, se esta desarrollando mucho trabajo en el campo del IoT. Sin embargo, en el área de la comunicación multicast no se ha hecho mucho esfuerzo. Hasta donde nosotros sabemos, solo se han desarrollado dos ideas para comunicación multicast en dispositivos con recursos limitados. El protocolo MPL (Constrained Application Protocol) y el protocolo SMRF (Stateless Multicast forwarding with RPL). MPL es un protocolo que no tiene gestión de grupos multicast y no es eficiente hablando de consumo energético y tampoco fiable. SMRF solo permite transmisión de datos en una dirección.

Muchas aplicaciones reales en redes de sensores necesitan comunicación multicast ya

que ayuda a ahorrar ancho de banda y energía porque evitamos envíos de paquetes repetidos y los dispositivos pueden estar más tiempo inactivos.

1.2. Desarrollo del proyecto

Este proyecto final de carrera ha sido desarrollado en el departamento de “*Data communications*” de la Universidad de Aalto situada en Espoo (Finlandia). En concreto, en el grupo de investigación MAMMoTH (*Massive Scale Machine-to-Machine Service*). MAMMoTH esta financiado por entidades académicas como la universidad de Aalto y la universidad de Oulu (Finlandia), y por entidades de la industria como Ericsson (Nomadic Lab), Sensinode Oy, Renesas Mobile, There Corporation and Kajaani Data Center Operator. El grupo MAMMoTH esta trabajando en el campo del IoT de forma activa desde el 1 de Julio de 2011 y continuará hasta el 31 de Diciembre de 2013.

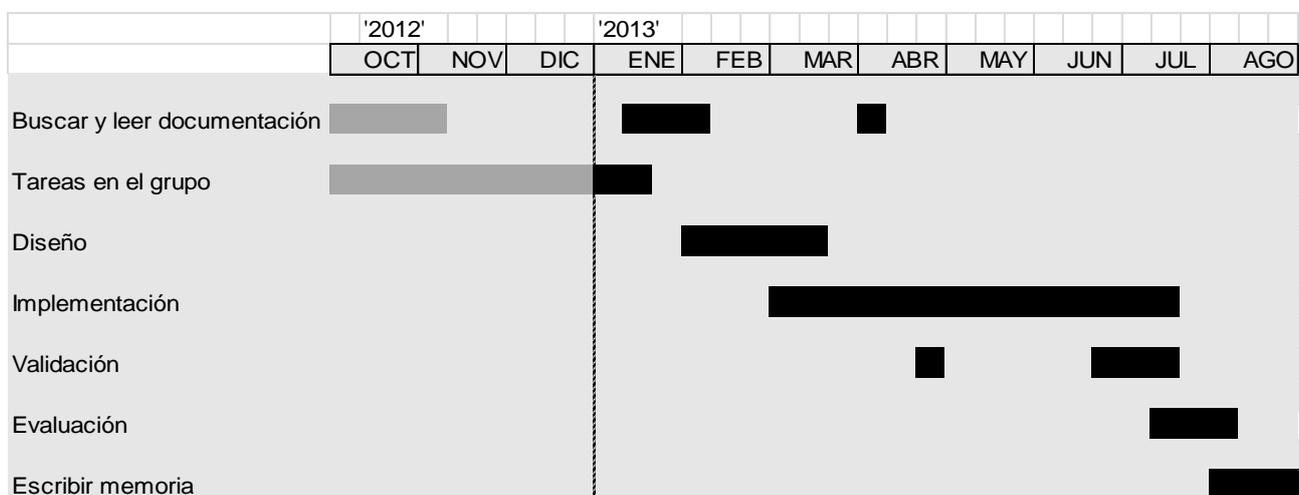
Dentro del grupo de investigación, mi posición era de “*research assistant*” bajo la supervisión de Zhonghong Ou (*Post-doc Researcher*). Semanalmente había una reunión con el grupo para comentar que habíamos hecho cada uno, que problemas y dificultades habíamos encontrado y que tareas se iban a hacer la siguiente semana.

El grupo estaba compuesto el profesor Antti Ylä-Jääski, dos “*Post-Doc researchers*”, Zhonghong Ou y Miika Komu, cuatro estudiantes de doctorado, Vilen Looga, Klaus Hartke, Yang Deng y Shichao Dong y dos “*research assistants*”, Ketan Devadiga y yo.

Mi trabajo estaba centrado en el diseño e implementación del protocolo Multicast Listener Discovery en constrained devices y también realizaba otras tareas para el departamento. Comencé a trabajar en el grupo en Octubre del 2012 y más enfocado a mi proyecto final de carrera en Enero del 2013 supervisado por Antti Ylä-Jääski y Enrique Torres Moreno y como instructor Yang Deng.

Con Enrique Torres Moreno, como supervisor de la universidad de Zaragoza, tenía reuniones on-line cada dos semanas y me ayudaba a centrar mis tareas hacia el proyecto final de carrera. Yang Deng como instructor en Aalto University seguía más mi día a día y los aspectos técnicos.

A continuación, mostramos un diagrama de Gantt donde podemos ver como se han repartido las diferentes fases del proyecto a lo largo del tiempo.



Cuando empecé a trabajar en mi proyecto final de carrera, mi primer objetivo era conseguir documentación útil para hacerme con el contexto de trabajo y entender correctamente todo lo relacionado con el tema. Una vez adquirida la idea sobre el protocolo, el campo en el que iba a trabajar, los trabajos relacionados existentes y más información relacionada, comencé a pensar en los problemas que había para aplicar MLD a los dispositivos con recursos limitados ya que no era posible la aplicación directa del protocolo especificado en el RFC 2710 MLDv1. Encontrados los problemas, busqué las soluciones y tras varias reuniones con mi instructor en la universidad de Aalto concluimos con un diseño que solucionaba el problema y se ajustaba a los dispositivos con recursos limitados.

Para implementar el diseño tuve que estudiar el sistema operativo Contiki, leyendo ejemplos de código y simulando escenarios. Tras obtener los conocimientos necesarios de Contiki para la implementación de mi protocolo, comencé a desarrollarlo.

Con el protocolo finalizado, comenzó el periodo de evaluación y de corregir fallos para conseguir el funcionamiento esperado propuesto en el diseño.

Con la aprobación de mi instructor sobre el funcionamiento del protocolo, comencé a escribir la memoria para Aalto University. Durante el desarrollo de mi proyecto, tuve varias reuniones sobre el protocolo MLD con mi instructor Yang Deng y con Ketan Kevadiga que estaba desarrollando el protocolo PIM-SM con SSM para LoWPANs para formar en sinergia el protocolo multicast apto para dispositivos con recursos limitados.

La duración del proyecto final de carrera ha sido desde mediados de Enero del 2013 hasta finales de Agosto del 2013. El número de horas trabajadas esta alrededor de 1000. En el departamento en la universidad de Aalto se trabajaba de lunes a viernes a tiempo completo.

1.3. Resultados

Un protocolo de comunicación multicast necesita dos protocolos. Uno para el encaminamiento de los paquetes y otro para el mantenimiento de los grupos multicast. Varios estudios afirman que un protocolo eficiente sería usando PIM-SM (Protocol Independent Multicast- Sparse Mode) adaptado a SSM (Source Specific Multicast) como protocolo de encaminamiento y MLDv2 como protocolo para la gestión de grupos multicast.

El objetivo del proyecto final de carrera ha consistido en diseñar e implementar el protocolo MLD (Multicast Listener Discovery) para LLNs (Low power and Lossy Networks) para en un futuro unirlo con una implementación de PIM-SM para dispositivos de recursos limitados y obtener un protocolo de comunicación multicast apto para LLNs.

Para el diseño del protocolo apto para dispositivos con recursos limitados se ha partido de la base de la especificación del RFC 2710 del IETF donde se describe el protocolo general MLD. Como el objetivo final es obtener un protocolo multicast tras unirlo con PIM-SM con SSM, nos hemos apoyado también en la especificación de MLDv2 en el RFC 3810 del IETF.

La aplicación directa del protocolo en dispositivos con recursos limitados no es posible. Ya que nos encontramos con unos problemas de definición correcta de área local en LLNs, complejidad de comportamiento del router y tamaño de paquetes para implementar el

filtrado por fuente que se especifica en MLDv2.

Hemos tenido que tomar ciertas decisiones en el diseño que se explican con más detalle en la memoria adjuntada en el anexo.

La implementación del protocolo MLD apto para dispositivos con recursos limitados ha sido implementado en el sistema operativo para el IoT, Contiki. El código está desarrollado en C y usa las librerías de Contiki.

El protocolo ha sido testeado sobre el simulador ofrecido por Contiki llamado Cooja. Se han realizado varios escenarios con diferentes flujos de datos para validar el correcto funcionamiento y verificar también que el tamaño del objeto cabe en los dispositivos usados. De este modo, hemos conseguido una implementación del protocolo Multicast Listener Discovery para dispositivos con recursos limitados ya que realiza la gestión de grupos multicast necesaria para formar un protocolo multicast, ocupa poca memoria por lo tanto el dispositivo dispone de espacio suficiente para su funcionamiento. Durante este proyecto, no se han verificado los consumos energéticos de los dispositivos pero sí que se han implementado métodos para tener que hacer menor gasto computacional.

1.4. Conclusión

La tecnología multicast, a menudo es vista como el mecanismo de comunicación más prometedor para el uso de aplicaciones de internet multiusuario, en tiempo real y multimedia. Los routers y hosts multicast, en redes basadas en IPv6, se comunican e intercambian sus competencias para la gestión de grupos multicast mediante el protocolo Multicast Listener Discovery.

Este proyecto final de carrera se ha realizado el diseño y la implementación del protocolo MLD apto para dispositivos con recursos limitados. El diseño es lo suficientemente simple para poder gestionar los grupos multicast y no ocupar mucha memoria.

La validación y evaluación de la implementación se ha realizado en el simulador Cooja

Como hemos comentado anteriormente, la implementación de MLD adecuada para dispositivos con recursos limitados forma parte de un proyecto mayor del grupo MAMMoTH que es crear un protocolo multicast completo para este tipo de dispositivos.

A continuación, se puede encontrar en el anexo la memoria realizada para la universidad de Aalto.

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Daniel Bailo Estallo

Design and implementation of Multicast Listener Discovery on constrained devices

Final project
Espoo, September 2013

Supervisors: Professor Antti Ylä-Jääski, Aalto University
Professor Enrique Fermin Torres Moreno, University of
Zaragoza
Advisor: Yang Deng M.Sc. (Tech.)

Author:	Daniel Bailo Estallo	
Title:	Design and implementation of Multicast Listener Discovery on constrained devices	
Date:	September 2013	Pages: 77
Major:	Data Communication Software	Code: T-110
Supervisors:	Professor Antti Ylä-Jääski Professor Enrique Fermín Torres Moreno	
Advisor:	Yang Deng M.Sc. (Tech.)	
<p>To enable and support IPv6 in Low-power Wireless Personal Area Networks (LoWPANs), there has been a significant research to contribute with protocols and mechanisms. Nonetheless, the area of multicast still remains an open research topic for LoWPANs. Multicast communication would be beneficial in different applications such as lightning in buildings by groups, temperature sensor networks organized by areas and so forth.</p> <p>The Multicast Listener Discovery (MLD) implementation for constrained devices has been done for a research group in Aalto University (Finland) called MAM-MoTH (Massive Scale Machine-to-Machine Service). One of the goals of the group is to build a complete multicast protocol suitable for devices with constraint resources. Hence, the group needs a protocol for multicast routing and a protocol for managing the multicast groups. This last protocol is the developed in this final project. Then, both protocols will work together creating a multicast protocol suitable for constrained devices.</p> <p>In this final project, we design, implement and analyse a MLD protocol suitable for constrained devices. MLD allows an IPv6 router manage multicast groups. However, the use of classic MLD in LoWPANs has several problems such as link-local area delimitations, packet size and complexity on the role of the routers. The protocol is implemented on Contiki OS, an operating system develop for the Internet of Things. Therefore, we have extended Contiki's TCP / IP stack to support MLD.</p> <p>The protocol was tested and evaluated in different scenarios to check that it works as we designed. The library object size was analyse as well to be fitted in the Z1 device.</p>		
Keywords:	multicast communication, MLD, contiki, smart objects, Internet of Things	
Language:	English	

Acknowledgements

First of all, I am grateful to Aalto University for providing me the opportunity to work in my final project using its facilities and services.

I wish to thank Yang Deng for his time and effort following my work. I also thank Zhonghong Ou for offering me the position in MAMMoTH and the allowance of working in his research group.

I wish to express my gratitude to the Professor Antti Ylä-Jääski for supervising my final project. I also want to thank to the Professor Enrique Torres for his support and disponibility all the time from Spain. I want to thank Miika Komu for his recommendations, suggestions and availability and the guidance extended to me.

I take this opportunity to record my sincere thanks to all the members of MaMMoTH which help me out, specially Ketan Kevadiga and Klaus Hartke.

I also place on record, my sense of gratitude to my family and friends for his unceasing encouragement and support.

Espoo, September 2013

Daniel Bailo Estallo

Abbreviations and Acronyms

IoT	Internet of Things
IP	Internet Protocol
MLD	Multicast Listener Discovery
ICMPv6	Internet Control Message Protocol version 6
LLN	Low power Lossy Network
6LowPAN	IPv6 over Low power Wireless Personal Area Networks
PIM	Protocol Independent Multicast
IETF	Internet Engineering Task Force
IPSO	Internet Protocol for Smart Objects
RPL	Routing Protocol Low power lossy networks

Contents

Abbreviations and Acronyms	4
1 Introduction	7
1.1 Scope of the final project	8
1.2 Structure of the final project	8
2 Background	9
2.1 Internet of Things	9
2.2 Wireless Sensor Networks	11
2.2.1 Smart objects	13
2.2.2 Categorization. Non IP-enabled sensor networks vs IP-enabled sensor networks	14
2.3 IP-based Wireless Sensor Networks	16
2.3.1 IEEE 802.15.4	18
2.3.2 The 6LowPAN adaptation layer	19
2.3.3 RPL	20
2.4 IP Multicasting	21
2.4.1 PIM	22
2.4.2 MLD	22
3 Design	24
3.1 Introduction	24
3.2 Motivation	24
3.3 Details of MLD suitable for constrained devices	25
3.4 Protocol description	26
3.4.1 MLD message format and types	27
3.4.2 Timers and default values	27
3.5 Host state transition diagram	28
3.5.1 Link-local area	30
3.6 Router state transition diagram	31
3.6.1 Querier and Non-Querier mode	33

4	Implementation	35
4.1	Operating system: Contiki OS	35
4.1.1	Event-based kernel	36
4.1.2	Threads and Protothreads	37
4.1.3	uIP	37
4.2	Architecture	38
4.3	Components	38
4.4	Data structures and timers	40
5	Evaluation	43
5.1	Experimental environment	43
5.1.1	Cooja	43
5.1.2	Shell modules	44
5.2	Experimental evaluation	45
5.2.1	Scenario 1: 1 router and 1 host	45
5.2.2	Scenario 2: 1 router and 2 hosts	47
5.2.3	Scenario 3: 2 routers and 1 host	49
6	Discussion	51
7	Conclusions	52
A	First appendix. MLD library code	58

Chapter 1

Introduction

The *Internet of Things* (IoT) is a novel paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications [7]. Cisco ¹ states that by 2020, there will be 50 billion 'things' connected to the Internet. IoT allows for the communication between devices, commonly referred to as Machine-to-Machine (M2M) communication. With this being possible, physical devices are able to communicate to people letting them know their condition and where it is located. Many applications can be useful since this idea, monitoring individual's health, a tool that can save people money within their households, businesses management and so forth.

Even with this foreseeable success, there are several obstacles in IoT. Devices whose batteries can run down, or which have to deal with spotty or weak signals, cannot always work with the protocols and mechanisms which already exists. For that reason, many organizations are putting their efforts together to standardize technologies for IoT. IPv6 is being supported as networking technology to connect the devices. A working group of the Internet Engineering Task Force (IETF) have designed the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [35] standard which enables low power wireless devices to use IPv6. The working group Routing Over Low power and Lossy networks(ROLL) is designing a protocol called RPL: IPv6 Routing Protocol for Low power and Lossy Networks [47]. An application layer is being also developed by an IETF team, the Constrained Application Protocol (CoAP) [39]. It is noticeable that there is a significant effort researching in the IoT field. However, the area of multicast communication has not drawn that much attention.

As far as we know, the Multicast Protocol for Low Power and Lossy Networks (MPL) [26] and the Stateless Multicast forwarding with RPL (SMRF)

¹<http://share.cisco.com/internet-of-things.html>

[36] are the only multicast protocol designed specifically for LLNs. In case of MPL, there is no management of multicast groups and it is not efficient energy-wise neither reliable. SMRF only allows data transmission in one direction.

Considerable real-world sensor network applications need multicast communication such as data reporting, data monitoring, activating a group of preselected devices. The devices operating in LoWPANs has a tiny processor, low memory and constrained battery. An efficient multicast communication protocol would help in saving bandwidth and energy.

As the authors of [30] state, a potential IPv6 multicast could be developed with the routing protocol PIM-SM adapted to Source Specific Multicast (SSM) [25] and MLDv2 [44].

This final project designs and implements a Multicast Listener Discovery protocol suitable for LLNs to be used in a future in conjunction with an implementation of PIM-SM adapted to SSM for LLNs and obtain a multicast protocol for constrained devices.

1.1 Scope of the final project

The scope of the final project is limited to the design, implementation and analysis of the MLD.

During the design, we have to study all the problems found when using MLD directly in LLNs and determine valid solutions in the constrained devices environment. The protocol has been implemented on Contiki OS and uses ICMPv6 for sending the MLD messages.

1.2 Structure of the final project

The remainder of the thesis is structured as follows: Chapter 2 makes a portrait to understand the technology field related to our work. In Chapter 3, we explain how we designed the protocol and its working. In Chapter 4, we describe the implementation details. In Chapter 5, we analyze the protocol based on our experiments. Chapter 6 gives a brief overview of the future work required to improve this protocol. In Chapter 6, we provide concluding remarks.

Chapter 2

Background

In this section, a general view of the Final Project's topic is given. First of all, a global vision of Internet of Things is presented focusing in the smart objects and the Wireless Sensor Networks. Then, it is introduced the IP Wireless Sensor Networks and the protocols and mechanisms which are used (IEEE 802.15.4, 6LowPAN and RPL). Eventually, the topic is centred in the IP multicasting explaining the protocols employed to build a multicast protocol in constrained devices (PIM and MLD).

2.1 Internet of Things

The term “*Internet of Things*” (IoT) is difficult to define accurately and can have different facets depending on the perspective taken. There are many groups that have define it, and its first use was probably meant by Kevin Aston, an expert on digital innovation, in 1999.

The main idea is that the information on the current Internet is taken by humans and in this next version of Internet the data should be created by things. In Aston words [6]: “*If we had computers that knew everything there was to know about things - using data they gathered without any help from us - we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best.*”

Hence, the point of this concept is the pervasive presence around us of a variety of things or objects - such as Radio-Frequency IDentification (RFID) [46] tags, sensors, actuators, mobile phones, etc. -which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals [22].

“Internet of Things” semantically means “*a world-wide network of in-*

terconnected objects uniquely addressable, based on standard communication protocols” [40].

As the authors explains in [7], there are different oriented visions for the concept, “*Internet*” oriented, “*Things*” oriented and “*Semantic*” oriented. The initial interpretation of IoT comes from a “*Things*” oriented perspective where the things were very simple elements, RFID tags. The terms “Internet of Things” is, in fact, attributed to The Auto-ID Labs [1] who are the leading global network of academic research laboratories in the field of networked RFID. These labs have been improving the object visibility since it is a fundamental key of the path to the full deployment of the IoT vision; but it is not the only one. In a ampler sense, IoT cannot be just a global system in which the only objects are RFIDs. IoT visions recognize that the term IoT implies a much broader vision than the idea of a mere objects identification. In this sense, comes the idea of smart objects that will have identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environment and user contexts. Making a union of the Internet and Things vision we get the idea of world where things can automatically communicate to computers and each other providing services to the benefit of the human kind.

Related with the “*Internet vision*”, appears the vision of the IPSO (IP for Smart Objects) Alliance, a forum formed in September 2008 by 25 founding companies to promote the Internet Protocol as the network technology for connecting Smart Objects around the world. According to the IPSO vision, the IP stack is a light protocol that already connects a huge amount of communicating devices and runs on tiny and battery operated embedded devices. This guarantees that IP has all the qualities to make IoT a reality. By reading IPSO whitepapers, it seems that through a wise IP adaptation and by incorporating IEEE 802.15.4 into the IP architecture, in the view of 6LoWPAN, the full deployment of the IoT paradigm will be automatically enabled.

“Semantic oriented” IoT visions is behind the idea of the items involved in the Future Internet will become extremely high. Therefore, issues related to how to represent, store, interconnect, search, and organize information generated by the IoT will become very challenging. In this context, semantic technologies could play a key role. In fact, these can exploit appropriate modeling solutions for things description, reasoning over data generated by IoT, semantic execution environments and architectures that accommodate IoT requirements and scalable storing and communication infrastructure [41].

For us, Internet of Things is the possibility of connect all kind of heterogeneous networks to the Internet. Every object can be connected to the Internet, no matter how it is the protocol or mechanism is using. The figure

2.1 gives a good vision about our idea in IoT.

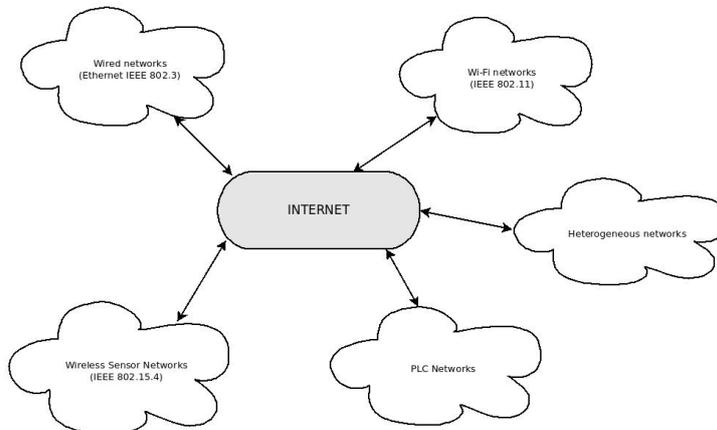


Figure 2.1: Internet of things vision

We will focus our project in the Wireless Sensor Networks which will be connected to the Internet. Therefore, the WSN will be a part of the IoT.

2.2 Wireless Sensor Networks

“Smart environments embody the following trace in building, utilities, industrial, home, shipboard and transportation systems automation. Sensor networks are the key to gather the information used by smart environments” [29]. Sensors integrated in these fields could provide extraordinary benefits to society.

“The optimal wireless sensor is networked and scalable, consumes very little power, is smart and software programmable, capable of fast data acquisition, reliable and accurate over the long term, costs little to purchase and install, and requires no real maintenance. A Wireless Sensor Network (WSN) practically consist of a gateway that can communicate with a number of wireless sensors via a radio link. Data is collected at the wireless sensor node, compressed, and transmitted to the gateway or other nodes to forward data to the gateway” [43]. There are different topologies for radio communications networks: star network (Single Point-to-Multipoint) (Fig. 2.2), mesh network topology (Fig. 2.3) and hybrid star-mesh network (Fig. 2.4).

Other important point is the election of the physical layer in WSN. The physical radio layer characterizes the operating frequency, modulation scheme, and hardware interface of the radio to the system. There are many

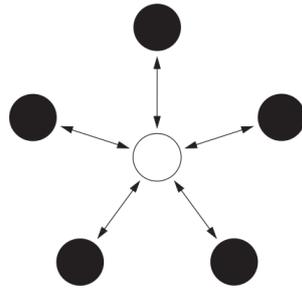


Figure 2.2: Star network topology

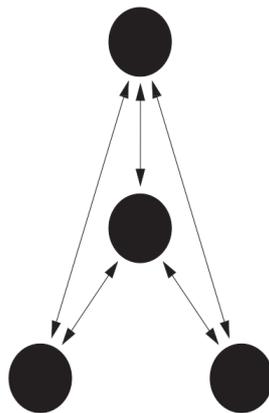


Figure 2.3: Mesh network topology

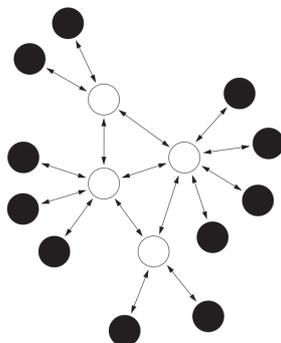


Figure 2.4: Hybrid star-mesh network topology

low power proprietary low power radio integrated circuits that are convenient choices for the radio layer in WSN, including those from companies such as Atmel, MicroChip, Micrel, Melexis, and ChipCon. It is appropriate to use a radio interface that is standards based since it allows interoperability among multiple companies [42]. Some of the existing radio standards -more information about them on the references- are IEEE802.11x [12], Bluetooth (IEEE802.15.1 and .2) [24], IEEE802.15.4 [23] , Zigbee [4] and IEEE1451.5 [28].

2.2.1 Smart objects

“The technical definition for a smart object would be an item formed by a sensor or actuator, a tiny microprocessor, a communication device and a power source. Thus, the sensor or actuator provides the smart object to transform the data captured from the sensors, albeit at limited speed and complexity” [43]. The communication device gives the communication ability to send and receive data and the power source enables the smart object to do its work. In smart objects is unquestionably important the size, they are significantly smaller than cellphones and their size cannot outreach few cubic centimeters. Devices of this kind are generally referred to as *“motes”*. A mote is an autonomous, compact device, a sensor unit that also has the capability of processing and communicating wirelessly. Even with the autonomy they present, the big strength of motes is that they can form wireless sensor networks and co-operate according to various models and architectures [5].

Despite the importance of the technical definition, we must define the

smart objects based on their behaviour, what they actually do, given their technical abilities. The behaviour completely depends on where and how it is used. The main behavioral properties are the interaction with the real world and communication. Smart objects utilize its sensors to sense physical properties ranging from simple and easy-to-measure properties such as light, temperature, and air humidity, to more complex properties such as air pollution, the presence of a car, or when an industrial machine is about to break down. Communication is fundamental to the behavior of smart objects, an smart object can be useful itself but the real power comes from the communication. The smart object that would previously switch on the door light is now able to communicate that the door was opened to every other nearby smart object. These smart objects may turn on other lights in the house, turn up the heat, and so forth [43].

2.2.2 Categorization. Non IP-enabled sensor networks vs IP-enabled sensor networks

Traditionally, sensors networks are not IP-enabled and for networking other mechanisms are used. The IP architecture was considered too heavy to use for low-power short-range networks. Therefore, several custom protocol stacks and architectures were developed. Their integration into IP-based Wide Area Network (WAN) infrastructures requires the deployment of proxies at the borders of both networking domains that transform between non-IP communication in the sensor network and IP communication in the Internet [32]. As you can see in Figure 2.5.

The conversion is operated usually at application level. Hence, the proxy queries sensor nodes for sensor data via non-IP communication and stores the data in a local database. Eventually, the proxy is inquired via TCP/IP by the users in the Internet. It might be done also at network level, the proxy would perform a communication protocol transformation. Connecting sensor networks to the Internet via proxies is not ideal. Proxies escapes the end-to-end principle of the internet and for deploying a non-IP network, there should be control and communication protocols standardized for IP communication.

As example of protocol specifications could be named: ZigBee and Z-Wave. The ZigBee specification, developed by the ZigBee Alliance, is based on the IEEE 802.15.4 radio standard and provides a set of mechanisms for creating networks of nodes as well as the establishment of applications on top of the network. The ZigBee specification is owned by the ZigBee Alliance and vendors need to join the alliance to commercialize ZigBee technology [3]. Z-Wave is another specification for low-power communication in wireless smart

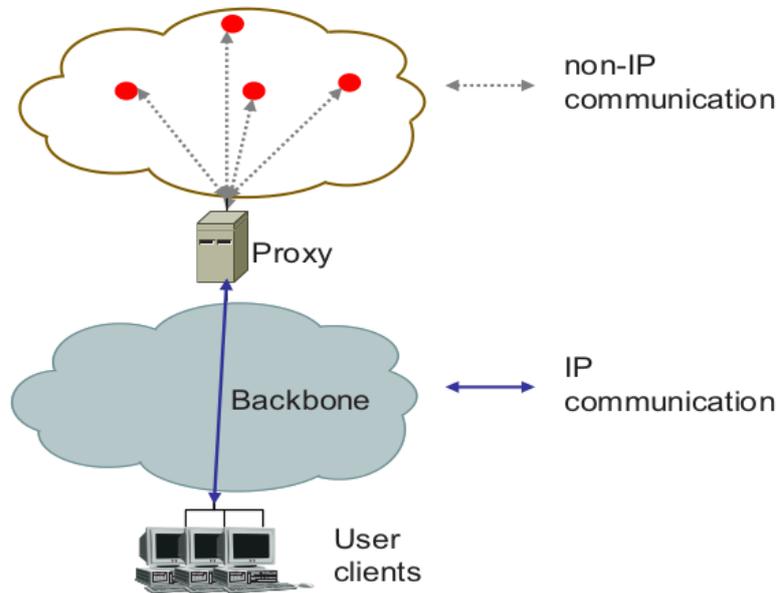


Figure 2.5: Non-IP enabled sensor network [32]

object systems. It is patented and owned by the Z-Wave Alliance. Z-Wave specifies an entire network stack from the physical layer to the application layer. The application layers are tailored to specific market segments such as home automation or energy management [2]. Neither ZigBee nor Z-Wave are compatible with IP, which is an important problem for emerging systems that need to integrate with IP-based networks and services.

By contrast, sensor networks with IP support would enable a seamless integration with WAN infrastructures since the IP-enabled sensor network would be just another part of the Internet *figura con* IP-enabled. Therefore, no special proxies are needed, taking out all the problems related to them. Evidently, gateways have to be deployed that interconnect WAN infrastructures and wireless sensor nodes but these can operate on IP level. As the Figure 2.6 represents. A lot of research has been performed in the IP world that could be reused in an IP-based wireless sensor networks, e.g, mechanisms, and protocols to support autoconfiguration, mobility, and security [32].

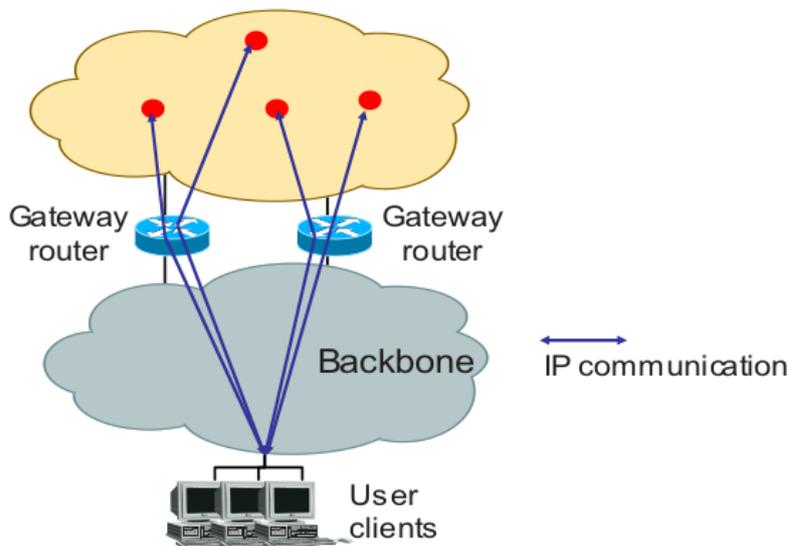


Figure 2.6: IP enabled sensor network [32]

2.3 IP-based Wireless Sensor Networks

”IP is the future for smart object networks” according to Vasseur and Dunkels [43]. Smart objects present some challenges and IP helps to solve them. Here recap some of them.

- **Interoperability.** IP runs over different link layers. Thus, IP network operates across both wired and wireless link layers without requiring any external systems. Smart object networks extend from low-power wireless nodes to high-power data coordination servers. Still, these systems need to communicate with each other and IP provides interoperability without any special mechanism that connects the systems because of its layered architecture. IP is available in most operating systems. The all-presence of IP is clear in the ever-growing number of communication technologies that support IP. Standardization portrays a significant part in the success of IP’s interoperability. IP is standardized by an established standardization organization that provides mechanisms through which new standards are reviewed and vetted. This process puts a large amount of effort into ensuring that the mechanisms and protocols proposed as standards can be efficiently implemented.

- **Evolvability.** The IP architecture has proven to be evolvable due to the end-to-end protocol principle. IP was designed to allow evolve autonomously. The end-to-end principle specifies that application layer functionality should be treat in the end points of the network. Hence, the network does not contain any application-level intelligence and only transport data between the end points. The network does not know about the information is transporting, only the applications make sense of the data.
- **Scalability.** The IP architecture has shown scalability through the use of IP over the public Internet. Ip has proven that it can be deployed over a large number of systems and it can run across huge amount of different implementations of its protocols. It is not necessary to be connected to the public Internet, large companies can span their networks with thousands of computers or servers.
- **Configuration and management.** As a consequence of the IP wide adoption and large-scale deployment, IP has developed plentiful mechanisms and protocols for network configuration and management. For smart object networks, configuration, management, installation, and commissioning are an issue. Although traditional mechanisms cannot be directly applied to smart object networks, the ability to improve existing mechanisms and tools is important.
- **Small footprint.** Low energy consumption, small physical size, and low cost are three of the node-level challenges of smart objects. These ideas means memory constraints and complexity on the nodes. IP was thought as a heavy protocol for its necessity of power and memory. A smart object has only few tens of kilobytes of memory, whereas existing IP implementations would need hundreds. That is why several non-IP stacks were developed. Later on, some lightweight implementations were developed keeping the essential mechanisms for IP.

The challenges of low-power operation and the large scale of smart object networks have impuled several years of research in the wireless sensor networks research community. Although wireless sensor networks are a subset of smart object networks, they share many of the properties such as the low-power operation, the large scale of the networks, and the resource constraints. In the following subsections, we focus on the low-power wireless networks. Therefore, it is explained the communication mechanism in the LR-WPANs (IEEE 802.15.4), the adaptation layer for IPv6 in LowPANs and RPL (IPv6 Routing Protocol for Low-power and Lossy Networks). In the Figure 2.7, it

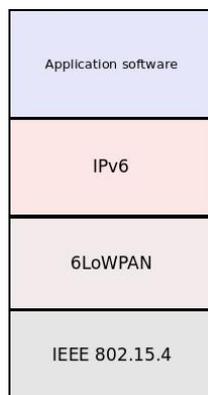


Figure 2.7: Protocol stack of an IP enabled sensor

is showed the protocol stack of an IP enabled sensor, layer by layer. Now, we will go in detail with the two firsts layers, IEEE 802.15.4 and the 6LoWPAN adaptation layer.

2.3.1 IEEE 802.15.4

IEEE Std 802.15.4 [10, 23] defines the physical layer (PHY) and medium access control (MAC) sublayer specifications for low-data-rate wireless connectivity with fixed, portable, and moving devices with no battery or very limited battery consumption requirements typically operating in the personal operating space (POS) of 10 m. The standard has been developed within the 802.15 personal area network (PAN) Working Group within the Institute of Electrical and Electronic Engineers (IEEE). IEEE 802.15 has a maximum data rate of 250,000 bits/s and a maximum output power of 1mW. The maximum packet size in 802.15.4 is 127bytes.

Two different device types can cooperate in an IEEE 802.15.4 network; a full-function device (FFD) and a reduced-function device (RFD). The FFD can work in three modes serving as a personal area network (PAN) coordinator, a coordinator, or a device. An FFD can talk to RFDs or other FFDs, while an RFD can talk only to an FFD. An RFD is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor; they do not have the need to send large amounts of data and may only associate with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity. The devices can form star and peer-to-peer network topologies. Each node in an 802.15.4 network has a 64-bit address that global uniquely identifies the device. 802.15.4 allows

nodes to use short addresses that are 16 bits long in the context of a PAN. The IEEE 802.15.4 standard does not specify any particular algorithm to be used by a PAN coordinator when assigning unique short addresses within the PAN.

The IEEE 802.15.4 architecture is specified in term of layers to facilitate the standard. An LR-WPAN device contains a PHY, which enclose the radio frequency (RF) transceiver along with its low-level control mechanism, and a MAC sublayer that provides access to the physical channel for all types of transfer.

The PHY gives two services: the PHY data service and the PHY management service interfacing to the physical layer management entity service access point. The PHY data service allows the transmission and reception of PHY protocol data units (PPDUs) across the physical radio channel. The abilities of the PHY are activation and deactivation of the radio transceiver, energy detection, link quality indication, channel selection, clear channel assessment, and transmitting as well as receiving packets across the physical medium. The radio operates at one or more of the following unlicensed bands:

- 868-868.6 MHz (e.g., Europe)
- 902-928 MHz (e.g., North America)
- 2400-2483.5 MHz (worldwide)

The MAC sublayer gives two services: the MAC data service and the MAC management service interfacing the MAC sublayer management entity service access point. The MAC data service allows the transmission and reception of MAC protocol data units (MPDUs) across the PHY data service. The abilities of the MAC sublayer are beacon management, channel access, Optional allocation of guaranteed time slots management, frame validation, acknowledged frame delivery, association, and disassociation. Like most links, IEEE 802.15.4 supports a Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) mechanism. In this algorithm the MAC first listens for energy or modulated data on the air. If none is detected, it can transmit immediately. If the channel is not clear the algorithm provides for random wait times (backoffs) before retrying the transmissions. In addition, it can be implemented application-appropriate security mechanisms in the MAC sublayer.

2.3.2 The 6LoPAN adaptation layer

6LoWPAN [13, 35] is a protocol definition to enable IPv6 packets to be carried on top of low power wireless networks, specifically IEEE 802.15.4.

6LoWPAN defines the adaptation layer's services:

- Packet fragmentation and reassembly
- Header compression
- Link layer forwarding when multi-hop is used by the link layer

The 6LoWPAN adaptation currently supports three headers: a mesh addressing header, the fragment header, and the IPv6 header compression header. The mesh addressing header is used with a mesh-under "routing" approach where nodes that are not in direct communication make use of multi-hop "routing" at the link layer using link layer addresses. According to IEEE 802.15.4, only FFDs perform mesh-under operation. RFDs send all their traffic to FFDs. 6LoWPAN supports router-over routing as well and the routing decisions are made at the network layer.

The fragmentation header is used when the IPv6 payload cannot be carried within a single IEEE 802.15.4 frame because it exceeds the MTU size. The header specifies the size, a tag and the offset. There is a maximum value of reassembling, if not all fragments have been received within this time, all fragments must be discarded. The maximum value timer is 60 seconds.

In RFC 4944 [33] is defined HC1, a stateless compression scheme optimized for link-local IPv6 communication. 6LoWPAN suppresses some fields by assuming commonly used values. HC2 is used for UDP compression. The best-case compression efficiency happens with link-local unicast communication, where HC1 and HC2 can compress a header down to 7bytes. The version, traffic class, flow label, payload length, next header and link-local prefixes for the IPv6 Source and Destination addresses are all suppressed. Improving the compression over a wider range of scenarios, the 6LoWPAN working group have standardized the header IPHC and NHC to be able to compress the Source and Destination addresses even when communicating with nodes outside the LoWPAN. When communicating with global addresses, IPHC can compress a UDP/IPv6 header down to 9 or 10 bytes (vs. 31 bytes with HC1).

2.3.3 RPL

The Internet Engineering Task Force (IETF) formed a new Working Group called ROLL (Routing Over Low-power and Lossy networks). The first objective was to determine whether or not existing IETF routing protocols would accomplish the requirements for Low-power and Lossy Networks (LLNs). The working group decided that none of the existing routing protocols would

accomplish the requirements. Hence, ROLL was reassigned to design a new routing protocol called RPL (Routing Protocol for Low-power and Lossy Networks).

This subsection describes an overview of RPL. RPL is still on progress and the IETF RFC [47] should be used as the final reference. RPL is a routing protocol for LLNs that defines how to build a Destination Oriented Directed Acyclic Graph (DODAG) using an Objective Function (OF) and a set of metrics/constraints. There could be different OFs in operation since there is a wide variety of possible objectives looking for the "best path".

The DODAG built by RPL is a logical routing topology over a physical network to meet a specific criteria and the network administrator may decide to have multiple DODAGs active to carry traffic with different set of objectives. A node can participate and join one or more graphs (RPL instances) and record the traffic according to the DODAG characteristic to support QoS and constraint based routing.

The DODAG building process is not explained here in more detail but to go deeper, the RFC and the white paper of IPSO about RPL are pretty interesting.

2.4 IP Multicasting

Multicast is an essential service in Wireless Sensor Networks, since it may allow an efficient way for group communication. A set of receivers show their interest in receiving a particular data stream. Multicast minimizes the number of transmission and forwarding in each sensor node [11]. The hosts that are interested in receiving packets join a multicast group using Internet Group Management Protocol (IGMP) and Multicast Listener Discover (MLD) protocol in IPv4 and IPv6, respectively. MLDv1 and MLDv2 are specified in [14, 44]. Both are in many ways similar to IGMPv2 [21] and IGMPv3 [9]. The main difference between MLD and IGMP is that MLD uses ICMP, whereas IGMP packets are encapsulated in IP packets. The protocols manage the communication between hosts and router. Each router keeps a list of active members per multicast group.

Forwarding of multicast packets is managed by the routing protocols. A number of multicast routing protocols have been designed for IPv4: Distance Vector Multicast Routing Protocol (DVMRP) [45], Multicast Open Shortest Path First (MOSPF) [34], Protocol Independent Multicast (PIM) [20], Core Based Trees (CBT) [8], and so forth.

As the author affirms in [31], PIM is the most common form of multicast. "Independent" means that it relies on the underlying network routing protocol

and this fact suits with the existence of standardized routing protocols in WSN. PIM has few variants.

In the following subsections, we will briefly describe PIM and MLD. Therefore, both protocols together would form a multicast protocol, PIM for routing and MLD for managing the multicast groups.

2.4.1 PIM

Protocol Independent Multicast is a notable multicast routing protocol for IPv4 and IPv6. PIM essentially necessitate that all routers in a routing domain support the protocol. PIM is called "Protocol Independent" because it is independent from the routing protocols. Within PIM, only multicast group information is changed among the routers. There are two modes of PIM: PIM Dense Mode (PIM-DM) and PIM Sparse Mode (PIM-SM). We will explain PIM-SM since in WSN is generally a sparse topology.

PIM-SM performs on the explicit-join model. In PIM-SM, a core router is called *Rendezvous Point*(RP). When a router receives a join message, it sends a PIM Join message containing the multicast group address toward the RP for the group address. Then the RP updates the rest of the routers. This process will create a packet distribution tree for the group from the RP to leaf networks. When a multicast source sends a multicast packet to a particular group, the first hop router encapsulates the packets inside a PIM Register message, and forwards it to the RP through unicast routing. The RP then decapsulates the original packet and distributes it along the distribution tree.

2.4.2 MLD

The Multicast Listener Discovery protocol is the multicast group management protocol for IPv6 and is used to exchange group information between multicast hosts and routers. MLDv1 is specified in the RFC 2710 [14] and MLDv2 in the RFC 3810 [44]. Both done by the IETF's Network Working Group. MLD is specified as part of ICMPv6. MLD messages are normally sent with a IPv6 link-local source IP address. The hop limit is always 1. Hence, it prevents the transmission of MLD messages by a router.

As we mentioned earlier, there is two versions of MLD, MLDv1 and MLDv2. The messages in MLD consists in the following ones: Multicast Listener Query, Multicast Listener Report and Multicast Listener Done. MLDv2 add some features. For example, MLDv2 allows a host to specify the source for a particular multicast group, this feature is called "Source-Filtering".

Through the work of this final project, none open source implementations has been found. We found a closed MLD implementation developed by Cisco

for networks with high-resources devices. But not even closed implementations for constrained devices.

There are some problems in the direct application of MLD on constrained devices such as the definition of the link-local areas, the packet size and the complex role of the routers. We will explain the MLD protocol in more detail in the next sections since is the protocol that we are designing and implementing in this final project.

Chapter 3

Design

In this section, we explain an introduction about the classic MLD. We also describe a more detailed description of the protocol. Finally, the host and router transitions diagrams and what decisions we have made to suit MLD on constrained devices.

3.1 Introduction

The Multicast Listener Discovery protocol for IPv6 is used to discover the presence of multicast listeners (nodes wishing to receive multicast packets) on its directly attached links and to discover specifically which multicast addresses are of interest to those neighboring nodes. The multicast groups information is provided to the routing protocol to ensure that multicast packets are delivered to all links where there are interested receivers. MLD is an asymmetric protocol, specifying different behaviors for multicast listeners and for routers.

3.2 Motivation

There is plenty of application in LLNs where multicast communication could be beneficial. In LoWPANS, there will be hosts and routers communicating for performing a task. A good example where the multicast communication would be useful is in a building management system. An actuator could switch on/off the devices in a particular floor organized by multicast groups for different floors. We design a management group protocol by learning if there are listeners for the needs of these applications. In this section, we briefly explain why we choose a listener discovery method and not an on-demand method.

On demand way, the router will be on a listening state waiting for potential reports from the interested nodes on the link-local area. Thus, the node is responsible for letting know to the router its interest. When a node want to join a multicast group has to send a report and then it will have a timer to send periodically a report to keep the multicast group router's list updated. Therefore, the node has two transition states, no listening and delaying listener. In the no listening state, it will not receive any multicast data message and in the delaying listener, it will be receiving all multicast data messages from the groups it joined and it would be informing about the continue interest in those multicast addresses.

Discovery way, this strategy is completely the opposite. The router holds the responsibility of its multicast addresses list's maintenance. The router queries periodically based on a timer and when the node receives this message, starts a timer per address and when the timer expires, it sends a report message to show the interest of the multicast address that it wants to listen. This method add another state in the node since when a node receive a report for a multicast address that it is interested, it will stop the timer and it will be in an idle state until it gets another query.

To compare and evaluate them, we will consider the node performance and the network traffic. In the on-demand way, the node has to be updating the router every time. Hence, when the node is listening, it always has to be running a timer for sending reports to the router. However, in the discovery way, the node only works when it receives a query. Until that moment, the node can be idle. The same idea comes up in the network traffic on the link local area, on-demand way every node sends a report thus the router handles all the reports from the nodes. On the contrary in the discovery strategy, the router receives only one report per multicast address if there are listeners.

To sum up, the discovery way has lower node performance because there is no need of having a timer running all the time and it can be in an idle state and it congests less the network since only one node sends report per multicast group. Therefore, MLD and its discovery way is better for constrained devices.

3.3 Details of MLD suitable for constrained devices

First of all, we have to consider that MLD will be used with PIM-SM with SSM to build a complete multicast protocol. Consequently, we have to design the source-filtering feature. In the MLDv2's RFC, there is an explanation

about the source-filtering but it is a bit heavy-weight since it requires big messages and it is not suitable for constrained devices. The point is to simplify the idea of MLDv2, therefore the router will keep not only the multicast address group but a unique source address also. We accomplish the idea of SSM, and in every report message, done message or storage for multicast group list there will be the pair (Group address, Source address). In the next section, we describe how MLD works based on the MLDv1's RFC.

3.4 Protocol description

Routers use MLD to learn which multicast addresses have listeners on their attached links. Each router has a list with the multicast addresses which has listeners and a timer associated with each of those addresses. The important part to consider is that the router only needs to know the existence of listeners per address, it does not need to know their identity (e.g. , unicast address). A router transmits all MLD packets on its link-local area. A router may have two possible roles: Querier and non Querier. All router starts as a Querier and if a router hears a Query message whose IPv6 Source Address is numerically lower than its own address for that link then, it must become a Non-querier for that link. A querier periodically sends a General query to ask for reports of all multicast addresses of interest of that link. General queries are delivered to the link-scope all-nodes multicast address (FF02::1). When a node receives a General Query, it sets a delay timer for each multicast address to which it is listening. Same behavior with the Specific-address query but only for the address required. When the timer expires, send a report to inform its interest. If a node receives a report from an interface for a multicast address while it has a timer delay running for that same address, it stops and does not send a Report for that address, therefore it avoids duplicate reports. When a router gets a report, if the reported address is not on the router's list of multicast addresses, the reported address is added to the list. If a report is received for a multicast address that already exists, just set the timer. When the timer expires, it is assumed there is no more interested listeners on the link and the address is deleted from the list. When a node want to start listening to a multicast address, it should send an unsolicited report. When a node does not want to listen anymore, it should transmit a Done message for that address. When a router receives a Done, it makes sure that there are no more listeners for that address, sending a specific-query.

3.4.1 MLD message format and types

MLD uses ICMPv6 message types. The ICMPv6 packet format is shown in Figure 3.1. All MLD messages are sent with a link-local IPv6 Source Address, an IPv6 Hop Limit of 1, and an IPv6 Router Alert option in a Hop-by-Hop Options header.

ICMPv6 packet			
Bit offset	0–7	8–15	16–31
0	Type	Code	Checksum
32	Message body		

Figure 3.1: ICMPv6 packet format

There are three types of MLD messages:

- Multicast Listener Query (ICMPv6 type = decimal 130). There are two subtypes:
 - General Query, used to learn which multicast addresses have listeners on a link.
 - Multicast-Address-Specific Query, used to determine if a particular multicast address has any listeners.
- Multicast Listener Report (ICMPv6 type = decimal 131)
- Multicast Listener Done (ICMPv6 type = decimal 132)

The code in all messages is set to 0.

3.4.2 Timers and default values

The timers are configurable. Here explain the more important values to understand how works the MLD protocol.

- **Robustness variable** allows tuning the expected packet loss on a link. The more possibilities to loss a packet, the bigger has to be the constant. The default value in the RFC is 2.
- **Query Interval** is the interval between General Queries sent by the router. The default value is 125 seconds and it can be selected depending on the network needs.

- **Query Response Interval** is the time that the host has to reply since it gets the query from the router. Default value: 10 seconds.
- **Multicast Listener Interval** is the amount of time that must pass before a router decide there are no more listeners. This value must be (the Robustness variable) times (the Query Interval)) plus (one Query Response Interval).
- **Last Listener Query Interval** is the time that the host has to reply when it get a multicast specific address query from the router. Default value 1 second.

3.5 Host state transition diagram

Host behavior is more specified by the state transition diagram in the figure 3.2 . A node may be in one of three possible states with respect to any single IPv6 multicast address on any single interface:

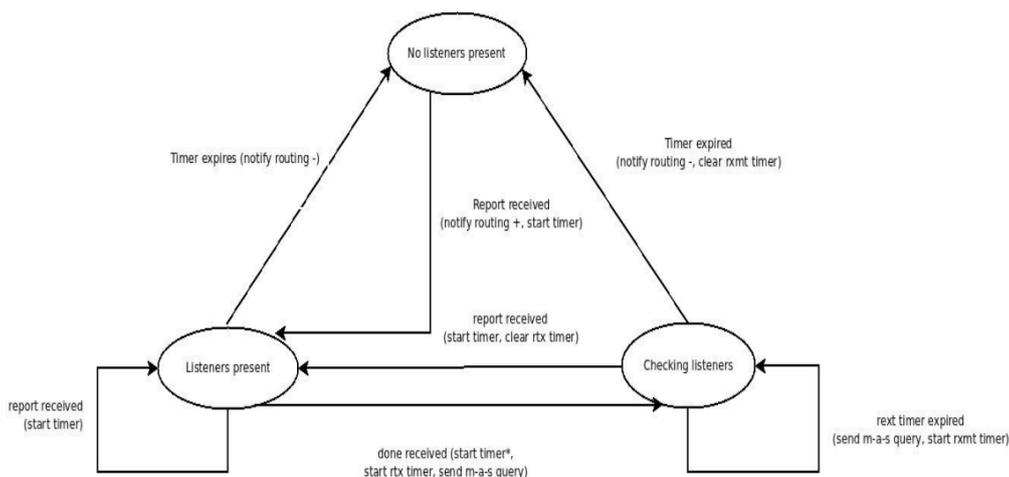


Figure 3.2: Host state transition diagram

- “Non-Listener” state, when the node is not listening to the address on the interface. This is the initial state for all multicast addresses on all interfaces; it requires no storage in the node.
- “Delaying Listener” state, when the node is listening to the address on the interface and has a report delay timer running for that address.

- “Idle Listener” state, when the node is listening to the address on the interface and does not have a report delay timer running for that address.

There are five significant events that can cause MLD state transitions:

- “start listening” occurs when the node starts listening to the address on the interface. It may occur only in the Non-Listener state.
- “stop listening” occurs when the node stops listening to the address on the interface. It may occur only in the Delaying Listener and Idle Listener states.
- “query received” occurs when the node receives either a valid General Query message, or a valid Multicast-Address-Specific Query message.
- “report received” occurs when the node receives a valid MLD Report message. It is ignored in the Non-Listener or Idle Listener state.
- “timer expired” occurs when the report delay timer for the address on the interface expires. It may occur only in the Delaying Listener state.

All other events, such as receiving invalid MLD messages or MLD message types other than Query or Report, are ignored in all states.

There are seven possible actions that may be taken in response to the above events:

- “send report”. The Report message is sent to the address being reported.
- “send done” for the address on the interface. If the flag saying we were the last node to report is cleared, this action MAY be skipped. The Done message is sent to the link-scope all-routers address (FF02::2).
- “set flag” that we were the last node to send a report for this address.
- “clear flag” since we were not the last node to send a report for this address.
- “start timer” for the address on the interface, using a delay value chosen uniformly from the interval $[0, \text{Maximum Response Delay}]$, where Maximum Response Delay is specified in the Query. If this is an unsolicited Report, the timer is set to a delay value chosen uniformly from the interval $[0, [\text{Unsolicited Report Interval}]]$.

- “reset timer” for the address on the interface to a new value, using a delay value chosen uniformly from the interval $[0, \text{Maximum Response Delay}]$, as described in “start timer”.
- “stop timer” for the address on the interface.

In the following subsection, we explain the problem related with the definition of link-local area in constrained devices. We have represented a flow sequence over the topology drawn in the Figure 3.3 and study the solutions for them.

3.5.1 Link-local area

Constrained devices works in Wireless Sensors Networks and the link-local areas are defined by the radio range of the routers. MLD explains clearly that all MLD messages are sent in the link-local area scenario. But how define link-local areas to make MLD work properly. Applying directly MLD in constrained devices, gives some undesired behaviors. To illustrate it the following example over the Figure 3.3.

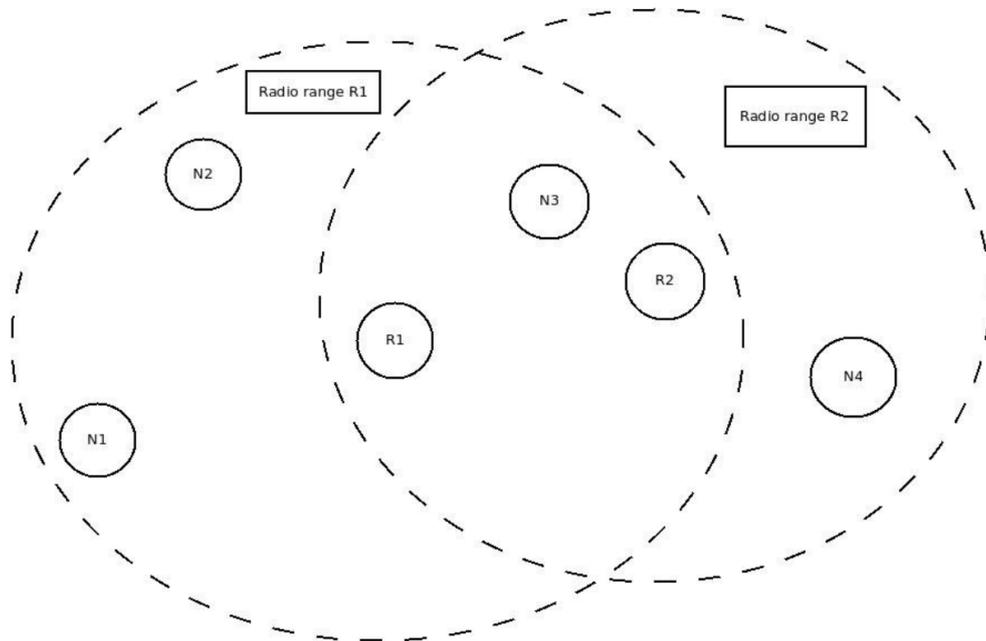


Figure 3.3: Example of scenario

1. N3 sends a report for (Group1, Source7) and reach R1 and R2

2. R1 receives the report, start a timer for this pair (G,S) and notify the routing protocol to get the multicast packets from this group and source
3. R2 does the same actions like R1
4. Source 7 sends a multicast data packet in the group1. R1 and R2 has notified they have listeners so they receive the message and broadcast it in the link-local area.
5. N3 is reachable from R1 and R2 thus N3 receives the multicast data message twice.

To solve this problem of reachability that the link-local areas define, the nodes send the report message with the pair (G,S) and also filtering the preferred parent for reachability. For instance, N3 should send a report with (Group1, Source7, R1)

3.6 Router state transition diagram

Router behavior is more specified by the state transition diagram in the figure 3.4 . A router may be in one of three possible states with respect to any single IPv6 multicast address on any single attached link:

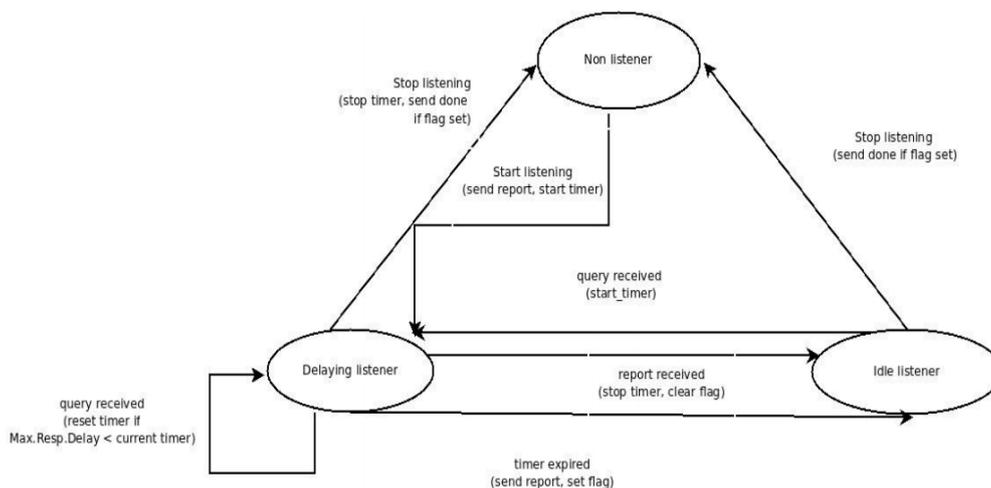


Figure 3.4: Router state transition diagram

- “No Listeners Present” state, when there are no nodes on the link that have sent a Report for this multicast address. This is the initial state for all multicast addresses on the router.
- “Listeners Present” state, when there is a node on the link that has sent a Report for this multicast address.
- “Checking Listeners” state, when the router has received a Done message but has not yet heard a Report for the identified address.

There are five significant events that can cause router state transitions:

- “report received” occurs when the router receives a Report for the address from the link.
- “done received” occurs when the router receives a Done message for the address from the link. This event is significant only in the “Listeners Present” state.
- “multicast-address-specific query received” occurs when a router receives a Multicast-Address-Specific Query for the address from the link. This event is significant only in the “Listeners Present” state.
- “timer expired” occurs when the timer set for a multicast address expires. This event is significant only in the “Listeners Present” or “Checking Listeners” state.
- “retransmit timer expired” occurs when the timer set to retransmit a Multicast-Address-Specific Query expires. This event is significant only in the “Checking Listeners” state.

There are seven possible actions that may be taken in response to the above events:

- “start timer” for the address on the link - also resets the timer to its initial value if the timer is currently running.
- “start timer*” for the address on the link - this alternate action sets the timer to the minimum of its current value and either $[\text{Last Listener Query Interval}] * [\text{Last Listener Query Count}]$ if this router is a Querier, or the Maximum Response Delay in the Query message $* [\text{Last Listener Query Count}]$ if this router is a non-Querier.
- “start retransmit timer” for the address on the link $[\text{Last Listener Query Interval}]$.

- “clear retransmit timer” for the address on the link.
- “send multicast-address-specific query” for the address on the link. The Multicast-Address-Specific Query is sent to the address being queried, and has a Maximum Response Delay of [Last Listener Query Interval].
- “notify routing +” internally notify the multicast routing protocol that there are listeners to this address on this link.
- “notify routing -” internally notify the multicast routing protocol that there are no longer any listeners to this address on this link.

The MLD has two modes for router: Querier and Non-Querier but we will not go on detail over them in this final project because for constrained devices, we will not use them.

In the following subsection, we explain why we do not use the non-querier router mode. We have created a flow sequence in the topology showed in the Figure 3.3 to explain it.

3.6.1 Querier and Non-Querier mode

The two modes of the router produces some undesired problems because of the definition of the link-local areas.

Given the routers R1 and R2 and the nodes N1, N2, N3 and N4. Describe an undesired situation. Supposing that R1 and R2 are set up as Querier and the nodes has already done some unsolicited reports to join some multicast groups.

1. R1 sends a general query
2. N1, N2, N3 and R2 receive the query
3. N1, N2 and N3 start their delay timers for their multicast addresses
4. R2 handles the query and realizes the source IP address is numerically lower and change its role as Non-querier
5. N4 will not be queried anymore and will be idle
6. N4 sends a done that only receives R2 since N4 can't reach R1 and R2 in Non-querier role ignores message

Hence, N4 could not drop out of the multicast group which sent in the done message. In conclusion, to solve this problem we take the measure of eliminate the non-querier role in the routers and the maintenance of the link-local areas with the multicast groups will be assured. Without non-querier role, there will be more network traffic since all nodes will query but also improve the performance of the router thus it does not need a timer to keep control of being Querier or non-querier and it does neither need the handler for MLD query messages.

Chapter 4

Implementation

In this section, we will discuss the implementation of our protocol. Firstly, we will introduce the operating system where we have developed the protocol, Contiki OS. Then, we will describe the architecture of the protocol and how it fits into the contiki communication system. We will then discuss the components in the implementation and also include a few points about the data structures used.

4.1 Operating system: Contiki OS

Contiki operating system [17] is an open source operating system for the IoT. Contiki connects tiny, low-cost, battery-operated and low-power systems to the Internet ¹. The first version was released in 2003. It is built around an event-driven kernel and features include dynamic loading and unloading of programs and services, and pre-emptive multithreading. Contiki supports a full TCP/IP stack by the uIP library, as well as the programming abstraction Protothreads. Contiki is implemented in the C language and has been designed to be easily portable to new platforms. A Contiki system is partitioned into two parts: the core and the loaded programs as shown in Figure 4.1. The partitioning is made at compile time and is specific to the deployment in which Contiki is used. As the figure shows, the core consists of the Contiki kernel, the program loader, the most commonly used parts of the language run-time and support libraries, and a communication stack with device drivers for the communication hardware. The core is compiled into a single binary image that is stored in the devices prior to deployment. The core is generally not modified after deployment, even though it should be

¹<http://www.contiki-os.org/>

noted that it is possible to use a special boot loader to overwrite or patch the core.

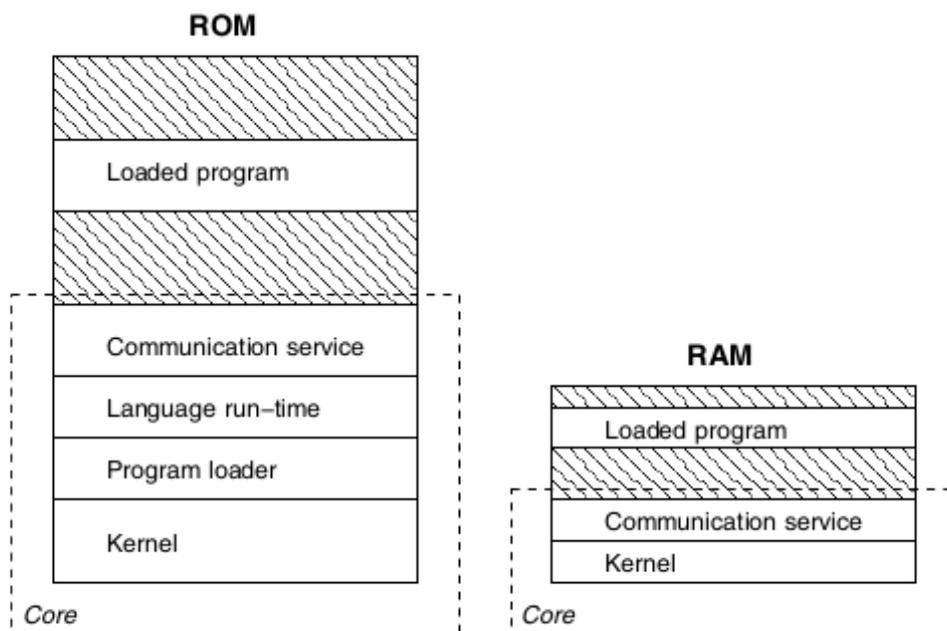


Figure 4.1: Partitioning into core and loaded programs

4.1.1 Event-based kernel

In an event-based system, a process is implemented as an event handler and holds the responsibility of executing different blocks of code depending on the given event. Once the event has been dispatched to a process, the respective event handler in the process cannot be pre-empted and has to run to completion. Since a single code block will never be interrupted, these blocks can be designed for sharing the same stack. Compared to a multi-threaded model this requires less memory and computation overhead when having several concurrent processes. In Contiki, a process consists of an event handler and an optional poll handler function. The Contiki kernel holds the event scheduler that dispatches events to processes and periodically polls processes that registered a poll handler function. It uses a single stack for all processes, which is rewound between each invocation of an event handler.

4.1.2 Threads and Protothreads

Unlike most other event-based systems, Contiki supports pre-emptive multi-threading [37]. This allows nodes to run applications that normally do not fit well in purely event-based systems. Besides pre-emptive threads, Contiki also supports Protothreads [18]. Generally, when writing programs for event-driven systems, these have to be written as explicit state machines with a resulting code that can be hard to understand and maintain. Protothreads is a programming abstraction used on top of these systems, with the purpose to simplify implementations of high-level functionality. By using Protothreads, programs can perform conditional blocking without the overhead of regular threads; Protothreads are stackless. A regular Contiki process consists of one single Protothread.

4.1.3 uIP

uIP (micro IP) is an implementation of the IP stack developed to fit in the memory requirements of smart objects and other networked embedded systems. uIP is the principal IP communication component of the Contiki OS. uIP has very low memory requirements. In its default configuration, it requires only about one kilobyte of RAM and few kilobytes of ROM [43]. This includes the IP, ICMP, UDP, and TCP protocols. uIP has only the minimum of required features for a full TCP/IP stack. uIP uses three methods to reduce code size and memory usage: an event-driven programming interface, an intentionally simple buffer management scheme, and a memory-efficient TCP implementation.

The first versions of the uIP stack only supported IPv4 communication but in 2008 Cisco Systems extended uIP with IPv6 capabilities [19]. uIPv6 stack was the first stack to comply with all the IPv6 requirements, which enabled it to use the IPv6 Ready logo.

The uIP's principle of operation is simple. It does three tasks: processing the packets that arrive from the communication device driver, processing requests for the application layer and it does periodic processing.

When a packet is received, the input processing function is called. The headers of the incoming packet are processed and passes to the application if it contains application data. The application may reply and the response packet will be handled by the outprocessing part of uIP. The outprocessing code starts when the application has given data to be sent. The outprocessing part adds protocol headers to the packet and deliver the packet over the communication device for transmission. There is a periodic processing for control the timer-based actions such as retransmissions[43].

4.2 Architecture

The communication stack in contiki is described in 4.2. In the figure we focus only on IPv6 packets in the interest of our implementation. An application would send packets using the IPv6 interface provided for the operating system. From the uIPv6 layer, the packet continue to the 6LoWPAN adaptation layer to compress the headers and possible fragmentation. Once compressed and fragmented, the packet flows to the CSMA MAC layer. The MAC layer puts the data packet in a queue. The radio duty cycling (RDC) layer sends the packet on this queue according to its duty cycling algorithm. The radio link layer send the packet. The MAC layer does not drop the packet from the queue until it gets the acknowledgment from the link layer. When a packet is received in the radio link layer, the process is the same but on the contrary direction. The 6LoWPAN holds the responsibility of decompress and reassemble the packets [15].

The communication layers in the Contiki stack are implemented as protothreads. Although to reduce the code size, IPv6, TCP and UDP layer are a single protothread. The MLD messages as we have already explained use ICMPv6. In contiki, there is a library which implements ICMPv6 functions such as send, receive, and so forth. These functions use the uIPv6 layer of Contiki since ICMPv6 packets are transported by IPv6.

The 6LoWPAN adaptation layer in Contiki OS is not a process, but initialized by the MAC layer. For outgoing packets, TCP/IP layer calls the function in 6LoWPAN. For incoming packets, MAC layer calls the function.

The MAC layer is a separate process itself. The default MAC layer is CSMA/CA implementation. The default RDC layer is ContikiMAC [16]. The radio link layer consists of the radio driver that is responsible for transmission and reception of packets.

4.3 Components

There are three main parts in MLD protocol processing. One is the handlers for the MLD messages. The second one is the sending of the different MLD messages. The third part is the callbacks and the timers. There are timers for every multicast group and when it expires it has to be handled.

- **Handlers.** As we explained in Section 3.4 in the protocol design, the hosts and the routers have different states per multicast group. The handlers are the responsible of keeping the maintenance and status in the correct way as the protocol design states. The handlers are trig-

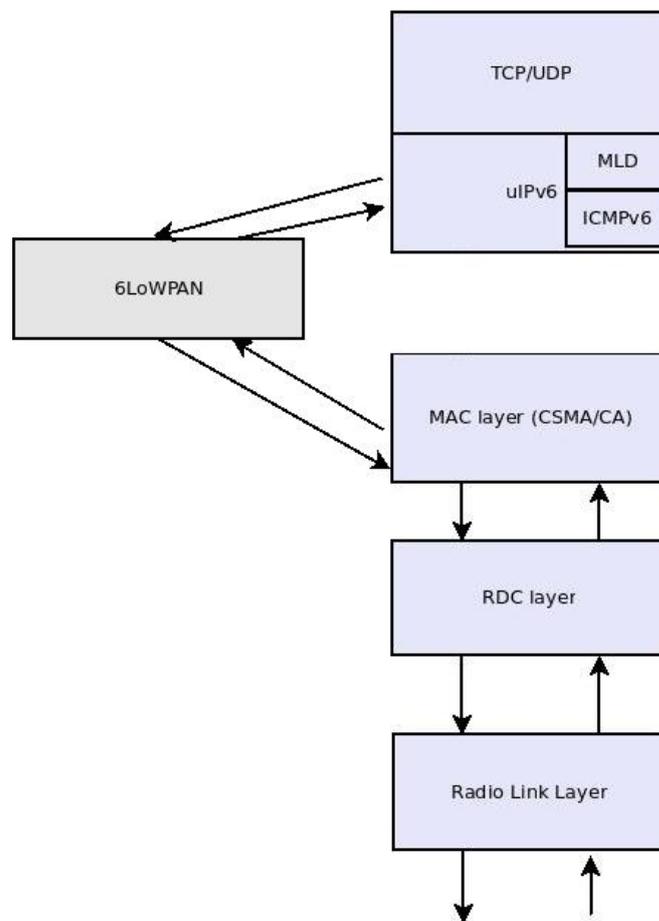


Figure 4.2: The communication stack in Contiki OS

gered when a node receives a MLD message. Therefore we have implemented the functions `mld_query_in()`, `mld_report_in()` and `mld_done_in()`. Since the MLD protocol is asymmetric, having different behaviors for multicast listeners and for routers. We have used the C preprocessor variables to implement the different roles. Using `UIP_CONF_ROUTER` for the routers and `RPL_CONF_LEAF_ONLY` for the multicast listeners.

The router's states are defined with the following constant variables `NO_LISTENERS`, `LISTENERS`, and `CHECKING_LISTENERS`. In the same way, the host's states with the following ones `DELAYING_LISTENER` and `IDLE_LISTENER`. The "Non listener" state does not need to be stored.

- Sending MLD messages. We have defined a function per MLD message type. Hence, we implement `mld_gen_query()`, `mld_mas_query()`,

`mld_report()` and `mld_done()`. For MLD general query, MLD specific query, MLD report and MLD done respectively. All messages go through ICMPv6. The destination of the messages is “*all link-local nodes*” but the MLD done message which flows to “*all link-local routers*”. To accomplish the design, depending of the message, the packet carries different payload. For the general query, no info extra is needed. Hence, the payload is 0. The specific query needs to ask about the group and source which is considering to turn to the *Non listeners* state. Therefore, the payload is the tuple (Multicast Group address, Source address). The report and done messages has the same payload, they need to inform about its (Multicast Group address, Source address, Preferent parent address).

- **Callbacks.** The callbacks are functions which are triggered when a specific timer expires. To implement the behavior of the nodes, we need the following callbacks `timer_host_expired()`, `timer_group_expired()` and `rtxtimer_expired()`. The first callback is called when the multicast group timer expires in the listener and it has to send a report to continue listening. The second callback is triggered when a multicast group in a router realizes that there is no more listeners. The third one is called when the router is trying to guess if there are still some interested listeners in the multicast group.

To conclude with the components, we have to explain how our protocol is connected with Contiki OS. In the file `uip6.c` when the process is checking the headers of the IPv6 packet, we have added our handlers if the packet belongs to the MLD protocol.

4.4 Data structures and timers

As we have already stated, both the routers and the hosts have to save some information per multicast address group. In Contiki OS, the memory is constrained. Hence, we have implemented some structures with the data that we strictly need it. The following code shows the information that each router has per multicast group.

```
struct mcast_group {
    int used;
    uip_ipaddr_t mcast_group;
    uip_ipaddr_t source;
    uint8_t state;
```

```

    struct ctimer timer;
    struct ctimer rtx_timer;
};

typedef struct mcast_group mcast_group_t;

static mcast_group_t mcast_group_table[MAX_NUM_OF_MCAST6_GROUPS];

```

The data type is called `mcast_group_t`. There are different fields to keep the maintenance correctly. The field `used` is admin's information and it is used to know if the multicast group is available and in use. If `used` is 0, you can overwrite the record with other multicast group. The two next fields `mcast_group` and `source` belong to the tuple (Group, Source) that we need to store for multicast group. The field `state` is to control the possible states of the multicast group. They were explained in Section 4.3. Finally, the both timers, `timer` and `rtx_time`. The first one, to know if there are no more listeners per group and the second one, to send the multicast specific query when the router want to check if there are more listeners.

The following code is the structure that the host stores per listening multicast group.

```

struct mld6_listen_group {
    int used;
    uint8_t mld_flag; /* Whether there is more listeners than you*/
    uint8_t state;
    uip_ipaddr_t mcast_group;
    uip_ipaddr_t source; /* Source-filtering*/
    uip_ipaddr_t preferred_parent;
    struct ctimer delay_timer; /* [0, Max.Respons.Delay] */
};

typedef struct mld6_listen_group mld6_listen_group_t;

static mld6_listen_group_t listening_table[MAX_NUM_OF_MCAST_LISTENING];

```

The field `used` is for the same purpose as in the router's structure. The fields `mld_flag` and `state` works like specified in Section 3.5 and Section 4.3, respectively. The next three fields are the addresses necessary for the multicast group, the tuple (Group, Source, Parent), `mcast_group`, `source` and `preferred_parent`. Finally, the timer `delay_timer` which is used to control when the host have to send a report to keep listening to the multicast group.

Both structures have their functions to keep their maintenance such as add groups, erase groups, find groups or print them. For the `mcast_group_table`, we have implemented `mcast_group_new()`, `mcast_group_find()` and `print_mld_router_table()`. For the `listening_table`, we have implemented `new_listener()`, `find_listener()`, `erase_listener()` and `print_mld_host_table()`. These functions are also used for debugging and testing.

About the timers, to implement the callbacks we have used the library `ctimer.h`. This library allows to dispatch callbacks when the timer expires. We have already talked about the callbacks in this chapter. To set the timer, you have to choose the timer, the time, the function which will be triggered and the parameters which will be used in the callback function. The parameters has to be referred as a void pointer. The next code present how to set the timer with the callback function.

```
ctimer_set(&aux_mcast->timer, MCAST_LISTENER_INTERVAL * CLOCK_SECOND,
          timer_group_expired , &aux_mcast);
```

There are some global constants in MLD protocol. All of them explained in the Chapter 3.

```
#define ROBUSTNESS_VARIABLE 2
#define QUERY_INTERVAL 125
#define QUERY_RESPONSE_INTERVAL 10
#define MCAST_LISTENER_INTERVAL 260
#define LAST_LISTENER_QI 1
```

Chapter 5

Evaluation

In this chapter we test and evaluate our protocol in some scenarios which uses the IPv6 stack in Contiki. Therefore, we want to show how MLD works integrated with other components in Contiki OS. We start by explaining the experimental environment. We follow with a description of the evaluation in the different scenarios to check if we achieve the behavior we designed.

5.1 Experimental environment

In this section we describe the evaluation environment we use for evaluating our protocol. First of all, we explained which simulator we are using. The similar is Cooja, a tool included in Contiki OS. Then, we explain what modules we have implemented to send asynchronous messages via shell.

5.1.1 Cooja

Cooja [38] is a Java-based simulator designed for simulating sensor networks on Contiki OS. The simulator is implemented in Java but the sensor node software is written in C. In Cooja is possible to simulate at different levels; Network level, Operating System level and Machine code level.

Cooja supports simulation of nodes of mixed types as well. In the same time, Cooja simulates the radio medium using simple models. Overall, Cooja provides a very usable environment that visualizes radio events, captures packets, monitors variables, visualizes the network, shows nodes' logs and supports scripting and test automations. The interface of Cooja is shown in Figure 5.1. It eases the development and evaluation of network protocols and embedded development by a great deal.

In Cooja, we can develop a contiki application in C, test the code and

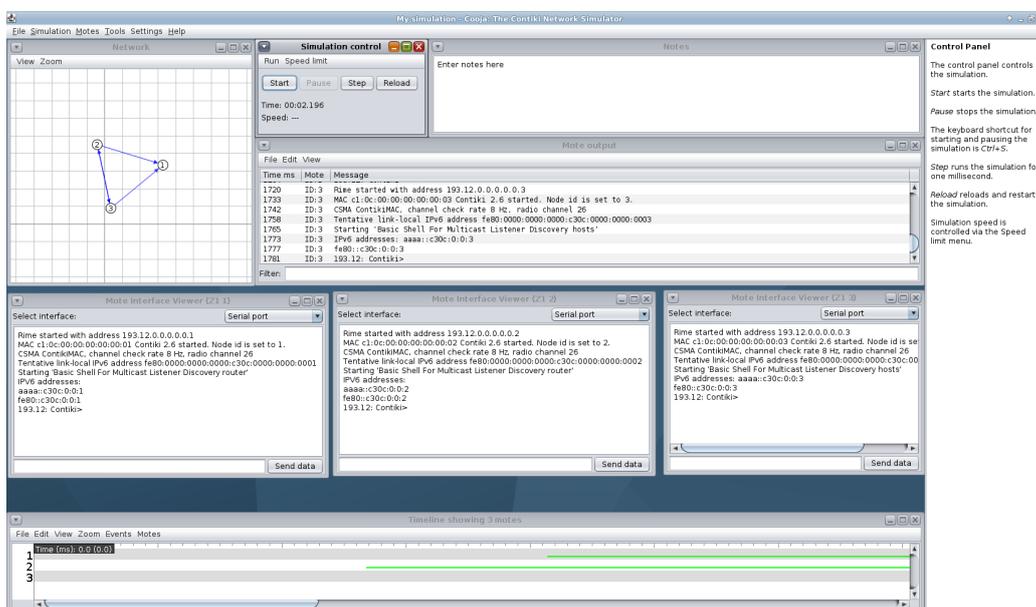


Figure 5.1: Cooja interface

then use the same code to run the application on a real hardware. The code can be compiled to be used in the native configuration of Cooja or based on a specific hardware and Cooja will simulate it.

In this final project, we have coded in C, compiled it for specific hardware, Zolertia Z1¹, and used the cooja simulator for testing.

5.1.2 Shell modules

Contiki provides an optional command-line shell with a set of commands that are useful during development and debugging of Contiki systems. With Unix-style pipelines, shell commands can be combined in powerful ways. Applications can define their own shell commands that work together with existing commands.

We have implemented two modules to have our own commands for the host and for the router. Therefore, we have developed the `shell-mld-host` and the `shell-mld-router` modules in the `apps` folder of Contiki OS.

The first module allows to send unsolicited reports, to send done messages and to print the multicast groups that the host is listening. The second module allows the router to print its information about its multicast groups and their states. The commands are sent via serial port. Here, an example

¹<http://www.zolertia.com/ti>

about how to use the commands.

```
send_report group source pref_parent
send_done group source pref_parent
print_host_table
print_router_table
```

5.2 Experimental evaluation

In this section, we describe the experiments we run to evaluate our MLD suitable for constrained devices. We have built three different scenarios in LLNs to generate all the possible behaviors of the routers and the hosts (normal working and corner cases). In the following subsections, we will explain each scenario, describing what we command via shell and what it should happen to agree with our protocol objectives. With this sequences, we want to show the proper working of the protocol.

5.2.1 Scenario 1: 1 router and 1 host

This scenario is represented in the Figure 5.2. The green circle denotes the transmission range of node 1 while the gray circle denotes its collision with other radios. The size of the radio range can be adapted in the code.

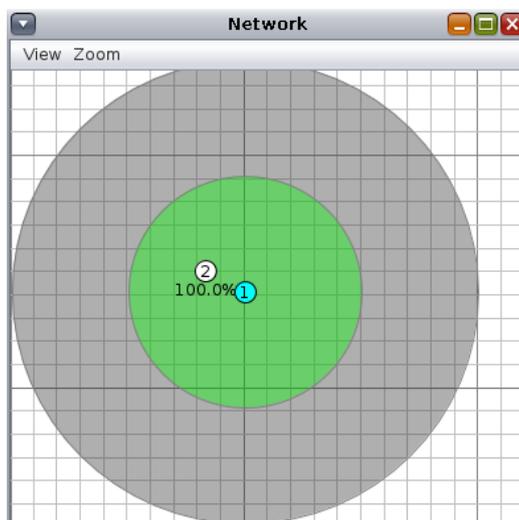


Figure 5.2: Scenario 1

First of all we create two multicast groups in the router via shell command. Once they are created -Figure 5.3-, we describe the flow diagram, step

by step.

```

Mote Interface Viewer (Z1 1)
Select interface: Serial port
fe80::c30c:0:0:1
193.12: Contiki>
> create_group 1 1
Creating mcast group
g= 1 , s= 1
193.12: Contiki>
> create_group 2 2
Creating mcast group
g= 2 , s= 2
193.12: Contiki>
> print_router_table
G = ff31::1, S = aaaa::c30c:0:0:1, state= 0
G = ff31::2, S = aaaa::c30c:0:0:2, state= 0
193.12: Contiki>
Send data

```

Figure 5.3: Router shell with 2 multicast groups on its records

The flow sequence continues in Figure Figure 5.4. We wait until the first General Query message from the router (1), the host will get it and it will not send any report back since it has no listener groups yet (2). We send via host shell, a `send_report 1 1 1` and the host will start a delay timer to send the next report(3). The router will get the report from the host and will notice that there are listeners for this multicast group `state = 1` (5). The multicast group timer delay in the host expires and it sends a report(6).

```

Mote Interface Viewer (Z1 1)
Select interface: Serial port
> print_router_table
G = ff31::1, S = aaaa::c30c:0:0:1, state= 0
G = ff31::2, S = aaaa::c30c:0:0:2, state= 0
193.12: Contiki>
MLD: GENERAL QUERY TO ff02::1
Received MLD report from
fe80::c30c:0:0:2 G = ff31::1
S = aaaa::c30c:0:0:1 P = aaaa::c30c:0:0:1
En la config de router, handle reports
Group found
Lanzo el timer de timer group con 80 sin haber listeners
> print_router_table
G = ff31::1, S = aaaa::c30c:0:0:1, state= 1
G = ff31::2, S = aaaa::c30c:0:0:2, state= 0
193.12: Contiki>
Send data

Mote Interface Viewer (Z1 2)
Select interface: Serial port
Aquí debo entrar si soy host
Received MLD general query from fe80::c30c:0:0:1
> print_host_table
Printing multicast host table
193.12: Contiki>
> send_report 1 1 1
Sending query request
MLD: MCAST REPORT TO
ff02::1 G = ff31::1
S = aaaa::c30c:0:0:1
Host group listener con delay 5
193.12: Contiki>
Previous timer host expired send report G= ff31::1
MLD: MCAST REPORT TO
ff02::1 G = ff31::1
S = aaaa::c30c:0:0:1
Send data

```

Figure 5.4: Scenario 1 Sequence 1 in router shell and host shell

Now the sequence continues in Figure 5.5, the listener group in the host for that multicast group will be in idle state (`state=1`) and `mld_flag = 1`, until it receives a query (7). We send an unsolicited done via shell, `send_done 1 1 1`. No more reports to this multicast group, and we stop the delay timer for

that multicast group(8). The router get the MLD done message and goes to CHECKING LISTENERS state (9). The router sends two MLD multicast address specific queries (10). The host gets the queries (11) and it does not reply since it has already erased that address while sending the MLD done. And finally the router realizes that there is no more listeners when the timer expires and it can be shown with (12) ($state=0$).

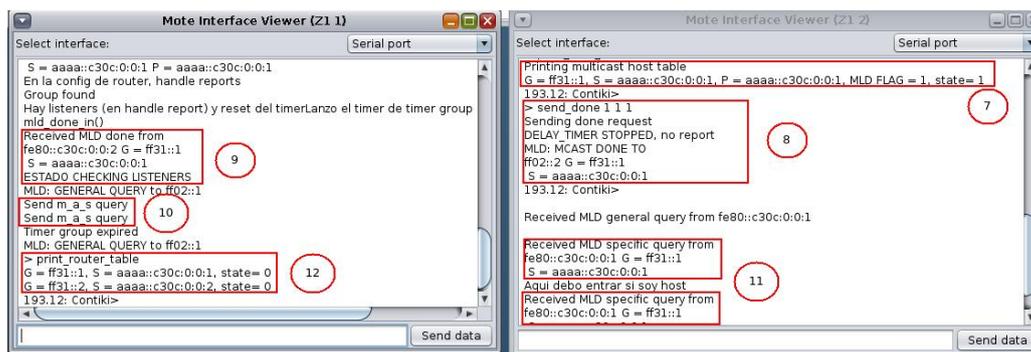


Figure 5.5: Scenario 1 Sequence 2 in router shell and host shell

With this simulation, we have tested the vast majority of the possible reactions and behaviors of the nodes. We have checked the correct general query operation, to join and leave a group, the 3 possible states of a router, the 2 possible states of a host, and the functions implemented to work with the data structures. But still, we have to test the interaction withing two hosts and a router. Therefore, this one is the next scenario.

5.2.2 Scenario 2: 1 router and 2 hosts

In this scenario, we are locating one router (node 1) and two hosts (node 2 and node 3) as it can be shown in Figure 5.6. We will check the interaction within 2 hosts when they join and leave a group.

Our given scenario will have a multicast group created (1,1,1) in the router and the node 2 has already join that group. The sequence is in the Figure 5.7. In the figure, it does not appear the router console since it is not that relevant. We show that the listener group is **DELAYING**, $state=0$ and $mld_flag=1$ since it is the only one listening the multicast group (1,1,1) (1). The node 3 receives the **send_report 1 1 1** from the node 2 and it does not do anything since it does not have that multicast group on its listener table(2). We command via shell in the node 3, **send_report 1 1 1** (3). And the router and the other host will get the message. They will do different processes, the router will restart its timer and the host will stop its timer,

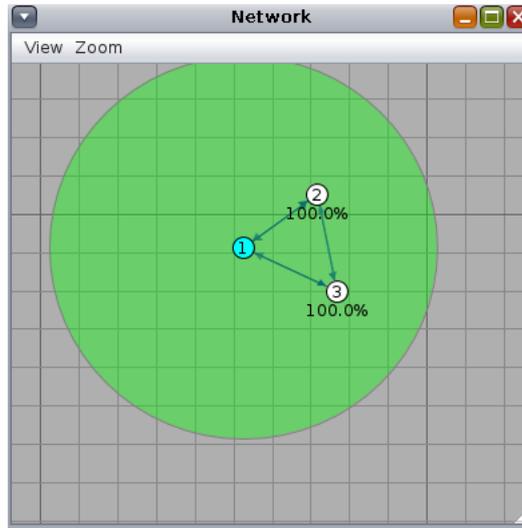


Figure 5.6: Scenario 2

go to IDLE state, $state=0$ and set its $ml_d_flag=0$, noticing that it is not the only one listening that multicast group(4).

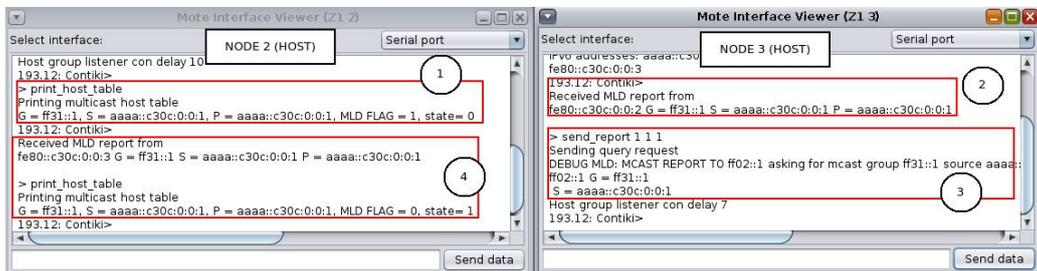


Figure 5.7: Scenario 2 Sequence 1 two hosts

The Figure 5.8 shows how follows the simulation. The sequence continue with an unsolicited `send_done 1 1 1` from the node 3 (1). The router get the MLD done message (2) and start in the `CHECKING LISTENERS` state, sending specific queries (3). The node 2 get the query (4) and send a report to the multicast group got it in the query (5). The router get the MLD report (6) and continue the normal performing, knowing that there are still listeners.

With this simulation, we have tested the interaction within hosts. How a host, get a report and check if it is for a multicast group that it has already to pass to idle. Therefore, it does not have to send a report. And, we have evaluated also how a router after getting a done, it checks if there are still listeners.

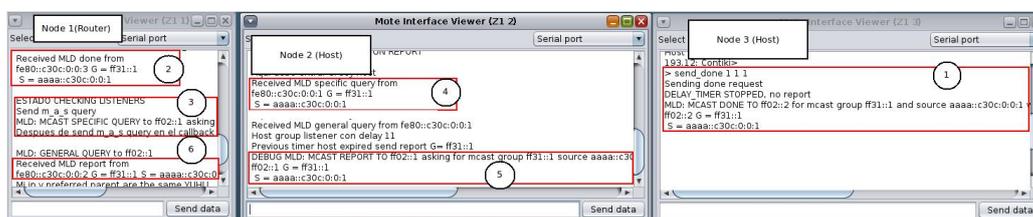


Figure 5.8: Scenario 2 Sequence 1 2 two hosts

5.2.3 Scenario 3: 2 routers and 1 host

With the both two earlier scenarios, we have tested most of the possible MLD protocol behaviors. But, we still need to test the *preferred_parent* selection. Hence, we have built and scenario with two router and one host. The scenario is seen in the Figure 5.9. Node 1 and 2 are routers, and node 3 is a host.



Figure 5.9: Scenario 3

In this sequence -Figure 5.10-, we command `create_group 1 1` in both routers(1). Hence, both have the same (G,S). The host `send_report 1 1 1` wanting to join the (G,S) = (1,1) but only expecting to receive packets from the node 1 (2). Both routers receive the MLD report from node 3, since MLD reports are sent to *all_link_local_nodes*(3)(4). We can see how the node 1 has written down the existence of listeners (*state=1*)(5). But node 2 no, *state=0* because in the MLD report the *preferred_parent* was 1.

The next sequence -Figure 5.11-. We evaluate same problem, the *preferred_parent* addressing, but with MLD done. We command `send_done 1`

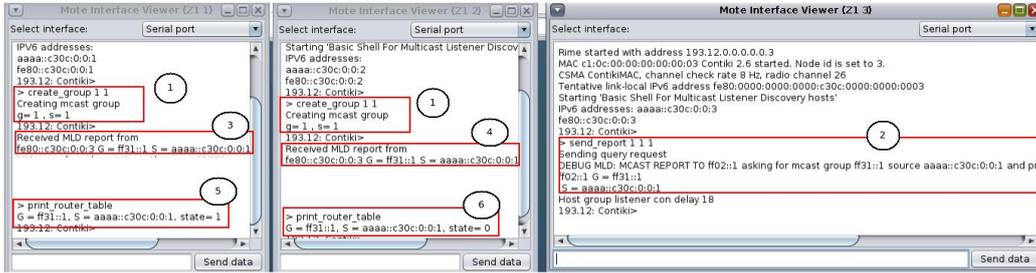


Figure 5.10: Scenario 3 Sequence 1

1 1(1). The router 1 receive it and check that it is for itself (2). The router 2 get it but notices that it has another preferred_parent address than its address(3). Finally, router 1 starts the process to handle the MLD done message, checking listeners(4).

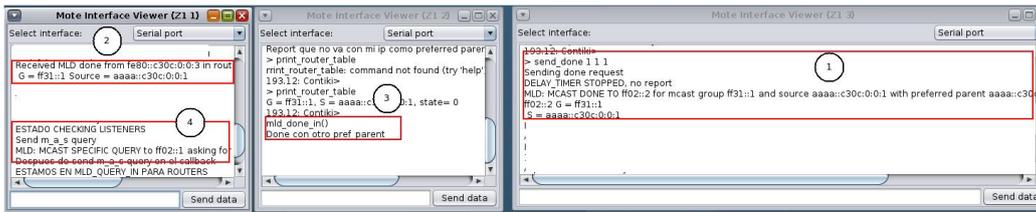


Figure 5.11: Scenario 3 Sequence 2

In this way, with these three scenarios we have tested and evaluated all the behaviors of the MLD protocol.

Since we are working with constrained devices to check if the compiled object fits in the device memory is a key point. There is a tool to know the object's size. The router and the host take up 4kb. Hence, they perfectly fit in our Z1 device which has more than 9kb on its memory.

Finally, we can state that our protocol works as we design it and it achieves the space goal on constrained devices.

Chapter 6

Discussion

The context of this master thesis was limited to the design and implementation of MLD suitable for constrained devices. However, several aspects should be studied further or improved in the future work in this field. We might want to consider to add the features that the MLDv2 protocol offer. The SSM filtering which adds some complexity to the message format and all the problems that it produces in constrained devices. We could investigate further to have reliability and avoid the effects of on-link forgery of MLD packets. The MLDv2 protocol does not provide protection from on-link attacks which can cause incorrect multicast records or disruption to packet delivery. The authors of [27] discuss about the trust models for MLD protocols. They also provide security and threat analysis for each model. Hence, we could implement some of these ideas about trustness and security.

Chapter 7

Conclusions

Multicasting is often viewed as the most promising communication mechanism for the deployment of multi-user, real-time and multimedia-rich Internet applications. Multicast hosts and routers, in IPv6 based networks, communicate and exchange their capabilities, requirements and manage group memberships using Multicast Listener Discovery protocol.

This final project has investigated the design and implementation of a MLD protocol suitable for constrained devices. We started from the simplest idea of MLDv1 and exploiting source-filtering. We avoided the complexity of the MLDv2 protocol. We have designed a protocol which fits in the constraint resources of the smart objects. The designed protocol is simple and enough to manage multicast groups.

The implementation has been developed for the Contiki OS, an open source operating system for the Internet of Things. Contiki connects tiny, low-cost, battery-operated and low-power systems to the Internet. The code has been developed in C language using the Contiki's libraries.

The evaluation has been done in Cooja in different scenarios to check the normal behavior and the corner cases of the protocol.

The idea behind designing and implementing a MLD protocol for constrained devices is to make it work with a multicast routing protocol suitable for constrained devices. Hence, the goal was to establish a multicast protocol for devices with constraint resources.

In this final project, we have accomplished our main goal. We have successfully developed a MLD protocol for constrained devices. The protocol works as we defined in the design and fits in the device's memory.

Bibliography

- [1] Home page auto id labs website. <http://auto-idlabs.autoidlabs.org/page.html>. Accessed: 2013-07-05.
- [2] Z-wave. web page. www.z-wave.com. Accessed: 2013-07-08.
- [3] Zigbee alliance. zigbee. web page. <http://www.zigbee.org>. Accessed: 2013-07-05.
- [4] ALLIANCE, Z. Zigbee specification. *Document 053474r06, Version 1* (2006).
- [5] ARAMPATZIS, T., LYGEROS, J., AND MANESIS, S. A survey of applications of wireless sensors and wireless sensor networks. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation* (2005), IEEE, pp. 719–724.
- [6] ASHTON, K. That internet of things thing. *RFiD Journal* 22 (2009), 97–114.
- [7] ATZORI, L., IERA, A., MORABITO, AND GIACOMO. The internet of things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.
- [8] BALLARDIE, A. Core based trees (cbt) multicast routing architecture.
- [9] CAIN, B., DEERING, S., KOUVELAS, I., FENNER, B., AND THYAGARAJAN, A. Internet group management protocol, version 3.
- [10] CALLAWAY, E., GORDAY, P., HESTER, L., GUTIERREZ, J. A., NAEVE, M., HEILE, B., AND BAHL, V. Home networking with ieee 802.15. 4: a developing standard for low-rate wireless personal area networks. *Communications Magazine, IEEE* 40, 8 (2002), 70–77.

- [11] CHUN, W., AND TANG, W. Multicasting in wireless sensor networks. In *Optical Internet and Next Generation Network, 2006. COIN-NGNCON 2006. The Joint International Conference on* (2006), IEEE, pp. 251–253.
- [12] CROW, B. P., WIDJAJA, I., KIM, L., AND SAKAI, P. T. Ieee 802.11 wireless local area networks. *Communications Magazine, IEEE* 35, 9 (1997), 116–126.
- [13] CULLER, D., AND CHAKRABARTI, B. S. 6lowpan: Incorporating ieee 802.15. 4 into the ip architecture.
- [14] DEERING, S., FENNER, W., AND HABERMAN, B. Multicast listener discovery (mld) for ipv6. Tech. rep., RFC 2710, October, 1999.
- [15] DEVADIGA, K. Development of a multicast routing protocol on contiki os. Master’s thesis, Aalto University, School of Science, 2013.
- [16] DUNKELS, A. The contikimac radio duty cycling protocol.
- [17] DUNKELS, A., GRONVALL, B., AND VOIGT, T. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on* (2004), IEEE, pp. 455–462.
- [18] DUNKELS, A., SCHMIDT, O., AND VOIGT, T. Using protothreads for sensor node programming. In *Proceedings of the REALWSN* (2005), vol. 5.
- [19] DURVY, M., ABEILLÉ, J., WETTERWALD, P., O’FLYNN, C., LEVERETT, B., GNOSKE, E., VIDALES, M., MULLIGAN, G., TSIFTES, N., FINNE, N., ET AL. Making sensor networks ipv6 ready. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (2008), ACM, pp. 421–422.
- [20] FENNER, B., HANDLEY, M., HOLBROOK, H., AND KOUVELAS, I. Rfc 4601: Protocol independent multicast-sparse mode (pim-sm): Protocol specification (revised).
- [21] FENNER, W. Rfc 2236 internet group management protocol. *IETF Document* (1997), 1.
- [22] GIUSTO, IERA, M., AND ATZORI. *The Internet of Things*. Springer, 2010.

- [23] GROUP, I. . W., ET AL. Standard for part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (lr-wpans). *ANSI/IEEE 802.15.4* (2003).
- [24] HAARTSEN, J. C. The bluetooth radio system. *Personal Communications, IEEE* 7, 1 (2000), 28–36.
- [25] HOLBROOK, H., AND CAIN, B. Rfc 4607: Source-specific multicast for ip. *Internet Engineering Task Force* (2006).
- [26] HUI, J., AND KELSEY, R. Multicast protocol for low power and lossy networks (mpl).
- [27] KURUP, G., DALEY, G., AND SEKERCIOGLU, Y. A. Trust models and security considerations in multicast listener discovery protocol version 2 (mldv2). In *TENCON 2005 2005 IEEE Region 10* (2005), IEEE, pp. 1–8.
- [28] LEE, K., AND SONG, E. Wireless sensor network based on ieee 1451.0 and ieee 1451.5-802.11. In *Electronic Measurement and Instruments, 2007. ICEMI'07. 8th International Conference on* (2007), IEEE, pp. 4–7.
- [29] LEWIS, F. L. Wireless sensor networks. *Smart environments: technologies, protocols, and applications* (2004), 11–46.
- [30] LI, Q., JINMEI, T., AND SHIMA, K. *IPv6 advanced protocols implementation*. The Morgan Kaufmann series in networking. Elsevier, Amsterdam, Boston, 2007.
- [31] MARCHIORI, A., AND HAN, Q. Pim-wsn: Efficient multicast for ipv6 wireless sensor networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a* (2011), IEEE, pp. 1–6.
- [32] MAYER, K., AND FRITSCH, W. Ip-enabled wireless sensor networks and their integration into the internet. In *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks* (New York, NY, USA, 2006), InterSense '06, ACM.
- [33] MONTENEGRO, G., KUSHALNAGAR, N., AND CULLER, D. Rfc 4944. transmission of ipv6 packets over ieee 802.15.4 networks.online-september 2007.

- [34] MOY, J. Rfc 1584–multicast extensions to ospf. *SRI Network Information Center* (1994).
- [35] MULLIGAN, G. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors* (2007), ACM, pp. 78–82.
- [36] OIKONOMOU, G., AND PHILLIPS, I. Stateless multicast forwarding with rpl in 6lowpan sensor networks. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on* (2012), IEEE, pp. 272–277.
- [37] ÖSTERLIND, F. A sensor network simulator for the contiki os. *SICS Research Report* (2006).
- [38] ÖSTERLIND, F., DUNKELS, A., ERIKSSON, J., FINNE, N., AND VOIGT, T. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on* (2006), IEEE, pp. 641–648.
- [39] SHELBY, Z., HARTZE, K., AND BORMANN, C. Constrained application protocol (coap) draft-ietf-core-coap-18. online-june 2013.
- [40] TAN, L., AND WANG, N. Future internet: The internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on* (2010), vol. 5, IEEE, pp. V5–376.
- [41] TOMA, I., SIMPERL, E., AND HENCH, G. A joint roadmap for semantic technologies and the internet of things. In *Proceedings of the 3rd STI Roadmapping Workshop* (2009).
- [42] TOWNSEND, C., AND ARMS, S. Wireless sensor networks. *Principles and Applications, Sensor Technology Handbook Newness Publication* (2005).
- [43] VASSEUR, J.-P., AND DUNKELS, A. *Interconnecting smart objects with ip: The next internet*. Morgan Kaufmann, 2010.
- [44] VIDA, R., COSTA, L., FDIDA, S., DEERING, S., FENNER, B., KOUVELAS, I., AND HABERMAN, B. Multicast listener discovery version 2 (mldv2) for ipv6. Tech. rep., RFC 3810, June, 2004.
- [45] WAITZMAN, D., DEERING, S., AND PARTRIDGE, C. Distance vector multicast routing protocol.

- [46] WANT, R. An introduction to rfid technology. *Pervasive Computing, IEEE* 5, 1 (2006), 25–33.
- [47] WINTER, T. Rpl: Ipv6 routing protocol for low-power and lossy networks.

Appendix A

First appendix. MLD library code

```
/**
 * \file
 *      Multicast Listener Discovery protocol library
 *
 * \author Daniel Bailo <dbailo1988@gmail.com>
 * \date 20/08/2013
 *
 */
#ifdef __MLD6_H__
#define __MLD6_H__

#include "net/uip.h"
#include "sys/clock.h"

#ifdef MAX_NUM_OF_MCAST6_GROUPS
#define MAX_NUM_OF_MCAST6_GROUPS 2
#endif

#ifdef MAX_NUM_OF_MCAST_LISTENING
#define MAX_NUM_OF_MCAST_LISTENING 4
#endif

/*MLD constants*/

#define ROBUSTNESS_VARIABLE 2
#define QUERY_INTERVAL 125 /*seconds within periodical general queries*/
#define QUERY_RESPONSE_INTERVAL 10
#define MCAST_LISTENER_INTERVAL 260
```

```

/*(ROBUSTNESS_VARIABLE x QUERY_INTERVAL)+ QUERY_RESPONSE_INTERVAL*/
/* time to decide there is no more listeners for an address*/
#define START_QI 32 /* (QUERY_INTERVAL / 4) */
#define START_COUNT ROBUSTNESS_VARIABLE
#define LAST_LISTENER_QI 5
#define COUNT_LAST_LISTENER ROBUSTNESS_VARIABLE
#define UNSOLICITED REPORT INTERVAL 10

/* host States*/

#define DELAYING_LISTENER 0
#define IDLE_LISTENER 1

/* Router States*/

#define NO_LISTENERS 0
#define LISTENERS 1
#define CHECKING_LISTENERS 2

/* Struct for hosts*/

struct mld6_listen_group {
    int used;
    uint8_t mld_flag; /* Whether there is more listeners than you*/
    uint8_t state;
    uip_ipaddr_t mcast_group;
    uip_ipaddr_t source; /* Source-filthering*/
    uip_ipaddr_t preferred_parent;
    struct ctimer delay_timer; /* [0, Max.Respons.Delay] */
};

typedef struct mld6_listen_group mld6_listen_group_t;

/*
 * List or array of mld6_groups, as structs as number
 * of mcast groups listening
 */

struct mcast_group {
    int used;
    uip_ipaddr_t mcast_group;

```

```
    uip_ipaddr_t source;
    uint8_t state;
    struct ctimer timer;
    struct ctimer rtx_timer;
};

typedef struct mcast_group mcast_group_t;

/*
 * Functions to send MLD packets
 */

void mld_gen_query();
void mld_mas_query(uip_ipaddr_t *group_addr, uip_ipaddr_t *source_addr);
void mld_report(uip_ipaddr_t *group_addr, uip_ipaddr_t *source_addr,
               uip_ipaddr_t *preferred_parent);
void mld_done(uip_ipaddr_t *group_addr, uip_ipaddr_t *source_addr,
              uip_ipaddr_t *preferred_parent);

/*
 * Functions to work with Multicast groups in routers
 */

#ifdef UIP_CONF_ROUTER
mcast_group_t *mcast_group_new(uip_ipaddr_t *group_addr,
                               uip_ipaddr_t *source_addr);
mcast_group_t *mcast_group_find (uip_ipaddr_t *group_addr,
                                 uip_ipaddr_t *source_addr);
void print_mld_router_table();
#endif

/*
 * Functions to work with listeners_group in hosts
 */

#ifdef RPL_CONF_LEAF_ONLY
mld6_listen_group_t *new_listener(uip_ipaddr_t *group_addr,
                                  uip_ipaddr_t *source, uip_ipaddr_t *preferred_parent);
mld6_listen_group_t *find_listener (uip_ipaddr_t *group_addr,
                                    uip_ipaddr_t *source);
int erase_listener (uip_ipaddr_t *group_addr, uip_ipaddr_t *source,
                  uip_ipaddr_t *preferred_parent);
void print_mld_host_table();
```

```
#endif

/*
 *      Handlers for MLD messages
 */

void mld_query_in();
void mld_report_in();
void mld_done_in();

/*
 *      Callbacks
 */

static void timer_host_expired (void *listen_group);
static void rtxtimer_expired (void *mcast_group);
static void timer_group_expired (void *mcast_group);

#endif /* __MLD6_H__ */

/**
 * \file
 *      Multicast Listener Discovery protocol library
 *
 * \author Daniel Bailo <dbailo1988@gmail.com>
 * \date 25/08/2013
 *
 */

#include "net/uip.h"
#include "contiki-net.h"
#include "net/uip-ds6.h"
#include "net/uip-icmp6.h"
#include "lib/random.h"
#include "mld6.h"
#include "sys/clock.h"
#include "sys/ctimer.h"

#define DEBUG 0
```

```
#include "net/uip-debug.h"

#define UIP_IP_BUF      ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])
#define UIP_ICMP_BUF    ((struct uip_icmp_hdr *)&uip_buf[uip_l2_l3_hdr_len])
#define UIP_ICMP_PAYLOAD ((unsigned char *)&uip_buf[uip_l2_l3_icmp_hdr_len])

#ifdef UIP_CONF_ROUTER
    static mcast_group_t mcast_group_table[MAX_NUM_OF_MCAST6_GROUPS];
#endif

#ifdef RPL_CONF_LEAF_ONLY
    static mld6_listen_group_t listening_table[MAX_NUM_OF_MCAST_LISTENING];
#endif

/*
 *   return A pseudo-random number between 0 and MRD.
 */

unsigned short
random_time (uint16_t MRD)
{
    random_init(clock_time());
    return (random_rand() % (MRD)) + 2;
}

/**
 *   send a general query in MLD to all link local nodes
 */

void
mld_gen_query()
{
    uip_ipaddr_t dest_addr;

    uip_create_linklocal_allnodes_mcast(&dest_addr);

    PRINTF("MLD: GENERAL QUERY to ");
    PRINT6ADDR(&dest_addr);
    PRINTF("\n");
}
```

```
        uip_icmp6_send(&dest_addr, ICMP6_MLD_QUERY, 0, 0);
    }

    /**
     *   send a multicast address specific query
     *   in MLD to all link local nodes asking for group_addr
     *   and source_addr
     */

    void
    mld_mas_query(uip_ipaddr_t *group_addr, uip_ipaddr_t *source_addr)
    {

        uip_ipaddr_t dest_addr;
        int pos;
        unsigned char *buffer;

        uip_create_linklocal_allnodes_mcast(&dest_addr);

        PRINTF("MLD: MCAST SPECIFIC QUERY to ");
        PRINT6ADDR(&dest_addr);
        PRINTF(" asking for mcast group ");
        PRINT6ADDR(group_addr);
        PRINTF(" and source ");
        PRINT6ADDR(source_addr);
        PRINTF("\n");

        buffer = UIP_ICMP_PAYLOAD;
        pos = 0;
        memcpy(buffer, group_addr, sizeof(uip_ipaddr_t));
        pos += sizeof(uip_ipaddr_t);
        memcpy(buffer + pos, source_addr, sizeof(uip_ipaddr_t));
        pos += sizeof(uip_ipaddr_t);

        uip_icmp6_send(&dest_addr, ICMP6_MLD_QUERY, 0, pos);
    }

    /**
```

```
*      send a report for listening the multicast group group_addr  
*      and source source_addr specifying the preferred_parent  
*/  
  
void  
mld_report(uiplib_addr_t *group_addr, uilib_addr_t *source_addr,  
           uilib_addr_t *preferred_parent)  
{  
    uilib_addr_t dest_addr;  
    int pos;  
    unsigned char *buffer;  
  
    uilib_create_linklocal_allnodes_mcast(&dest_addr);  
  
    PRINTF("MLD: MCAST REPORT TO ");  
    PRINT6ADDR(&dest_addr);  
    PRINTF(" G=");  
    PRINT6ADDR(group_addr);  
    PRINTF(" S=");  
    PRINT6ADDR(source_addr);  
    PRINTF(" P= ");  
    PRINT6ADDR(preferred_parent);  
    PRINTF("\n");  
  
    buffer = UIP_ICMP_PAYLOAD;  
    pos = 0;  
    memcpy(buffer, group_addr, sizeof(uiplib_addr_t));  
    pos += sizeof(uiplib_addr_t);  
    memcpy(buffer + pos, source_addr, sizeof(uiplib_addr_t));  
    pos += sizeof(uiplib_addr_t);  
    memcpy(buffer + pos, preferred_parent, sizeof(uiplib_addr_t));  
    pos += sizeof(uiplib_addr_t);  
  
    uilib_icmp6_send(&dest_addr, ICMP6_MLD_REPORT, 0, pos);  
  
}  
  
/**  
 *      send a done message to the multicast group group_addr
```

```

    *      and source source_addr specifying the preferred_parent
    */

void
mld_done(uiplib_t *group_addr, uip_ipaddr_t *source_addr,
         uip_ipaddr_t *preferred_parent)
{
    uip_ipaddr_t dest_addr;
    int pos;
    unsigned char *buffer;

    uip_create_linklocal_allrouters_mcast(&dest_addr);

    PRINTF("MLD: MCAST DONE TO ");
    PRINT6ADDR(&dest_addr);
    PRINTF(" G=");
    PRINT6ADDR(group_addr);
    PRINTF(" S=");
    PRINT6ADDR(source_addr);
    PRINTF(" P= ");
    PRINT6ADDR(preferred_parent);
    PRINTF("\n");

    buffer = UIP_ICMP_PAYLOAD;
    pos = 0;
    memcpy(buffer, group_addr, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);
    memcpy(buffer + pos, source_addr, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);
    memcpy(buffer + pos, preferred_parent, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);

    uip_icmp6_send(&dest_addr, ICMP6_MLD_DONE, 0, pos);
}

/*
 *      callback when timer in listener_group (host) expires
 */

static void

```

```
timer_host_expired (void *listen_group)
{
    mld6_listen_group_t *aux = listen_group;
    aux->mld_flag = 1; /*Set flag, we are the last host*/
    PRINTF("Timer host expired send report G=");
    PRINT6ADDR(&aux->mcast_group);
    PRINTF("\n");
    aux->state= IDLE_LISTENER;
    mld_report(&aux->mcast_group, &aux->source, &aux->preferred_parent);
}

/*
 *      callback function when rtx timer expires in router
*/

static void
rtxtimer_expired (void *mcast_group)
{
    mcast_group_t *aux = mcast_group;
    PRINTF("Send m_a_s query\n");
    mld_mas_query(&aux->mcast_group, &aux->source);
    ctimer_set(&aux->rtx_timer, LAST_LISTENER_QI * CLOCK_SECOND,
               rtxtimer_expired, aux);
}

/*
 *      callback function when timer* expires in router
*/

static void
timer_group_expired (void *mcast_group)
{
    mcast_group_t *aux = mcast_group;
    //NOTIFY ROUTING -
    PRINTF("Timer group expired \n");
    aux->state = NO_LISTENERS;
    ctimer_stop(&aux->rtx_timer);
}

/**
```

```
* ROUTER MANAGEMENT MULTICAST GROUPS FUNCTIONS
*/

/**
 *   Function to introduce a new mcast_group_t in the table
 *   If there are no available entries return NULL
 */
#ifdef UIP_CONF_ROUTER
mcast_group_t *
mcast_group_new(uiplib_ipaddr_t *group_addr, uip_ipaddr_t *source_addr)
{
    mcast_group_t *aux;
    int i;

    for(i=0; i < MAX_NUM_OF_MCAST6_GROUPS; i++)
    {
        aux= &mcast_group_table[i];
        if(aux->used == 0)
        {
            aux->used = 1;
            aux->mcast_group = *group_addr;
            aux->source= *source_addr;
            aux->state = NO_LISTENERS;
            /* timers will start when receive a report or done*/
            return aux;
        }
    }
    return NULL;
}

/**
 *   Function to find a multicast group in a multicast table in
 *   a router
 */
mcast_group_t *
mcast_group_find (uip_ipaddr_t *group_addr, uip_ipaddr_t *source_addr)
{
    mcast_group_t *aux;
    int i;
```

```

    for(i=0; i< MAX_NUM_OF_MCAST6_GROUPS; i++)
    {
        aux= &mcast_group_table[i];
        if(aux->used)
        {
            if((uip_ipaddr_cmp(&aux->mcast_group, group_addr)) &&
                (uip_ipaddr_cmp(&aux->source, source_addr)) )
            {
                PRINTF("Same G and S \n");
                return aux;
            }
        }
    }
    return NULL;
}

void
print_mld_router_table()
{
    mcast_group_t *aux;
    int i;

    for(i=0; i< MAX_NUM_OF_MCAST6_GROUPS; i++)
    {
        aux= &mcast_group_table[i];
        if(aux->used)
        {
            printf("G = ");
            uip_debug_ipaddr_print(&aux->mcast_group);
            printf(", S = ");
            uip_debug_ipaddr_print(&aux->source);
            printf(", state= %d\n", aux->state);
        }
    }
}
#endif
/*
 * Functions to work with listeners_group in hosts
 */
#ifdef RPL_CONF_LEAF_ONLY

```

```

mld6_listen_group_t *
new_listener(uiplib_t *group_addr, uip_ipaddr_t *source,
            uip_ipaddr_t *preferred_parent)
{
    mld6_listen_group_t *aux;
    int i;
    unsigned short mrd;

    for(i=0; i < MAX_NUM_OF_MCAST_LISTENING; i++)
    {
        aux= &listening_table[i];
        if(aux->used == 0)
        {
            memset(aux, 0, sizeof(mcast_group_t));
            aux->used = 1;
            aux->mld_flag = 1; /* We are the last host*/
            aux->mcast_group = *group_addr;
            aux->source = *source;
            aux->preferred_parent = *preferred_parent;
            mrd=random_time(QUERY_RESPONSE_INTERVAL);
            PRINTF("Host group listener with delay %d \n",mrd);
            ctimer_set(&aux->delay_timer, mrd * CLOCK_SECOND,
                    timer_host_expired , aux);

            return aux;
        }
    }
    return NULL;
}

mld6_listen_group_t *
find_listener (uip_ipaddr_t *group_addr, uip_ipaddr_t *source_addr)
{
    mld6_listen_group_t *aux;
    int i;

    for(i=0; i< MAX_NUM_OF_MCAST_LISTENING; i++)
    {
        aux= &listening_table[i];
        if(aux->used)
        {
            if(uiplib_cmp(&aux->mcast_group, group_addr) == 0)

```

```
        uip_ipaddr_cmp(&aux->source, source_addr))
    {
        return aux;
    }
}
return NULL;
}

int
erase_listener (uip_ipaddr_t *group_addr, uip_ipaddr_t *source,
               uip_ipaddr_t *preferred_parent)
{
    mld6_listen_group_t *aux;
    aux= find_listener(group_addr, source);
    if (aux !=NULL)
    {
        if(uip_ipaddr_cmp(&aux->preferred_parent, preferred_parent))
        {
            aux->used=0;
            return 1;
        }
    }
    return 0;
}

void
print_mld_host_table()
{
    mld6_listen_group_t *aux;
    int i;

    printf("Printing multicast host table\n");
    for(i=0; i< MAX_NUM_OF_MCAST_LISTENING; i++)
    {
        aux= &listening_table[i];
        if(aux->used)
        {
            printf("G = ");
            uip_debug_ipaddr_print(&aux->mcast_group);
            printf(", S = ");

```

```

        uip_debug_ipaddr_print(&aux->source);
        printf(", P = ");
        uip_debug_ipaddr_print(&aux->preferred_parent);
        printf(", MLD FLAG = %d, state= %d\n", aux->mld_flag, aux->state);
    }
}
}

#endif

void
mld_query_in()
{ // Receive query

    uip_ipaddr_t group_addr;
    uip_ipaddr_t source_addr;
    mld6_listen_group_t *aux;
    unsigned char *buffer;
    uint8_t buffer_length;
    int pos;
    unsigned short mrd;

    buffer = UIP_ICMP_PAYLOAD;
    buffer_length = uip_len - uip_l3_icmp_hdr_len;

    uip_len=0;

    #if UIP_CONF_ROUTER
        uip_len=0;
        PRINTF("HANDLING MLD_QUERY IN A ROUTER\n");
    #endif /*UIP_CONF_ROUTER*/

    #if RPL_CONF_LEAF_ONLY
        PRINTF("HANDLING MLD_QUERY IN A HOST\n");
        if (buffer_length != 0)
        { /* Mcast address specific query*/
            pos = 0;
            memcpy(&group_addr, buffer, sizeof(uip_ipaddr_t));
            pos += sizeof(uip_ipaddr_t);
            memcpy(&source_addr, buffer + pos, sizeof(uip_ipaddr_t));
            pos += sizeof(uip_ipaddr_t);
        }
    #endif
}

```

```

PRINTF("Received MLD query from \n");
PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
PRINTF(" G = ");
PRINT6ADDR(&group_addr);
PRINTF("\n");
PRINTF(" S = ");
PRINT6ADDR(&source_addr);
PRINTF("\n");

aux = find_listener(&group_addr, &source_addr);

if (aux==NULL){
    PRINTF("host is not listening requested group in m-a-s query\n");
} else
{
    if (uip_ipaddr_cmp(&aux->preferred_parent,
                      &UIP_IP_BUF->srcipaddr))
    {
        if (aux->state == IDLE_LISTENER)
        {
            aux->state = DELAYING_LISTENER;
            ctimer_set(&aux->delay_timer, QUERY_RESPONSE_INTERVAL,
                      timer_host_expired, aux);
        } else
        {
            if (aux->state == DELAYING_LISTENER)
            {
                ctimer_restart(&aux->delay_timer);
            }
        }
    }
}
} else
{ //General query

    PRINTF("Received MLD general query from ");
    PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
    PRINTF("\n");
    int i;
    for (i=0; i< MAX_NUM_OF_MCAST_LISTENING; i++)

```

```

    {
        aux = &listening_table[i];
        if (aux->used == 1)
        {
            aux->state = DELAYING_LISTENER;
            mrd=random_time(QUERY_RESPONSE_INTERVAL);
            PRINTF("Host group listener with delay %d \n",mrd);
            ctimer_set(&aux->delay_timer, mrd * CLOCK_SECOND,
                timer_host_expired , aux);
        }
    }
}

uip_len = 0;
#endif /*RPL_CONF_LEAF_ONLY*/
}

void
mld_report_in()
{
    uip_ipaddr_t group_addr, source, preferred_parent;
    mld6_listen_group_t *aux_listener;
    mcast_group_t *aux_mcast;
    unsigned char *buffer;
    uint8_t buffer_length;
    int pos;

    buffer = UIP_ICMP_PAYLOAD;
    buffer_length = uip_len - uip_l3_icmp_hdr_len;

    pos = 0;
    memcpy(&group_addr, buffer, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);
    memcpy(&source, buffer + pos, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);
    memcpy(&preferred_parent, buffer + pos, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);

    PRINTF("Received MLD report from ");
    PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
    PRINTF("in host \n");
    PRINTF(" G = ");

```

```

PRINT6ADDR(&group_addr);
PRINTF(" Source = ");
PRINT6ADDR(&source);
PRINTF("\n");

uip_len=0;

#if UIP_CONF_ROUTER
if(uip_ds6_is_my_addr(&preferred_parent))
{
    PRINTF("My router ip and the preferred parent are the same\n");

    aux_mcast = mcast_group_find(&group_addr, &source);
    if (aux_mcast == NULL)
    { // Requested Group is not on the list of mcast group
        PRINTF("Report discard, no mcast group in router\n");
    }
    else
    {
        PRINTF("Group found \n");
        switch(aux_mcast->state)
        {
            case NO_LISTENERS:
                // notify routing +
                aux_mcast->state = LISTENERS;
                PRINTF("Timer group delay %d, no listeners\n", MCAST_LISTENER_INTERVAL);
                ctimer_set(&aux_mcast->timer, MCAST_LISTENER_INTERVAL * CLOCK_SECOND,
                    timer_group_expired , &aux_mcast);

                break;
            case LISTENERS:
                PRINTF("Timer group delay %d, listeners\n", MCAST_LISTENER_INTERVAL);
                ctimer_set(&aux_mcast->timer, MCAST_LISTENER_INTERVAL * CLOCK_SECOND,
                    timer_group_expired , &aux_mcast);

                break;
            case CHECKING_LISTENERS:
                aux_mcast->state = LISTENERS;
                ctimer_set(&aux_mcast->timer, MCAST_LISTENER_INTERVAL * CLOCK_SECOND,
                    timer_group_expired , &aux_mcast);
                ctimer_stop(&aux_mcast->rtx_timer);
                break;
            default:

```

```

        PRINTF("Error in router's state getting report message\n");
        break;
    }
}
} else PRINTF("Report with preferred parent not my router ip\n");

#endif /*UIP_RPL_ROUTER*/

#if RPL_CONF_LEAF_ONLY
    PRINTF("REPORT in HOST \n");
    aux_listener = find_listener(&group_addr, &source);
    if (aux_listener->state == DELAYING_LISTENER)
    {
        if (aux_listener->used)
        {
            ctimer_stop(&aux_listener->delay_timer);
            //Clear flag we were not the last host
            aux_listener->mld_flag = 0;
            aux_listener->state = IDLE_LISTENER;
        }
    }
}

#endif /*RPL_CONF_LEAF_ONLY*/
    uip_len=0;
}

void mld_done_in()
{
    uip_ipaddr_t group_addr, source, preferred_parent;
    mcast_group_t *aux;
    unsigned char *buffer;
    uint8_t buffer_length;
    int pos;
    uip_len = 0;
    #if UIP_CONF_ROUTER
    buffer = UIP_ICMP_PAYLOAD;
    pos = 0;
    memcpy(&group_addr, buffer, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);
    memcpy(&source, buffer + pos, sizeof(uip_ipaddr_t));
    pos += sizeof(uip_ipaddr_t);

```

```

memcpy(&preferred_parent, buffer + pos, sizeof(uiplib_t));
pos += sizeof(uiplib_t);

if(uiplib_ds6_is_my_addr(&preferred_parent))
{

    PRINTF("Received MLD done from ");
    PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
    PRINTF(" in router \n");
    PRINTF(" G = ");
    PRINT6ADDR(&group_addr);
    PRINTF(" Source = ");
    PRINT6ADDR(&source);
    PRINTF("\n");

    aux= mcast_group_find(&group_addr, &source);
    if (aux == NULL) { // Requested Group is not in the list of mcast group
        PRINTF("Report discard, no this mcast group in router\n");
    }
    else
    {
        switch(aux->state)
        {
            case LISTENERS:
                aux->state = CHECKING_LISTENERS;
                PRINTF("STATE CHECKING LISTENERS \n");
                ctimer_set(&aux->timer, MCAST_LISTENER_INTERVAL * CLOCK_SECOND,
                    timer_group_expired, aux);
                ctimer_set(&aux->rtx_timer, LAST_LISTENER_QI * CLOCK_SECOND,
                    rtxtimer_expired, aux);

                break;
            default:
                PRINTF("Error in router's state receiving mld_done\n");
                break;
        }
    }
}
else printf("Done with other preferred_parent address\n");

#endif
uip_len = 0;

```

}