

Universidad de Alcalá Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación



Trabajo Fin de Grado

Implementación de algoritmos distribuidos en Smart Grids
mediante el uso de tecnología Raspberry

ESCUELA POLITECNICA
SUPERIOR

Autor: Paula Bartolomé Mora

Tutora: Elisa Rojas Sánchez

2022

UNIVERSIDAD DE ALCALÁ
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

**Implementación de algoritmos distribuidos en Smart Grids
mediante el uso de tecnología Raspberry**

Autora: Paula Bartolomé Mora

Tutora: Elisa Rojas Sánchez

Tribunal:

Presidente: Jaime José García Reinoso

Vocal 1º: Isaías Martínez Yelmo

Vocal 2º: José Manuel Arco Rodríguez

Resumen

Este Trabajo Final de Grado (TFG) se centra en el desarrollo de un algoritmo de distribución de cargas computacionales para redes de sensores de baja capacidad o dispositivos *Internet of Things* (IoT), denominado como Dedenne. Se hará uso como base del mismo el protocolo de encaminamiento IoTorii, ya implementado anteriormente.

El estudio del concepto de las redes inteligentes de electricidad y de los protocolos estandarizados para las redes de sensores (*Low Power and Lossy Network*, LLN) servirá de soporte para establecer el contexto y el ámbito de aplicación del algoritmo.

Se expondrán las herramientas empleadas llevar a cabo su implementación a nivel software y así, poder plantear las correspondientes simulaciones en diferentes escenarios y topologías de red. En particular, este TFG se enfocará en el empleo de Contiki con el objetivo de posibilitar el despliegue en una tarjeta Raspberry Pi. Finalmente, se justificarán los resultados del experimento y se evidenciará el funcionamiento y la eficiencia del algoritmo.

Palabras clave: Smart grids, IoT, LLN, RPL, Contiki-ng .

Abstract

This Bachelor's Degree Final Project is focused on the development of a computational load distribution algorithm for low-capacity sensor networks or *Internet of Things* (IoT) devices, called Dedenne. The IoTorii routing protocol, already implemented previously, will be used as its basis.

The study of the concept of smart grids and standardized protocols for Low Power and Lossy Networks (LLN) will serve as support to establish the environment and scope of the algorithm.

The tools used to carry out software implementation will be also presented. In this way, it will be possible to run simulations in different scenarios and network topologies. In particular, this project will focus on the use of Contiki in order to enable deployment on a Raspberry Pi card. Finally, the results of the experiment will be clearly justified and the functioning and efficiency of the algorithm will be evidenced.

Keywords: Smart grids, IoT, LLN, RPL, Contiki-ng , L^AT_EX, .

Índice general

Resumen	v
Abstract	vii
Índice general	ix
Índice de figuras	xiii
Índice de tablas	xvii
Índice de listados de código fuente	xix
Índice de algoritmos	xxi
1 Introducción	1
1.1 Presentación	1
1.2 Objetivo y plan de trabajo	2
1.3 Estructura del trabajo	3
2 Estado del arte	5
2.1 Smart grids	5
2.1.1 Seguridad en smart grids	7
2.1.2 Impacto medioambiental	8
2.1.3 La figura del prosumidor	9
2.1.3.1 Sistema de transacciones energéticas	10
2.2 Redes LLN	12
2.2.1 Clases de nodos en redes LLN	14
2.2.1.1 Clase 0	14
2.2.1.2 Clase 1	14
2.2.1.3 Clase 2	14
2.2.2 Estándares y protocolos en redes LLN	14
2.2.2.1 IEEE 802.15.4	15

2.2.2.2	6LoWPAN	16
2.2.2.3	RPL	17
2.2.2.3.1	Ruta <i>Upward</i>	18
2.2.2.3.2	Ruta <i>Downward</i>	18
2.2.2.3.3	Tipos de mensajes en RPL	18
2.2.3	Encaminamiento eficiente	19
2.2.3.1	Optimización del protocolo RPL	19
2.2.3.2	IoTorii	20
2.3	Simuladores de red y herramientas software	22
2.3.1	Contiki-ng	22
2.3.1.1	Simulador Cooja	22
2.3.1.2	Ejemplo Hello World	23
2.3.1.3	Implementación de IoTorii	24
2.3.2	BRITE	26
2.3.3	Mininet	27
2.4	Implementación hardware y tecnología Raspberry Pi	27
2.4.1	Raspberry Pi 4	27
3	Diseño y análisis	29
3.1	Código base	29
3.1.1	Archivos de configuración y compilación	30
3.2	Empleo de Contiki	31
3.3	Herramientas de despliegue de redes	32
4	Desarrollo	35
4.1	Introducción	35
4.2	Cronología del desarrollo	36
4.2.1	Preimplementación y estudio del software	36
4.2.2	Planteamiento del nuevo modelo	36
4.2.2.1	Asignación de una única etiqueta	36
4.2.2.2	Creación de la estructura de datos del nodo	38
4.2.2.3	Clasificación de nodos para el reparto de cargas	39
4.2.2.4	Identificación de nodos frontera	40
4.2.2.4.1	Método 1: identificación mediante la longitud de las etiquetas	41
4.2.2.4.2	Método 2: identificación mediante comparación de listas	41
4.2.2.4.3	Método 3: identificación mediante el uso de flags locales	42
4.3	Implementación completa por módulos	45
4.3.1	Inicialización de los nodos	45

4.3.1.1	Mensajes Hello	45
4.3.1.2	Mensajes HLMAC y proceso de asignación de etiquetas	46
4.3.1.3	Mensajes de notificación de carga	47
4.3.1.4	Mensajes de traspaso de carga y proceso de reparto	49
4.3.1.4.1	Caso 1: un nodo hijo con excedente de carga	51
4.3.1.4.2	Caso 2: un nodo hijo con escasez de carga	52
4.3.1.4.3	Caso 3: varios nodos hijos. Planteamiento del procesamiento de los paquetes recibidos	52
4.4	Tiempos de ejecución	55
4.4.1	Inicio del intercambio de mensajes Hello	55
4.4.2	Inicio del proceso de asignación de etiquetas	56
4.4.2.1	Nodo raíz	56
4.4.2.2	Nodos comunes	56
4.4.3	Visualización de las primeras estadísticas	57
4.4.4	Inicio de notificación de las cargas	57
4.4.5	Inicio del proceso de reparto de cargas	58
5	Evaluación de resultados	61
5.1	Introducción	61
5.2	Selección y despliegue de la topología	61
5.3	Resultados de la simulación	63
6	Conclusiones y líneas futuras	69
6.1	Conclusiones	69
6.2	Líneas futuras	70
6.2.1	Mejoras en la programación del algoritmo	70
6.2.1.1	Cambio de la versión base de IoTorii	70
6.2.1.2	Mejora de la asignación de etiquetas para nodos remotos	71
6.2.1.3	Conversión de paquetes multicast en unicast	72
6.2.2	Posibles implementaciones futuras	72
Bibliografía		73
Apéndice A Manual de usuario		77
A.1	Introducción	77
A.2	Instalación del entorno de simulación	77
A.2.1	Virtualización del sistema operativo Linux	77
A.2.2	Instalación y configuración de Contiki	78
A.2.2.1	Compilación en Cooja	79

A.3 Configuración para la implementación hardware	80
A.3.1 Instalación del sistema operativo para Raspberry Pi	80
A.3.2 Establecimiento del entorno hardware	80

Índice de figuras

1.1	Esquema característico de una red inteligente de energía [1]	2
2.1	Diferencias entre el modelo de red convencional (a) y el de red inteligente (b) [2]	5
2.2	Esquema de sistema distribuido en una smart grid [3]	6
2.3	Ejemplo de una microrred inteligente [4]	9
2.4	Esquema del encadenado de bloques en la tecnología blockchain [5]	10
2.5	Secuencia de acciones en una transacción de venta [6]	11
2.6	Esquema de actualización de la cadena en la tecnología blockchain [7]	12
2.7	Estructura de una red LLN [8]	13
2.8	Ejemplo de red con tecnología PLC [9]	13
2.9	Pila de protocolos en redes LLN [10]	14
2.10	Diagrama de flujo de CSMA [11]	15
2.11	Estructura de una red LLN [12]	16
2.12	Ejemplo de arquitectura RPL [13]	17
2.13	Representación de los mensajes de información en RPL [14]	18
2.14	Diferencias de enrutamiento entre los algoritmos ZTR y STR [15]	20
2.15	Asignación de etiquetas en el algoritmo IoTorii (I) [16]	21
2.16	Asignación de etiquetas en el algoritmo IoTorii (II) [16]	21
2.17	Inicio de la herramienta Cooja	23
2.18	Ejecución en modo nativo	23
2.19	Ventana de salida en Cooja	24
2.20	Topología de red empleada	24
2.21	Inicialización del nodo con ID 5	25
2.22	Número de mensajes Hello enviados en cada nodo	25
2.23	Procedimiento de asignación de direcciones HLMAC	25
2.24	Estadísticas de los nodos	26
2.25	Esquemático de Raspberry Pi 4 [17]	28
3.1	Esquema de creación de varios namespaces con Docker [18]	31

3.2 Ejemplo de topología diseñada en Mininet	33
4.1 Ejemplo 1 de topología de red (a)	37
4.2 Procedimiento de asignación de etiquetas a los nodos	38
4.3 Ejemplo 1 de topología de red (b)	40
4.4 Ejemplo 2 de topología de red	41
4.5 Idea de clasificación de nodos mediante dos listas [19]	42
4.6 Mensajes de notificación de nodos frontera de la red	44
4.7 Mensajes de notificación de nodos no frontera de la red	45
4.8 Mensajes de notificación del envío de carga	48
4.9 Envío y recepción de la carga de un nodo	48
4.10 Actualización de la lista de vecinos con la carga informada	49
4.11 Ejemplo de cálculo de carga para el caso 1	51
4.12 Ejemplo de cálculo de carga para el caso 2	52
4.13 Cálculo de la carga de un nodo a partir de los traspasos de varios hijos	53
4.14 Línea temporal de la secuencia de procesos en el algoritmo para la topología	60
5.1 Diseño de la topología de red empleada	61
5.2 Despliegue de la topología 5.1 en Mininet-WiFi	62
5.3 Despliegue de la topología 5.1 en Cooja	62
5.4 Árbol de red resultante del proceso de asignación de etiquetas	63
5.5 Información de los nodos tras la inicialización y asignación de etiquetas	64
5.6 Visualización de los nodos de la red clasificados como nodos frontera	64
5.7 Envío y recepción de las notificaciones de carga	65
5.8 Procedimiento de traspaso en el nodo ID:9	66
5.9 Procedimiento de traspaso en el nodo ID:12	66
5.10 Actualización de los flujos de carga de los hijos en la lista del nodo ID:16	66
5.11 Actualización del flujo de carga del padre en la lista del nodo ID:16	66
5.12 Situación final de cargas en los nodos de la red	67
6.1 Ejemplo de topología con una alta densidad de nodos	70
6.2 Resultado del proceso de identificación de nodos frontera	71
6.3 Ejemplo de topología en la que uno de los nodos es remoto	71
6.4 Visualización del número de etiquetas asignadas a cada nodo	72
A.1 Especificaciones del sistema operativo y del equipo	77
A.2 Características de la configuración de Contiki	79
A.3 Error de compilación en Cooja	79
A.4 Instalación y escritura de Raspberry Pi OS	80

A.5 Mensaje de finalización de la configuración [20]	81
A.6 Esquema de conexiones en la tarjeta Raspberry Pi [20]	81

Índice de tablas

2.1 Ejemplos de posibles ataques a smart grids	7
2.2 Tablas de direcciones HLMAC almacenadas en los nodos	26
4.1 Estructura del mensaje HLMAC (payload)	46

Índice de listados de código fuente

3.1	Introducción en el makefile de los archivos de código para su compilación	30
4.1	Definición de la estructura del nodo <code>this_node</code>	38
4.2	Definición de la estructura de vecinos <code>neighbour_table_entry</code>	43
4.3	Inicio del proceso de reparto de cargas en cada nodo	49
4.4	Preparación del buffer para el envío de paquetes	50
4.5	Procesamiento de los mensajes de traspaso	53
4.6	Retardos en la función de inicialización	56
4.7	Retardo de envío de mensajes HLMAC para los nodos comunes	57
4.8	Retardo de envío de mensajes de notificación de carga	58

Índice de algoritmos

4.1	Procesamiento de mensajes HLMAC	47
4.2	Diferenciación del procesamiento para cada tipo de paquete	50
4.3	Procedimiento de envío de los mensajes de traspaso	51
4.4	Retardos definidos para el proceso de notificación de cargas	58

Capítulo 1

Introducción

En el presente capítulo se pretende explicar de forma resumida los conceptos más relevantes del trabajo desarrollado. A modo de una breve introducción, se definirá el papel de las smart grids en el ámbito de las redes de electricidad y se expondrá el algoritmo que se desea implementar como fin principal del proyecto.

En un siguiente apartado, se hará hincapié en los objetivos que se marcan previamente al desarrollo del TFG. Su establecimiento elaborará la secuencia de acciones que deben llevarse a cabo para orientar la programación del algoritmo hacia el cumplimiento de los mismos. Este se justificará en el Capítulo 6, dedicado a las conclusiones finales.

Por último, se indicará la estructura de capítulos que presenta este documento, con una explicación general del contenido que se tratará en cada uno de ellos.

1.1 Presentación

El creciente auge del uso de las *smart grids* o redes eléctricas inteligentes ha supuesto la transformación del modelo energético centralizado en uno distribuido y mucho más eficiente. En otros términos, se puede decir que las nuevas redes inteligentes de electricidad consiguen descentralizar la producción de energía y conducir a una optimización del consumo [21].

Para conseguir este objetivo, se basan en algoritmos de monitorización, los cuales realizan un estudio continuo de las acciones de los usuarios de la red. Además, llevan un control de los dispositivos que están conectados y que participan en el proceso energético. Con todo esto, consiguen crear patrones de comportamiento que conllevan una mejor gestión de la distribución y del consumo energético [6].

En este contexto cabe destacar la figura del prosumidor [22]. Se puede definir como el usuario que toma el papel de consumidor o productor de energía según las necesidades que se presentan en cada instante en la red inteligente. Suponen una de las claves principal para ofrecer un servicio energético eficiente en las smart grids.

Haciendo referencia al hardware en el que se basan este tipo de redes, es preciso enfocarse en los dispositivos IoT [23]. El Internet de las Cosas (del inglés *Internet of Things*, IoT) constituye la base de las comunicaciones e interconexiones entre los equipos que participan en el proceso de distribución energético.

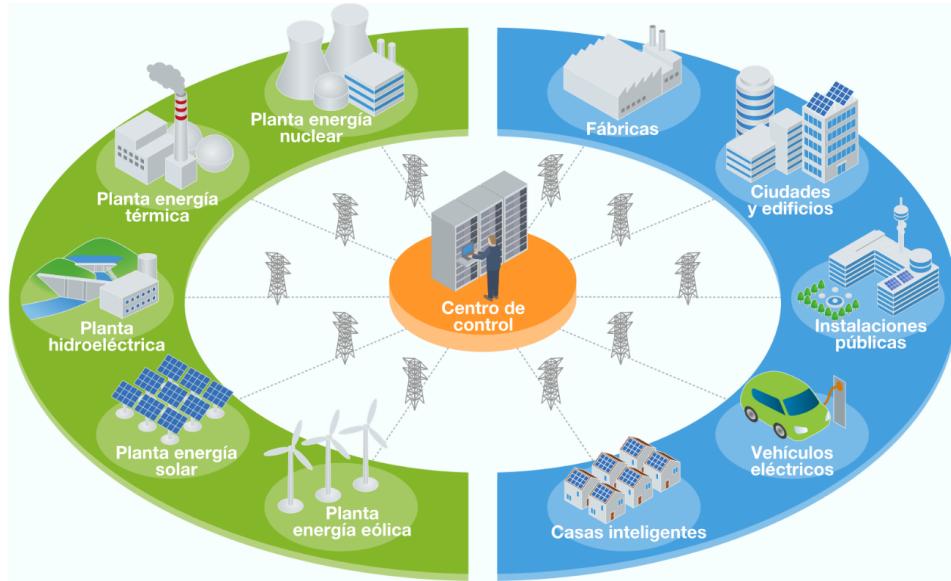


Figura 1.1: Esquema característico de una red inteligente de energía [1]

De forma general, actúan como sensores de baja capacidad y determinarán en cada instante la situación en la que se puede encontrar un usuario, un sector de la red o un equipo determinado. Es por ello que, en este trabajo, se planteará un protocolo de encaminamiento y de distribución aplicable a dispositivos IoT en el ámbito de las redes inteligentes de energía.

El algoritmo en sí, denominado como Dedenne por el equipo de investigación NetIS de la Universidad de Alcalá, pretende establecer un proceso de reparto de cargas computacionales eficiente entre el conjunto de sensores que existen en una topología de red. Se pretende utilizar como base del mismo el código de IoTorii¹ [24], algoritmo ya existente y desarrollado como mejora frente a las desventajas que presenta RPL [16][25], protocolo de encaminamiento para redes de baja capacidad basadas en dispositivos IoT. Es decir, el procedimiento de implementación de Dedenne se fundamentará en la transformación del modelo centralizado de IoTorii en uno nuevo distribuido.

1.2 Objetivo y plan de trabajo

A partir de la presentación anterior, se puede expresar que el objetivo principal de este trabajo es el desarrollo de un algoritmo distribuido para el reparto de cargas computacionales en sensores de baja capacidad o dispositivos IoT.

¹<https://github.com/NETSERV-UAH/IoTorii>

Para alcanzar este fin, será necesario establecer la secuencia de pasos a tomar mediante el siguiente plan de trabajo:

- **Documentación previa:** Se realiza una investigación y una búsqueda amplia de artículos, trabajos y noticias en internet o bibliotecas sobre el tema en cuestión. Como primer paso, se requiere profundizar sobre el contexto en el que se engloba el trabajo.
- **Instalación del software:** Se estudian las diferentes aplicaciones que se pueden utilizar para llevar a cabo la implementación del algoritmo. Una vez examinadas, se selecciona el software necesario para proceder a su instalación.
- **Estudio del algoritmo base:** Se desarrolla un análisis en profundidad del código ya existente, sobre el que se realizarán posteriormente las modificaciones, para comprender su funcionamiento. Es preciso distinguir entre los diferentes módulos y partes que lo componen, además de identificar las funciones de cada uno de los procesos.
- **Desarrollo del nuevo algoritmo:** Es la fase más importante del trabajo. Consiste en el planteamiento de la transformación a realizar para crear la estructura que tendrá el modelo distribuido. Se programan los nuevos procesos en base a las especificaciones iniciales para cumplir con los objetivos del proyecto.
- **Realización de pruebas de funcionamiento:** Durante la etapa anterior de desarrollo, es necesaria la constante depuración del código para seguir la evolución del mismo y obtener las funcionalidades deseadas. Una vez finalizada la programación del algoritmo, se lleva a cabo la evaluación de su funcionamiento y de su grado de eficiencia mediante su implementación. Esta se basará en la ejecución de simulaciones en diferentes escenarios y topologías de red, a partir de las cuales se obtendrán los resultados.
- **Informe final del proyecto:** Se elabora el presente documento para reunir todos los puntos realizados anteriormente y exponerlos de forma detallada. Se trata, por tanto, de redactar la presente memoria de TFG.

1.3 Estructura del trabajo

En este Apartado se puede visualizar de forma general la estructura que tendrá el TFG. Se dividirá en los siguientes Capítulos:

1. **Introducción:** Presentación de los aspectos más importantes que se van a tratar y de los pasos a realizar. También, se añade un breve resumen de los objetivos de la realización del trabajo.
2. **Estado del arte:** Establecimiento del marco teórico en el que se sitúa el proyecto y documentación de los conceptos principales. Se introduce en contexto y se relaciona el desarrollo práctico, que se llevará a cabo posteriormente, al ámbito de las redes energéticas inteligentes. Además, se expone el estudio de las herramientas de las que se dispone y se definen sus características.

3. **Diseño y análisis:** Discusión de las diferentes posibilidades que existen a nivel de software para llevar a cabo la implementación del algoritmo. En base a las características de cada una de las herramientas, se justifica finalmente cuáles de ellas se emplearán y de qué forma.
4. **Desarrollo:** Es el capítulo dedicado al proceso de programación del algoritmo. Por un lado, se describe de forma cronológica las fases del planteamiento del nuevo modelo y las decisiones que se han decidido tomar según ha ido evolucionando el código. Por otro lado, se definen en detalle los módulos que contiene el algoritmo y se establecen los tiempos de ejecución, en base a los retardos impuestos para cada proceso.
5. **Evaluación de resultados:** Justificación del correcto funcionamiento del algoritmo. Se describen los resultados obtenidos de la simulación de un ejemplo de topología de red mediante las evidencias correspondientes.
6. **Conclusiones y líneas futuras:** Exposición de un capítulo de finalización para comentar las conclusiones del desarrollo realizado. Se plantean las tareas y acciones que seguiría este trabajo en una futura continuación.
7. **Bibliografía:** Se escriben todas las referencias a artículos, páginas web, ensayos o libros que se han consultado para escribir el presente documento. Se hace uso del estilo de citación del IEEE.
8. **Anexo:** Se añade un manual de usuario que comprende la instalación de las herramientas a nivel de software necesarias para desarrollar la parte práctica del trabajo.

Capítulo 2

Estado del arte

Como objetivo del presente capítulo, se expondrán en detalle los conceptos más relevantes relacionados con el proyecto, además de las herramientas y el software empleado para el desarrollo del mismo. El fin principal de este apartado es entrar en el marco teórico que engloba a este TFG y, así, conseguir una mayor compresión de su diseño en los siguientes capítulos.

2.1 Smart grids

Una *smart grid* [21] o red energética inteligente se puede definir como aquella red que es capaz de suministrar electricidad de forma eficiente a sus usuarios según la información que adquiere de los mismos. Se plantea un esquema energético distribuido basado en una comunicación bidireccional constante entre la compañía eléctrica y los clientes. En la Figura 2.1 se visualizan los aspectos diferenciales entre el modelo perteneciente a la red eléctrica convencional respecto al de una smart grid.

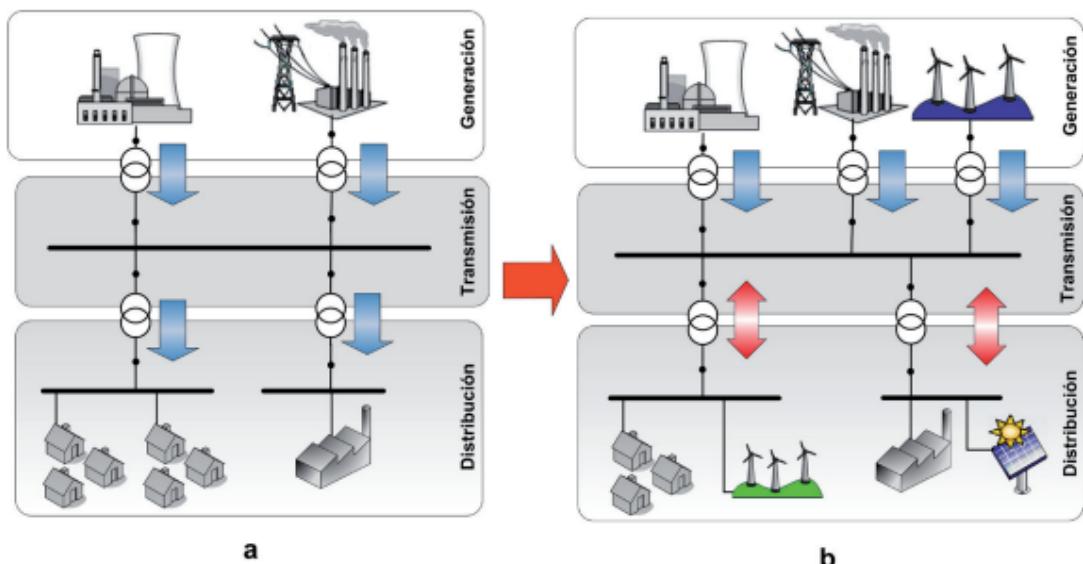


Figura 2.1: Diferencias entre el modelo de red convencional (a) y el de red inteligente (b) [2]

A partir de este esquema de intercambio de información, el sistema registrará las acciones de cada uno de los usuarios conectados y logrará distinguir entre aquellos que se comportan como consumidores, los que son generadores de energía o los prosumidores, los cuales desarrollan ambos papeles. Por lo tanto, el pilar fundamental de una smart grid son los usuarios de la propia red. La interacción que se produce entre estos y la compañía consigue proporcionar un seguimiento constante y eficaz del estado del sistema.

En otros términos, una red inteligente necesita la introducción de sensores y equipos automatizados capaces de gestionar y controlar los datos relacionados con la cantidad de energía que es entregada a cada uno de los consumidores. Se requiere crear una red de contadores que determinen los niveles de transferencia energética que se están produciendo en cada momento y la demanda que notifican los clientes [6]. En la Figura 2.2 se visualizan los diferentes elementos que participan en el funcionamiento de una smart grid y que componen el esquema distribuido.

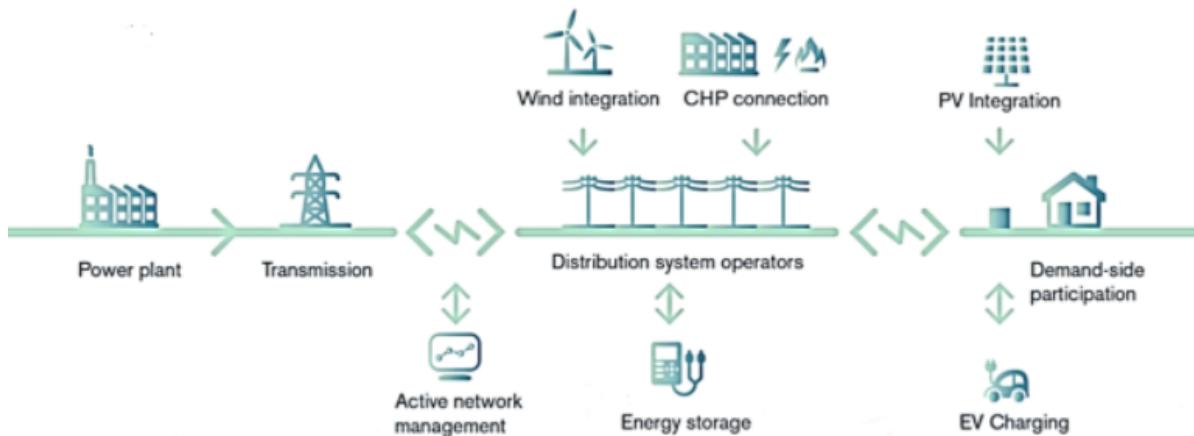


Figura 2.2: Esquema de sistema distribuido en una smart grid [3]

La monitorización a tiempo real evitara proporcionar más energía de la necesaria a cada una de las líneas finales. En consecuencia a ello, se conseguirá uno de los objetivos principales de una smart grid: reducir costes. Esencialmente, el seguimiento de las acciones de los usuarios se fundamenta en la recopilación de grandes cantidades de información de los mismos y en la consecuente asignación de patrones de uso de electricidad a partir de estos datos. La gestión energética se vuelve más concreta y personalizada para los usuarios, ya que serán divididos en diferentes grupos de consumo según determinados parámetros, como la zona geográfica o los horarios de uso.

Gracias a este proceso, el sistema será capaz de predecir el comportamiento futuro de cada uno de los consumidores según al tipo al que pertenezcan. En el procedimiento de adquisición y análisis de datos entrará la figura del centro de operaciones o *service center*, el cual llevará a cabo la gestión de la información recibida por parte de las líneas finales a las que da servicio [26]. Será posible conocer a tiempo real el estado de la red o de una parte de ella y actuar en caso de fallos. En el caso de haber incidencias, podrá identificarlas, notificar sobre ellas a la compañía y a los propios usuarios y resolverlas por sí mismo de forma automatizada.

2.1.1 Seguridad en smart grids

Por cuestiones de seguridad es preciso que un centro de operaciones tenga la capacidad de actuar de forma inmediata ante problemas en la red. Si no se toma esto en cuenta es posible que se produzcan interrupciones del suministro de electricidad a gran escala, agravando las consecuencias de la incidencia.

Es decir, en algunos casos un problema de red se puede propiciar por errores de configuración del propio sistema, pero también a veces por ataques externos que atenten contra este. Los equipos o dispositivos IoT que engloban a la red deben ser programados para protegerse de ellos, conociendo las amenazas a las que se pueden enfrentar y prepararse con antelación. En la Tabla 2.1 se describen algunos ejemplos de amenazas conocidas [27] y el mecanismo de seguridad que debería de imponer la red para protegerse ante ellas:

Ataque	Riesgos en la red	Soluciones
Denegación de servicio (Ataque DoS) [28]	Bloqueo de equipos e interrupción de servicio, pérdidas de suministro y de conexiones entre los usuarios.	<ul style="list-style-type: none"> - Gestión de las conexiones entre dispositivos mediante filtros de tráfico (firewalls) - Detección inmediata de equipos intrusos - Autenticación criptográfica - Restricciones del acceso físico a los equipos de la red
ARP spoofing [29]	<p>Envenenamiento de las tablas ARP y suplantación. El atacante envía mensajes falsos ARP para obtener el control de tráfico y de los paquetes que se transmiten por la red. Pérdida de privacidad de datos y posibles alteraciones de estos.</p>	<ul style="list-style-type: none"> - Monitorización del tráfico - División de la red en subredes - Empleo del protocolo SEND (Secure Neighbor Discovery) para enrutamiento
Man in the Middle y Eavesdropping [30]	Intrusión del atacante en la comunicación entre dos dispositivos o equipos para obtención, descarte o alteración de paquetes de datos.	<ul style="list-style-type: none"> - Cifrado de comunicaciones - Autenticación - Comprobación de la integridad de los datos recibidos mediante firma digital

Tabla 2.1: Ejemplos de posibles ataques a smart grids

El análisis de los posibles riesgos tendrá que ser actualizado de forma continua en el tiempo para conseguir cierta efectividad. Además, debe de ser exhaustivo y preventivo: la red tendrá que conocer la probabilidad de que se den ciertas amenazas y el nivel de impacto que tendrían si se producen. Una vez que se domina esta información, es imprescindible minimizar las vulnerabilidades de la red

mediante mecanismos propios de protección. En una smart grid participan multitud de dispositivos que mantienen interacciones con equipos externos y ajenos a la red. Es importante que exista un control en estas comunicaciones, ya que pueden suponer posibles entradas de ataques al sistema.

2.1.2 Impacto medioambiental

Uno de los fines [31] que se persiguen con las smart grids es la reducción del impacto medioambiental en la obtención y distribución del suministro. El modelo de red distribuido permite conocer el estado constante de los recursos y determinar las zonas geográficas de mayor consumo. Además, la capacidad de generar energía en mayor o menor cantidad según la situación de demanda en la red elimina sobrecostes de producción.

Es importante el papel de las compañías eléctricas [32] en el ahorro energético. Una de las aportaciones que pueden llevar a cabo en el contexto de una red inteligente es el asesoramiento al usuario. Es decir, pueden recomendar tanto a clientes particulares como a empresas sobre la cantidad de potencia que deberían contratar y así, optimizar el suministro.

Por otro lado, es imprescindible que el sistema pueda detectar rápidamente las incidencias que se producen en los equipos a lo largo de la red y sobre todo, que sea capaz de solventarlas automáticamente. Internamente, un fallo en uno o varios equipos de la red podría conducir a pérdidas del suministro, interrupciones de los procesos de distribución o a desconexiones entre los prosumidores. Es necesario el mantenimiento de los equipos de la red y una monitorización constante de los mismos, además de su actualización para prevenir riesgos ante amenazas externas.

Las smart grids introducen la posibilidad de ofrecer un abastecimiento seguro al consumidor. Se presentan como una solución ante la descarbonización y ante el agotamiento de las fuentes energéticas provenientes de los combustibles fósiles. Esto es porque fomentan la integración de las energías renovables en el sistema eléctrico, favoreciendo la conexión de la red a este tipo de fuentes frente a las convencionales [33].

Para asegurar el abastecimiento y dar un uso eficiente a las fuentes de energía renovables es imprescindible que exista una adaptación del sistema a la situación en cada instante en el tiempo. Esto se debe a que por norma general, no son recursos que se pueden obtener de forma regular. Por ejemplo, la disponibilidad de fuentes como la energía solar o la eólica depende directamente de las condiciones climáticas dadas. A esto se le suma también, la situación de demanda de consumo eléctrico en la red.

Por consiguiente, la red debería de estar dotada de un sistema de almacenamiento, en el que los prosumidores puedan acumular sus excedentes para tener suministro disponible en un futuro o para repartirlo con otros prosumidores [34]. En la actualidad, los sistemas de almacenamiento se encuentran en proceso de estudio. La búsqueda de su optimización es un punto clave para poder incrementar las fuentes de energía renovables y realizar una implementación progresiva de su uso en el mercado energético. Garantizan la sostenibilidad del sistema y son la solución principal a la desventaja de la irregularidad de suministro que provocan estas fuentes [33].

2.1.3 La figura del prosumidor

Como se ha comentado anteriormente, un prosumidor [22] es aquel usuario o comunidad de usuarios en la red inteligente con la capacidad de consumir y generar electricidad de forma simultánea. Tiene cierta independencia de la red al posibilitar el autoconsumo, aunque también puede compartir la energía producida con los demás usuarios del sistema.

Con la aparición del papel del prosumidor se puede disponer de diversas microrredes de distribución y almacenamiento a lo largo de la *smart grid*, las cuales abarcarián a comunidades de usuarios. Es de gran importancia, ya que constituye la descentralización de la red eléctrica, además de que el esquema de microrred eléctrica se podría aplicar tanto en entornos urbanos, rurales o industriales. En el caso urbano, surge el concepto de *smart cities* [32].

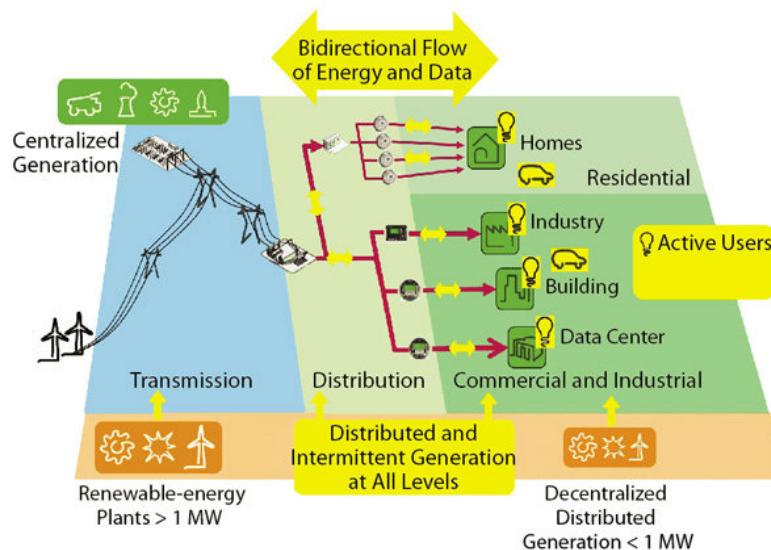


Figura 2.3: Ejemplo de una microrred inteligente [4]

Como concepto, el modelo de ciudad inteligente se basa principalmente en el empleo de fuentes de energías limpias, integrando la red distribuida que presenta una *smart grid*. En la actualidad, muchos núcleos urbanos están fomentando la tendencia a convertirse en *smart cities*. Esto se produce a través del mobiliario urbano, con la construcción de edificios sostenibles con el uso de paneles solares o mediante el transporte, promoviendo el uso de autobuses urbanos con motor eléctrico e incorporando estaciones de carga para los vehículos eléctricos particulares.

En cualquier ámbito, para generar una microrred sería necesario un conjunto de dispositivos generadores de electricidad que puedan abarcar el suministro de una zona geográfica determinada. Se deben ubicar con cierta cercanía al conjunto de usuarios finales a los que van a abastecer para reducir los costes del transporte y de distribución. Además, es imprescindible la cooperación y coordinación de los diferentes equipos para que la energía producida sea gestionada de forma eficiente.

Por otro lado, tiene que existir una gestión de los flujos de electricidad que provienen o se dirigen al exterior de la misma. De esta gestión se encarga el operador del sistema distribuido (DSO) [35] para asegurar el abastecimiento de recursos en el interior de la microrred.

2.1.3.1 Sistema de transacciones energéticas

La eficiencia de una smart grid se basa en el conjunto de transacciones energéticas que se producen entre los prosumidores y las cuales se fundamentan en la tecnología blockchain [36]. Esta se puede describir principalmente, como la creación de un libro mayor compartido por los equipos de la red en el que registran la información de las transacciones realizadas. Los dispositivos autorizados podrán acceder a estos datos de forma inmediata y transparente y se asegurarán de la integridad de la información.

En otros términos, la tecnología *blockchain* registra cada una de las transacciones como bloques de datos y los encadena uno a uno de forma irreversible. A cada bloque añadido se le aplica un timestamp y una función *hash*. Así, cuando llegue otro bloque este almacenará el *hash* del bloque anterior y repetirá el proceso con todos los posteriores. En consecuencia, se consigue que toda la información introducida respecto a las transacciones resulte inalterable. Para cambiar los datos de un bloque sería necesario recalcular su propio *hash*, cambiar el *hash* que guarda del bloque anterior y llevar a cabo el mismo procedimiento uno por uno en toda la cadena.

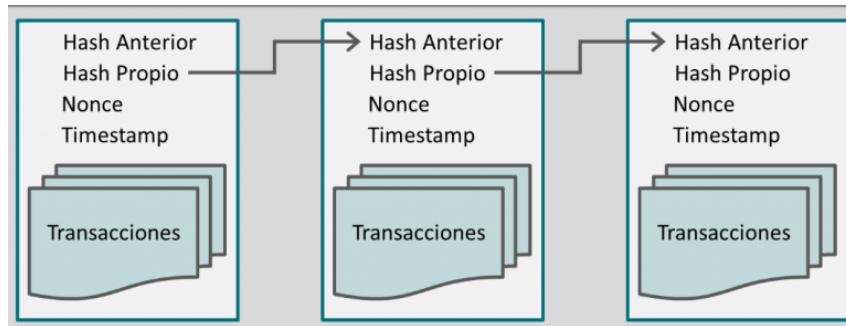


Figura 2.4: Esquema del encadenado de bloques en la tecnología blockchain [5]

Este proceso se vuelve prácticamente imposible, ya que se cuenta con que una función hash es de un solo sentido, y no es fácil obtener los datos que se introdujeron en un principio para alterarlos. Si se realizará la más mínima modificación en un bloque, el sistema lo detectaría y rechazaría automáticamente el cambio realizado [7].

Mediante este método de protección, los usuarios activos en la red pueden realizar negociaciones seguras de los recursos energéticos ante las diferentes situaciones de demanda en las que se encuentren. En este proceso entra el papel de los smart meters o contadores inteligentes. Se componen de sensores físicos que miden de manera precisa la cantidad de energía generada y consumida en cada línea final.

El procedimiento de ejecución de una transacción entre dos o más prosumidores se puede dividir en los siguientes pasos [6][37]:

1. **Creación del *smart contract*** [38]: Dos o más usuarios de la red escriben un contrato inteligente y plantean en él un acuerdo de compra-venta de energía. Como definición, se puede decir que un *smart contract* hace referencia a un protocolo informático que realiza una serie de acciones de forma automática. Verifica una transacción y asegura que los prosumidores que participan en el acuerdo cumplen con las reglas que han aceptado en el mismo.

2. **Aceptación del *smart contract*:** Los demás usuarios y equipos de la red deben aprobar el contrato para que este tenga validez. Se deben verificar sus parámetros (timestamp y firma) para que se lleve a cabo la transacción energética.
3. **Proceso de venta:** Uno de los prosumidores que forma parte del acuerdo observa que sobrepasa en su contador un valor máximo y decide crear una transacción de venta. El vendedor tiene que demostrar a la red que posee el recurso para poder originar la oferta de venta. Esta se almacenará en el sistema haciendo referencia al recurso y de forma anónima para preservar la privacidad del usuario.

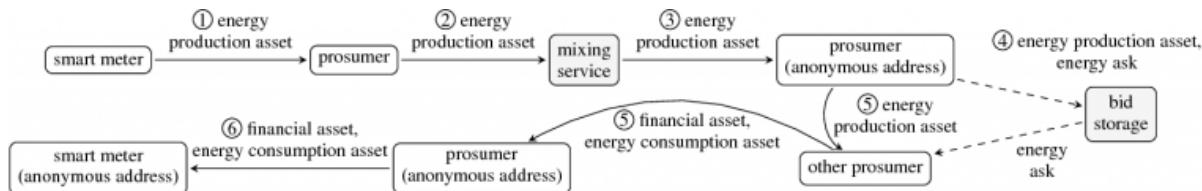


Figura 2.5: Secuencia de acciones en una transacción de venta [6]

4. **Proceso de compra:** Otro usuario comprueba en su contador inteligente que la cantidad de recursos de la que dispone está por debajo de un valor umbral. Por tanto, tiene que generar una transacción de compra y comunicar su demanda al servicio de ofertas (Bid Storage Service). Las transacciones se realizan de forma directa entre los prosumidores (peer to peer), por lo que el servicio de ofertas no actúa como intermediario entre estos, si no que les da soporte para almacenar las ofertas. En la Figura 2.5 se puede observar su funcionalidad y el orden de las acciones que lleva a cabo.
5. **Casamiento:** Si las transacciones anteriores se reciben en el smart contract en un mismo período de tiempo se verifica primero que se cumplan las condiciones necesarias en la red. En caso afirmativo, se lleva a cabo la transferencia de recursos.
6. **Confirmación del traspaso:** El usuario comprador contabiliza en su contador la cantidad recibida y lo comunica al contrato. Cuando se produce la confirmación, automáticamente se abona al vendedor la cuantía establecida en el acuerdo.
7. **Creación de bloque:** Las transacciones validadas se van almacenando en el sistema y, una vez se ha agrupado un gran número de ellas, se forma un bloque de datos. Después, se añade el timestamp y se aplica la función hash.
8. **Actualización de la cadena:** El nuevo bloque solicita registrarse en la cadena y el sistema confirma que es válido. Se finaliza por tanto, el procedimiento de la transacción entre los prosumidores.

Una de los requerimientos que debe cumplir este proceso es dotar de privacidad a los usuarios. Internamente, todas las transacciones que se realizan en la red son anónimas y únicamente el propio prosumidor conoce su historial de ofertas.

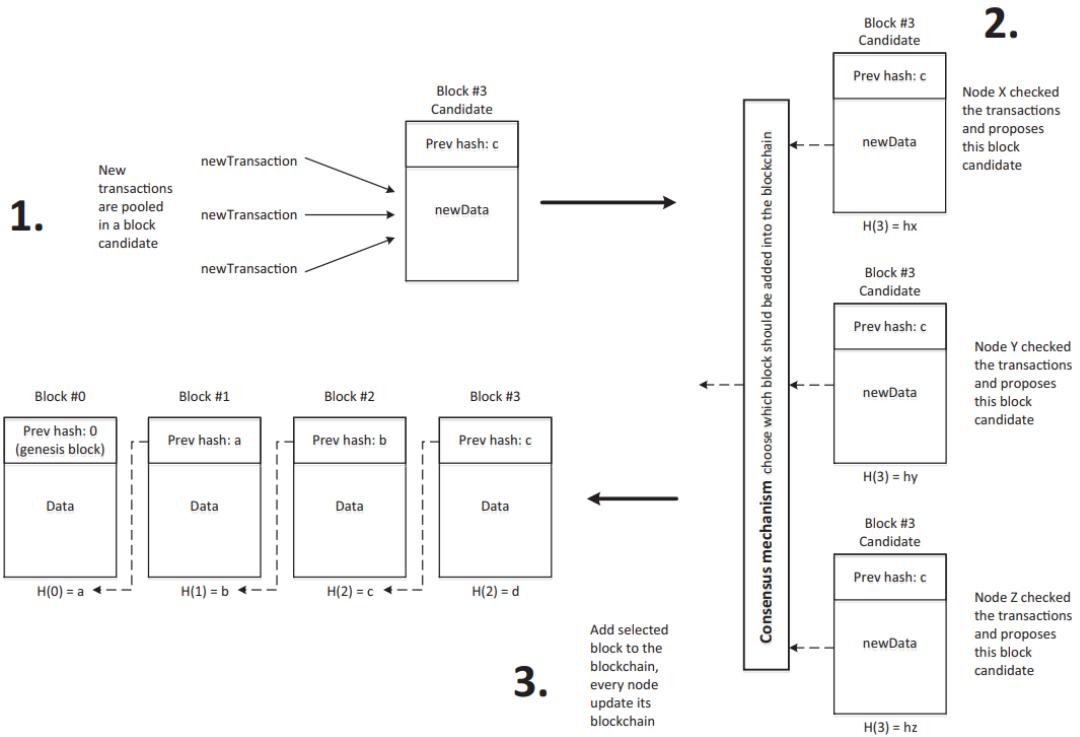


Figura 2.6: Esquema de actualización de la cadena en la tecnología blockchain [7]

Se debe contar con un servicio de generación de direcciones aleatorias o de mezcla de datos (ver Figura 2.5) que permita proteger los datos privados de los usuarios. También, es importante que exista cierta coordinación entre los usuarios para la ejecución de las transacciones de forma sincronizada y estable en el tiempo.

2.2 Redes LLN

Las redes LLN (*Low Power and Lossy Network*) [39] se pueden definir como redes de baja potencia y con pérdidas. Es decir, están compuestas por equipos o nodos interconectados que tienen limitaciones en términos de potencia, memoria y recursos de procesamiento. Por lo general, se puede denominar LLN a aquellas redes que están conformadas por dispositivos IoT, que integran sensores y/o actuadores con limitaciones de capacidad y batería (por ejemplo, alimentados por pequeños paneles solares).

El desarrollo de algoritmos de enrutamiento que garanticen la eficiencia en redes LLN es una tarea complicada. Además de las limitaciones de recursos que presentan los dispositivos, el modelo de red LLN se caracteriza por ser dinámico y poseer bajas tasas binarias de transmisión datos (en rangos de Kbps). Es

por ello, que es imprescindible que las tareas de enrutamiento sean adaptables a los cambios de topologías que pueden suceder y también, que cada nodo sea capaz de tener un control local de los datos que se transmiten para no producir redundancia.

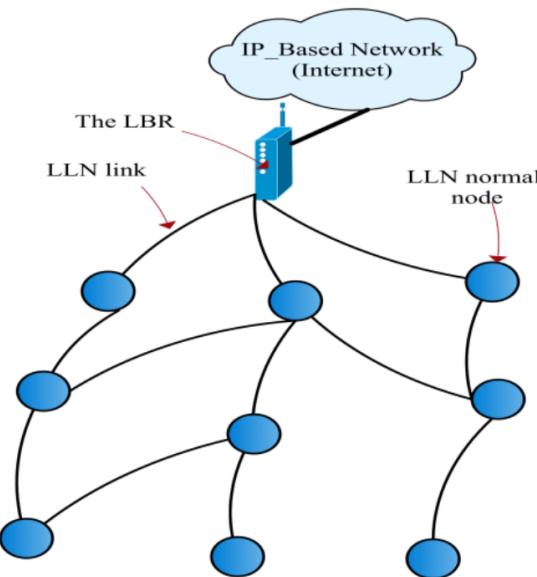


Figura 2.7: Estructura de una red LLN [8]

Por otro lado, el radio de comunicación que proporcionar cada nodo abarca un área muy reducida. Para posibilitar conexiones entre un gran número de dispositivos repartidos en el espacio de la red se deben realizar comunicaciones multisalto [40][10].

En cuanto a la estructura de las redes de baja potencia y con pérdidas, los dispositivos pueden interconectarse por un gran variedad de tipos de enlace [41]. Entre ellos, se encuentra el estándar IEEE 802.15.4 o tecnologías como *Bluetooth*, *WiFi* de baja potencia o PLC (*Powerline Communication*). En el caso de PLC, se produce la transformación de la red eléctrica en líneas de alta velocidad de transmisión para permitir acceso a internet a los nodos.

Es decir, se hace uso del propio cableado y se distingue entre la señal eléctrica y la señal digital de datos mediante filtrado de frecuencias. La separación [9] será relativamente simple, ya que la corriente eléctrica hace uso de frecuencias muy bajas (50Hz) y la señal de datos, de frecuencias muy altas (mayores a 10MHz).

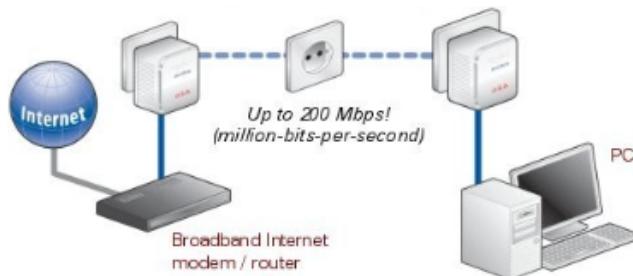


Figura 2.8: Ejemplo de red con tecnología PLC [9]

2.2.1 Clases de nodos en redes LLN

En la Imagen 2.7 se representa la arquitectura básica de una red LLN. Por defecto, se compone de una serie de nodos IoT y de un router de frontera (LBR) [42], que se encarga de producir la unión entre el mundo físico y el internet [8]. Su funcionamiento se visualiza en mayor detalle en la Figura 2.11.

Además, en cuanto a los nodos o dispositivos IoT, cada uno de ellos poseerá diferentes propiedades internamente en la red. Según sus características en términos de memoria y procesamiento, se pueden clasificar en las siguientes clases [43]:

2.2.1.1 Clase 0

Esta enfocada a sensores con un tamaño de memoria mínimo, no superando generalmente los 10KB. No tienen la capacidad suficiente para desarrollar una comunicación directa con internet, por lo que deben de utilizar como soporte nodos gateway o servidores proxy. Suelen preconfigurarse para el envío de señalización a través de la red.

2.2.1.2 Clase 1

A diferencia de la clase anterior, los nodos de este tipo sí son capaces de llevar a cabo una comunicación con internet y con otros dispositivos IoT por sí mismos. Siguen teniendo una memoria bastante limitada, pero es suficiente para mantener una pila de protocolos ligeros. Para este caso se emplea el protocolo de aplicación restringido CoAP (*Constrained Application Protocol*) [44], que aporta funcionalidades similares a HTTP y UDP, como protocolo de transporte.

2.2.1.3 Clase 2

Engloba a los nodos que poseen menores restricciones de memoria, proporcionando en torno a 50KB. Con este tamaño se posibilita el soporte de una pila de protocolos tradicional y más funcionalidades en la comunicación.

2.2.2 Estándares y protocolos en redes LLN

La progresiva implementación de la tecnología IoT en diferentes ámbitos de aplicación ha provocado el desarrollo de distintos estándares con el fin de cubrir las especificaciones impuestas en las redes LLN.

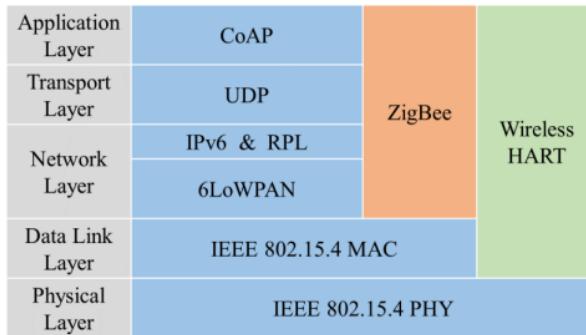


Figura 2.9: Pila de protocolos en redes LLN [10]

2.2.2.1 IEEE 802.15.4

En las redes inalámbricas es el estándar que determina la capa física y el control de acceso al medio. Proporciona un coste mínimo en la transmisión de datos con tasas máximas de transferencias de 250 Kbps y con un alcance típico de 10 metros para la comunicación entre nodos. Estas especificaciones derivan en un estándar de comunicación básico, flexible a redes dinámicas y de bajo consumo energético [45].

IEEE 802.15.4 introduce adaptabilidad a la situación de la red en tiempo real. Para ello, emplea el protocolo CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance*) [46]. Como concepto, se trata de un tipo de protocolo de acceso múltiple que regula el uso que realizan los dispositivos de la red de los canales de comunicación disponibles.

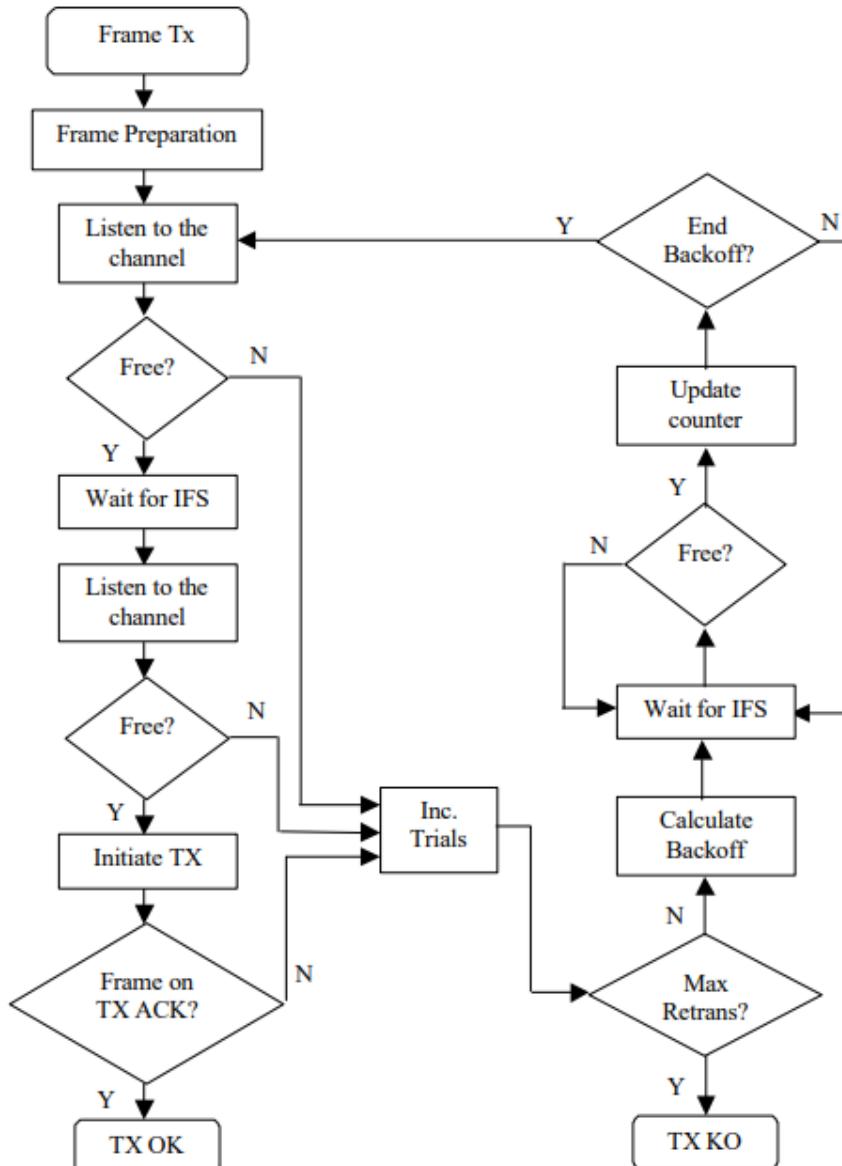


Figura 2.10: Diagrama de flujo de CSMA [11]

En la Figura 2.10 se visualiza mediante un diagrama de flujo la secuencia de acciones que sigue el protocolo CSMA para realizar de forma segura las transmisiones [11]. El principal objetivo es la compartición del mismo medio de transmisión. Es indispensable una monitorización constante del estado de los canales (por detección de portadora) y la organización de los mismos en ranuras temporales o slots para evitar colisiones. Es decir, si un nodo pretende realizar una transmisión, debe primero consultar a la red si el canal que quiere utilizar se encuentra ocupado por otro nodo. En caso de no poder acceder en ese instante o si se producen fallos de conexión, el nodo espera a realizar otro intento.

Posteriormente, una vez el nodo consigue acceso al canal, realiza la transmisión de datos y espera la llegada de la confirmación de que el paquete fue entregado correctamente en el punto de acceso (AP). Si no la recibe, asume que se ha producido una colisión e intenta su retransmisión. Con este proceso, el protocolo CSMA-CA consigue eliminar la posibilidad de que se produzcan colisiones de paquetes y se consigue una eficiencia en la transmisión de datos en las redes de baja potencia.

No obstante, presenta ciertas desventajas como la introducción de tiempos de espera fijos determinados. Estos son consecuencia del empleo de los slots temporales para planificar el uso del canal compartido. Además, en CSMA-CA se puede ocasionar un tráfico adicional de datos debido a las retransmisiones.

2.2.2.2 6LoWPAN

Se define como el protocolo de capa de red que posibilita la transmisión de paquetes IPv6 en el contexto de redes inalámbricas de baja potencia. Es decir, funciona como una subcapa de adaptación entre las restricciones impuestas por el estándar IEEE 802.15.4 y las especificaciones de IPv6.

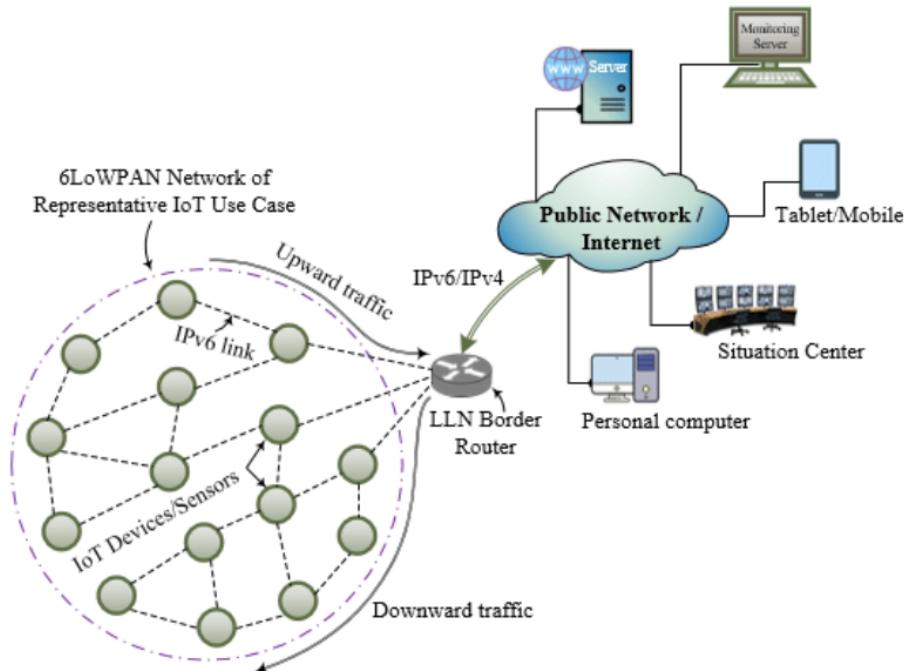


Figura 2.11: Estructura de una red LLN [12]

6LoWPAN implementa la transición entre el tamaño mínimo de datagrama de IPv6 (1280 bytes) y la unidad máxima de transmisión (MTU) de IEEE 802.15.4 (127 bytes). Por ello, se reduce el tamaño de los paquetes a través del encapsulado de los datos, la comprensión de cabeceras y la codificación de las direcciones IPv6 [47]. En la Figura 2.11 se ilustra la unión que se produce entre un ejemplo de red LLN, compuesta por múltiples dispositivos IoT, y la red externa o internet.

2.2.2.3 RPL

RPL [25] (*Routing Protocol for Low-Power and Lossy Networks*), como sus siglas indican, se define como el protocolo de enrutamiento definido para las redes LLN, el cual presenta dos modos posibles de funcionamiento: *storing* y *non-storing* [48].

Principalmente, se puede expresar que se fundamenta en la configuración de topologías de red en forma de árbol, con comunicaciones multipunto-punto y mediante grafos acíclicos dirigidos (DAG) [49]. En otros términos, para determinar las tablas de enrutamiento en una red LLN, se genera un grafo a partir de las interconexiones del conjunto de nodos que existen en la red. Se definen las direcciones y sentidos de los flujos de datos que se producen en cada una de las líneas y se crean relaciones únicas y de un solo sentido para cada uno de los dispositivos que interviene en la red.

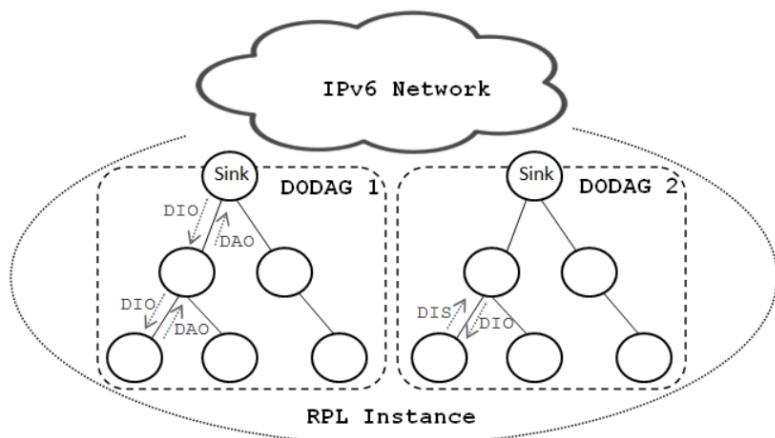


Figura 2.12: Ejemplo de arquitectura RPL [13]

Con el uso de vectores se asegura que no se producen bucles de información en la estructura completa del grafo. Internamente, se produce una especie de sumidero, en el que cada nodo hace referencia a uno o varios nodos. Como se representa en la Figura 2.12, la estructura tiene como inicio un primer nodo denominado como nodo raíz (DODAG). En el contexto de las redes de baja potencia, es preciso especificar que el nodo raíz representa al router de frontera (LBR), ya que este es el destino final del tráfico en el dominio de la red LLN. RPL define dos tipos de rutas en función del sentido que toman sus vectores, pudiendo ser upward (hacia arriba) o downward (hacia abajo).

2.2.2.3.1 Ruta *Upward*

Este tipo de ruta determina un flujo de tráfico que parte de los nodos con destino el router frontera (LBR). Es preciso que cada nodo defina a uno de sus vecinos como parente, indicando así el siguiente salto que debe tomar el flujo de tráfico para ascender hasta el router.

2.2.2.3.2 Ruta *Downward*

En cambio, en este caso se hace referencia a tráfico generalmente externo (Internet) y que se dirige a uno o varios nodos de la red LLN. Cada nodo debe anunciar al menos a uno de sus padres para construir el camino hasta los nodos más lejanos al router frontera [41].

2.2.2.3.3 Tipos de mensajes en RPL

Internamente, existen tres tipos de mensajes [14] en RPL para configurar el enrutamiento. En las Figuras 2.12 y 2.13 se expone de forma gráfica el funcionamiento de estos y el sentido que toman.

- **DIS (*DODAG Informational Solicitation*):** Los nodos envían solicitudes de información para encontrar DODAG cercanos.
- **DIO (*DODAG Information Object*):** Es la respuesta a los mensajes DIS por parte del DODAG y se envía de forma periódica en sentido *downward*. Cada nodo actualiza la información sobre la topología en el instante en el que lo recibe y lo reenvía a sus hijos.
- **DAO (*Destination Advertisement Object*):** Al contrario del mensaje DIO, se envía en sentido *upward* para actualizar la información de cada uno de los nodos a sus padres.

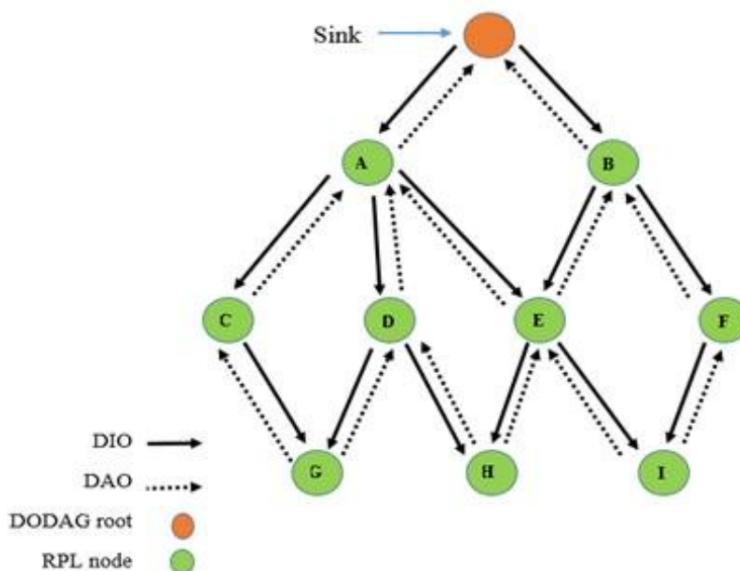


Figura 2.13: Representación de los mensajes de información en RPL [14]

2.2.3 Encaminamiento eficiente

En el Apartado 2.2.2.3 se ha definido el protocolo RPL como el protocolo de enrutamiento por defecto para las comunicaciones entre dispositivos IoT en las redes LLN. Sin embargo, este presenta algunas desventajas en términos de eficiencia y escalabilidad.

Por un lado, en el caso de empleo del protocolo RPL en modo *non-storing*, no existe un almacenamiento de rutas en los nodos. A medida que aumente el tamaño de la red y el número de dispositivos que participan en ella, se introducirán multitud de mensajes de encaminamiento para posibilitar las comunicaciones. Se desencadena un proceso de enrutamiento muy ineficiente, ya que los nodos se ven obligados a dirigir los paquetes al nodo DODAG o router frontera en la red LLN (LBR) para añadir las cabeceras necesarias. Después, este reencamina los datos al destino indicado por el nodo emisor.

Por otro lado, para el modo *storing* [48], se requiere que cada uno de los nodos de la red tenga disponible a nivel local las rutas hacia cualquier destino. Es decir, cada nodo da la responsabilidad a su padre de conocer los siguientes saltos que se deben realizar para comunicarse con un nodo determinado. En este caso también, si se incrementan las dimensiones de la red, surgen problemas de eficiencia. Se produce un gran consumo de memoria y un aumento innecesario del tamaño de las cabeceras.

2.2.3.1 Optimización del protocolo RPL

Con el fin de mejorar el protocolo RPL y eliminar los problemas indicados anteriormente se han desarrollado nuevas versiones. Algunos de los algoritmos más relevantes:

- **OSR (*Opportunistic Source Routing Protocol*)** [50]: Introduce una optimización del enruteamiento que se produce con sentido desde el nodo raíz o DODAG hacia los demás nodos (sentido downward). Realiza un proceso de codificación de cabeceras para comprimir y reducir de forma significante su longitud. Además, en las tablas de encaminamiento de cada nodo se almacena un único hijo, forzando el siguiente salto hacia este.
- **DT-RPL (*Diverse Bidirectional Traffic Delivery Protocol*)** [51]: Su funcionamiento se basa en la actualización del estado de los enlaces de forma periódica para soportar distintos patrones de tráfico. El proceso se produce tanto en sentido upward como downward y consigue optimizar el consumo energético, ya que determina las transmisiones de paquetes en función del estado de la red en cada instante.
- **ZTR (*ZigBee Tree Routing*)** [52]: Se aplica en la red un encaminamiento basado en etiquetas para el estándar ZigBee. En primera instancia, se le asigna a cada nodo una etiqueta, partiendo desde el DODAG y siguiendo un esquema de árbol. En términos de escalabilidad, es un algoritmo eficiente, pero no optimiza de forma significante el enruteamiento.

- **STR (Shortcut Tree Routing)** [52][53]: Se presenta como una versión mejorada del algoritmo ZTR para el encaminamiento de los paquetes. Como su nombre indica, se determinan atajos en el esquema de árbol para acortar el camino entre nodos adyacentes. De otra forma, se produce el siguiente salto al nodo que tenga una cuenta menor de saltos restantes hacia el destino. En la Figura 2.14 se aprecia visualmente esta optimización y se representan las diferencias entre la ruta que toma STR respecto a ZTR.

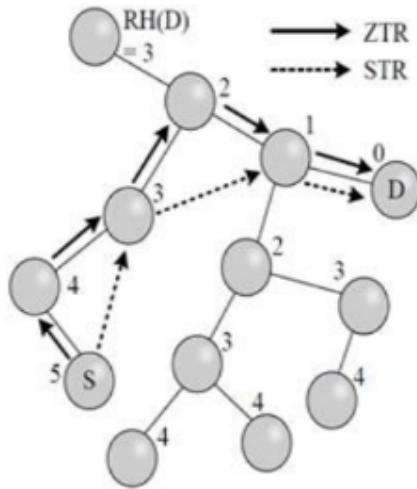


Figura 2.14: Diferencias de enrutamiento entre los algoritmos ZTR y STR [15]

En términos de optimización RPL, este TFG se centrará principalmente en el protocolo IoTorii. Este supondrá la base del diseño y desarrollo del algoritmo Dedenne como se define en el Capítulo 4.

2.2.3.2 IoTorii

IoTorii [16] se presenta como un algoritmo jerárquico que realiza un enrutamiento basado en la exploración de la red. Se pueden encontrar los detalles de su implementación en el sistema operativo Contiki (más información en la Sección 2.3.1) en el repositorio de GitHub¹.

Generalmente, como se ha comentado en el Apartado 2.2.3.1, los algoritmos de mejora de RPL requieren un conocimiento completo de la red nodo a nodo. Esta cualidad provoca un aumento de los retardos y un intercambio mayor de mensajes, enviándose en ocasiones incluso informaciones duplicadas. IoTorii como protocolo, evita que esto suceda y transmite mensajes de difusión para propagar los datos rápidamente por toda la red. Se trata de un algoritmo altamente escalable con independencia del tamaño de la red.

El proceso de creación de rutas comienza analizando primero los múltiples caminos existentes entre cualquier origen y destino, en vez de calcular los vectores distancia como se realiza en RPL. En concreto, reutiliza información de los nodos (dirección MAC) para establecer la jerarquía de las motas en la red mediante la asignación de etiquetas. En otros términos, IoTorii concede como etiqueta o identificación una dirección MAC jerárquica (HLMAC) a cada uno de los nodos.

¹<https://github.com/NETSERV-UAH/IoTorii>

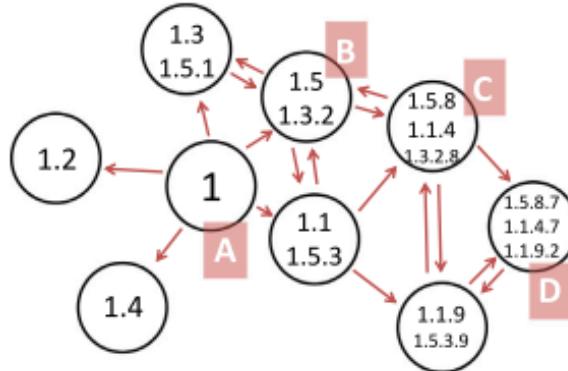


Figura 2.15: Asignación de etiquetas en el algoritmo IoTorii (I) [16]

Como se puede observar en la Figura 2.15 la asignación comienza desde el nodo raíz o DODAG, el cual envía un mensaje a sus nodos vecinos informando de su etiqueta y añadiendo a esta un sufijo. El proceso se repite en estos nodos con sus respectivos vecinos y así, uno a uno cada nodo guardará en su tabla mínima una dirección MAC jerárquica (HLMAC). Además, no se producirán bucles porque cada nodo se encarga de no propagar direcciones de sus propios hijos.

De esta forma, cada dirección almacenada significará una ruta distinta para encaminar paquetes hacia un destino. En la Figura 2.16 se representan los diferentes posibles caminos que surgen a partir de una múltiple asignación de etiquetas a cada nodo.

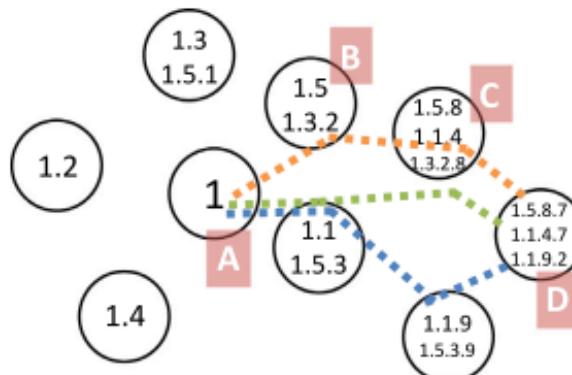


Figura 2.16: Asignación de etiquetas en el algoritmo IoTorii (II) [16]

El número máximo de direcciones HLMAC que obtendrá cada nodo es configurable en el algoritmo. También, el tamaño de estas etiquetas con los parámetros de longitud y anchura. Respectivamente, establecen la cuenta máxima de saltos permitidos respecto al nodo raíz y el número de vecinos que recibirá direcciones. A través del procedimiento de investigación de rutas, consigue reducir las entradas de las tablas de encaminamiento de cada nodo y los mensajes de control que se intercambian en la red. En consecuencia, se minimiza el consumo de memoria y de energía y se logran tiempos de convergencia menores con un mismo número de saltos.

Por otro lado, no se requiere una computación compleja, lo cual lo convierte en un algoritmo más simple respecto a RPL [24][54]. En la Sección 2.3.1.3 se realiza la implementación de IoTorii en el programa Contiki, empleando un ejemplo de topología de red de sensores.

2.3 Simuladores de red y herramientas software

En esta sección se definen de forma detallada las características principales de las herramientas software que se analizarán más adelante para plantear cómo se va a llevar a cabo la evaluación del funcionamiento del algoritmo (ver Capítulo 3).

2.3.1 Contiki-ng

Contiki-ng o, simplemente, Contiki [55] se trata de un sistema operativo ligero para el tratamiento de redes de sensores o dispositivos IoT de baja capacidad. El proyecto fue creado en 2002 por Adam Dunkels, Bjorn Gronvall y Thiemo Voigt, ingenieros del *Swedish Institute of Computer Science* y se desarrolló para posibilitar la implementación del protocolo RPL de forma práctica en redes LLN. Es un software de código abierto y gracias a esto y a la colaboración de otras empresas, con el tiempo ha podido ir evolucionando y actualizándose. Se puede acceder a su última versión desde su repositorio de GitHub².

En términos de software, Contiki tiene como núcleo un kernel y su programación se basa en el tratamiento de los eventos de los diferentes procesos implicados para ofrecer una ejecución multithreading con requisa. Además, se define como un sistema operativo portable [56], que soporta la concurrencia. Es preciso que sea eficiente en el uso de memoria, para soportar múltiples hilos, ya que todos ellos comparten una misma pila.

2.3.1.1 Simulador Cooja

En redes IoT a gran escala la existencia de multitud de dispositivos dificultan el desarrollo y la depuración de los procesos de cada uno de los nodos. Para simplificar el funcionamiento de Contiki se ofrece como opción la herramienta Cooja³. Se puede ejecutar mediante el comando `ant` desde el directorio `contiki-ng/tools/cooja` como se puede visualizar en la Figura 2.17.

Concretamente, Cooja es un simulador gráfico basado en Java que permite monitorizar y visualizar el comportamiento de redes de sensores IoT de cualquier tamaño o topología. Para ello, se debe configurar en primer lugar la red definiendo un número determinado de motas. También, se debe seleccionar el tipo de plataforma [57] en la que se va a basar la simulación de los nodos.

Para ello, se dispone de varios paquetes que proveen diferentes tipos de dispositivos hardware para emular. Así, se permite probar el código antes de ejecutarlo en un controlador hardware real. Además, Cooja permite guardar la información y los resultados de las simulaciones realizadas en una estructura XML mediante ficheros de extensión `*.csc`.

²<https://github.com/contiki-ng/contiki-ng>

³<https://github.com/contiki-ng/cooja/>

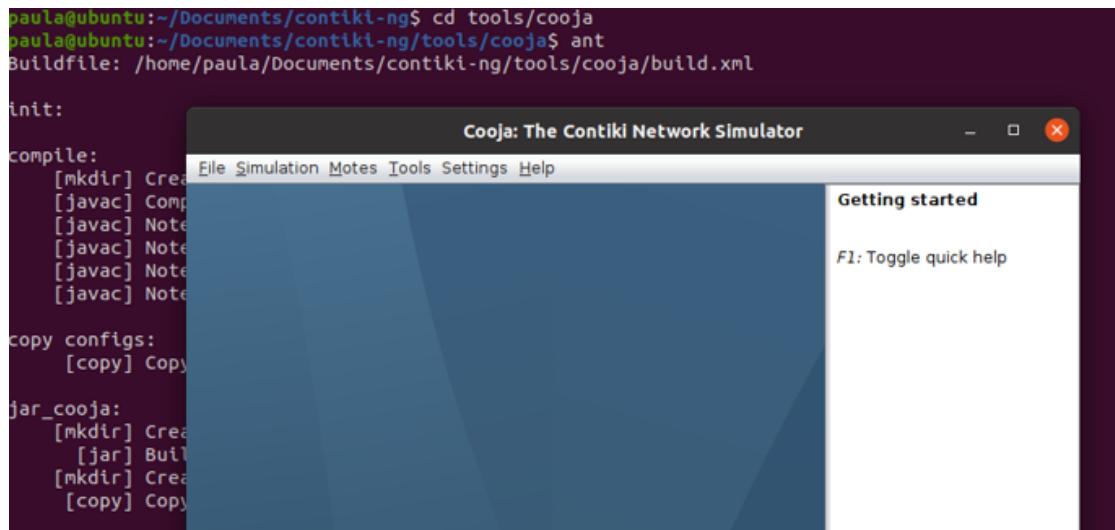


Figura 2.17: Inicio de la herramienta Cooja

2.3.1.2 Ejemplo Hello World

Para tener un primer contacto con el software de Contiki se recomienda probar su funcionamiento con algunos de los ejemplos de proyectos que este incluye en el directorio `examples/`. En este caso, el programa Hello World⁴ comenzará un proceso que imprimirá por consola el mensaje Hello, World cada diez segundos. A continuación, se muestran las ejecuciones en modo nativo o por defecto y en Cooja.

En el modo nativo se ejecutará el programa directamente desde el terminal. Para compilar en este modo se necesita el comando `make hello-world.native TARGET=native`. En la Figura 2.18 se puede observar la secuencia de mensajes obtenida.

```
paula@ubuntu:~/Documents/contiki-ng/examples/hello-world$ ./hello-world.native
[WARN: Tun6      ] Failed to open tun device (you may be lacking permission). Running without network.
[INFO: Main     ] Starting Contiki-NG-release/v4.7-45-g705541797
[INFO: Main     ] - Routing: RPL Lite
[INFO: Main     ] - Net: tun6
[INFO: Main     ] - MAC: nullmac
[INFO: Main     ] - 802.15.4 PANID: 0xabcd
[INFO: Main     ] - 802.15.4 Default channel: 26
[INFO: Main     ] Node ID: 1800
[INFO: Main     ] Link-layer address: 0102.0304.0506.0708
[INFO: Main     ] Tentative link-local IPv6 address: fe80::302:304:506:708
[INFO: Native   ] Added global IPv6 address fd00::302:304:506:708
Hello, world
Hello, world
Hello, world
Hello, world
Hello, world
```

Figura 2.18: Ejecución en modo nativo

Por otra parte, con el uso del simulador gráfico Cooja, la compilación se podrá realizar a través del propio simulador o mediante el comando `make hello-world.cooja TARGET=cooja`. Para este caso, se han definido dos motas que intercambiarán los mensajes de Hello entre sí como se muestra en la ventana de salida capturada en la Figura 2.19.

⁴<https://github.com/contiki-ng/contiki-ng/wiki/Tutorial:-Hello,-World!>

Time	Mote	Message
00:00.236	ID:2	[INFO: Main] Starting Contiki-NG-release/v4.7-45-g705541797-dirty
00:00.236	ID:2	[INFO: Main] - Routing: RPL Lite
00:00.236	ID:2	[INFO: Main] - Net: nicslowpan
00:00.236	ID:2	[INFO: Main] - MAC: CSMA
00:00.236	ID:2	[INFO: Main] - 802.15.4 PANID: 0xabcd
00:00.236	ID:2	[INFO: Main] - 802.15.4 Default channel: 26
00:00.236	ID:2	[INFO: Main] Node ID: 2
00:00.236	ID:2	[INFO: Main] Link-layer address: 0002.0002.0002
00:00.236	ID:2	[INFO: Main] Tentative link-local IPv6 address: fe80::202:2::2
00:00.236	ID:2	Hello, world
00:09.382	ID:1	[INFO: Main] Starting Contiki-NG-release/v4.7-45-g705541797-dirty
00:09.382	ID:1	[INFO: Main] - Routing: RPL Lite
00:09.382	ID:1	[INFO: Main] - Net: nicslowpan
00:09.382	ID:1	[INFO: Main] - MAC: CSMA
00:09.382	ID:1	[INFO: Main] - 802.15.4 PANID: 0xabcd
00:09.382	ID:1	[INFO: Main] - 802.15.4 Default channel: 26
00:09.382	ID:1	[INFO: Main] Node ID: 1
00:09.382	ID:1	[INFO: Main] Link-layer address: 0001.0001.0001
00:09.382	ID:1	[INFO: Main] Tentative link-local IPv6 address: fe80::201:1::1
00:09.382	ID:1	Hello, world
00:10.236	ID:2	Hello, world
00:10.236	ID:1	Hello, world
00:20.236	ID:2	Hello, world
00:20.236	ID:1	Hello, world
00:30.236	ID:2	Hello, world
00:30.236	ID:1	Hello, world

Figura 2.19: Ventana de salida en Cooja

2.3.1.3 Implementación de IoTorii

Con el fin de implementar de una forma gráfica el algoritmo IoTorii y de probar su funcionamiento comentado en el Apartado 2.2.3.2, se hace empleo de Contiki y en particular, de Cooja. Como ejemplo se diseña la topología representada en la Figura 2.20 y que se compone de 6 motas (tipo sky motes) [57]: 1 nodo definido como raíz y 5 nodos comunes.

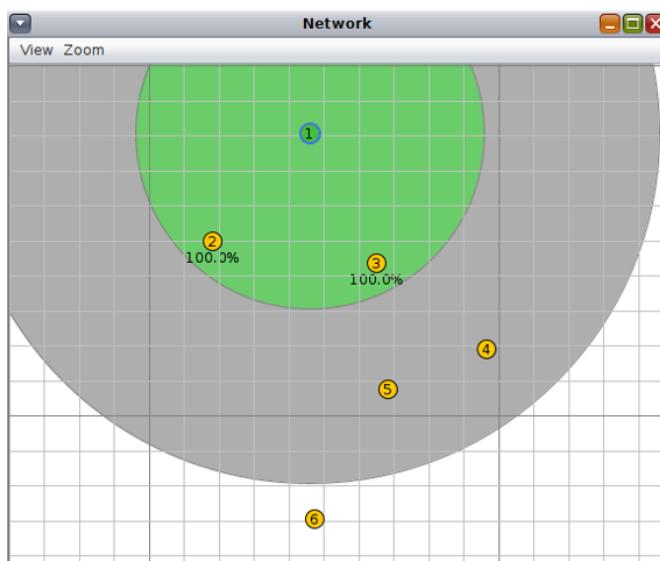


Figura 2.20: Topología de red empleada

De esta forma, ejecutando la simulación el protocolo sigue la siguiente secuencia de pasos:

- Inicialización de los nodos:** El algoritmo define las características del nodo (ID) y del tipo de mota seleccionado (sky mote) [57]. Se determina una dirección MAC en función del protocolo IoTorii CSMA y se informa del canal por defecto. En la Figura 2.21 se muestran los mensajes de inicialización de uno de los nodos comunes (ID:5).

```
00:01.001 ID:5 Scheduling a statistic timer after 2560 ticks in the future
00:01.006 ID:5 [INFO: Main      ] Starting Contiki-NG-release/v4.7-45-g705541797-dirty
00:01.009 ID:5 [INFO: Main      ] - Routing: nullrouting
00:01.012 ID:5 [INFO: Main      ] - Net: nullnet
00:01.014 ID:5 [INFO: Main      ] - MAC: IoTorii CSMA
00:01.018 ID:5 [INFO: Main      ] - 802.15.4 PANID: Oxabcd
00:01.021 ID:5 [INFO: Main      ] - 802.15.4 Default channel: 26
00:01.023 ID:5 [INFO: Main      ] Node ID: 5
00:01.029 ID:5 [INFO: Main      ] Link-layer address: 0505.0500.0574.1200
00:01.032 ID:5 [INFO: Sky       ] CC2420 CCA threshold -45
```

Figura 2.21: Inicialización del nodo con ID 5

- Intercambio de mensajes Hello:** Cada nodo propaga mensajes Hello a los nodos cercanos para poder descubrir quiénes son sus vecinos y cuántos tiene.

```
00:02.664 ID:2 Number of Hello messages: 1
00:03.697 ID:5 Number of Hello messages: 1
00:03.958 ID:1 Number of Hello messages: 1
00:04.085 ID:6 Number of Hello messages: 1
00:04.178 ID:3 Number of Hello messages: 1
00:04.479 ID:4 Number of Hello messages: 1
```

Figura 2.22: Número de mensajes Hello enviados en cada nodo

- Intercambio de mensajes HLMAC:** Se lleva a cabo el procedimiento de asignación de direcciones, comenzando desde el nodo raíz y repitiendo el proceso en cada uno de los nodos hasta el final de la red. En la figura 2.23 se imprime la secuencia de acciones realizadas por el nodo raíz (ID:1) para enviar a sus nodos vecinos (ID:2, ID:3) la dirección HLMAC que les corresponde. Se ha configurado una asignación de un máximo de 3 direcciones para cada nodo.

```
00:10.672 ID:1 Periodic Statistics: HLMAC address: 01. saved to HLMAC table.
00:10.675 ID:1 Periodic Statistics: node_id: 1, convergence_time_start
00:10.679 ID:1 Number of SetHLMAC messages: 1
00:10.732 ID:2 Periodic Statistics: HLMAC address: 01.01. saved to HLMAC table.
00:10.732 ID:3 Periodic Statistics: HLMAC address: 01.02. saved to HLMAC table.
00:10.737 ID:2 Periodic Statistics: node_id: 2, convergence_time_end
00:10.739 ID:3 Periodic Statistics: node_id: 3, convergence_time_end
00:10.803 ID:3 Number of SetHLMAC messages: 1
00:10.804 ID:2 Number of SetHLMAC messages: 1
00:10.825 ID:1 Periodic Statistics: node_id: 1, convergence_time_end
```

Figura 2.23: Procedimiento de asignación de direcciones HLMAC

- Visualización de estadísticas:** Se imprimen en la ventana de salida las estadísticas completas de cada nodo. Como se visualiza en la figura 2.24 se obtiene la información de los parámetros relacionados al número de mensajes de control, los vecinos de cada nodo, las direcciones HLMAC almacenadas y la cuenta de saltos.

```

node_id: 2, number_of_hello_messages: 1, number_of_sethlmac_messages: 2, number_of_neighbours: 2, number_of_hlmac_addresses: 2, sum_hop: 5
node_id: 6, number_of_hello_messages: 1, number_of_sethlmac_messages: 0, number_of_neighbours: 1, number_of_hlmac_addresses: 3, sum_hop: 14
node_id: 4, number_of_hello_messages: 1, number_of_sethlmac_messages: 3, number_of_neighbours: 2, number_of_hlmac_addresses: 3, sum_hop: 11
node_id: 1, number_of_hello_messages: 1, number_of_sethlmac_messages: 1, number_of_neighbours: 2, number_of_hlmac_addresses: 1, sum_hop: 1
node_id: 5, number_of_hello_messages: 1, number_of_sethlmac_messages: 3, number_of_neighbours: 3, number_of_hlmac_addresses: 3, sum_hop: 11
node_id: 3, number_of_hello_messages: 1, number_of_sethlmac_messages: 2, number_of_neighbours: 4, number_of_hlmac_addresses: 2, sum_hop: 5

```

Figura 2.24: Estadísticas de los nodos

Una vez desarrollado el proceso de simulación, se obtiene como resultado una topología de red de árbol, con una jerarquía de nodos basada en las etiquetas asignadas. En la Tabla 2.2 se indica la lista de direcciones HLMAC que han sido almacenadas finalmente en cada uno de los nodos de la red.

Nodo ID	Lista de direcciones HLMAC
01	01
02	01.01, 01.02.01
03	01.02, 01.01.02
04	01.02.04, 01.01.02.04, 01.02.02.03
05	01.02.02, 01.01.02.02, 01.02.04.01
06	01.02.02.01, 01.02.04.01.01, 01.01.02.02.01

Tabla 2.2: Tablas de direcciones HLMAC almacenadas en los nodos

Debido a la capacidad del algoritmo de evitar bucles en la asignación de etiquetas, los nodos más cercanos al nodo raíz recibirán un menor número de direcciones. Esto es también, porque se trata de una topología con pocas motas y los nodos no pueden propagar direcciones HLMAC de sus propios hijos.

2.3.2 BRITE

BRITE⁵ (*Boston University Representative Internet Topology Generator*) se presenta como una herramienta de apoyo para la creación de topologías de red aleatorias. El software fue desarrollado por miembros del Departamento de Ciencias Computacionales de la Universidad de Boston con el fin de establecer un algoritmo flexible que soporte la generación de diversos modelos de topologías. El proceso de diseño y creación de topologías que sigue BRITE se basa en la siguiente secuencia de acciones:

- 1. Definición de la posición de los nodos:** Es posible colocarlos de forma aleatoria o por zonas, concentrando los nodos en diversos puntos del plano.
- 2. Establecimiento de enlaces:** Con el fin de interconectar a los nodos entre sí.
- 3. Introducción de las características:** De la red y de sus dispositivos. Se definen los retardos temporales y el ancho de banda.
- 4. Especificación del formato:** Se establece el formato de salida .brite para la topología creada.

⁵<https://github.com/NETSERV-UAH/BRITE>

La configuración de cada uno de los modelos de red se basa principalmente en los parámetros de entrada que se definen. Estos especificarán propiedades de la red [58] como la posición de los nodos, las características del ancho de banda de los enlaces o el número de conexiones de cada uno de los nodos.

2.3.3 Mininet

Mininet [59] se trata de un software de diseño y despliegue de redes virtuales. Es de código abierto y su desarrollo y actualización se basa en las contribuciones de los propios usuarios en su repositorio⁶. Tiene la capacidad de emular topologías de red completas de nodos o hosts con sus respectivas interconexiones, además de switches y enlaces. Como característica, Mininet es compatible con algunos tipos de controladores de redes definidas por software (del inglés *Software-Defined Networking*, SDN) [60] y, por ello, introduce la posibilidad de transferir los prototipos de red diseñados a una ejecución en hardware. El programa en sí se fundamenta en el empleo de una virtualización basada en procesos y en namespaces [61] (ver Apartado 3.2).

A nivel de software, cada uno de los hosts o nodos que participan en la topología de red diseñada se define como un proceso bash con su propio namespace. En otros términos, cada nodo tendrá su interfaz de red privada y podrá ver sus propios procesos. Se puede decir que este tipo de procesos equivale al código que se podría ejecutar en un servidor Linux. Teniendo en cuenta la limitación de recursos y memoria que puede presentar un equipo, realiza una repartición de los mismos para cada uno de los hosts de la red. Por otro lado, en cuanto a los enlaces, se definirán pares Ethernet virtuales que conecten los switches a los hosts o procesos de forma interna en la red.

2.4 Implementación hardware y tecnología Raspberry Pi

En el Apartado 2.3 se comentaban las herramientas estudiadas para evaluar el desarrollo del algoritmo, objetivo de este TFG, a nivel de software. Sin embargo, en esta sección, se presentará la tecnología que sería necesaria para poder llevar a cabo su implementación hardware.

Cabe destacar que la implementación hardware se trataba de uno de los objetivos marcados inicialmente para este proyecto, pero debido a su grado de dificultad (que ya se anticipaba en el anteproyecto) y la escasez de material se tomó la decisión de descartar este tipo de pruebas. Como se comentará en el Capítulo 5, referente a los resultados del experimento, se hará hincapié en que la comprobación del correcto funcionamiento del algoritmo se basa en la ejecución de simulaciones en diferentes topologías de red.

2.4.1 Raspberry Pi 4

Para transportar las funcionalidades del algoritmo a un escenario real de red de sensores se presenta como mejor elección el uso de tarjetas Raspberry Pi [62], y en particular del modelo Raspberry Pi 4⁷. Este tipo de dispositivos se basan en ordenadores de placa simple o placa reducida, cuyo software es de

⁶<https://github.com/mininet/mininet>

⁷<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>

código abierto. Pueden soportar también, la instalación de otros sistemas operativos, como Windows 10 o diferentes versiones de Ubuntu. El primer modelo de tarjeta producido por The Raspberry Pi Foundation fue lanzado en 2012, con el fin de promover el ámbito de la informática y la programación a nivel educativo [63]. Tanto su bajo coste en tecnología, como su diseño y formato posibilitan el desarrollo de una gran diversidad de proyectos y la accesibilidad a la tecnología a cualquier usuario.

En cuanto a la tarjeta en la que se enfoca este TFG, la Raspberry Pi 4, se trata del modelo más nuevo. Respecto a las versiones anteriores, se optimiza la arquitectura y se introduce un procesador mucho más eficiente, reduciendo su consumo energético. Posibilita una conexión Ethernet con mayor ancho de banda (Gigabit Ethernet) y conexión WiFi y Bluetooth. Además, aumenta su capacidad USB con dos puertos USB2.0 y USB3.0, lo que produce transferencias de datos incluso diez veces mayores [64]. En la Figura 2.25 se muestra la estructura de la tarjeta y se especifican cada una de sus partes, entre las que se encuentran el procesador, la RAM y cada uno de los puertos de entrada y salida.

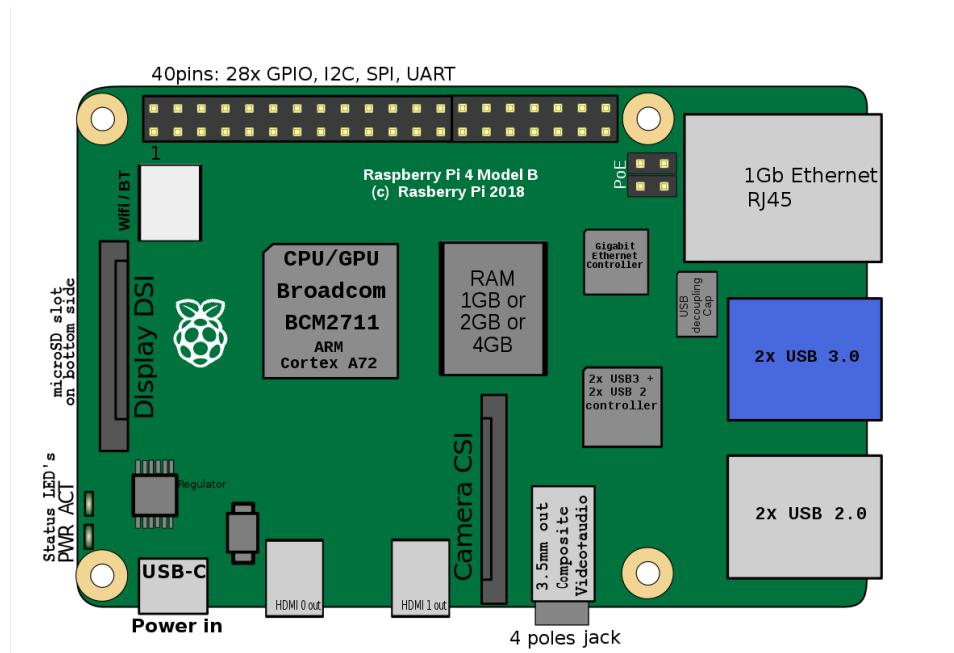


Figura 2.25: Esquemático de Raspberry Pi 4 [17]

Capítulo 3

Diseño y análisis

En este capítulo, se llevará a cabo una fase de pre-implementación anterior a la parte del desarrollo completo (ver Capítulo 4). Se planteará el análisis práctico de las tecnologías explicadas en el Capítulo 2 y, también, se justificará el empleo de cada una de ellas para llevar a cabo la implementación final del algoritmo objetivo del presente TFG. Además, en base al marco teórico y práctico en el que se encuentra el proyecto, se detallará finalmente las herramientas que se emplearán y de qué forma.

3.1 Código base

Principalmente, el origen del diseño de un nuevo algoritmo para redes de sensores se encuentra en el protocolo IoTorii. Esto es, porque se requiere un protocolo que sea eficiente para gestionar la comunicación entre nodos y repartir su carga computacional. Entonces, analizándolo en profundidad respecto a RPL, se prueba que es preferible para que se convierta en la base del algoritmo Dedenne, desarrollado en el presente TFG (ver Capítulo 4). Es importante destacar además, que este es más similar en su naturaleza y definición a IoTorii que a RPL.

Como se ha comentado anteriormente en el Apartado 2.2.3.2, IoTorii es característico por proporcionar escalabilidad en las redes de baja potencia. Lo consigue mediante la asignación de etiquetas a cada uno de los nodos que participan en el esquema de red, creando una topología de árbol. Transportando las funcionalidades de IoTorii al ámbito de las smart grids, el nodo raíz se comportaría como la distribuidora energética y los nodos comunes, como los usuarios.

Entrando en detalle en la creación del nuevo algoritmo, se planteará un modelo distribuido, en el que el nodo raíz no suponga el centro del esquema. En este caso, cada nodo tendrá un conocimiento de la información de la red a nivel local y solo percibirá los datos de sus vecinos a través de las transmisiones de los mensajes de control. Esta característica supondrá la realización de una serie de modificaciones en el proceso de asignación de etiquetas, ya que el nodo raíz tampoco conoce el estado y la posición de los demás nodos (solo la de sus vecinos).

Además, afectará sobre todo a los nodos ubicados en los extremos de la red. Requerirán de vecinos a su alrededor para poder implementar una comunicación multisalto con cualquiera de los demás nodos que existen en la red.

En el caso de IoTorii, los nodos extremos no tienen un tratamiento especial, y es por ello que se presenta la problemática con la recepción de los recursos que reparte el nodo raíz. Se implementará, según se detalla en el Capítulo 4 y como modificación al algoritmo, una clasificación de los nodos comunes, que diferencie entre los nodos que son extremos de red y los que no.

3.1.1 Archivos de configuración y compilación

En concreto, para desarrollar el algoritmo Dedenne, se empleará la versión `iotorii-n-hlmac`¹. Se trata de la versión original y completa del protocolo IoTorii, en la cual se puede configurar el parámetro relacionado al número máximo de direcciones HLMAC que se le puede asignar a cada nodo (`HLMAC_CONF_MAX_HLMAC`).

En esta versión de IoTorii existe un directorio que contiene, por un lado, los ficheros de código con extensiones `.c` y `.h` y otros dos directorios, pertenecientes cada uno de ellos a los dos tipos de nodo que pueden existir en la red (nodo raíz o nodo común). En estos directorios se encuentran los siguientes archivos:

- **Fichero con extensión `.c`:** Declara y define el proceso de activación e inicialización del nodo. Por cada nodo del tipo indicado, se crea un hilo, permitiendo una ejecución concurrente y multihilo para soportar la simulación de la red. Este fichero supone la base del comienzo del funcionamiento del algoritmo.
- **Archivo de configuración de parámetros con extensión `.h`:** Principalmente, se define el valor del número máximo de direcciones HLMAC que puede almacenar cada nodo y los tiempos de delay para cada una de las ejecuciones que se desarrollan en el algoritmo.
- **Archivo `makefile`:** Para compilar todas las partes necesarias (ficheros `.c` y `.h`) para la ejecución de cada hilo y en consecuencia, para la simulación de cualquier escenario de IoTorii. Como se determina en el Código 3.1, se introducen los archivos de código que completan el funcionamiento de cada uno de los módulos y procesos del algoritmo.

Listado 3.1: Introducción en el makefile de los archivos de código para su compilación

```
PROJECT_SOURCEFILES += ../iotorii-csma.c
PROJECT_SOURCEFILES += ../hlmacaddr.c
PROJECT_SOURCEFILES += ../hlmac-table.c
PROJECT_SOURCEFILES += ../csma-security.c
PROJECT_SOURCEFILES += ../csma-output.c
```

¹https://github.com/NETSERV-UAH/IoTorii/tree/master/Contiki-ng_4_2/iotorii-n-hlmac

A partir de todos los ficheros anteriores, se realizarán las modificaciones necesarias para convertir el modelo centralizado de IoTorii en uno distribuido y se implementará en la red la posibilidad de un reparto de cargas computacionales entre nodos. Además, las nuevas funcionalidades del algoritmo a desarrollar serán aplicables en el ámbito de las smart grids como objetivo de este TFG.

3.2 Empleo de Contiki

El sistema operativo Contiki será el programa principal utilizado para simular el funcionamiento del algoritmo Dedenne. Como se comenta en el Apartado 2.3.1 tiene la capacidad de implementar el protocolo RPL y cualquier algoritmo que sea aplicable al ámbito de las redes de baja potencia (LLN) y de dispositivos IoT [56].

Entrando en detalle en el proceso de desarrollo del código, se empleará como herramienta principal su simulador gráfico Cooja. Debido a que en el proceso de implementación, se produce una ejecución multihilo, suponiendo un hilo por nodo, Cooja se vuelve una aplicación imprescindible. Esto es, porque mediante terminal no sería factible realizar ejecuciones de varios nodos de forma concurrente.

No obstante, cabe la posibilidad de convertir la máquina de Linux internamente en varios nodos virtuales. Se podría hacer uso de contenedores mediante *namespaces* [61]. Generalmente, su concepto se aplica al número de identificación de un proceso, pero existen 8 tipos de *namespaces* según el recurso al que se haga referencia.

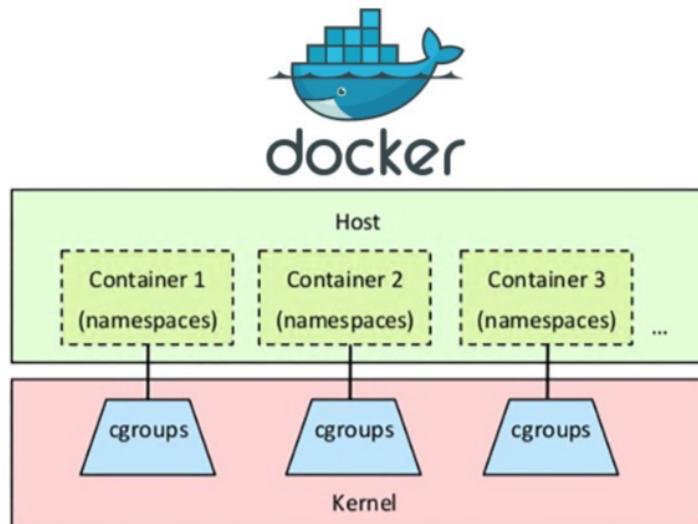


Figura 3.1: Esquema de creación de varios namespaces con Docker [18]

De forma inicial, siempre se comienza con un solo *namespace*, en el que se ubican los procesos que se están ejecutando en el sistema operativo y en el cual todos ellos comparten los recursos de forma global. Por lo tanto, es preciso realizar una serie de particiones del kernel para crear tantas divisiones del total de recursos como nodos existan en la topología a simular. Es decir, se aíslan en entornos independientes para evitar que los hilos que se ejecutan de forma paralela interfieran entre sí [65].

De este modo, el grupo de procesos perteneciente a un *namespace* verán un conjunto de recursos distinto a los procesos externos a él. Para llevar a cabo este procedimiento sería de utilidad la aplicación Docker². En la Figura 3.1 se visualiza el esquema de creación de varios namespaces diferentes empleando esta aplicación.

Otra opción alternativa al uso de contenedores, sería la creación de varias máquinas virtuales interconectadas entre sí. En este caso habría que habilitar una comunicación entre todas ellas y se presentaría un escenario en el que cada máquina actuaría como un nodo de la red. Sin embargo, esta posibilidad se ha desechado rápidamente debido a su ineficiencia y al alto consumo que supondría. Además, no sería posible simular una topología de red a gran escala por el gran coste de memoria.

Finalmente, valorando tanto la opción de empleo de la herramienta Cooja, como la aplicación Docker para la creación de de contenedores, se ha decidido desarrollar la implementación del algoritmo en la primera. Como ventaja, permitirá una mejor comprobación del funcionamiento del código, mediante la visualización detallada de los mensajes de depuración. Esta funcionalidad será indispensable sobre todo en el caso de diseñar topologías de red con multitud de dispositivos IoT.

Además, es importante seleccionar Contiki como software si se procede a realizar una implementación del algoritmo en un escenario real de red de sensores o dispositivos IoT. Esto es, porque se trata de una aplicación planteada principalmente, para cargarse en tarjetas Raspberry Pi (ver Apartado 2.4.1) en forma de procesos de usuario. Más adelante, en la Sección 6.2, se definirá en detalle el procedimiento de implementación a nivel hardware como trabajo futuro de este proyecto.

3.3 Herramientas de despliegue de redes

Como se ha expuesto en los Apartados 2.3.2 y 2.3.3 y haciendo referencia al marco teórico en el que se encuentra este trabajo, se ha procedido al estudio de BRITE y Mininet como herramientas de generación y despliegue de redes de dispositivos IoT. Ha sido de utilidad comprender el funcionamiento de ambas para poder probar y depurar el algoritmo en el proceso de desarrollo. Además, una vez finalizado su código se ha conseguido validar este de forma completa mediante la simulación en múltiples topologías de red.

En particular, para ello, ha sido imprescindible el software Mininet. Se han empleado varios esquemas diseñados en el mismo y se han trasladado a Contiki para realizar la implementación de forma gráfica con ayuda de Cooja. Como resultado de salida de Mininet, se dispone de un fichero con extensión .txt para cada topología generada. En el mismo, se indica la posición de cada uno de los nodos mediante sus coordenadas en formato [y,x,z].

Como paso previo a la implementación, se requiere el ajuste de las áreas de cobertura y, por tanto, de estos valores numéricos. Es decir, los rangos aplicados en Mininet no corresponden de la misma forma que en Contiki y de forma general, los nodos abarcarán una mayor área de la que requerida.

²<https://www.docker.com/>

Como se expondrá en el Capítulo 5, referente a los resultados obtenidos tras la etapa de desarrollo, será necesario aumentar de forma proporcional las distancias entre nodos para poder simular de forma correcta el algoritmo en cada una de las topologías. En la Figura 3.2 se representa de forma gráfica un ejemplo de esquema de red que se podría emplear como base para ejecutar las simulaciones en Contiki.

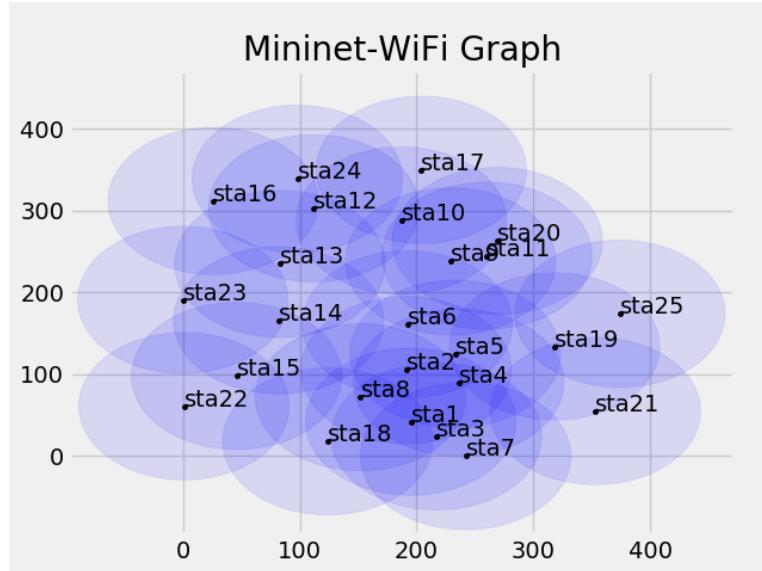


Figura 3.2: Ejemplo de topología diseñada en Mininet

Capítulo 4

Desarrollo

4.1 Introducción

El propósito de este capítulo es realizar una descripción detallada del algoritmo Dedenne, desarrollado para cumplir el principal objetivo de este TFG. Como se expresa en la introducción (ver Capítulo 1) se planteará un nuevo protocolo eficiente en el ámbito de las redes de baja potencia y de los dispositivos IoT con la base del algoritmo IoTorii ya implementado. Más concretamente, se procederá a modificar su modelo de red, de un esquema centralizado a uno distribuido. Además, se presentarán nuevas funcionalidades relacionadas con el proceso de asignación de las etiquetas jerárquicas y la clasificación de nodos en el esquema de red.

No obstante, el fin fundamental del algoritmo Dedenne es capacitar a los nodos de una forma inteligente y automática de elaborar un reparto de carga computacional entre los mismos. El desarrollo de esta nueva función será detallada en las siguientes páginas y se demostrará la eficiencia y escalabilidad de su procedimiento.

El capítulo se estructura en varias secciones. En primer lugar, se encuentra un apartado en el que se describen de forma cronológica las fases del planteamiento del algoritmo y cómo se ha producido el desarrollo. Además, se exponen las decisiones que se han decidido tomar con el progreso del código, siguiendo la evolución temporal del mismo. Esta primera sección presenta, por un lado, una primera parte de estudio previa a la implementación. En ella, se expondrán las herramientas que se van a emplear y las funciones que tendrán cada una de ellas. Por otro lado, también se desarrolla una parte de planteamiento, en la que se describirán de forma detallada las modificaciones principales que se han aplicado en la programación del modelo del algoritmo.

Volviendo a las secciones del capítulo, en segundo lugar, se encontrará el apartado de implementación. En él, se definirá en profundidad las funcionalidades implementadas en el algoritmo Dedenne. Para ello, se divide su descripción en los diferentes módulos conceptuales que lo componen y se representan los puntos más importantes en forma de código y de algoritmos.

Finalmente, se introduce una sección referente a los tiempos de ejecución, en la que se justifican los retardos impuestos para cada uno de los procesos que intervienen en el algoritmo.

4.2 Cronología del desarrollo

Como se mencionó previamente, se expondrá a continuación una descripción de forma cronológica de las diferentes partes en las que se ha dividido el desarrollo del algoritmo.

4.2.1 Preimplementación y estudio del software

Para comenzar con la implementación del nuevo algoritmo se precisó en primera instancia de la creación del entorno de simulación. Se tuvo que tomar la decisión de emplear el sistema operativo para redes de baja potencia Contiki como herramienta principal (ver Apartado 3.2). Como se expondrá en el Apéndice A, correspondiente al manual de usuario, se procedió primero a su instalación en una máquina virtual de Linux. Se comentará además, que fue necesario resolver algunas incidencias relacionadas con el procedimiento de compilación en Cooja (ver Apartado A.2.2.1).

Después de esta parte, se estableció un primer contacto con el software mediante el empleo de algunos de los programas de ejemplo que aporta Contiki en su repositorio. En el Apartado 2.3.1.2 se muestra una de las simulaciones realizadas y los resultados obtenidos en la misma.

Continuando con la fase de estudio de las herramientas, se analizó el funcionamiento de Brite y Mininet como programas de diseño y despliegue de topologías de red. En particular, al finalizar el desarrollo completo del algoritmo, se realizaron las respectivas simulaciones en diferentes topologías de red diseñadas en Mininet. Como comprobación del funcionamiento del algoritmo, se presentarán los resultados de estas en el Capítulo 5.

4.2.2 Planteamiento del nuevo modelo

Las características del algoritmo Dedenne que se pretendía implementar produjeron la necesidad de empleo del protocolo de encaminamiento IoTorii como base. Este último, ya desarrollado anteriormente, presentaba un modelo centralizado que se tuvo que modificar para cumplir con las nuevas especificaciones. Por ello, el primer paso de esta fase de planteamiento fue el estudio de las diferentes posibilidades que existían para realizar el cambio de esquema.

Es preciso recordar que en el protocolo IoTorii existían dos tipos de nodo: nodo raíz y nodos comunes. El primero de ellos se determinaba como un nodo "maestro" porque potencialmente podía poseer el total conocimiento de los demás nodos. Además, tenía la responsabilidad de establecer la jerarquía dentro de la topología de red. En otros términos, se puede decir que era el encargado de comenzar el proceso de asignación de etiquetas, informando a sus vecinos de la propia más un sufijo añadido. En cuanto a este proceso, cabe destacar que cada nodo tenía la capacidad de obtener múltiples etiquetas por parte de sus vecinos.

4.2.2.1 Asignación de una única etiqueta

Para el diseño de Dedenne, se tomó la decisión de que cada nodo tuviera únicamente una dirección HLMAC almacenada internamente. Esto es, porque se estimó que, para el proceso de reparto de cargas computacionales que se iba a desarrollar posteriormente, solo se iba a precisar de una dirección para su

funcionamiento. Así, se conseguiría aumentar la eficiencia del algoritmo, reduciendo de forma considerable el uso de memoria y eliminando retardos innecesarios en la asignación de etiquetas. Entonces, por definición, se puede expresar que la etiqueta que almacenaría el nodo sería la primera que recibiera¹. El valor máximo de direcciones que se podía guardar en la tabla HLMAC lo imponía la variable definida en los archivos de configuración pertenecientes al nodo root y a los nodos comunes:

```
|| #define HLMAC_CONF_MAX_HLMAC 1
```

A modo de ejemplo de esta aplicación, se probó el proceso de asignación de etiquetas, empleando la topología de red de la Figura 4.1.

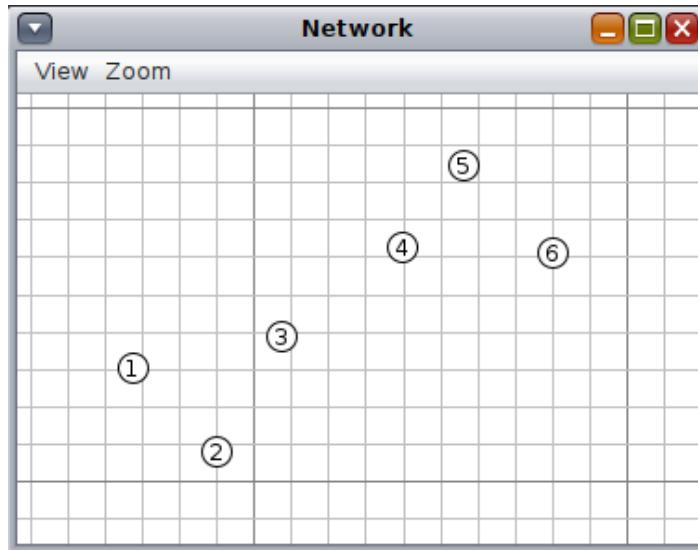


Figura 4.1: Ejemplo 1 de topología de red (a)

Tras desplegarla en Cooja, se procedió a ejecutar la simulación del proceso de asignación de etiquetas. Como era de esperar, en la ventana de resultados (ver Figura 4.2) se pudo observar que cada nodo, aunque recibió varias direcciones por parte de sus vecinos, solo almacenó la primera de ellas. Por ejemplo, en el caso del nodo root, este envió las etiquetas 01.01. y 01.02. a sus vecinos con ID 2 y 3 respectivamente. Al no haber recibido anteriormente otras direcciones, se quedó con estas mismas. Siguiendo con el proceso de propagación, como estos nodos también eran vecinos entre ellos, se comunicaron entre sí posteriormente nuevas etiquetas:

- El nodo 2 con etiqueta 01.01. generó y envió a 01.01.02. al nodo 3. Se recibió 01.02.01. por parte del nodo 3, pero no se almacenó.
- El nodo 3 con etiqueta 01.02. generó y envió 01.02.01. al nodo 2. Se recibió 01.01.02. por parte del nodo 2, pero no se almacenó.

¹Alternativamente, se podría haber guardado solo la HLMAC más corta (con menor número de saltos hasta el nodo raíz), que no necesariamente tiene por qué ser la más rápida en recibirse, o siguiendo otros criterios. Este análisis del mejor criterio para la selección de la HLMAC se considera ortogonal al desarrollo del TFG y, por tanto, por simplicidad, se considera solo la HLMAC más rápida en ser recibida.

```

00:05.671 ID:1 SetHLMAC prefix (addr:01.) added to queue to advertise to 2 nodes.
00:05.699 ID:2 //INFO INCOMING HLMAC// HLMAC recibida: 01.01.
00:05.699 ID:3 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.
00:05.714 ID:2 SetHLMAC prefix (addr:01.01.) added to queue to advertise to 1 nodes.
00:05.715 ID:3 SetHLMAC prefix (addr:01.02.) added to queue to advertise to 2 nodes.
00:05.733 ID:3 //INFO INCOMING HLMAC// HLMAC recibida: 01.01.02.
00:05.743 ID:4 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.03.
00:05.743 ID:2 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.01.
00:05.760 ID:4 SetHLMAC prefix (addr:01.02.03.) added to queue to advertise to 2 nodes.
00:05.783 ID:5 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.03.01.
00:05.784 ID:6 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.03.02.
00:05.802 ID:6 SetHLMAC prefix (addr:01.02.03.02.) added to queue to advertise to 1 nodes.
00:05.802 ID:5 SetHLMAC prefix (addr:01.02.03.01.) added to queue to advertise to 1 nodes.
00:05.814 ID:5 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.03.02.01.
00:05.830 ID:6 //INFO INCOMING HLMAC// HLMAC recibida: 01.02.03.01.01.

```

Figura 4.2: Procedimiento de asignación de etiquetas a los nodos

Por lo tanto, tras realizar el estudio de los resultados en la Figura 4.2 y siguiendo el proceso, se podía exponer de forma clara que las direcciones restantes asignadas a los demás nodos de la red derivarían todas de la etiqueta 01.02., almacenada en el nodo 3.

4.2.2.2 Creación de la estructura de datos del nodo

Respecto a IoTorii, fue necesario introducir como novedad la definición de una nueva estructura que contuviera información respectiva a cada uno de los nodos de la red. Estos datos serían de utilidad tanto para facilitar la depuración del algoritmo como para llevar a cabo el reparto de las cargas de forma correcta. En relación a este último proceso, es importante comentar que los elementos de la estructura definirían cómo se produciría su comienzo. Más adelante, se expondrán las diferentes opciones que se plantearon para iniciar los traspasos de cargas computacionales en el Apartado 4.2.2.4.

Listado 4.1: Definición de la estructura del nodo `this_node`

```

struct this_node
{
    struct this_node *next;
    char* str_addr;
    char* str_top_addr;
    uint8_t payload;
    uint8_t nhops;
    int load;
};

```

Entrando en una descripción detallada de los elementos definidos en la estructura de datos:

- **Dirección HLMAC del nodo (`str_addr`):** Variable de tipo string que determinaría, mediante una conversión del campo perteneciente a la estructura de direcciones², la etiqueta que habría sido asignada al nodo.
- **Dirección HLMAC del nodo padre (`str_top_addr`):** Como información adicional para la depuración, almacenaría la etiqueta del padre. Esta se podría obtener mediante la copia de la que había sido asignada al propio nodo, sin incluir el último sufijo añadido (último campo ID).

²Sería necesario previamente el empleo de una función que devolviera el campo address como un string.

- **Longitud de la carga útil (*payload*):** Almacenamiento del valor de payload, que se relacionaría directamente con la longitud de la etiqueta asignada al nodo.
- **Número de saltos (*nhops*):** Cuenta de saltos hacia el nodo raíz calculada a partir de la longitud de la etiqueta.
- **Carga computacional (*load*):** Se inicializaría con un valor aleatorio de carga computacional en un rango configurable³.

Respecto a los elementos anteriores, fue importante que en cada nodo de la red se definieran sus valores previamente al comienzo del proceso de envío de los paquetes HLMAC. En otros términos, se informaría de la carga computacional que se había configurado en el nodo en cuestión para que los vecinos la pudieran añadir a su tabla interna.

4.2.2.3 Clasificación de nodos para el reparto de cargas

Una vez se impuso la especificación de almacenar una sola HLMAC en cada uno de los nodos, ya se podía definir la topología de red. Es preciso exponer, que en cuanto al proceso de asignación de etiquetas, se iba a generar una especie de árbol o grafo a partir de las interconexiones que se habían producido. En otros términos, se establecerían múltiples ramificaciones nodo a nodo hasta llegar al final del esquema de red. Es por ello, que en este proceso cada uno de los nodos podía adquirir dos tipos de papeles posibles según la relación que presentara con cada uno de los vecinos de su lista:

- **Padre:** Como mínimo, alguno de sus vecinos había almacenado la etiqueta que había generado para el mismo. En este caso, se establecería una nueva rama del árbol y se indicaría que este vecino es hijo directo.
- **Hijo:** Cualquier nodo sería siempre hijo de un vecino como mínimo. Exceptuando el nodo raíz, los demás nodos dependerían de la asignación de su etiqueta por parte de un nodo padre.

Como consecuencia a la decisión que se tomó sobre el almacenamiento de una sola etiqueta (ver Apartado 4.2.2.1), se probó que se establecería un único parente para cada nodo y por tanto, una única ruta. En cuanto a la topología de red, esta se dibujaría de forma simple a través del conjunto de asignaciones de etiquetas. Continuando con el ejemplo que se expuso en el apartado anterior, se puede examinar en la Figura 4.3 el árbol que se creó tras la asignación de etiquetas.

En rojo, se muestran las rutas que se establecieron entre los nodos a través de la relación parente-hijo y en azul, las relaciones simples que se crearon entre vecinos. Es decir, en el primer caso el nodo que actuaba como hijo almacenó la etiqueta recibida, presentándose como parente al nodo emisor de la misma y produciendo un camino para futuros flujos de información. Por otro lado, en el segundo caso, se recibió la etiqueta en el nodo, pero no se guardó en la tabla HLMAC porque ya se disponía de otra que había llegado antes. Por ejemplo, esta situación ocurrió entre los nodos con ID 2 y 3 (ver Apartado 4.2.2.1).

³Por defecto se estableció el rango [0, 200]

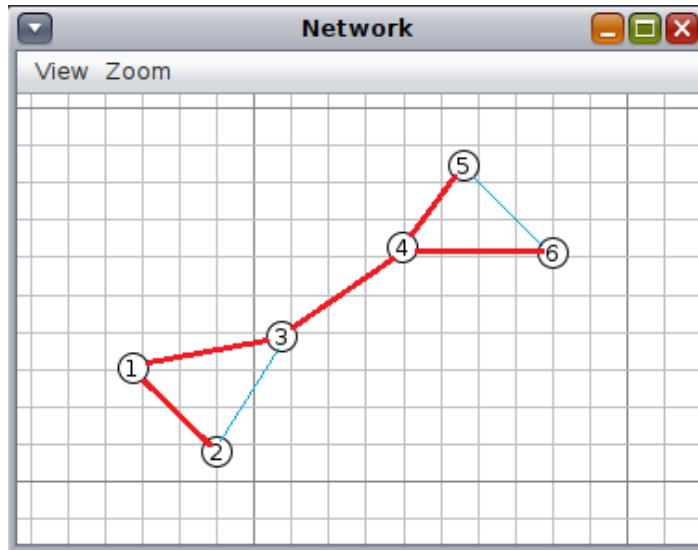


Figura 4.3: Ejemplo 1 de topología de red (b)

Tras analizar los diferentes papeles que podían poseer los nodos se volvió imprescindible implementar una clasificación de los mismos para posibilitar la implementación de las nuevas funcionalidades al algoritmo. Principalmente, se tuvo que tener en cuenta el modelo distribuido de red que este presentaba y la particularidad de que todos los nodos poseían un conocimiento a nivel local de la red. Como su fin principal era establecer un reparto eficiente de cargas computacionales entre todos ellos, se tuvo que buscar una manera de iniciar este proceso a partir de las especificaciones comentadas.

En consecuencia, se propuso la idea de comenzar los traspasos desde los nodos más lejanos al nodo raíz, debido a que aquellos que se encontraban más aislados precisarían de un reparto más urgente de su carga. Además, se iba a requerir generalmente una comunicación con el nodo raíz⁴ con un mayor número de saltos que el resto de los nodos. Es por ello, que tuvo que entrar en juego el papel de nodo frontera o nodo edge.

4.2.2.4 Identificación de nodos frontera

Teniendo en cuenta que la topología de red presentaba un esquema en forma de árbol, se definió como frontera aquel nodo que suponía un fin de ramificación. En otras palabras, el mismo no iba a contar con hijos a su cargo y se conectaría al resto de la red por medio de un nodo padre. Con la definición de esta característica y, volviendo al ejemplo de topología de red mostrado en la Figura 4.3, se pudo determinar que los nodos con ID 2, 5 y 6 son frontera.

Después, tras tomar la decisión de iniciar el reparto de cargas desde los nodos frontera, surgió la necesidad de plantear las diferentes opciones que se podían llevar a cabo para la implementación de esta idea en el algoritmo. Cada nodo debía de tener la capacidad de conocer si era frontera para, en caso afirmativo, iniciar el traspaso de la carga a su padre.

⁴Cabe mencionar que el nodo raíz normalmente sería un nodo que actuaría como punto de acceso a otra red más potente, por lo que, de partida, era un nodo capaz de igualar cargas siempre que fuera necesario

4.2.2.4.1 Método 1: identificación mediante la longitud de las etiquetas

En primera instancia, se desarrolló la idea de relacionar el concepto de nodo frontera con el número de saltos y el tamaño de la carga útil (payload) de los mensajes HLMAC. Como se había expuesto ya en el Apartado 4.2.2.2, los valores de los elementos payload y nhops de la estructura estarían directamente relacionados con la longitud de las etiquetas. Por lo tanto, se determinó que cuanto mayor fuera esta, mayor sería la posibilidad de que el nodo al que le correspondía la misma fuera un nodo frontera.

Sin embargo, se probó que era falsa la afirmación de que un nodo por haber almacenado una etiqueta larga aumentaría su probabilidad de ser frontera. Es decir, en cualquier topología de red, los nodos frontera podrían tener tanto una posición cercana como lejana al nodo raíz. En consecuencia, aumentó la creencia de que la dirección HLMAC asignada un nodo determinado sería independiente al hecho de suponer un final de ramificación en el esquema de red.

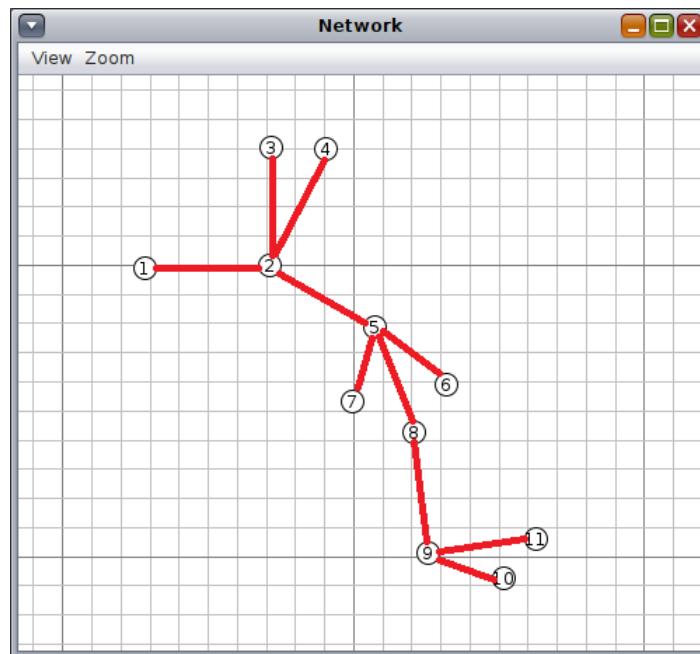


Figura 4.4: Ejemplo 2 de topología de red

Con el ejemplo de topología de la Figura 4.4 se pudo justificar gráficamente la posibilidad de existencia de nodos frontera a distintas distancias del nodo raíz. En este caso, se habían clasificado como nodos frontera los nodos con ID 3, 4, 6, 7, 10 y 11. Se observó que para los dos últimos sí funcionó la idea expuesta, pero los demás no se pudieron determinar como extremo de red. Tras realizar esta comprobación, se comenzaron a valorar otras opciones de programación.

4.2.2.4.2 Método 2: identificación mediante comparación de listas

Como segunda idea a implementar, se planteó la creación de dos listas de elementos. Una de ellas contendría las etiquetas de todos los nodos y la otra, solo las pertenecientes a los que eran padres. Es decir, en primer lugar, se almacenarían respectivamente los valores de los elementos str_addr y str_top_addr de la estructura para cada uno de los nodos de la topología (ver Apartado 4.2.2.2).

Por consiguiente, se realizaría mediante un doble bucle una comprobación de los elementos que contenían ambas listas con el objetivo de determinar qué etiquetas pertenecían a nodos frontera. Denominando a la lista con el total de direcciones de la red como lista A y a la que contendría solo las de los nodos padres, como lista B, se podría afirmar de forma clara que las etiquetas que se encontraran en A y no en B serían nodos frontera. Al contrario, aquellas que estuvieran contenidas tanto en A como en B, pertenecerían a nodos con al menos un hijo. En este caso, por descarte, no serían nodos frontera. En la Figura 4.5 se representa de forma gráfica el planteamiento descrito. El área pintada de blanco haría referencia a las direcciones de los nodos frontera.

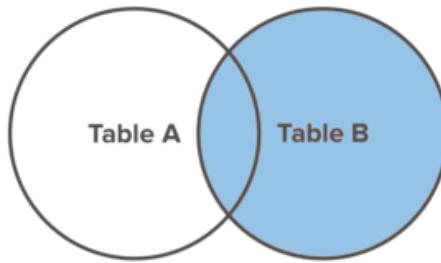


Figura 4.5: Idea de clasificación de nodos mediante dos listas [19]

El razonamiento de esta idea se basó en la aplicación de la característica principal que poseía un nodo frontera. El procedimiento comprobaría de forma directa qué nodos no tenían hijos y en consecuencia de ello, sería posible confirmar que se encontraban al final de una ramificación de la red. No obstante, el esquema de diseño de Dedenne, con un modelo distribuido de red, desechó instantáneamente este planteamiento. Es decir, se analizó que era prácticamente imposible implementar una programación por listas en un algoritmo en el que ninguno de los nodos tenía una visión global de la red.

Además, la particularidad de realizar una ejecución multihilo, con un hilo por nodo, provocó que no se pudiera hacer uso de estructuras globales o que contuvieran los datos del conjunto de todos los nodos de la topología. Como se exponía en la Sección 3.2 de la parte de diseño, cada hilo supondría un proceso con sus propios recursos. Entonces, un nodo determinado tendría conocimiento únicamente de su propia información y de la que hubiera recibido por parte de sus vecinos.

4.2.2.4.3 Método 3: identificación mediante el uso de flags locales

Finalmente, después de que se rechazaran los planteamientos iniciales, se concluyó con una tercera idea para conseguir identificar a los nodos frontera dentro de la red. Como concepto, esta consistía en llevar un control de las etiquetas que iría recibiendo cada nodo por parte de sus vecinos. En este planteamiento fue necesario tener en cuenta que aunque en cada nodo solo se pudiera almacenar una dirección⁵, se recibirían múltiples más que se irían descartando. Entonces, para llevar a cabo la monitorización de los mensajes HLMAC que se recibían en un nodo, era preciso contabilizar cuáles de sus vecinos no se lo habían enviado. Esto se convirtió en un claro indicador, por definición, de que dichos vecinos eran hijos del nodo en cuestión.

⁵Según la configuración impuesta para nuestra implementación (ver Sección 4.2.2.1)

Por otro lado, es preciso comentar que también entró la condición que implementaba IoTorii para evitar bucles en las asignaciones de etiquetas. En otros términos, se debieron realizar diversas comprobaciones en la programación del algoritmo para que un nodo nunca llegara a procesar un mensaje HLMAC emitido por su propio hijo. Por lo tanto, ahora al concluir el proceso de asignación de etiquetas, se podía verificar de forma clara si cada nodo había recibido un número de mensajes igual a la cantidad de vecinos que tenía. En caso afirmativo, se indicaría que se trataba de un nodo frontera, ya que se confirmaba que ninguno de sus vecinos era hijo suyo. Por el contrario, si existían vecinos restantes se podía determinar por descarte que no suponía un extremo de la red.

Cabe mencionar una vez más la independencia de este proceso al hecho de que se hubiera impuesto anteriormente como especificación el almacenamiento de una única etiqueta en cada nodo. Es decir, el procesamiento de los paquetes seguiría produciéndose para todos los siguientes, indistintamente de que se almacene la dirección que porten los mismos. Este aspecto se podía visualizar de forma gráfica en la Figura 4.2, mostrada previamente. En ella se imprimían todas las direcciones recibidas del conjunto de vecinos y las notificaciones de aquellas que realmente sí se almacenaban en cada nodo.

Para trasladar esta idea al código, se hizo uso de la estructura de información de los vecinos, ya definida en IoTorii, y se decidió realizar algunas modificaciones sobre la misma. Como se ha comentado, se pretendía realizar a nivel local una monitorización de las características que presentaban los vecinos de un nodo determinado y para poder llevarla a cabo, era imprescindible añadir nuevos elementos en la estructura (ver Listado 4.2).

Listado 4.2: Definición de la estructura de vecinos neighbour_table_entry

```
struct neighbour_table_entry
{
    struct neighbour_table_entry *next;
    linkaddr_t addr;
    uint8_t number_id;
    int flag;
    uint8_t load;
    int in_out;
};
```

A continuación, se describen en profundidad las funciones principales de cada uno de los elementos:

- **Puntero al siguiente vecino (**next*):** Necesario para la creación de una lista enlazada con todos los vecinos.
- **Dirección física del vecino (*addr*):** Se emplearía para identificar, entre todos los vecinos que tendría un nodo, al emisor de cada mensaje HLMAC.
- **Número de identificación (*number_id*):** Solo se utilizaría a nivel local en la lista de vecinos como un elemento secundario. Ya había sido definido en IoTorii y no tenía relación alguna con el ID global que poseía cada nodo para identificarse en el esquema completo de la red.

- **Marca o distintivo del papel del vecino (*flag*):** Su valor indica el comportamiento que presenta el vecino respecto a un nodo determinado.
 - **Valor 0:** El nodo no ha recibido mensaje HLMAC del vecino y entonces, este se trata claramente de un hijo.
 - **Valor 1:** El nodo ha recibido mensaje HLMAC del vecino y ha almacenado la dirección que le ha asignado. El vecino, por lo tanto, es su padre.
 - **Valor -1:** El nodo ha recibido mensaje HLMAC del vecino, pero no ha almacenado la dirección porque ya tiene una asignada. El vecino no es hijo y se puede decir que es un parente indirecto.
- **Carga computacional (*load*):** Al igual que en la estructura de información del nodo, se tuvo que añadir un campo que almacenara cada una de las carga computacionales que tenían los vecinos de un nodo determinado. Su valor indicaría la carga inicial informada previa a la fase del reparto.
- **Flujo de carga (*in_out*):** Su valor y su signo (positivo o negativo) indicaba la dirección y el sentido del flujo que se formaba entre dos nodos cuando se producían traspasos de carga computacional. Se convirtió en uno de los elementos más importantes, ya que se podrían optimizar los repartos en el esquema de red a partir del análisis y almacenamiento de su valor en varios instantes temporales.

La información de los vecinos referente a la carga computacional se podría obtener mediante la recepción de paquetes de datos previos al proceso de reparto de cargas (ver Sección 4.3.1.3). Por otro lado, el sentido y dirección de los flujos de datos que se originarían entre pares de nodos serían almacenados en cada uno de los traspasos que se produjeran. En cuanto al flag que determinaba la función de cada uno de los vecinos, se indicaría su valor en la fase de procesamiento de los mensajes HLMAC recibidos. Como se ha expuesto, esta se convirtió en la fase primordial para poder diferenciar entre los nodos que eran frontera y los que no. En el Algoritmo 4.1 se representan las comprobaciones que se programaron y el comportamiento que se impuso al protocolo ante la llegada de los mensajes HLMAC.

Empleando ahora este método en el ejemplo de topología que se había representado anteriormente en la Figura 4.4 se consiguió evaluar de forma correcta el procedimiento de identificación de los nodos frontera que existían en la red. Con el objetivo de facilitar la depuración y la comprensión del proceso, se programaron las notificaciones para imprimir en la ventana de salida de Cooja los resultados.

Mote output		
Time	Mote	Message
00:08.478	ID:11	//INFO STATISTICS// El nodo 01.01.01.03.02.03. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.538	ID:6	//INFO STATISTICS// El nodo 01.01.01.04. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.635	ID:4	//INFO STATISTICS// El nodo 01.01.04. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.689	ID:7	//INFO STATISTICS// El nodo 01.01.01.02. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.844	ID:10	//INFO STATISTICS// El nodo 01.01.01.03.02.02. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:09.186	ID:3	//INFO STATISTICS// El nodo 01.01.03. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge

Figura 4.6: Mensajes de notificación de nodos frontera de la red

En las Figuras 4.6 y 4.7 se expone para cada uno de los nodos su cantidad total de vecinos y la comparación de este valor con la cuenta que se había llevado a cabo de todos aquellos que al final del proceso no habían enviado una etiqueta. Mediante la herramienta de filtro de mensajes que aporta Cooja, en la primera (Figura 4.6) se muestran los nodos identificados como frontera, mientras que en la segunda (Figura 4.7), los que no los son.

Mote output		
Time	Mote	Message
00:08.360	ID:8	//INFO STATISTICS// El nodo 01.01.01.03. tiene 4 vecinos y no ha recibido HLMAC de 1 vecinos: no es edge
00:08.523	ID:2	//INFO STATISTICS// El nodo 01.01. tiene 4 vecinos y no ha recibido HLMAC de 3 vecinos: no es edge
00:08.669	ID:1	//INFO STATISTICS// El nodo 01. tiene 1 vecinos y no ha recibido HLMAC de 1 vecinos: no es edge
00:08.996	ID:9	//INFO STATISTICS// El nodo 01.01.01.03.02. tiene 3 vecinos y no ha recibido HLMAC de 2 vecinos: no es edge
00:09.002	ID:5	//INFO STATISTICS// El nodo 01.01.01. tiene 4 vecinos y no ha recibido HLMAC de 3 vecinos: no es edge

Figura 4.7: Mensajes de notificación de nodos no frontera de la red

4.3 Implementación completa por módulos

El algoritmo Dedenne se decidió dividir en una serie de módulos conceptuales, que unificados, lograron implementar la funcionalidad del reparto de cargas computacionales de forma eficiente. En esta sección, se exponen en profundidad y de forma secuencial cada uno de ellos.

4.3.1 Inicialización de los nodos

En cuanto al procedimiento de inicialización, fue necesario separar el mismo en diferentes fases. Es importante comentar que en el algoritmo cada nodo supondría un hilo y por lo tanto, un proceso distinto e independiente. Por ello, de primeras, solo tendría sus propios recursos y después, toda la información que obtendría sería a partir de las comunicaciones que se produjeran con los demás nodos de la red. La inicialización sigue la siguiente secuencia de acciones:

- Inicialización de los recursos a nivel físico y de enlace, siguiendo las especificaciones del protocolo CSMA [46] (ver Sección 2.2.2).
- Activación del módulo de radio para posibilitar la recepción de paquetes.
- Temporización⁶ y definición de varios contadores en paralelo para iniciar los procesos:
 - Transmisión de mensajes Hello para buscar nodos vecinos en el área que abarca el nodo.
 - Transmisión del primer mensaje HLMAC⁷ para comenzar el proceso de asignación de etiquetas.
 - Visualización de las primeras estadísticas.

4.3.1.1 Mensajes Hello

Para el estudio de la transmisión de los mensajes Hello, se tuvo que diferenciar entre dos acciones posibles dentro de este proceso: envío y recepción de mensajes Hello. Por un lado, en referencia al envío, se programó una función que manejara la propagación de los paquetes hacia aquellos nodos ubicados dentro del área que abarcaba el nodo emisor. Además, cada uno llevaría la cuenta de los mensajes enviados, siendo este número igual a la cantidad de vecinos que tuviera.

Por otro lado, en cuanto a la recepción de mensajes, fue imprescindible realizar primero la comprobación de que el paquete recibido tenía longitud nula. Esto es, porque posteriormente para la llegada de los mensajes que portaban información (ej. asignación de dirección HLMAC) sería necesario plantear un procesamiento distinto de los mismos.

⁶La temporización impuesta en estos casos es mayor a la duración de la fase de inicialización para no interferir con ella (ver más en la Sección 4.4).

⁷Se temporiza el envío solo desde el nodo raíz.

Una vez se determinara que el paquete recibido estaba vacío, se procedería a leer el buffer para identificar a su emisor. El nodo verificaría que su tabla de vecinos no había superado el máximo de entradas disponibles (256) y que el vecino que había mandado el mensaje Hello no se encontraba ya registrado en la misma. En caso negativo de ambas condiciones, se reservaría espacio en la lista y se inicializaría la estructura de los datos del vecino. Cada vez que se registrara una nueva entrada se incrementaría la cuenta total de los vecinos.

4.3.1.2 Mensajes HLMAC y proceso de asignación de etiquetas

Cuando ya se hubiera ejecutado la fase de inicialización completamente, los nodos habrían adquirido el conocimiento del total de sus vecinos y podrían llevar a cabo la propagación de los mensajes de asignación de etiquetas. Dedenne, al igual que IoTorii, comenzaría este proceso desde el nodo raíz. En consecuencia a esta característica, se tuvieron que programar dos versiones del manejador⁸ correspondiente al envío de los mensajes HLMAC: una para el nodo raíz y otra para los demás nodos, denominados como nodos comunes.

En el primer caso, solo se ejecutaría una vez. Es decir, al inicio del proceso el nodo raíz de la red se establecería como nodo raíz y realizaría el envío de etiquetas a sus vecinos directos. De forma similar al algoritmo IoTorii, para transmitir los mensajes HLMAC se debería llevar a cabo un procedimiento de empaquetado con el fin de incluir en la carga útil la etiqueta correspondiente a cada vecino. La dirección emitida se compondría de la etiqueta propia del nodo emisor más un sufijo determinado aleatoriamente para cada vecino⁹.

Longitud del prefijo	Prefijo	ID1	MAC1	ID2	MAC2	IDn	MACn
----------------------	---------	-----	------	-----	------	-------	-----	------

Tabla 4.1: Estructura del mensaje HLMAC (payload)

En la Tabla 4.1 se presenta la estructura característica de un mensaje HLMAC. Se puede ver que las etiquetas, a medida que se vayan propagando los mensajes a través de la red, se ampliarán y se harán cada vez más largas. Es por ello, que se programó la posibilidad de imponer una longitud máxima de carga útil (`max_payload`) para topologías de red a gran escala. Una vez el nodo raíz hubiera enviado los mensajes HLMAC a todos sus vecinos, se seguiría la propagación de las etiquetas hasta llegar a los nodos más lejanos de la topología.

Como se ha expuesto en la Sección 4.2.2.2, se implementó como novedad respecto a IoTorii la definición de una estructura de datos para cada nodo. Por lo tanto, en el proceso de envío de paquetes se produciría su escritura, añadiendo la información del propio nodo mediante una función auxiliar.

Por otro lado, en cuanto a la recepción de este tipo de paquetes en los nodos, es preciso recordar el comentario realizado en el Apartado 4.3.1.1. En este caso, a diferencia de los mensajes *Hello*, se tratarían de mensajes con contenido, y es por ello que fue preciso plantear un procesamiento distinto para los mismos (ver Algoritmo 4.1).

⁸Función *handler* responsable de la transmisión de las etiquetas

⁹Es preciso que en el envío de las direcciones HLMAC todos los vecinos de un nodo tengan la misma prioridad de ser elegidos.

```

if Dirección HLMAC especificada en el mensaje then
    if No existe bucle then
        Añadir dirección a la tabla HLMAC
        if Se ha asignado la dirección al nodo then
            Propagación de etiquetas hacia los vecinos
        else
            Se libera memoria
        for Número de vecinos do
            if Nodo vecino es el nodo emisor then
                if Se ha asignado la dirección al nodo then
                    Se tacha el vecino de la lista de pendientes
                    Se marca como padre (flag = 1)
                else
                    Se tacha el vecino de la lista de pendientes
                    No se marca como padre (flag = -1)
            Se decrementa el número de vecinos de los que falta por recibir HLMAC
        else
            Se libera memoria
    else
        No se procesa como mensaje HLMAC

```

Algoritmo 4.1: Procesamiento de mensajes HLMAC

Para posibilitar posteriormente los repartos de carga en los nodos, fue necesario en este procesamiento implementar la clasificación de los vecinos, según la relación que presentaban con un nodo determinado (padre o hijo). Por ello, se procedería a asignar valor al nuevo campo definido `flag` para cada entrada de la estructura de vecinos. Así, nodo a nodo, se conseguiría establecer la estructura de la topología completa de red. Finalmente, con este proceso, cada uno de los nodos conocería cuál de sus vecinos es su padre y por tanto, el que le asignó la primera etiqueta (`flag = 1`). En el caso de los demás, solo se tacharían de la lista en el momento en el que se recibiera un mensaje HLMAC de los mismos (`flag = -1`).

Como soporte a la programación, se introdujo una cuenta descendente que se inicializaría al comienzo con el número total de vecinos y que iría decrementándose con cada llegada de un nuevo mensaje HLMAC. Al concluir, los vecinos restantes por marcar se identificarían como los hijos del nodo y por tanto, sería posible conocer de forma clara la cantidad total de estos. Como se comentaba en la Sección 4.2.2.4.3, este valor se convirtió en la condición principal para identificar a los nodos frontera de la red.

Finalmente, se visualizarían las primeras estadísticas del algoritmo con la información relativa al número de mensajes *Hello* y HLMAC recibidos para cada nodo de la red. Además, se indicaría para cada nodo si es frontera o no (ver Figuras 4.6 y 4.7) y, en el primer caso se activaría un distintivo o flag para el mismo (`edge = 1`).

4.3.1.3 Mensajes de notificación de carga

Una vez creada la topología de red, se procedería a iniciar el envío de mensajes de notificación de las cargas computacionales de los nodos. Es decir, cada uno de ellos informaría a sus vecinos de la carga inicial que posee para que la pudiera añadir a su lista de vecinos. Para llevar a cabo este proceso se hizo uso del elemento `load` de la nueva estructura de información del nodo `this_node`, comentada en el Apartado 4.2.2.2.

En primer lugar, se planteó comenzar con la transmisión de los mensajes desde los nodos frontera hacia sus vecinos, debido a que cuando se mostraran las estadísticas, cada nodo ya tendría conocimiento de si es frontera o no. Por lo tanto, se tomó esta condición para iniciar el contador respectivo al envío de las notificaciones.

Mote output		
Time	Mote	Message
00:11.480	ID:11	//INFO HANDLE LOAD// carga enviada: 58 de direccion 01.01.01.03.02.03.
00:11.540	ID:6	//INFO HANDLE LOAD// carga enviada: 14 de direccion 01.01.01.04.
00:11.638	ID:4	//INFO HANDLE LOAD// carga enviada: 135 de direccion 01.01.04.
00:11.691	ID:7	//INFO HANDLE LOAD// carga enviada: 47 de direccion 01.01.01.02.
00:11.846	ID:10	//INFO HANDLE LOAD// carga enviada: 61 de direccion 01.01.01.03.02.02.
00:12.188	ID:3	//INFO HANDLE LOAD// carga enviada: 2 de direccion 01.01.03.
00:13.506	ID:9	//INFO HANDLE LOAD// carga enviada: 113 de direccion 01.01.01.03.02.
00:15.598	ID:5	//INFO HANDLE LOAD// carga enviada: 170 de direccion 01.01.01.
00:17.596	ID:8	//INFO HANDLE LOAD// carga enviada: 113 de direccion 01.01.01.03.
00:17.658	ID:2	//INFO HANDLE LOAD// carga enviada: 91 de direccion 01.01.
00:20.686	ID:1	//INFO HANDLE LOAD// carga enviada: 35 de direccion 01.

Figura 4.8: Mensajes de notificación del envío de carga

Volviendo a tomar como ejemplo la topología de los apartados anteriores (ver Figura 4.4), en la que se indicaba que existían 6 nodos frontera (ID: 3, 4, 6, 7, 10 y 11), se puede observar en la Figura 4.8 que, de forma correcta, se notificaría primero de sus cargas¹⁰. En consecuencia, estas serían las primeras en escribirse en las tablas correspondientes a los vecinos de cada uno de los nodos frontera. En cuanto a los nodos restantes (los nodos identificados como no frontera), informarían de su carga con un retardo aleatorio introducido para que no incidan los mensajes entre sí.

Como los mensajes de notificación de carga se definieron como paquetes con contenido, se tuvo que hacer uso de la función de procesamiento que se había empleado para los mensajes HLMAC. Como novedad, sería necesario imponer ciertas condiciones para diferenciar entre estos dos tipos de paquetes a la llegada del nodo. Es por ello, que se programó la comprobación, en primera instancia, del contenido del mensaje para analizar si este portaba una etiqueta. En caso negativo, se impuso por descarte que la información comprendida en el paquete haría referencia a la carga informada.

Mote output		
Time	Mote	Message
00:11.480	ID:11	//INFO HANDLE LOAD// carga enviada: 58 de direccion 01.01.01.03.02.03.
00:11.491	ID:10	//INFO INCOMING LOAD// carga recibida: 58 del nodo 0B.
00:11.491	ID:9	//INFO INCOMING LOAD// carga recibida: 58 del nodo 0B.

Figura 4.9: Envío y recepción de la carga de un nodo

En la Figura 4.9 se visualiza la transmisión de la carga de uno de los nodos frontera (ID: 11) de la topología de ejemplo. Este informó de su carga a sus dos vecinos (ID: 9, 10), los cuales, al recibirla, tenían que buscar la entrada perteneciente al nodo emisor en la lista de vecinos para rellenarla con el dato. Por

¹⁰En concreto, primero el ID 11, luego 6, 4, 7, 10 y finalmente 3.

consiguiente, en cada una de las transmisiones, los nodos continuarían completando su lista de vecinos con las cargas que poseía cada uno. En la Figura 4.10 se toma como ejemplo el nodo con ID 10, el cual muestra por pantalla la modificación de su lista de vecinos con el valor de carga informada por parte del vecino con ID 11.

Time	Mote	Message
00:11.508	ID:10	--> Vecino padre (flag, carga, in_out) --> 1, 0, 0
00:11.512	ID:10	--> Vecino padre (flag, carga, in_out) --> -1, 58, (0)
00:11.516	ID:10	//INFO STATISTICS// Faltan 1 nodos por conocer su carga

Figura 4.10: Actualización de la lista de vecinos con la carga informada

En el instante en el que se concluyera con la transmisión de las notificaciones, ya se podría comenzar con la fase de reparto en la topología de red. Para ello, todos los nodos tendrían que haber completado antes su lista de vecinos con los valores de cada una de las cargas informadas. Se observó que era imprescindible informar en cada actualización de las estadísticas sobre la cantidad de ellos que todavía faltaban por notificar su carga. Para ello, cada nodo debía recorrer su lista y comprobar que ninguna de las cargas que tenía era nula. En caso de haber llenado toda la lista, se modificaría el valor del flag (`load_null = 0`) y después, se planificarían los mensajes de traspaso (ver Listado 4.3).

4.3.1.4 Mensajes de traspaso de carga y proceso de reparto

La idea principal de este procedimiento fue desarrollar una propagación de la carga computacional de forma progresiva desde los nodos frontera hacia el nodo raíz. Se pretendía dejar al final del proceso un esquema de red en el que cada uno de los nodos conservara una carga final de valor 100¹¹, independientemente de la que tenía inicialmente.

Listado 4.3: Inicio del proceso de reparto de cargas en cada nodo

```

if ((edge == 1 || new_edge == 1) && load_null == 0)
{
    ctimer_set(&share_timer, IOTORII_SHARE_START_TIME * CLOCK_SECOND,
              iotorii_handle_share_upstream_timer, NULL);
}

```

El reparto en sí, en términos globales, tendría un sentido upstream. Los nodos frontera iniciarían los traspasos a sus padres y estos a su vez, a los suyos. Por ello, en primera instancia, fue preciso diferenciar dentro del proceso entre los nodos que se habían identificado como frontera y los que no. Como se había comentado ya en el Apartado 4.3.1.2, esto se produciría mediante el flag `edge = 1`, el cual activaría primero el comienzo de la transmisión solo para los nodos frontera.

¹¹Enfocando el algoritmo en el ámbito de las smart grids, sería el equivalente a un porcentaje de uso del 100 %

De la misma forma, una vez estos hubieran finalizado el intercambio de mensajes, se continuaría con el proceso habilitando los traspasos para los siguientes nodos en la topología¹². Los nodos padre que ahora entrarián en juego se convertirían en una especie de nodos frontera mediante la asignación de un nuevo flag (`new_edge = 1`) y así, ya podrían iniciar los traspasos de sus cargas. En el Listado 4.3 se muestran las condiciones impuestas para el inicio del proceso de reparto. La comunicación terminaría una vez se llegase hasta el nodo raíz, el cual sería el único que podría tener al final del proceso un valor de carga distinto de 100. Al concluir, sería posible que este se encontrara sobrecargado si se hubiera acumulado un gran excedente de carga computacional en la propagación.

Es importante hacer hincapié en que los mensajes portarían el valor de carga que se traspasa de un nodo a otro. En otros términos, al igual que los mensajes HLMAC y los de notificación de carga, se tratarían de paquetes con contenido. En todos ellos se requeriría una previa preparación del buffer como se puede visualizar en el Listado 4.4.

Listado 4.4: Preparación del buffer para el envío de paquetes

```

packetbuf_clear();
memcpy(packetbuf_dataptr(), &(extra_load), sizeof(extra_load));
packetbuf_set_datalen(sizeof(extra_load));

packetbuf_set_addr(PACKETBUF_ADDR_RECEIVER, &linkaddr_null);

```

Primero, se debía resetear el buffer para poder copiar el contenido que llevaría cada paquete, además de establecer su longitud. En el caso del proceso de traspasos, se insertaría la parte de carga extraída del nodo. (`extra_load`). Tras el estudio del procedimiento de preparación del buffer, se observó que era necesario plantear cómo se iban a diferenciar todos estos mensajes entre sí al llegar a un nodo, ya que se produjo una situación en la que cada paquete iba a requerir de un procesamiento distinto.

```

if Mensaje tiene contenido then
  if Dirección HLMAC no especificada en el mensaje then
    if Se han identificado los nodos frontera y no se han iniciado los traspasos then
      Se procesa como mensaje de notificación de carga
    else if Se han iniciado los traspasos then
      Se procesa como mensaje de traspaso de carga
    else
      Se procesa como mensaje HLMAC
  else
    Se procesa como mensaje Hello

```

Algoritmo 4.2: Diferenciación del procesamiento para cada tipo de paquete

Como ya se había expuesto en apartados anteriores, la evolución de la programación del algoritmo se había basado en la progresiva introducción de comprobaciones para poder llevar a cabo una diversificación del tratamiento de los paquetes. Por ello, en el Algoritmo 4.2 se describen finalmente, todas las condiciones que se impusieron en el código para poder identificar de forma correcta cada tipo de mensaje.

¹²Se puede expresar que se produciría una modificación en el esquema de red. A nivel del algoritmo se interpretaría que los nodos frontera ya habrían concluido su tarea y, por tanto, sería como eliminarlos.

Volviendo al proceso de reparto, fue preciso determinar que la carga contenida en los mensajes podría ser tanto positiva como negativa. En el Algoritmo 4.3 se describe el procedimiento seguido por el protocolo Dedenne para llevar a cabo el envío de los mensajes. Como concepto, a partir del valor de carga del nodo, se calcularía la cantidad a enviar a su padre y se establecería el signo la misma. Se podrían definir varios casos de aplicación en función de la situación de cargas que presentaran los nodos implicados en el traspaso.

```

if Nodo frontera o nodo nuevo frontera then
    if carga >100 then
        cargo a enviar = carga del nodo - 100
    else if carga del nodo <100 then
        if carga del nodo >0 then
            cargo a enviar = carga del nodo - 100
        else
            cargo a enviar = - carga del nodo + 100
    else
        cargo a enviar = - carga a enviar

    if nodo es nodo común then
        cargo del nodo = 100
    for Número de vecinos do
        if Vecino es padre then
            Flujo del vecino = - cargo a enviar
            Preparación del buffer
    Envío del paquete

```

Algoritmo 4.3: Procedimiento de envío de los mensajes de traspaso

4.3.1.4.1 Caso 1: un nodo hijo con excedente de carga

Se podría definir como caso por defecto el traspaso de carga positiva desde un nodo hijo hacia su padre (sentido upstream). Es decir, sería la situación en la que un nodo determinado tuviera un valor de carga mayor a 100. En consecuencia, este nodo debería proceder a calcular cuál es su cantidad sobrante y empaquetarla para enviársela al padre. Después, siempre que fuera un nodo común, fijaría su carga computacional a un valor de 100¹³. Una vez el nodo indicara el fin de la transferencia de carga, daría paso a la participación de los siguientes nodos y ya no volvería a transmitir más mensajes de carga por el momento¹⁴.

Time	Mote	Message
00:25.700	ID: 4	//INFO HANDLE SHARE UP// En este nodo sobra una carga de 35
00:25.703	ID: 4	//INFO HANDLE SHARE UP// Carga actualizada: 100
00:25.706	ID: 4	//INFO HANDLE SHARE UP// Carga informada: 35
00:27.705	ID: 4	Carga actual del nodo: 100

Figura 4.11: Ejemplo de cálculo de carga para el caso 1

¹³Este paso se aplicaría en todos los nodos menos en el nodo raíz, ya que es el único que tendría la capacidad de soportar un valor de carga distinto a 100.

¹⁴Siempre que no se produjeran cambios en la topología o se añadieran nuevos nodos a la red.

En la Figura 4.11 se presenta el cálculo de la carga excedente para uno de los nodos de la topología empleada en apartados anteriores (ver Figura 4.4). En este caso, el nodo con ID 4 poseía inicialmente una carga computacional igual a 135. Por ello, tuvo que traspasar una cantidad de 35 a su padre y actualizar su carga a un valor de 100.

4.3.1.4.2 Caso 2: un nodo hijo con escasez de carga

Se podría dar también el caso contrario si un nodo tuviera un valor de carga menor a 100. Esto supondría un traspaso de carga negativa hacia su padre, al que debería enviarle un mensaje informándole sobre la cantidad que podría llegar a absorber hasta alcanzar el valor de 100. Para realizar el cálculo correcto de dicha cantidad, sería necesario comprobar primero si la carga es positiva o negativa. Además, como se comentaba en el caso 1, si se diera la condición de que el nodo fuera común se establecería un valor de carga computacional igual a 100.

Mote output		
Time	Mote	Message
00:31.658	ID:8	//INFO INCOMING SHARE// carga actual del nodo: 45
00:41.675	ID:8	//INFO HANDLE SHARE UP// En este nodo permite una carga adicional de valor 55
00:41.679	ID:8	//INFO HANDLE SHARE UP// Carga actualizada: 100
00:41.682	ID:8	//INFO HANDLE SHARE UP// Carga informada: -55
00:41.718	ID:5	//INFO INCOMING SHARE// carga actual del nodo: -24
00:51.732	ID:5	//INFO HANDLE SHARE UP// En este nodo permite una carga adicional de valor 124
00:51.736	ID:5	//INFO HANDLE SHARE UP// Carga actualizada: 100
00:51.739	ID:5	//INFO HANDLE SHARE UP// Carga informada: -124

Figura 4.12: Ejemplo de cálculo de carga para el caso 2

Volviendo a la topología de ejemplo, se puede visualizar en la Figura 4.12 el proceso de traspaso que se produjo entre dos nodos que necesitaban más carga para alcanzar el valor de 100. Ambos enviaron un paquete que contenía un valor de carga negativa a su padre.

4.3.1.4.3 Caso 3: varios nodos hijos. Planteamiento del procesamiento de los paquetes recibidos

Por último, y como caso adicional, se encontraría la situación de un nodo con varios hijos a su cargo. En ciertas topologías, sobre todo a gran escala, sería muy probable que se llegue a esta tesitura, ya que existirían multitud de nodos en la red. Por tanto, en el desarrollo del código se observó que surgía la necesidad de monitorizar el proceso de reparto y de llevar un control de los traspasos que habían realizado cada uno de los hijos a su padre.

Como se había expuesto anteriormente, para continuar el proceso nodo a nodo en sentido upward (en dirección al nodo raíz) era preciso marcar que un nodo determinado había completado el intercambio de mensajes con todos sus hijos. Entonces, si existieran varios, se debería contabilizar que todos ellos habían terminado el traspaso correspondiente hacia su padre y además, que habían impuesto internamente un valor de carga igual a 100.

Como se puede ver en la Figura 4.13, el nodo procedía a modificar su valor de carga y a notificar sobre el cambio realizado con cada recepción de un nuevo paquete. Se configuró el mensaje que se imprimiría por pantalla para informar de que el nodo ya había actualizado el valor de flujo en su lista de vecinos. Para ello, debía buscar dentro de la lista la entrada correspondiente al hijo emisor y escribir la cantidad que había sido traspasada.

Time	Mote	Message
00:25.687	ID:5	//INFO INCOMING SHARE// carga actual del nodo: 117
00:25.708	ID:5	//INFO INCOMING SHARE// carga actual del nodo: 31
00:41.718	ID:5	//INFO INCOMING SHARE// carga actual del nodo: -24
00:43.713	ID:5	Carga actual del nodo: -24
00:43.717	ID:5	--> Vecino padre (flag, carga inicial, in_out) --> 1, 91, 0
00:43.722	ID:5	--> Vecino hijo (flag, carga inicial, in_out) --> 0, 47, -53
00:43.727	ID:5	--> Vecino hijo (flag, carga inicial, in_out) --> 0, 113, -55
00:43.731	ID:5	--> Vecino hijo (flag, carga inicial, in_out) --> 0, 14, -86

Figura 4.13: Cálculo de la carga de un nodo a partir de los traspasos de varios hijos

Después de establecer los casos anteriores y, teniendo en cuenta el hecho de que los traspasos podrían implicar tanto mensajes con carga positiva como con carga negativa, se volvió recomendable por eficiencia plantear un procesamiento de paquetes generalizado (ver Listado 4.5). Se debía englobar tanto una situación en la que un hijo pretendiera ceder carga a su padre como en la que se la solicitara. También, había que comprobar el número de hijos que tenía cada nodo.

Listado 4.5: Procesamiento de los mensajes de traspaso

```

for (nb = list_head(neighbour_table_entry_list); nb != NULL; nb = list_item_next(nb))
{
    if (linkaddr_cmp(&nb->addr, sender))
    {
        if (nb->flag == 0 && edge == 0 && n_hijos != 0)
        {
            node->load = node->load + *p_extra;
            n_hijos--;
        }

        if (n_hijos == 0)
        {
            new_edge = 1;
            ctimer_set(&statistic_timer, IOTORII_STATISTICS2_TIME * CLOCK_SECOND,
                      iotorii_handle_statistic_timer, NULL);
        }
    }

    if (*p_extra != 0 && nb->in_out == 0)
        nb->in_out = *p_extra;
}
}

```

En consecuencia de las condiciones anteriores, se introdujo en la función de procesamiento un contador inicializado con el total existente de hijos, el cual iría decrementándose. Se seguiría por tanto, un procedimiento similar al proceso de identificación de los nodos frontera (ver Sección 4.2.2.4.3). Entonces, cuando se comprobara que la cuenta era nula se activaría el flag `new_edge = 1` y se planificaría el siguiente intercambio de mensajes.

Entrando en detalle en la función, se programó la siguiente secuencia de acciones para llevar a cabo el procesamiento de los mensajes de traspaso:

1. **Identificación del emisor:** Se recorría la lista de vecinos del nodo para comprobar cuál de ellos es el emisor del mensaje. Al ser una comunicación upstream (desde los nodos frontera hacia el nodo raíz), se debía cumplir la condición de que el emisor del paquete era uno de los hijos.
2. **Actualización de la carga:** Se añadía la carga computacional (positiva o negativa) a la que ya existía almacenada anteriormente en el nodo.
3. **Ajuste de la cuenta de hijos:** Se decrementaba en uno el contador de los hijos que faltaban por realizar el traspaso.
4. **Comprobación del valor de cuenta:** Si esta era nula, el propio nodo se marcaría como nueva frontera y planificaría las estadísticas para actualizar los resultados.
5. **Actualización de la información del flujo:** Cada vez que se procesara un nuevo traspaso entre dos pares de nodos, se escribiría en la lista de vecinos del receptor el flujo de carga que se había producido.

Visualizando el Listado 4.5 hay que hacer hincapié en el bucle que se planteó para llevar a cabo la búsqueda del nodo emisor en la lista de vecinos. Implementar este paso fue imprescindible, ya que en el procedimiento de envío de los paquete no se había especificado la dirección del receptor (ver Listado 4.4)¹⁵. Es decir, el paquete sería difundido a los vecinos del nodo emisor y todos ellos lo recibirían. Sin embargo, el único que debería realmente procesarlo sería el padre.

Por otro lado, en cuanto a la progresiva actualización de los flujos de datos, al final del proceso se podría obtener a nivel global un historial de todos los movimientos realizados en cada uno de los nodos. La entrada de cada flujo indicaría la dirección que tomó al realizar el traspaso o el hijo al que haría referencia. En cuanto al signo, se relacionaría con el sentido en el que se produjo la transferencia:

- **Valor positivo:** El flujo en la conexión padre-hijo tuvo sentido upward (hacia el nodo raíz). El hijo poseía un excedente de carga que traspasó al padre y entonces, la carga se vio incrementada.
- **Valor negativo:** El flujo en la conexión padre-hijo tuvo sentido downward (hacia los nodos frontera). El hijo necesitó absorber carga de su padre para obtener un valor de 100 y por lo tanto, se actualizó la carga de este con un valor menor al que existía.
- **Valor nulo:** Si la entrada en la lista de vecinos tuviera un 0, se podría determinar que todavía no se habría producido el traspaso con el nodo relacionado a la entrada de la lista.

¹⁵Se determinaría como nula la dirección del receptor, tratándose de un envío de paquetes en modo *broadcast* o por difusión.

Es importante recalcar que se podría dar el caso inusual en el que un nodo hijo inicialmente tuviera un valor de carga igual a 100. Es decir, cuando se produjera el proceso de reparto informaría de un valor nulo a su padre. En este caso, no existiría un flujo como tal entre el par de nodos, por lo que la lista seguiría conservando un 0.

4.4 Tiempos de ejecución

En esta sección se exponen los retardos impuestos por defecto para cada uno de los procesos del algoritmo. Se tuvo que tener en cuenta la duración de cada uno de ellos para que no interfirieran entre sí y para que existiera un correcto funcionamiento a nivel global. Además, fue preciso fijar un número medio máximo de nodos para las topologías que se pretendían simular. Si se quisiera incrementar el mismo de forma considerable, sería preciso aumentar también los tiempos.

Por optimización, se observó que era recomendable que los retardos fueran muy pequeños para que las acciones que se producían en los nodos se elaboraran lo antes posible. En otros términos, se debía plantear el equilibrio entre la eficiencia del algoritmo y la compatibilidad con una gran diversidad de topologías. Como se había comentado en el Apartado 3.3, para evidenciar el correcto funcionamiento del algoritmo se decidió adaptar topologías diseñadas en el software Mininet a la plataforma Contiki¹⁶. Entrando en profundidad en este tipo de redes, se observó que se componían de multitud de nodos (en un rango entre 20 y 30). En consecuencia, se tuvieron que ajustar¹⁷ los tiempos en base a las mismas, por lo que se puede expresar que los retardos se optimizaron principalmente para redes con un número de nodos en torno a un valor de 30.

Para determinar los ajustes temporales se llevó a cabo el estudio de los resultados obtenidos en la simulación de un ejemplo de topología con estas características. Es decir, de forma aproximada, se estimó qué retardos se debían imponer, buscando un cierto margen temporal para que no existieran incidencias en el funcionamiento del algoritmo. Al final de esta sección, se mostrará en la Figura 4.14 la línea temporal de la secuencia de ejecución de los procesos, indicando en qué instantes se ejecutarían cada uno de ellos.

4.4.1 Inicio del intercambio de mensajes Hello

En la función de inicialización de cada uno de los nodos se iniciarían en primer lugar, los contadores de inicio de los procesos de búsqueda de vecinos y de asignación de las etiquetas jerárquicas:

En cuanto al primero, haría referencia al instante temporal en el que comenzaría el intercambio de los mensajes Hello entre los nodos. Se definió por defecto un retardo de 2 segundos, teniendo en cuenta un cierto margen con el procedimiento de inicialización interno que se producía en todos los nodos:

```
|| #define IOTORII_HELLO_START_TIME 2
```

Como se determina en el Listado 4.6, referente a la función de inicialización, para cada uno de los nodos de la red se establecería un tiempo aleatorio de inicio. El valor del tiempo de espera sería como mínimo igual al retardo por defecto de 2 segundos y podría llegar a valer un máximo de 4 segundos. Con este rango de valores sería posible realizar los intercambios en distintos instantes de tiempo.

¹⁶El principal motivo de implementarlas de este modo fue poder comparar el desarrollo de este TFG con otros trabajos similares del grupo de investigación que se habían realizado en Contiki.

¹⁷El ajuste de los tiempos se basó en un procedimiento de prueba y error mediante la simulación de varias topologías extraídas de Mininet

Listado 4.6: Retardos en la función de inicialización

```

static void init (void)
{
    ...

    clock_time_t hello_start_time = IOTORII_HELLO_START_TIME * CLOCK_SECOND;
    hello_start_time = hello_start_time + (random_rand() % hello_start_time);
    ctimer_set(&hello_timer, hello_start_time, iotorii_handle_hello_timer, NULL);

    ctimer_set(&statistic_timer, IOTORII_STATISTICS1_TIME * CLOCK_SECOND,
              iotorii_handle_statistic_timer, NULL);

#if IOTORII_NODE_TYPE == 1
    ctimer_set(&sethlmac_timer, IOTORII_SETHLMAC_START_TIME * CLOCK_SECOND,
              iotorii_handle_sethlmac_timer, NULL);
#endif

    ...
}

```

4.4.2 Inicio del proceso de asignación de etiquetas

4.4.2.1 Nodo raíz

Haciendo referencia al proceso de asignación de etiquetas, se debía fijar el inicio del mismo desde el nodo raíz. Según se había expuesto ya en el Apartado 4.3.1.2, se indicaría el retardo existente desde que se inicializaba el nodo raíz hasta que este mismo enviaba el primer mensaje HLMAC a sus vecinos. Se decidió marcar con un valor de 5 segundos:

```
#define IOTORII_SETHLMAC_START_TIME 5
```

Es decir, volviendo a la fase de intercambio de mensajes Hello, se había determinado previamente que esta poseería una duración aproximada de 2 segundos. Al iniciarse con un retardo de otros 2 segundos, se analizó que era obligatorio comenzar como muy pronto el proceso de asignación a los 5 segundos desde que se había iniciado la ejecución. Tenía que existir un cierto margen temporal en el caso de querer aumentar el número de nodos de la red. En el Listado 4.6 se visualiza la expresión relativa al contador en la función de inicialización.

4.4.2.2 Nodos comunes

Para el caso de los nodos comunes y, en concreto para los vecinos del nodo raíz, se iniciarían el retardo cuando este hubiera completado el envío de etiquetas. En el instante en el que el nodo raíz finalizara su asignación, se determinaría el tiempo de espera aleatoriamente, de forma parecida a como se producía para los mensajes Hello. Al igual que para el nodo raíz, se impuso un retardo por defecto de 5 segundos:

```
#define IOTORII_SETHLMAC_DELAY 5
```

En las expresiones descritas en el Listado 4.7 se indica cómo se establecería el rango de valores posibles, partiendo de un retardo mínimo igual a IOTORII_SETHLMAC_DELAY/2.

Listado 4.7: Retardo de envío de mensajes HLMAC para los nodos comunes

```
|| void iotorii_send_sethlmac (hlmacaddr_t addr, linkaddr_t sender_link_address)
{
    ...
    ...
    clock_time_t sethlmac_delay_time = IOTORII_SETHLMAC_DELAY/2 * (CLOCK_SECOND / 128);
    sethlmac_delay_time = sethlmac_delay_time + (random_rand() % sethlmac_delay_time);

    ctimer_set(&send_sethlmac_timer, sethlmac_delay_time, iotorii_handle_send_sethlmac_timer
              , NULL);
    ...
}
```

4.4.3 Visualización de las primeras estadísticas

En la función de inicialización (ver Listado 4.6), se definió además, el instante temporal en el que se imprimirían las primeras estadísticas. Este evento tenía que suceder una vez hubiera ocurrido la asignación completa de direcciones para todos los nodos de la red. Por lo tanto, al estimar que de media en una topología a gran escala se requerirían 2 segundos para ello, se planteó un retardo de 8 segundos:

```
|| #define IOTORII_STATISTICS1_TIME 8
```

En otros términos, la asignación finalizaría en torno al segundo 7. En ese instante todos los nodos tendrían almacenada una etiqueta y se podrían visualizar las estadísticas correctamente. Se podría visualizar por pantalla la información de cada nodo, como su carga computacional, la etiqueta que ha almacenado o el número de mensajes transmitidos de cada tipo. También, se mostraría la información recibida de parte de sus vecinos, referente al número total que tiene el nodo y a la relación existente con ellos (padre o hijo).

4.4.4 Inicio de notificación de las cargas

Una vez se ejecuten las primeras estadísticas, habría que continuar con la siguiente fase: la notificación de las cargas. Como se exponía en el Apartado 4.3.1.3, se comenzaría el proceso por los nodos frontera. Es decir, en el instante en el que se mostraran las estadísticas los nodos ya se habrían identificado y se conocería cuáles de ellos serían extremos de la red.

Por ello, se tuvo que imponer la condición de nodo frontera para iniciar el contador y que estos pudieran informar de su carga a sus vecinos. Como se exponía en el Algoritmo 4.4, según el tipo de nodo se establecería un retardo distinto.

En el caso de los nodos frontera, al ser los primeros en ejecutarse, habría que estimar antes la duración de la visualización de las estadísticas para que estas no se cruzaran con el inicio del proceso. Se definió un retardo inicial de 3 segundos, que sería impuesto para todos los nodos identificados como extremos de la red:

```
|| #define IOTORII_LOAD_START_TIME 3
```

Una vez se hubieran enviado los primeros mensajes de notificación de cargas se podría continuar con el resto de nodos. En este caso, se estableció la posibilidad de obtener un retardo incluso menor, ya que no existirían restricciones impuestas por otros procesos que se estuvieran ejecutando. Se aplicó un tiempo de espera aleatorio que podía llegar a un valor máximo igual al valor definido por defecto de 3 segundos:

Listado 4.8: Retardo de envío de mensajes de notificación de carga

```
|| timer_set(&load_timer, random_rand() % (IOTORII_LOAD_START_TIME*3) * CLOCK_SECOND,
           iotorii_handle_load_timer, NULL);
```

Esto implicó que en algunos nodos pudieran suceder ejecuciones casi instantáneas y sin apenas retardo, cumpliéndose el objetivo de minimizar los tiempos de espera. Debido a esta optimización temporal, se mejoró la eficiencia del algoritmo. En cuanto a las topologías extraídas de Mininet, se observó que se producía el inicio del proceso en torno al segundo 11 (ver Figura 4.14).

```
if El nodo no tiene hijos then
    Se notifica que el nodo es frontera
    edge = 1
    if No se ha iniciado la notificación de cargas then
        Inicio de contador con retardo fijo
    else
        Se notifica que el nodo no es frontera
        if Los nodos frontera han notificado de la carga y los nodos no frontera no then
            Inicio de contador con retardo variable
        Se marca el nodo como informado
```

Algoritmo 4.4: Retardos definidos para el proceso de notificación de cargas

4.4.5 Inicio del proceso de reparto de cargas

Para especificar el instante de comienzo de la fase de traspasos de cargas entre los nodos sería preciso recordar el código 4.3. En él, se exponía la programación de las condiciones que se habían impuesto para establecer el temporizador de este proceso. Por un lado, se encontraba el requisito de que el nodo fuera uno de los extremos de la red (`edge = 1`) o de que no lo fuera, pero se hubiera activado el flag indicador de que le había llegado el turno de ejecutar el traspaso (`new_edge = 1`).

Por otro lado, existía la condición imprescindible de que se hubiera rellenado de forma completa la lista de vecinos con todos los valores de carga de estos. Esto debía de suceder una vez se hubieran recibido en el nodo todos los mensajes de notificación de carga computacional (`load_null = 0`).

Por lo tanto, se planificarían los traspasos cada vez que se ejecutaran las estadísticas y además, un nodo determinado se encontrara en las circunstancias anteriores. Dado que esta situación se daría en primera instancia en los nodos frontera, el proceso de reparto se iniciaría siempre desde los mismos.

Como se había expuesto en el apartado referente a este procedimiento (ver Sección 4.3.1.4.3), un nodo no frontera no activaría su turno hasta que llegaran a él todos los mensajes de traspaso de sus hijos, por lo que se podría definir este proceso como una ejecución en cadena. En otros términos, hasta que no se activase el flag `new_edge = 1`, las estadísticas se mostrarían, pero no se planificaría el reparto para ese nodo determinado. Si se iniciara el tiempo de espera desde la función de estadísticas, se debería tener en cuenta el instante en el que en un nodo frontera ya hubiera completado su lista de vecinos con las cargas de cada uno.

Entrando en las topologías de red de gran escala (aprox. 30 nodos) que se pretendían simular, se estimó que las primeras planificaciones se producirían a partir del segundo 11, según se fueran recibiendo los mensajes de notificación (ver Apartado 4.4.4). Entonces, calculando la duración de este procedimiento, se llegó a la conclusión de que se necesitarían al menos unos 8 segundos para que existiera un margen mínimo entre el proceso de notificación y el de traspasos:

```
|| #define IOTORII_SHARE_START_TIME 8
```

Así, se podrían iniciar las primeras ejecuciones en los nodos frontera al segundo 20 aproximadamente (ver Figura 4.14). La duración total de este proceso dependería principalmente del esquema de nodos que presentara la topología y del número total de estos.

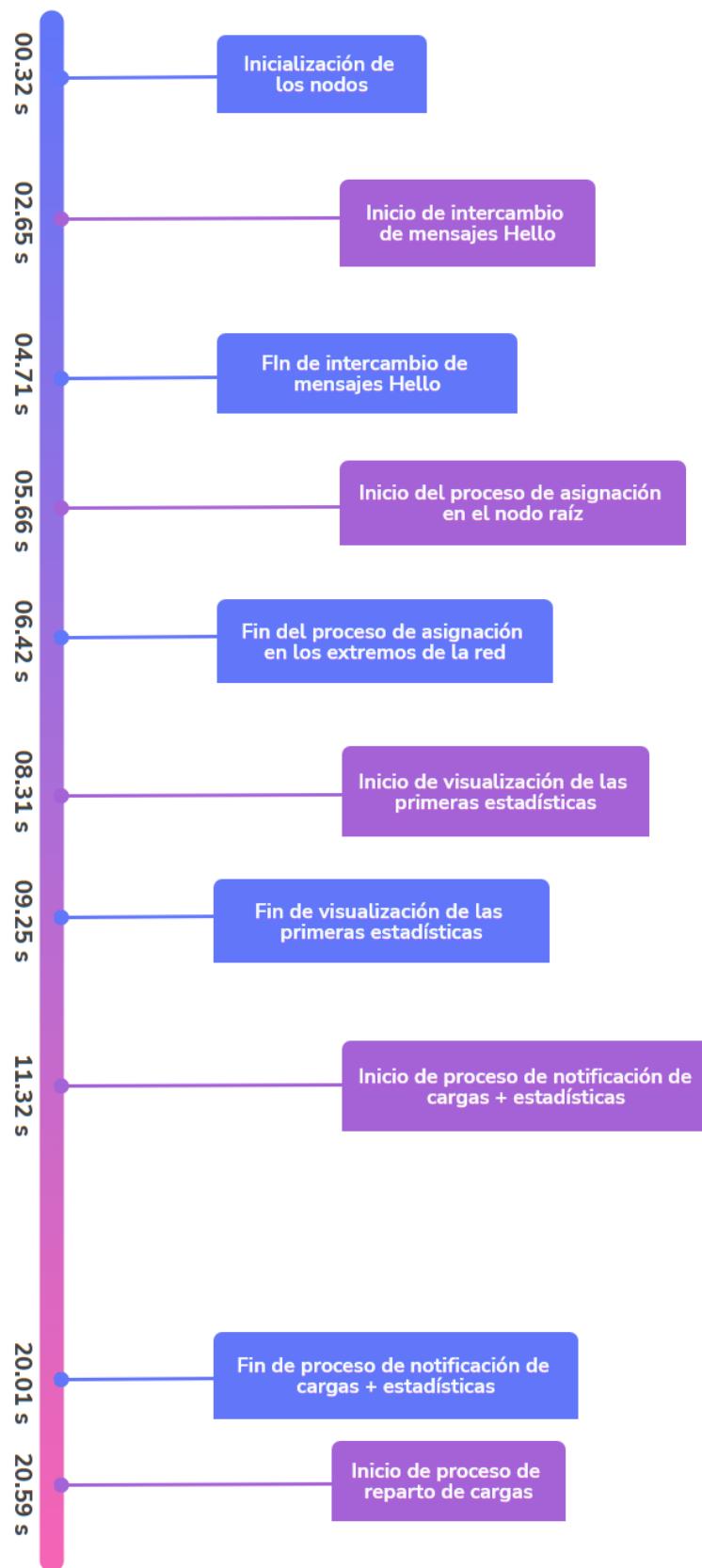


Figura 4.14: Línea temporal de la secuencia de procesos en el algoritmo para la topología

Capítulo 5

Evaluación de resultados

5.1 Introducción

En este Capítulo, con el fin de probar el funcionamiento del algoritmo desarrollado, se describirán los resultados obtenidos a partir de la simulación en Contiki de un ejemplo de topología de red. Se aportará en forma de capturas las muestras de la correcta ejecución de cada uno de los procesos en el nuevo modelo distribuido. Estas serán de utilidad para aprobar también, la implementación de las funcionalidades del algoritmo Dedenne en todos los nodos de la red. Además, se justificarán los resultados en base al marco teórico en el que se encuentra el trabajo (ver Capítulo 2) y haciendo referencia al proceso de desarrollo del código (ver Capítulo 4).

5.2 Selección y despliegue de la topología

Para exponer los resultados, se decidió hacer uso del diseño del esquema de red representado en la Figura 5.1.

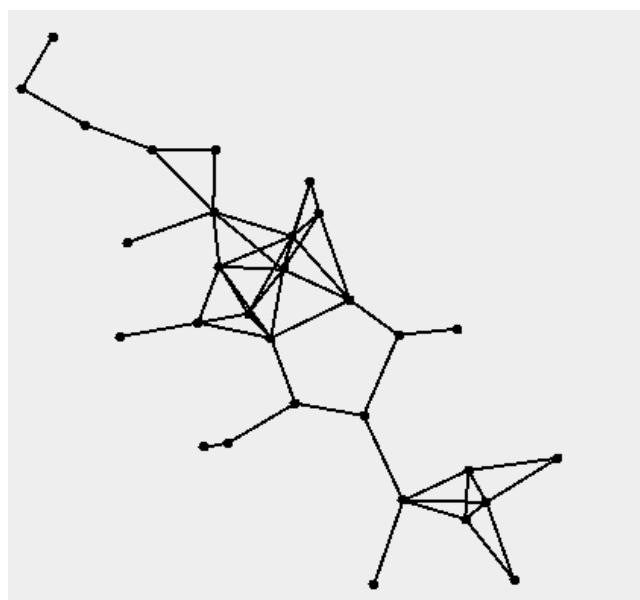


Figura 5.1: Diseño de la topología de red empleada

Como ya se había ido introduciendo en capítulos anteriores, el análisis principal de los resultados de este trabajo se basaría en el empleo de topologías de redes desplegadas en la aplicación de Mininet (2.3.3). En la Figura 5.2 se determina el gráfico que se produjo como resultado de la introducción del diseño anterior en el software.

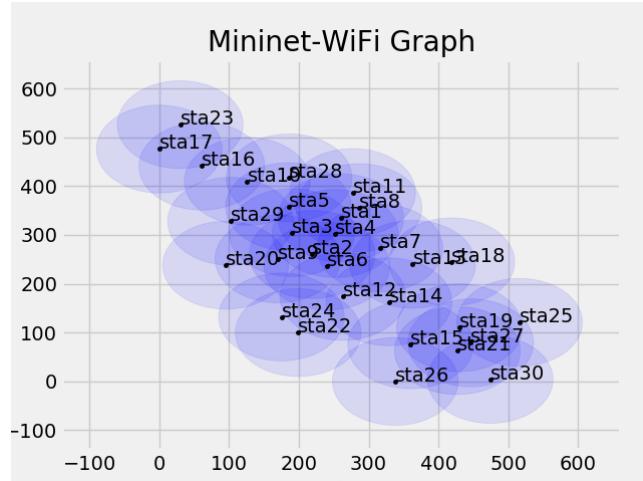


Figura 5.2: Despliegue de la topología 5.1 en Mininet-WiFi

Para posibilitar la simulación de esta red en Contiki, habría que hacer referencia al Apartado 3.3, en el que se exponía que Mininet aportaría a la salida un fichero de texto con las coordenadas de cada uno de los nodos de la topología. La importación del mismo a Contiki, y en particular a Cooja, supondría en primera instancia, un ajuste de los valores del fichero para desplegar la topología de forma proporcional.

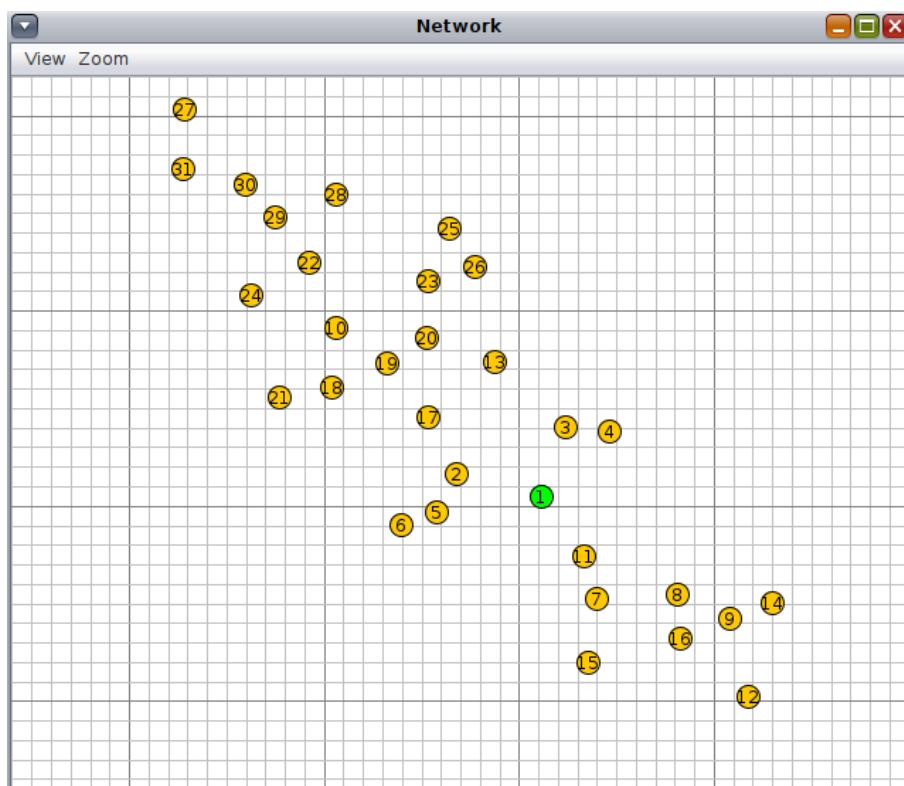


Figura 5.3: Despliegue de la topología 5.1 en Cooja

Con este procedimiento, se consiguió que, las áreas de cobertura de cada uno de los nodos y que venían descritas en Mininet, coincidieran con las introducidas en Cooja. En la Figura 5.3 se muestra el resultado de la adaptación resultante entre los diferentes soportes.

5.3 Resultados de la simulación

Una vez se desplegó la topología deseada en el simulador gráfico Cooja, ya se podía proceder a ejecutar la simulación del funcionamiento del algoritmo. Siguiendo la secuencia de procesos expuesta en la Sección 4.3, primero, cada nodo se inicializaría y enviaría mensajes Hello para elaborar una búsqueda dentro de su área y descubrir a sus vecinos.

Posteriormente, se llevaría a cabo la asignación de etiquetas con el fin de establecer la jerarquía de la red desde el nodo raíz hasta los extremos de la red. La topología dibujaría el esquema de las conexiones que resultaría entre cada uno de los pares de nodos, siguiendo una forma de árbol como se muestra en la Figura 5.4. En la misma también se puede observar la secuencia de pasos que se produjo en el procedimiento de asignación. En función de la longitud de la etiqueta almacenada en cada nodo (payload) y, como consecuencia, del número de saltos que se necesitaría para llegar al nodo raíz, se estableció una clasificación por colores de cada una de las uniones.

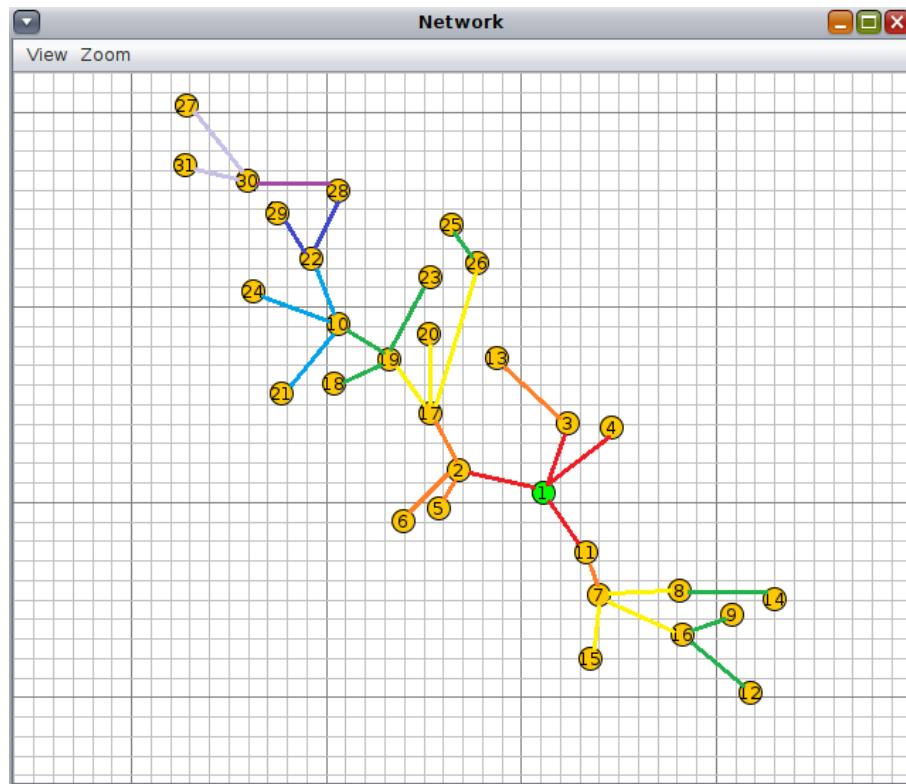


Figura 5.4: Árbol de red resultante del proceso de asignación de etiquetas

Al haber impuesto un almacenamiento de una dirección como máximo en cada nodo (ver Apartado 4.2.2.1), se obtuvo una topología simplificada respecto al diseño inicial de la Figura 5.1. En otros términos, esto supuso la existencia de un número menor de conexiones. Cada nodo procedió a llenar su estructura

de datos y se informó de ello por la ventana de salida de Cooja. En la Figura 5.5 se puede ver cómo se determinaba la información respectiva a la dirección asignada al nodo y a la de su padre, aparte del número de saltos que existían hasta el nodo raíz (igual a la longitud de la etiqueta o payload) y de la carga computacional inicial¹.

Time	Mote	Message
00:05.667	ID:1	[addr: 01. top_addr: / payload/hops: 1 node_load: 182]
00:05.717	ID:11	[addr: 01.03. top_addr: 01. payload/hops: 2 node_load: 58]
00:05.718	ID:4	[addr: 01.04. top_addr: 01. payload/hops: 2 node_load: 135]
00:05.718	ID:3	[addr: 01.02. top_addr: 01. payload/hops: 2 node_load: 115]
00:05.719	ID:2	[addr: 01.01. top_addr: 01. payload/hops: 2 node_load: 91]
00:05.749	ID:6	[addr: 01.01.04. top_addr: 01.01. payload/hops: 3 node_load: 65]
00:05.749	ID:5	[addr: 01.01.01. top_addr: 01.01. payload/hops: 3 node_load: 181]
00:05.750	ID:17	[addr: 01.01.03. top_addr: 01.01. payload/hops: 3 node_load: 151]
00:05.760	ID:13	[addr: 01.02.03. top_addr: 01.02. payload/hops: 3 node_load: 56]
00:05.784	ID:7	[addr: 01.03.01. top_addr: 01.03. payload/hops: 3 node_load: 132]
00:05.789	ID:19	[addr: 01.01.03.02. top_addr: 01.01.03. payload/hops: 4 node_load: 25]
00:05.791	ID:20	[addr: 01.01.03.04. top_addr: 01.01.03. payload/hops: 4 node_load: 170]
00:05.809	ID:26	[addr: 01.02.03.01. top_addr: 01.02.03. payload/hops: 4 node_load: 41]
00:05.828	ID:10	[addr: 01.01.03.02.02. top_addr: 01.01.03.02. payload/hops: 5 node_load: 153]
00:05.849	ID:15	[addr: 01.03.01.03. top_addr: 01.03.01. payload/hops: 4 node_load: 54]
00:05.850	ID:8	[addr: 01.03.01.02. top_addr: 01.03.01. payload/hops: 4 node_load: 113]
00:05.850	ID:16	[addr: 01.03.01.01. top_addr: 01.03.01. payload/hops: 4 node_load: 14]
00:05.851	ID:25	[addr: 01.02.03.01.03. top_addr: 01.02.03.01. payload/hops: 5 node_load: 176]
00:05.868	ID:24	[addr: 01.01.03.02.02.01. top_addr: 01.01.03.02.02. payload/hops: 6 node_load: 8]
00:05.868	ID:21	[addr: 01.01.03.02.02.02. top_addr: 01.01.03.02.02. payload/hops: 6 node_load: 180]
00:05.868	ID:22	[addr: 01.01.03.02.02.04. top_addr: 01.01.03.02.02. payload/hops: 6 node_load: 175]
00:05.913	ID:28	[addr: 01.01.03.02.02.04.01. top_addr: 01.01.03.02.02.04. payload/hops: 7 node_load: 140]
00:05.914	ID:29	[addr: 01.01.03.02.02.04.04. top_addr: 01.01.03.02.02.04. payload/hops: 7 node_load: 47]
00:05.937	ID:14	[addr: 01.03.01.02.03. top_addr: 01.03.01.02. payload/hops: 5 node_load: 78]
00:05.964	ID:30	[addr: 01.01.03.02.02.04.01.03. top_addr: 01.01.03.02.02.04.01 payload/hops: 8 node_load: 163]
00:06.044	ID:27	[addr: 01.01.03.02.02.04.01.03.04. top_addr: 01.01.03.02.02.04.01 payload/hops: 9 node_load: 106]
00:06.044	ID:31	[addr: 01.01.03.02.02.04.01.03.02. top_addr: 01.01.03.02.02.04.01 payload/hops: 9 node_load: 166]
00:06.286	ID:23	[addr: 01.01.03.02.01. top_addr: 01.01.03.02. payload/hops: 5 node_load: 156]
00:06.287	ID:18	[addr: 01.01.03.02.05. top_addr: 01.01.03.02. payload/hops: 5 node_load: 10]
00:06.385	ID:9	[addr: 01.03.01.01.02. top_addr: 01.03.01.01. payload/hops: 5 node_load: 119]
00:06.387	ID:12	[addr: 01.03.01.01.03. top_addr: 01.03.01.01. payload/hops: 5 node_load: 12]

Figura 5.5: Información de los nodos tras la inicialización y asignación de etiquetas

Por tanto, ahora que se conocía el grafo en forma de árbol que había producido la red, era ya posible realizar la clasificación de los nodos y distinguir entre los que eran frontera y los que no. La forma estrecha que presentaba la topología supondría la existencia de un gran porcentaje de nodos frontera respecto a la cantidad total de nodos (54,83 %).

Time	Mote	Message
00:08.323	ID:24	//INFO STATISTICS// El nodo 01.01.03.02.02.01. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.492	ID:31	//INFO STATISTICS// El nodo 01.01.03.02.02.04.01.03.02. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.538	ID:6	//INFO STATISTICS// El nodo 01.01.04. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.614	ID:18	//INFO STATISTICS// El nodo 01.01.03.02.05. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.635	ID:4	//INFO STATISTICS// El nodo 01.04. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.674	ID:27	//INFO STATISTICS// El nodo 01.01.03.02.02.04.01.03.04. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.738	ID:14	//INFO STATISTICS// El nodo 01.03.01.02.03. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.750	ID:15	//INFO STATISTICS// El nodo 01.03.01.03. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.755	ID:25	//INFO STATISTICS// El nodo 01.02.03.01.03. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.760	ID:20	//INFO STATISTICS// El nodo 01.01.03.04. tiene 6 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.882	ID:12	//INFO STATISTICS// El nodo 01.03.01.01.03. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.925	ID:29	//INFO STATISTICS// El nodo 01.01.03.02.02.04.04. tiene 4 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:08.996	ID:9	//INFO STATISTICS// El nodo 01.03.01.01.02. tiene 4 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:09.002	ID:5	//INFO STATISTICS// El nodo 01.01.01. tiene 3 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:09.194	ID:21	//INFO STATISTICS// El nodo 01.01.03.02.02.02. tiene 2 vecinos y no ha recibido HLMAC de 0 vecinos: es edge
00:09.233	ID:13	//INFO STATISTICS// El nodo 01.02.03. tiene 4 vecinos y no ha recibido HLMAC de 1 vecinos: es edge

Figura 5.6: Visualización de los nodos de la red clasificados como nodos frontera

¹Es importante recordar que en este proceso el valor inicial se había establecido de forma aleatoria mediante la función `random.rand()`

Es decir, en este caso, se tuvieron que crear numerosas ramificaciones para poder llegar a las zonas más remotas de la red, suponiendo un aumento de la cantidad de nodos extremos que se encontrarían en el esquema. Además, se produjo una asignación de etiquetas relativamente largas para aquellos nodos más lejanos al nodo raíz, implicando por tanto, la existencia de una cuenta mayor de saltos para posibilitar la comunicación entre los anteriores.

Siguiendo la secuencia de procesos del algoritmo, el siguiente paso en ejecutarse fue la notificación de las cargas de cada uno de los nodos a sus vecinos para que todos ellos pudieran completar su lista con las cargas. En la Figura 5.7 se muestra en forma de ejemplo cómo algunos de los nodos (ID 24, 31 y 6) informaron de su valor de carga mediante un paquete de difusión a sus vecinos. También, se visualiza la recepción de los mensajes en cada uno de ellos, indicando el nodo emisor del paquete.

Mote output		
Time	Mote	Message
00:11.324	ID:24	//INFO HANDLE LOAD// carga enviada: 8 de dirección 01.01.03.02.02.01.
00:11.367	ID:29	//INFO INCOMING LOAD// carga recibida: 8 del nodo 0x18.
00:11.368	ID:10	//INFO INCOMING LOAD// carga recibida: 8 del nodo 0x18.
00:11.368	ID:22	//INFO INCOMING LOAD// carga recibida: 8 del nodo 0x18.
00:11.494	ID:31	//INFO HANDLE LOAD// carga enviada: 166 de dirección 01.01.03.02.02.04.01.03.02.
00:11.521	ID:27	//INFO INCOMING LOAD// carga recibida: 166 del nodo 0x1F.
00:11.521	ID:30	//INFO INCOMING LOAD// carga recibida: 166 del nodo 0x1F.
00:11.540	ID:6	//INFO HANDLE LOAD// carga enviada: 65 de dirección 01.01.04.
00:11.591	ID:5	//INFO INCOMING LOAD// carga recibida: 65 del nodo 0x06.
00:11.591	ID:2	//INFO INCOMING LOAD// carga recibida: 65 del nodo 0x06.

Figura 5.7: Envío y recepción de las notificaciones de carga

Según terminó este proceso, cuando cada uno de los nodos comprobó que había completado todos los valores en su lista de vecinos, comenzó el reparto de cargas computacionales. En la ventana de salida de Cooja se fueron mostrando paquete a paquete las cargas que se transferían entre los pares de nodos. Comenzando por los nodos frontera, se puede tomar como ejemplo de aplicación el caso de los nodos con ID 9 y 12.

Inicialmente, estos poseían una carga computacional con valor 119 y 12 respectivamente y ambos eran hijos del nodo con ID 16, con carga inicial de valor 16. Por tanto, se trataba de una situación en la que un nodo padre tenía más de un hijo (ver 4.3.1.4.3) y en la que cada uno de ellos se encontraba en un estado distinto de carga: el primero, con un valor mayor a 100 (ver 4.3.1.4.1) y el segundo, con un valor menor (ver 4.3.1.4.2):

- El nodo con ID 9 procedió a efectuar un traspaso de carga de valor 19 hacia el nodo con ID 16. Despues, escribió en su lista el flujo producido con un valor negativo² y se impuso a sí mismo una carga de 100.
- El nodo con ID 12 informó al padre de que podía absorber carga. El mensaje de traspaso portó una carga igual a -88 mientras que el flujo se añadió a la lista con un valor positivo. También, se actualizó internamente con un valor de 100.

²El flujo indicaría el valor de carga que se está añadiendo o restando en el nodo hijo

```

00:25.849 ID:9 //INFO HANDLE SHARE UP// En este nodo sobra una carga de 19
00:25.853 ID:9 //INFO HANDLE SHARE UP// Carga actualizada: 100
00:25.856 ID:9 //INFO HANDLE SHARE UP// Carga informada: 19
00:27.854 ID:9 Carga actual del nodo: 100
00:27.859 ID:9 --> Vecino padre (flag, carga inicial, in_out) --> 1, 14, -19

```

Figura 5.8: Procedimiento de traspaso en el nodo ID:9

```

00:20.853 ID:12 //INFO HANDLE SHARE UP// En este nodo permite una carga adicional de valor 88
00:20.857 ID:12 //INFO HANDLE SHARE UP// Carga actualizada: 100
00:20.860 ID:12 //INFO HANDLE SHARE UP// Carga informada: -88
00:22.858 ID:12 Carga actual del nodo: 100
00:22.862 ID:12 --> Vecino padre (flag, carga inicial, in_out) --> 1, 14, 88

```

Figura 5.9: Procedimiento de traspaso en el nodo ID:12

Obteniendo la perspectiva que existía desde el nodo padre para todos los traspasos anteriores, se visualiza cómo se actualizó su propia carga de forma casi instantánea con la llegada de cada paquete. Al concluir el proceso, se obtuvo un valor igual a -55 y logró completar su lista de vecinos con los flujos que se habían creado a raíz de las transferencias. En la Figura 5.10 se observa esta modificación, indicando desde qué vecinos se había realizado el traspaso. Se puede comprobar que coincidió con aquellos que habían marcado anteriormente el flag a 0, ya que así se indicaba que estos vecinos se estaban comportando como hijos para el nodo.

```

00:17.833 ID:16 Carga actual del nodo: 14
00:17.837 ID:16 --> Vecino padre (flag, carga, in_out) --> 1, 132, 0
00:17.841 ID:16 --> Vecino hijo (flag, carga, in_out) --> 0, 119, 0
00:17.845 ID:16 --> Vecino hijo (flag, carga, in_out) --> 0, 12, 0
00:17.849 ID:16 --> Vecino padre (flag, carga, in_out) --> -1, 113, (0)
00:17.853 ID:16 --> Vecino padre (flag, carga, in_out) --> -1, 54, (0)
00:20.896 ID:16 //INFO INCOMING SHARE// carga actual del nodo: -74
00:25.876 ID:16 //INFO INCOMING SHARE// carga actual del nodo: -55
00:27.877 ID:16 Carga actual del nodo: -55
00:27.881 ID:16 --> Vecino padre (flag, carga inicial, in_out) --> 1, 132, 0
00:27.886 ID:16 --> Vecino hijo (flag, carga inicial, in_out) --> 0, 119, 19
00:27.891 ID:16 --> Vecino hijo (flag, carga inicial, in_out) --> 0, 12, -88
00:27.896 ID:16 --> Vecino padre (flag, carga inicial, in_out) --> -1, 113, (0)
00:27.901 ID:16 --> Vecino padre (flag, carga inicial, in_out) --> -1, 54, (-46)

```

Figura 5.10: Actualización de los flujos de carga de los hijos en la lista del nodo ID:16

En el siguiente paso, se continuó el procedimiento hacia arriba, siguiendo la ejecución en cadena. En ese instante, el nodo con ID 16 inició el traspaso de carga hacia su padre (ver Apartado 4.3.1.4). Actuando de la misma forma que antes, se actualizó un valor de carga igual a 100 y se completó la lista de vecinos.

```

00:35.904 ID:16 //INFO HANDLE SHARE UP// En este nodo permite una carga adicional de valor 155
00:35.907 ID:16 //INFO HANDLE SHARE UP// Carga actualizada: 100
00:35.911 ID:16 //INFO HANDLE SHARE UP// Carga informada: -155
00:37.908 ID:16 Carga actual del nodo: 100
00:37.913 ID:16 --> Vecino padre (flag, carga inicial, in_out) --> 1, 132, 155
00:37.918 ID:16 --> Vecino hijo (flag, carga inicial, in_out) --> 0, 119, 19
00:37.922 ID:16 --> Vecino hijo (flag, carga inicial, in_out) --> 0, 12, -88
00:37.927 ID:16 --> Vecino padre (flag, carga inicial, in_out) --> -1, 113, (-9)
00:37.932 ID:16 --> Vecino padre (flag, carga inicial, in_out) --> -1, 54, (-46)

```

Figura 5.11: Actualización del flujo de carga del padre en la lista del nodo ID:16

El proceso finalizó cuando los mensajes llegaron hasta el nodo raíz. En este momento todos los nodos de la topología de red debían poseer un valor de carga igual a 100, indicando a modo de depuración que ya habían efectuado su transferencia. En la Figura 5.12 se confirma que se llevó a cabo la ejecución de forma correcta y se presenta la situación de cargas que existía en la topología al concluir el proceso de repartos. El nodo raíz fue el único que podía poseer un valor diferente respecto a los demás nodos y en este caso, terminó sobrecargado.

Time	Mote	Message
01:33.059	ID:1	Carga actual del nodo: 418
01:34.882	ID:15	Carga actual del nodo: 100
01:36.490	ID:21	Carga actual del nodo: 100
01:36.543	ID:18	Carga actual del nodo: 100
01:36.548	ID:24	Carga actual del nodo: 100
01:37.702	ID:6	Carga actual del nodo: 100
01:37.963	ID:14	Carga actual del nodo: 100
01:37.964	ID:9	Carga actual del nodo: 100
01:38.016	ID:8	Carga actual del nodo: 100
01:38.048	ID:11	Carga actual del nodo: 100
01:38.049	ID:16	Carga actual del nodo: 100
01:38.063	ID:7	Carga actual del nodo: 100
01:38.825	ID:25	Carga actual del nodo: 100
01:38.899	ID:26	Carga actual del nodo: 100
01:38.936	ID:13	Carga actual del nodo: 100
01:38.998	ID:3	Carga actual del nodo: 100
01:39.760	ID:4	Carga actual del nodo: 100
01:39.892	ID:5	Carga actual del nodo: 100
01:39.950	ID:23	Carga actual del nodo: 100
01:40.001	ID:20	Carga actual del nodo: 100
01:40.127	ID:29	Carga actual del nodo: 100
01:42.665	ID:27	Carga actual del nodo: 100
01:42.670	ID:31	Carga actual del nodo: 100
01:42.789	ID:30	Carga actual del nodo: 100
01:42.819	ID:28	Carga actual del nodo: 100
01:42.880	ID:22	Carga actual del nodo: 100
01:42.976	ID:10	Carga actual del nodo: 100
01:42.983	ID:12	Carga actual del nodo: 100
01:42.998	ID:19	Carga actual del nodo: 100
01:43.033	ID:17	Carga actual del nodo: 100
01:43.038	ID:2	Carga actual del nodo: 100

Figura 5.12: Situación final de cargas en los nodos de la red

Capítulo 6

Conclusiones y líneas futuras

Como Capítulo de finalización de este trabajo, se expondrá a continuación, en un primer Apartado, las conclusiones principales que han surgido del desarrollo realizado. Se valorará si se han cumplido los objetivos planteados inicialmente y se relacionarán los mismos con los resultados obtenidos. Por consiguiente, se presentarán los caminos a tomar para realizar trabajos en un futuro como opciones de continuación de este proyecto de investigación. Se analizarán las posibles vías que fueron derivando de la etapa de desarrollo y la viabilidad de su implementación.

6.1 Conclusiones

A modo de resumen, se puede expresar que se ha conseguido el cumplimiento del fin principal de este trabajo: desarrollar un nuevo algoritmo distribuido, denominado como Dedenne, para llevar a cabo un reparto de cargas computacionales en redes de sensores y dispositivos IoT.

Los resultados obtenidos mediante múltiples simulaciones en la aplicación de Contiki soportan la justificación del correcto funcionamiento del algoritmo. Sin embargo, no fue posible llevar a cabo su implementación a nivel de hardware como se planteó inicialmente. En concreto, se decidió suprimir el proceso de despliegue de Contiki a dispositivos Raspberry Pi, debido a su alta complejidad y, principalmente, a la falta de disposición de tiempo y de tarjetas 802.15.4 para realizar las pruebas necesarias. Cabe destacar también, que se descubrió en la fase del desarrollo que el despliegue de Contiki con dichas tarjetas no era trivial¹. Por estos motivos, se optó por enfocar el TFG en una programación más intensiva del algoritmo y en evaluar el mismo en base a una implementación a nivel de software.

La elaboración de cada uno de los módulos que contiene Dedenne fue posible gracias al planteamiento continuo de las funcionalidades que se pretendían aplicar. Esto supuso en ocasiones la reorganización del algoritmo y de su programación. Fueron necesarias constantes modificaciones en el trazado del código con el fin de cumplir con las características del nuevo modelo distribuido. Así, de forma progresiva, se fueron transformando las propuestas de programación de cada uno de los procesos para perfeccionar el funcionamiento de los mismos y optimizarlos.

¹En relación con otras herramientas, el despliegue se podía realizar de una forma más sencilla en Contiki, pero seguía implicando una alta complejidad.

Además, en la programación fue imprescindible abordar con incidencias y problemas de eficiencia, tanto en los razonamientos que se fueron produciendo, como en el diseño de la implementación de cada parte. Según se ha detallado en cada uno de los apartados, la evolución fue favorable para lograr los objetivos. A nivel de desarrollo, se puede determinar que la mayor dificultad se percibió en el planteamiento del inicio del proceso de reparto de las cargas computacionales en los nodos. En otros términos, fue imprescindible la prueba y ejecución de diversas opciones de código hasta concluir con el método correcto.

6.2 Líneas futuras

Como se ha comentado a lo largo del TFG, las redes inteligentes de electricidad son de gran interés en la actualidad y su estudio está en constante crecimiento. Cada vez existe un mayor asentamiento y es por ello, que este proyecto podría presentar una posible prolongación, definiendo diferentes vías de trabajo futuro.

6.2.1 Mejoras en la programación del algoritmo

Durante el desarrollo del código, surgió el planteamiento de ciertas modificaciones que se podrían realizar con el fin de optimizar los procesos y mejorar ciertos aspectos de la estructura del algoritmo:

6.2.1.1 Cambio de la versión base de IoTorii

En topologías en las que existían multitud de nodos en áreas reducidas, apareció la posibilidad de que no se produjera de forma correcta el proceso de asignación de etiquetas. Debido a la limitación de la capacidad de los paquetes impuesta en IEEE 802.15.4 (ver Sección 2.2.2.1), podría ocurrir que algunos de los nodos no informaran de su dirección hacia sus vecinos.

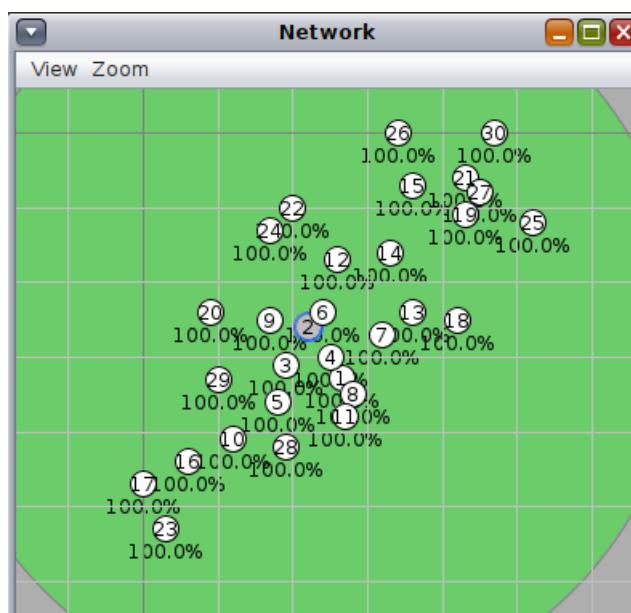


Figura 6.1: Ejemplo de topología con una alta densidad de nodos

En consecuencia, tampoco se conseguiría identificar de forma correcta a aquellos nodos que eran frontera en el esquema de red. Se puede apreciar en la ventana de resultados de la Figura 6.2 cómo se producía esta incidencia al simular un ejemplo de topología con una alta densidad de nodos (ver Figura 6.1). Al encontrarse el total de los nodos de la red en un mismo área, se dio una situación en la que todos eran vecinos entre sí. Por lo tanto, no se recibieron la mayoría de los mensajes HLMAC por la saturación de la red.

Mote output		
Time	Mote	Message
00:08.322	ID:24	//INFO STATISTICS// El nodo 01.07. tiene 29 vecinos y no ha recibido HLMAC de 16 vecinos: no es edge
00:08.360	ID:8	//INFO STATISTICS// El nodo 01.11.0A. tiene 29 vecinos y no ha recibido HLMAC de 21 vecinos: no es edge
00:08.429	ID:28	//INFO STATISTICS// El nodo 01.04.02. tiene 28 vecinos y no ha recibido HLMAC de 23 vecinos: no es edge
00:08.478	ID:11	//INFO STATISTICS// El nodo 01.18. tiene 29 vecinos y no ha recibido HLMAC de 20 vecinos: no es edge
00:08.524	ID:2	//INFO STATISTICS// El nodo 01.0C.01. tiene 29 vecinos y no ha recibido HLMAC de 23 vecinos: no es edge
00:08.532	ID:30	//INFO STATISTICS// El nodo 01.17.1C. tiene 24 vecinos y no ha recibido HLMAC de 21 vecinos: no es edge
00:08.538	ID:6	//INFO STATISTICS// El nodo 01.0C.12. tiene 29 vecinos y no ha recibido HLMAC de 25 vecinos: no es edge
00:08.614	ID:18	//INFO STATISTICS// El nodo 01.0L.17. tiene 29 vecinos y no ha recibido HLMAC de 25 vecinos: no es edge
00:08.636	ID:4	//INFO STATISTICS// El nodo 01.10.19. tiene 29 vecinos y no ha recibido HLMAC de 22 vecinos: no es edge
00:08.669	ID:1	//INFO STATISTICS// El nodo 01. tiene 29 vecinos y no ha recibido HLMAC de 29 vecinos: no es edge

Figura 6.2: Resultado del proceso de identificación de nodos frontera

En el Apartado 3.1.1 se había especificado que el desarrollo del algoritmo Dedenne tenía como base la versión original de IoTori (iotorii-n-hlmac). Por ello, para solucionar el defecto, se debería usar como extensión su versión posterior (iotorii-n-hlmac-2), en la cual se aseguraría la asignación de etiquetas para todos los nodos de la red mediante un reenvío de los mensajes HLMAC.

6.2.1.2 Mejora de la asignación de etiquetas para nodos remotos

En la fase de pruebas del funcionamiento del algoritmo, se percibieron incidencias en el envío de los mensajes para informar de las etiquetas cuando se daba la condición de que un nodo determinado encontraba un solo vecino y además, se situaba a cierta lejanía del resto de la red.

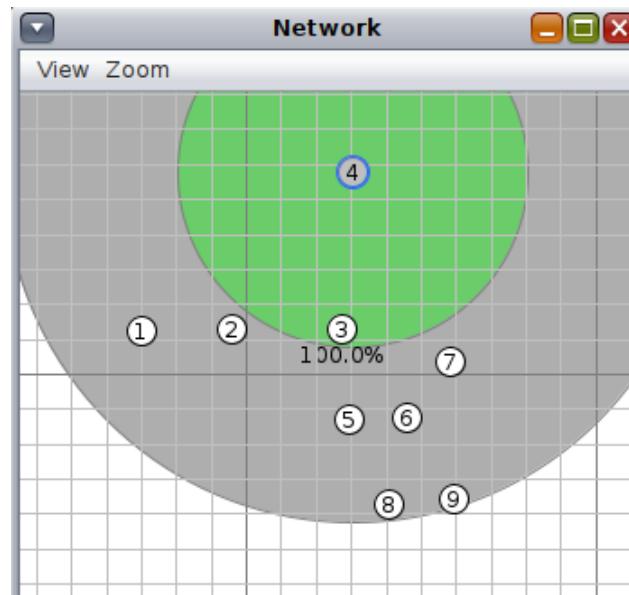


Figura 6.3: Ejemplo de topología en la que uno de los nodos es remoto

En el protocolo base, IoTorii, ya se originaba este inconveniente, por lo que en este caso, sería necesario prolongar el estudio del proceso de asignación de direcciones. Para exponer el problema gráficamente, se puede tomar como ejemplo la topología de la Figura 6.3. En ella, existía un nodo remoto (ID:4) que se conectaba al resto de la red por medio de un único nodo vecino (ID:3). En este caso, se demostró que no recibía el mensaje de asignación de etiqueta (ver Figura 6.4) y, por lo tanto, no participaría en sí en los procesos del esquema de red.

Mote output		
Time	Mote	Message
00:08.353	ID:8	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:08.516	ID:2	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:08.531	ID:6	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:08.629	ID:4	//INFO STATISTICS// n_hello: 1, n_sethlmac: 0
00:08.662	ID:1	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:08.682	ID:7	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:08.989	ID:9	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:08.995	ID:5	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1
00:09.180	ID:3	//INFO STATISTICS// n_hello: 1, n_sethlmac: 1

Figura 6.4: Visualización del número de etiquetas asignadas a cada nodo

6.2.1.3 Conversión de paquetes multicast en unicast

En el Apartado 4.3.1.4.3 se especificaba que, para programar la transmisión de los mensajes de traspaso de carga computacional en cada nodo, se decidió aplicar como base la forma de envío de los mensajes Hello y HLMAC. En otros términos, se planteó un envío por difusión hacia todos sus vecinos, por lo que cada vez que se recibiera un paquete, habría que comprobar si el mismo iba dirigido al nodo en cuestión (según el estado de los flags). Por lo tanto, en caso afirmativo, habría que procesarlo.

La modificación hacia un envío unicast eliminaría este proceso, y en consecuencia, optimizaría el código del algoritmo. Además, al requerir un menor número de mensajes para llevar a cabo el proceso de reparto, se reduciría la carga de la red.

6.2.2 Posibles implementaciones futuras

Según se ha especificado en detalle en el Apartado 6.1, referente a las conclusiones, la evaluación del desarrollo del algoritmo Dedenne se basó en la implementación del mismo en Contiki. Mediante los resultados obtenidos gráficamente en el programa, se logró evidenciar su correcto funcionamiento.

Inicialmente, se pretendía llevar a cabo un despliegue a nivel de hardware, creando un entorno de redes distribuidas mediante una infraestructura basada en tecnología Raspberry Pi. Debido a la decisión final de omitir este procedimiento como parte del trabajo, se propuso como posible opción dejar esta implementación como línea a seguir en un futuro.

Por otro lado, es preciso hacer hincapié en que las simulaciones realizadas en Contiki se basaron en escenarios sin pérdidas. Realmente, una de las características que presentaba el modelo de red LLN es que, debido a sus limitaciones en capacidad y en ancho de banda, existía una probabilidad alta de pérdidas de paquetes. Teniendo en cuenta este factor, sería de utilidad proseguir la ejecución de simulaciones en escenarios con pérdidas y realizar un análisis en comparativa.

Bibliografía

- [1] (2022) Energy management system. <https://www.gepowerconversion.com/product-solutions/EMS-Energy-Management-System>.
- [2] J. C. H. Carlos Andrés Díaz Andrade, “Smart grid: Las tics y la modernización de las redes de energía eléctrica â estado del arte,” *Revista ST*, no. 9, pp. 53–81, 2011.
- [3] (2022) What is a dso? <https://www.edsoforsmartgrids.eu/home/why-smart-grids/>.
- [4] S. Gillani, F. Laforest, and G. Picard, “A generic ontology for prosumer-oriented smart grid,” *CEUR Workshop Proceedings*, vol. 1133, 2014.
- [5] Cecilia Pastorino. (2018) Blockchain: qué es. <https://www.welivesecurity.com/la-es/2018/09/04/blockchain-que-es-como-funciona-y-como-se-esta-usando-en-el-mercado/>.
- [6] (2018) Smart grid basada en iot y blockchain. <https://iotfutura.com/2018/12/smargrid-basada-en-iot-y-blockchain/>.
- [7] R. H. A.A.G. Agung, “Blockchain for smart grid,” *Journal of King Saud University â Computer and Information Sciences*, no. 34, p. 666â675, 2022.
- [8] B. Ghaleb, A. Al-Dubai, E. Ekonomou, A. Alsarhan, Y. Nasser, L. Mackenzie, and A. Boukerche, “A survey of limitations and enhancements of the ipv6 routing protocol for low-power and lossy networks: A focus on core operations,” *IEEE Communications Surveys Tutorials*, vol. PP, 2018.
- [9] P. Rodríguez. (2013) Redes plc. <https://www.xatakahome.com/la-red-local/redes-plc-i-que-son-y-para-que-sirven>.
- [10] N. D. K. S. Carlos M García Algora, Vitalio Alfonso Reguera, “Review and classification of multi-channel mac protocols for low-power and lossy networks,” *IEEE*, vol. 5, pp. 19 536–19 561, 2017.
- [11] A. E. Miquel Oliver, “Study of different csma/ca ieee 802.11-based implementations, universitat politÃ“cnica de catalunya,” 1999.
- [12] K. Bhandari, I.-h. Ra, and G. Cho, “Multi-topology based qos-differentiation in rpl for internet of things applications,” *IEEE Access*, vol. PP, pp. 1–1, 2020.
- [13] I. Alsukayti, “The support of multipath routing in ipv6-based internet of things,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, p. 2208, 2020.
- [14] M. Tabari and Z. Mataji, “Detecting sinkhole attack in rpl-based internet of things routing protocol,” 2020.
- [15] D. P. Nangina and D. Ganesh, “Rotatory shortcut tree routing in zigbee networks,” *International Journal of Computer Sciences and Engineering*, vol. 5, pp. 143–146, 12 2017.

- [16] I. M. E. Rojas, J. Alvarez-Horcajo. (2019) Iotorii implementation in github. <https://github.com/gistnetserv-uah/>.
- [17] W. Commons, “File:raspberrypi 4 model b.svg — wikimedia commons, the free media repository,” 2020. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:RaspberryPi_4_Model_B.svg&oldid=444138243
- [18] (2021) Docker namespace vs cgroup. what is namespace? <https://bikramat.medium.com/namespace-vs-cgroup-60c832c6b8c8>.
- [19] (2022) Sql right join. <https://www.tutorialrepublic.com/sql-tutorial/sql-right-join-operation.php>.
- [20] (2022) Comenzando con raspberry pi. <https://projects.raspberrypi.org/es-ES/projects/raspberry-pi-getting-started>.
- [21] (2017) Smart grid. <https://blog.gruponovelec.com/electricidad/como-funciona-smart-grid/>.
- [22] (2020) Prosumidor. <https://www.miwnergia.com/la-revolucion-del-prosumidor-energetico/>.
- [23] (2022) Iot - internet of things. <https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html>.
- [24] E. Rojas, H. Hosseini, C. Gomez, D. Carrascal, and J. Rodrigues Cotrim, “Iotorii: Outperforming rpl with scalable routing,” 2020. [Online]. Available: <https://dx.doi.org/10.21227/cjw4-kr75>
- [25] A. Arroyo Castro, “Protocolo de enrutamiento rpl,” 2019. [Online]. Available: <https://bonga.unisimon.edu.co/handle/20.500.12442/3508>
- [26] (2022) Smart grids. <https://www.fundacionendesa.org/es/educacion/endesa-educa/recursos/smart-grid>.
- [27] V. S. K. O. y. L. M. D. Faquir, N. Chouliaras, “Cybersecurity in smart grids, challenges and solutions,” *Electronics and Electrical Engineering (AIMS)*, no. 1, pp. 24–37, 2021.
- [28] *Tutorial T-06: Denial of Service Attacks on the Smart Grid: Classification, Solutions, and Challenges*. <https://icc2016.ieee-icc.org/sites/icc2016.ieee-icc.org/files/u44/T6.pdf>, 2016.
- [29] Javier Jiménez. (2021) Ataques arp spoofing. <https://www.redeszone.net/tutoriales/redes-cable/ataques-arp-spoofing-evitar/>.
- [30] (2021) Wi-fi eavesdropping. <https://www.redeszone.net/tutoriales/redes-wifi/ataques-wifi-eavesdropping-redes/>.
- [31] (2022) Smart grids. <https://www.iberdrola.com/conocenos/energetica-del-futuro/smart-grids/>.
- [32] (2021) Smart cities. <https://www.aura-energia.com/smart-cities-empresas-de-luz/>.
- [33] (2022) Almacenamiento de energía. <https://www.segurorenovables.com/energia-renovable/almacenamiento-de-energia/>.
- [34] (2022) Smart grids. <https://aserta.com.es/smart-grids-inteligencia-para-la-transicion-energetica/>.
- [35] (2022) Dso. <https://www.smartgridsinfo.es/dso>.
- [36] (2022) What is blockchain? <https://www.ibm.com/es-es/topics/what-is-blockchain>.

- [37] F. Lombardi, L. Aniello, S. D. Angelis, A. Margheri, and V. Sassone, “A blockchain-based infrastructure for reliable and cost-effective iot-aided smart grids,” in *IoT 2018*, 2018.
- [38] L. Fernandez, “Smart contracts: Contratos basados en blockchain,” 2022. [Online]. Available: <https://www.bbva.com/es/smart-contracts-los-contratos-basados-blockchain-no-necesitan-abogados/>
- [39] (2022) Low power and lossy network. https://itlaw.fandom.com/wiki/Low_power_and_lossy_network.
- [40] M. J. Martínez Morfa, “Mecanismo de asociación dinámica para redes lln bajo el estándar ieee 802.15.4,” 2021.
- [41] D. Culler, J. Hui, J. Vasseur, and V. Manral, “An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL),” 2012. [Online]. Available: <https://www.rfc-editor.org/info/rfc6554>
- [42] (2013) Lbr in lln network. <http://acronymsandslang.com/definition/7888248/LBR-meaning.html>.
- [43] C. Bormann, M. Ersue, and A. KerÄonen, “Terminology for Constrained-Node Networks,” RFC 7228, 2014. [Online]. Available: <https://www.rfc-editor.org/info/rfc7228>
- [44] B. C. Villaverde, D. Pesch, R. De Paz Alberola, S. Fedor, and M. Boubekeur, “Constrained application protocol for low power embedded networks: A survey,” in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012.
- [45] “El estándar ieee 802.15.4,” http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo4.pdf.
- [46] (2018) Csma/ca: protocolo para redes inalÃ¡mbricas. <https://www.ionos.es/digitalguide/servidores/know-how/csmaca-protocolo-de-acceso-al-medio-para-redes-inalambricas>.
- [47] G. Mulligan, “The 6lowpan architecture,” in *Proceedings of the 4th Workshop on Embedded Networked Sensors*, ser. EmNets ’07. New York, NY, USA: Association for Computing Machinery, 2007, p. 78â82. [Online]. Available: <https://doi.org/10.1145/1278972.1278992>
- [48] W. Gan, Z. Shi, C. Zhang, L. Sun, and D. Ionescu, “Merpl: A more memory-efficient storing mode in rpl,” in *2013 19th IEEE International Conference on Networks (ICON)*, 2013, pp. 1–5.
- [49] J. Maldonado. (2020) What is a dag. <https://es.cointelegraph.com/explained/what-is-a-dag-and-how-do-it-work>.
- [50] X. Zhong and Y. Liang, “Scalable downward routing for wireless sensor networks actuation,” *IEEE Sensors Journal*, vol. 19, no. 20, pp. 9552–9560, 2019.
- [51] H.-S. Kim, H. Cho, H. Kim, and S. Bahk, “Dt-rpl: Diverse bidirectional traffic delivery through rpl routing protocol in low power and lossy networks,” *Computer Networks*, vol. 126, pp. 150–161, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389128617302748>
- [52] L. Wadhwa, R. S. Deshpande, and V. Priye, “Extended shortcut tree routing for zigbee based wireless sensor network,” *Ad Hoc Networks*, vol. 37, pp. 295–300, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1570870515001912>
- [53] R. M. Brigitta and P. Samundiswary, “A survey on shortcut tree routing protocols for zibee based wireless networks,” in *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2016, pp. 441–445.

- [54] “Iotorii: Beating rpl with simpler routing,” 2018. [Online]. Available: <https://dx.doi.org/10.21227/ha0r-mh78>
- [55] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 455–462.
- [56] A. Velinov and A. Mileva, “Running and testing applications for contiki os using cooja simulator,” 2016, <http://eprints.ugd.edu.mk/16096/>.
- [57] (2018) Platform sky. <https://github.com/contiki-ng/contiki-ng/wiki/Platform-sky>.
- [58] A. Medina, I. Matta, and J. Byers, “Brite: Boston university representative internet topology generator: A flexible generator of internet topologies,” 2000. [Online]. Available: <https://open.bu.edu/handle/2144/3752>
- [59] (2022) Mininet: An instant virtual network on your laptop. <http://mininet.org/>.
- [60] D. Kreutz, F. M. V. Ramos, P. E. VerÅssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.
- [61] (2022) Linux namespaces. https://en.wikipedia.org/wiki/Linux_namespaces.
- [62] Wikipedia, “Raspberry pi — wikipedia, la enciclopedia libre,” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=143571372
- [63] R. SolÃ©. (2021) Raspberry pi: Crea proyectos diy por muy poco dinero. <https://www.profesionalreview.com/2021/07/18/que-es-raspberry-pi/>.
- [64] (2022) Raspberry pi 4. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>.
- [65] D. Quiroga. (2020) Namespaces. <https://clibre.io/blog/por-secciones/hardening/item/384-namespaces-aislar-los-procesos-de-linux-en-sus-propios-entornos-de-sistema>.
- [66] (2020) Iotorii. https://github.com/NETSERV-UAH/IoTorii/tree/master/Contiki-ng_4_2.
- [67] (2022) Raspberry pi documentation. <https://www.raspberrypi.com/documentation/computers/getting-started.html>.

Apéndice A

Manual de usuario

A.1 Introducción

Como motivo de facilitar la implementación del algoritmo Dedenne, en este capítulo se van a desarrollar de forma detallada los procedimientos de instalación del entorno. La herramienta esencial, conforme se ha comentado en anteriores capítulos, será el software de simulación de redes de baja potencia Contiki. Por ello, se hará especial hincapié en su proceso de instalación y configuración en Linux.

Por otro lado, en otro apartado, se introducirá el método de instalación seguido para posibilitar implementaciones a nivel de hardware mediante el uso de placas Raspberry Pi 4. Se especificarán además, las interconexiones a realizar para poder visualizar la interfaz gráfica a través de un monitor.

A.2 Instalación del entorno de simulación

A.2.1 Virtualización del sistema operativo Linux

Memory	1.9 GiB
Processor	AMD® Ryzen 5 3400g with radeon vega graphics × 2
Graphics	llvmpipe (LLVM 12.0.0, 256 bits)
Disk Capacity	37.6 GB
OS Name	Ubuntu 20.04.1 LTS
OS Type	64-bit
GNOME Version	3.36.3
Windowing System	X11
Virtualization	VMware

Figura A.1: Especificaciones del sistema operativo y del equipo

El principal requerimiento que necesita nuestro equipo para disponer de Contiki es la instalación previa de una máquina virtual que soporte un archivo con extensión .iso de Ubuntu. En este TFG se ha procedido a emplear el software VMware¹ con la versión del sistema operativo Ubuntu 20.04 LTS² como se puede ver en la figura A.1.

A.2.2 Instalación y configuración de Contiki

Contiki posibilita dos procedimientos diferentes de instalación. Por un lado, se puede instalar con contenedores (Docker), realizando particiones de los recursos del kernel y por otro lado, de forma nativa para hacer uso de Cooja. Como se ha comentado en el apartado 3.2 del capítulo dedicado a la preimplementación, en el presente TFG se ha seleccionado la segunda opción. Por ello, se expondrá a continuación la secuencia de pasos que se ha seguido para realizar la instalación [66] en Linux³.

En primer lugar, es recomendable proceder en Linux a actualizar los paquetes ya instalados con sus últimas versiones. Se necesitará descargar e instalar el paquete de java para poder después compilar los ficheros de código en Cooja:

```
$ sudo apt update
$ sudo apt install default-jdk ant
```

Es esencial clonar el proyecto desde el repositorio de GitHub y para ello, se necesita instalar previamente el paquete git en nuestra máquina:

```
$ sudo apt-get install git
$ git clone https://github.com/contiki-ng/contiki-ng.git
```

Una vez realizados los pasos anteriores, ya es posible acceder al software. Por último, se debe acceder al directorio de Contiki y actualizar sus módulos:

```
$ cd contiki-ng
$ git submodule update --init --recursive
```

Como se comenta en la sección 2.3.1.1, la cual hace referencia a la herramienta Cooja, para ejecutar la misma se procede a insertar los comandos:

```
$ cd contiki-ng/tools/cooja
$ ant
```

En el mismo apartado mencionado, se visualiza una captura de la ventana de inicio de la herramienta (ver figura 2.17). Además, es posible visualizar las características del software Contiki y de la versión instalada en nuestra máquina. Será necesario acceder a un directorio en el que se encuentre un programa compuesto por ficheros de código y un archivo makefile.

¹<https://www.vmware.com/es/products.html>

²<https://ubuntu.com/#download>

³Procedimiento de instalación válido también para Raspberry Pi OS

A modo de ejemplo se ha utilizado el programa Hello World. Como se aprecia en la figura A.2, se indica la versión de Contiki que se ha instalado en la línea CONTIKI_VERSION_STRING.

```
paula@ubuntu:~/Documents/contiki-ng/examples/hello-world$ make viewconf
TARGET not defined, using target 'native'
----- Make variables: -----
##### "TARGET": native
##### "BOARD": 
##### "MAKE_MAC": MAKE_MAC_NULLMAC
##### "MAKE_NET": MAKE_NET_IPV6
##### "MAKE_ROUTING": MAKE_ROUTING_RPL_LITE
----- C variables: -----
##### "PROJECT_CONF_PATH": ><
##### "CONTIKI_VERSION_STRING": == "Contiki-NG-release/v4.7-45-g705541797-dirty"
##### "FRAME802154_CONF_VERSION": == FRAME802154_IEEE802154_2006
##### "IEEE802154_CONF_PANID": == 0xabcd
##### "IEEE802154_CONF_DEFAULT_CHANNEL": == 26
##### "RPL_CONF_MOP": -> RPL_MOP_NON_STORING
##### "RPL_CONF_OF_OCP": -> RPL_OCP_MRHOFR
##### "QUEUEBUF_CONF_NUM": == 64
##### "NBR_TABLE_CONF_MAX_NEIGHBORS": == 300
##### "NETSTACK_MAX_ROUTE_ENTRIES": == 300
##### "UIP_CONF_BUFFER_SIZE": == 1280
##### "UIP_CONF_UDP": == 1
```

Figura A.2: Características de la configuración de Contiki

A.2.2.1 Compilación en Cooja

Para llevar a cabo el proceso de compilación de cualquier programa en Cooja es imprescindible haber definido anteriormente la ruta de instalación de la última versión de Java en la máquina de Linux. En otros términos, Cooja requiere conocer el valor de la variable \$JAVA_HOME para funcionar correctamente.

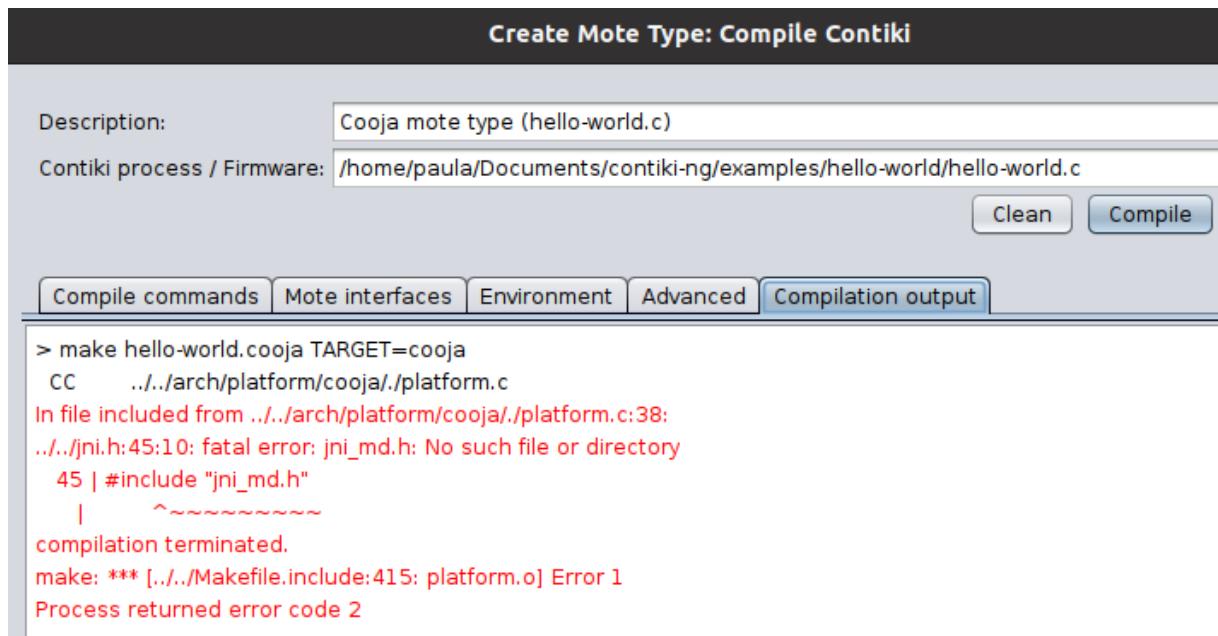


Figura A.3: Error de compilación en Cooja

En la figura A.3 se puede observar en la ventana de salida el error de compilación que se produce cuando no encuentra la carpeta. Por lo tanto, es preciso insertar el siguiente comando en el fichero `./profile` de la máquina para definir `$JAVA_HOME` de forma permanente.

```
$ export JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"
```

A.3 Configuración para la implementación hardware

Para llevar a cabo una implementación a nivel hardware y poder realizar pruebas en placas, en concreto en Raspberry Pi 4, es necesario seguir una secuencia de pasos.

A.3.1 Instalación del sistema operativo para Raspberry Pi

Principalmente, una Raspberry Pi requiere de un sistema operativo⁴ para funcionar. Por ello, se debe proceder a su instalación [67], descargando el software Raspberry Pi Imager⁵. Este servirá de soporte para introducir el sistema operativo en la placa mediante una tarjeta microSD. Una vez se inicializa el programa, se debe seleccionar tanto el sistema operativo como la tarjeta donde se almacenará el mismo para poder después, arrancar el proceso de escritura.



Figura A.4: Instalación y escritura de Raspberry Pi OS

A.3.2 Establecimiento del entorno hardware

Para iniciar el funcionamiento de la Raspberry Pi, se requiere en primera instancia, aplicar una serie de interconexiones de diferentes accesorios con esta. Una vez realizado este paso, ya será posible inicializar el sistema operativo a través de un monitor.

⁴Por defecto, el sistema operativo oficial, Raspberry Pi OS, previamente denominado como Raspbian

⁵<https://www.raspberrypi.com/software/>

Si el procedimiento de configuración se origina de forma correcta se obtendrá como resultado la ventana mostrada en la Figura A.5. Por consiguiente, se instalará el software Contiki en el sistema operativo de la tarjeta Raspberry Pi, siguiendo la secuencia de pasos expuesta en el Apartado A.2.2.

Instalación Completada

Tu Rapsberry Pi está configurada y lista para funcionar.

Pulsa "Restart" para reiniciar ahora tu Pi y que las nuevas configuraciones tengan efecto, o presiona "Later" para cerrar el asistente y reiniciar Pi tu mismo.

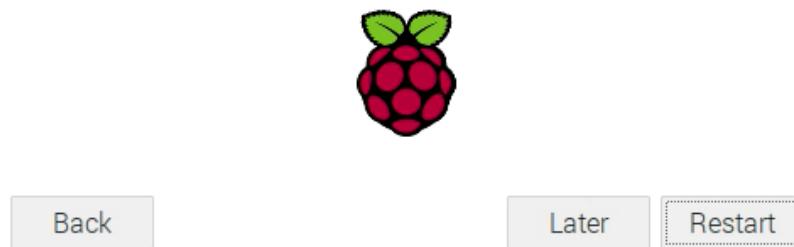


Figura A.5: Mensaje de finalización de la configuración [20]

A continuación, se indican las interconexiones elaboradas, siguiendo el orden de izquierda a derecha que presenta en la Figura A.6:

- Tarjeta microSD con Raspberry Pi OS instalado.
- Cable de alimentación de tipo C.
- Cable HDMI para conectar la Raspberry a un monitor y poder visualizar su interfaz gráfica.
- Cable USB para entrada de teclado
- Cable USB para entrada de ratón

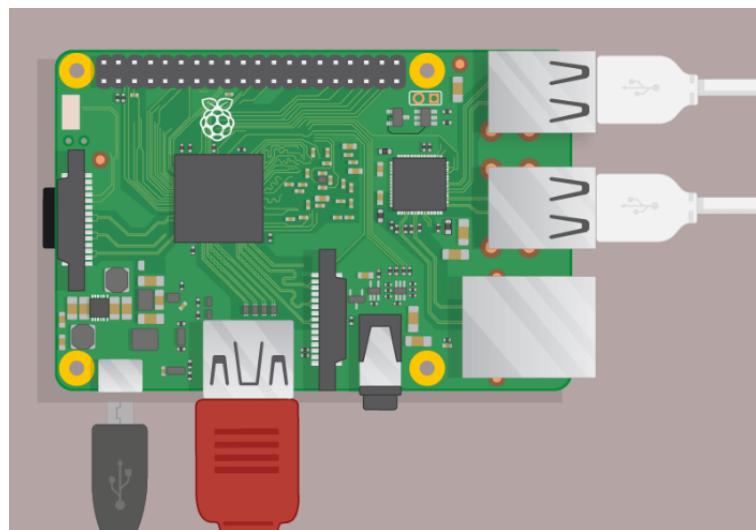


Figura A.6: Esquema de conexiones en la tarjeta Raspberry Pi [20]

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR

