ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN

(Internet of the Things service provision on 6LoWPAN sensor networks)

Para acceder al Titulo de

INGENIERO DE TELECOMUNICACIÓN

Autor: Óscar Vadillo Gutiérrez

Diciembre - 2014

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Vadillo Gutiérrez, Óscar
Director del PFC: Lanza Calderón, Jorge
Título: "Provisión de servicios de la Internet de las Cosas sobre redes de sensors basadas en 6LoWPAN"
Title: "Internet of the Things service provision on 6LoWPAN sensor networks"

Presentado a examen el día: 12 de diciembre de 2014

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Sánchez González, Luis
Secretario (Apellidos, Nombre): Lanza Calderón, Jorge
Vocal (Apellidos, Nombre): Mediavilla Bolado, Maria Elena

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

"Nuestra recompensa se encuentra en el esfuerzo y no en el resultado. Un esfuerzo total es una victoria completa."

- Mahatma Gandhi

Resumen ejecutivo

El interés por las redes de sensores inalámbricas ha ido creciendo durante los últimos años, permitiendo así la interconexión y el intercambio de información con dispositivos de bajo coste, dando lugar a lo que se conoce como la Internet de las Cosas. Aunque el paradigma de dicho concepto plantea que cualquier dispositivo estará conectado a Internet, las propias capacidades de los nodos y la escasez de direcciones *Internet Protocol Version 4 (IPv4)* ha hecho inviable hacer realidad completamente esta aproximación, siendo necesario el planteamiento de metodologías centralizadas por medio de pasarelas o *gateways* intermedios.

Sin embargo, es el despliegue de redes basadas en *Internet Protocol Version 6 (IPv6)* el que parece que definitivamente ha abierto la vía hacia una Internet de las Cosas donde se permita la conectividad directa entre cualquier dispositivo, que además estará unívocamente identificado. El propósito de este trabajo es el despliegue real de una red de sensores sobre el estándar *IPv6 over Low power Wireless Personal Area Networks (6LoWPAN)*. Tras un previo estudio del estándar y análisis de las ventajas que se consiguen con el uso del mismo, este proyecto considera distintas soluciones abiertas disponibles a día de hoy, teniendo como meta el despliegue real de una red de este tipo. Además de llevar a cabo dicho despliegue y analizar el comportamiento de la red, se evaluarán y desarrollarán alternativas para encontrar la mejor solución de abastecer de servicios a los nodos de la misma.

Palabras clave: Redes de sensores inalámbricas, 6LoWPAN, IEEE 802.15.4, TCP/IP, IPv6, Internet de las Cosas, REST, HTTP, CoAP, OML

Abstract

The interest in Wireless Sensor Networks (WSN) has been growing for the last few years, allowing the interconnection and data exchange with low-cost devices, making a reality the concept known as Internet of Things (IoT). Despite the fact that IoT suggests that all devices will be connected to the Internet, the nodes' features and the limitations of Internet Protocol Version 4 (IPv4) has made it impossible to enable end to end communications, forcing the use of centralized networks with gateways.

However, the deployment of networks based on Internet Protocol Version 6 (IPv6) seems to have opened the way to an IoT where end to end connectivity is enabled and where each device is univocally identified. In this sense, after analysing the state of the art, this project aims at deploying an IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) network. The work also includes the experimental assessment of different alternatives for service provision on 6LoWPAN networks.

Keywords: Wireless Sensor Network, 6LoWPAN, IEEE 802.15.4, TCP/IP, IPv6, Internet of Things, REST, HTTP, CoAP, OML

Agradecimientos

Quiero aprovechar estas líneas para agradecer a todas las personas que me han ayudado y apoyado durante toda esta etapa académica, esas personas que son las culpables de que hoy esté aquí presentando este trabajo.

En primer lugar dar las gracias a mi familia, en especial a mis padres Laura y José Manuel por haberme facilitado siempre todo lo necesario para que llegase hasta aquí. También a mi hermana Raquel por su paciencia conmigo durante todos estos años.

Asimismo, agradecer a mi tutor de proyecto, Jorge Lanza, por darme la posibilidad de realizar este trabajo, con el que además de ampliar los conocimientos adquiridos en la carrera, me ha permitido llevarlos a la práctica. Aún recuerdo el día de: *"A partir de ahora, puedes hacer lo que quieras con las placas, menos pisarlas..."*

Por otra parte y NO menos importante, dar las gracias a todos mis amigos que han sido siempre una fuente muy importante de apoyo y motivación. Gracias Héctor, Laura, Bea, Fran y Rafa por ser tan únicos y especiales. Gracias Jhoe, Irene, Javier, Diego, Mario y Txavi por todos los momentos que han hecho que esta carrera sea mucho más interesante y llevadera. Gracias a ti también, Andrea.

Índice general

Resumen ejecutivo iv

Abstract v

Agradecimientos vi

Índice general vii

Índice de figuras x

Índice de tablas xiii

Índice de cuadros de texto xiv

Índice de acrónimos xv

1. Introducción 1

1.1. Motivación . . . 2

1.2. Objetivos . . . 3

1.3. Estructura de la memoria . . . 3

2. Redes de Sensores 5

2.1. Conceptos generales . . . 6

2.2. Arquitectura funcional de un sensor . . . 6

2.3. Arquitecturas de red . . . 8

2.3.1. Arquitectura centralizada . . . 8

2.3.2. Arquitectura distribuida . . . 8

2.4. Topologías de red . . . 8

2.4.1. Estrella . . . 9

2.4.2. *Peer-to-peer* . . . 9

2.5. Protocolos de comunicación . . . 10

2.5.1. IEEE 802.15.4 . . . 10
2.5.1.1. Capa física . . . 11
2.5.1.2. Capa de enlace . . . 12
2.5.1.3. Elementos de red . . . 13
2.5.2. IPv6 . . . 14
2.5.2.1. Formato del paquete . . . 14
2.5.2.2. Direccionamiento . . . 15
2.5.2.3. ICMPv6 . . . 17
2.5.2.4. Neighbor Discovery . . . 17
2.5.3. 6LoWPAN . . . 18
2.5.3.1. Dispositivos y Arquitecturas . . . 19
2.5.3.2. Formato de paquete . . . 20
2.5.3.3. Direccionamiento mallado o *mesh addressing* . . . 21
2.5.3.4. Fragmentación y reensamblado . . . 22
2.5.3.5. Compresión de cabeceras . . . 23
Compresión HC1-HC2 . . . 24
Compresión basada en contexto: IPHC-NHC . . . 26
2.5.3.6. Neighbor Discovery . . . 29
2.5.3.7. Encaminamiento . . . 30
2.5.3.8. Seguridad . . . 32
2.6. Plataformas de implementación . . . 32

3. Despliegue de la red **34**
3.1. Descripción del escenario . . . 34
3.2. Plataforma de desarrollo . . . 35
3.2.1. *Software* - Contiki OS 2.7 . . . 36
3.2.1.1. Arquitecturas de red . . . 37
Capa Radio . . . 37
Capa Framer . . . 37
Capa RDC . . . 37
Capa MAC . . . 38
Capa de red . . . 38
3.2.2. *Hardware* - STM32W108C . . . 39
3.3. Configuración base . . . 40
3.4. Comunicación básica entre nodos . . . 43
3.5. Interconexión de la LoWPAN con red externa IPv6 . . . 45

3.5.1. Comunicación con enrutamiento multisalto 50
3.5.2. Comunicación con una red IPv4 - Servicio Tunnel Broker 52

4. Servicios de monitorización **57**
4.1. Introducción a REST 57
4.2. Diseño de aplicacion RESTful 58
4.3. Servicio REST sobre HTTP 59
4.3.1. Formato 60
4.3.2. Implementación 61
4.3.3. Validación 62
4.4. Servicio REST sobre CoAP 65
4.4.1. Formato 66
4.4.2. Implementación 67
4.4.3. Validación 68
4.5. Interoperabilidad HTTP/CoAP 71

5. Diseño de Servicios OML **74**
5.1. Conceptos teóricos 74
5.2. Implementación 77
5.3. Resultados 78

6. Conclusiones y Líneas Futuras **81**
6.1. Conclusiones 81
6.2. Líneas futuras 82

Bibliografía **84**

Índice de figuras

2.1. Esquema genérico de un nodo sensor o actuador 7
2.2. Esquema de arquitectura centralizada 8
2.3. Esquema de arquitectura distribuida 9
2.4. Topología de red en estrella y *peer-to-peer* 9
2.5. Pila de protocolos de 6LoWPAN y su equivalencia en OSI 10
2.6. Formato de trama de datos IEEE 802.15.4 12
2.7. Formato de paquete IPv6 15
2.8. Formato cabecera principal de IPv6 15
2.9. Formato cabecera de extensión de IPv6 16
2.10. Formato paquete ICMPv6 17
2.11. Diagrama del estado de 6LoWPAN 19
2.12. Arquitecturas LoWPAN 20
2.13. Formato de paquete 6LoWPAN 21
2.14. Dispatch Byte 21
2.15. Formato cabecera Mesh 21
2.16. Formato cabecera BC0 22
2.17. Formato cabecera de primer fragmento 22
2.18. Formato cabecera de continuación fragmentos 22
2.19. Encapsulación sin compresión en trama IEEE 802.15.4 23
2.20. Compresión de cabeceras HC1-HC2 24
2.21. Formato cabecera HC1 25
2.22. Formato cabecera HC2 25
2.23. Compresión de cabeceras IPHC-NHC 27
2.24. Formato cabecera IPHC 27
2.25. Formato cabecera IDs Contexto 28
2.26. Formato cabecera IDs Contexto 28
2.27. Formato cabecera UDP NHC 29
2.28. *Bootstrapping* 6LoWPAN-ND 30

3.1. Descripción general del escenario 35
3.2. Pila de protocolos de Contiki 38
3.3. Stack de 6LoWPAN de Contiki 39
3.4. Contenido del STM32W108C-SK 40
3.5. Escenario de comunicación entre nodos 44
3.6. Captura de comunicación entre nodos 46
3.7. Escenario Interconexión de la LoWPAN con IPv6 46
3.8. Conexión border router al PC 47
3.9. Petición `ping6` en el interfaz tun0 48
3.10. Petición `ping6` en la red LoWPAN (Fragmento 1) 49
3.11. Petición `ping6` en la red LoWPAN (Fragmento 2) 49
3.12. Escenario Multisalto 51
3.13. Escenario Tunnel Broker 53
3.14. Detalles del tunel 54
3.15. Testeo del tunneling 56
3.16. Petición HTTP desde terminal móvil 56

4.1. Diagrama de comunicación HTTP 59
4.2. Escenario HTTP 61
4.3. Diagrama de flujo del servidor HTTP 63
4.4. Validación de HTTP con tecnología HTML5 63
4.5. Petición HTTP en el interfaz tun0 64
4.6. Petición HTTP en el interfaz LoWPAN 65
4.7. Diagrama de comunicación CoAP 66
4.8. Formato cabecera CoAP 66
4.9. Descubrimiento de recursos en CoAP 67
4.10. Escenario CoAP 68
4.11. Aspecto del cliente Cupper 69
4.12. Peticion CoAP /.well-know/core 69
4.13. Reconocimiento CoAP /.well-know/core 70
4.14. Petición CoAP en el interfaz tun0 70
4.15. Petición CoAP en el interfaz LoWPAN 71
4.16. Escenario Proxy HTTP/CoAP 72
4.17. Petición HTTP utilizando Cf-proxy 72
4.18. Petición CoAP utilizando Cf-proxy 72
4.19. Respuesta CoAP utilizando Cf-proxy 73

4.20. Respuesta HTTP utilizando Cf-proxy 73

5.1. Arquitectura de OML 75
5.2. Pila de protocolos de OML 75
5.3. Diagrama de comunicación OML 76
5.4. Escenario OML 77
5.5. Diagrama de flujo del cliente OML 79
5.6. Base de datos OML 80

Índice de tablas

2.1. Bandas frecuenciales de trabajo en IEEE 802.15.4 11
2.2. Tipos de direcciones IPv6 16
2.3. Estado del estándar 6LoWPAN 18
2.4. *Payload* a nivel de transporte sin compresión 23
2.5. *Payload* a nivel de transporte con compresión HC1-HC2 26
2.6. *Payload* a nivel de transporte con compresión IPHC-NHC 29
2.7. Implementaciones *software* de 6LoWPAN 33

3.1. *Hardware* soportado por Contiki 36
3.2. Funcionalidades plataforma MB851 40
3.3. Funcionalidades plataforma MB951 40

4.1. Resumen de recursos basados en REST 59
4.2. Resumen de códigos de estado HTTP 61
4.3. Tipos de mensaje CoAP 67
4.4. Petición CoAP a través del proxy 71

Índice de cuadros de texto

3.1. Primer ejemplo en Contiki 41
3.2. Compilación de la aplicación para la plataforma MB851 42
3.3. Compilación de la aplicación para la plataforma MB951 42
3.4. Generación del fichero binario de la aplicación 42
3.5. Carga del fichero binario 42
3.6. Lectura del puerto serie de la placa 43
3.7. Puerto serie del servidor UDP 44
3.8. Puerto serie del cliente UDP 45
3.9. Ejecución de tunslip6 47
3.10. Configuración del forwarding para IPv6 50
3.11. Configuración de la tabla de rutas de PC2 50
3.12. Tabla de rutas del router de borde 50
3.13. Respuesta al aplicativo ping6 desde PC2 52
3.14. Configuración del interfaz de red para habilitar el servicio de tunnel 54
3.15. Comprobación del interfaz de red 54
3.16. Ejecución de tunslip6 usando el servicio de tunnel 55
3.17. Tablas de rutas usando el servicio de tunnel 55
5.1. Mensaje OML 80

Índice de acrónimos

6LoWPAN *IPv6 over Low power Wireless Personal Area Networks.*

ADC *Analog-to-Digital Converter.*

AES *Advanced Encryption Standard.*

AJAX *Asynchronous JavaScript And XML.*

AODV *Ad-Hoc On Demand Distance Vector.*

API *Application Programming Interface.*

ARP *Address Resolution Protocol.*

BAN *Body Area Network.*

BSD *Berkeley Software Distribution.*

CCA *Clear Channel Assessment.*

CoAP *Constrained Application Protocol.*

CORS *Cross-origin resource sharing.*

CRUD *Create, Read, Update and Delete.*

CSMA/CA *Carrier Sense Multiple Access with Collision Avoidance.*

DAO *Destination Advertisement Object.*

DHCP *Dynamic Host Configuration Protocol.*

DIO *DODAG Information Object.*

DIS *DODAG Information Solicitation.*

DODAG *Destination Oriented DAG.*

DSSS *Direct Sequence Spread Spectrum.*

DTLS *Datagram Transport Layer Security.*

DYMO *Dynamic MANET On-demand.*

EUI-64 *Extended Unique Identifier - 64.*

FCS *Frame Check Sequence.*

FFD *Full Function Device.*

FTP *File Transfer Protocol.*

GPL *General Public License.*

HATEOAS *Hypermedia as the Engine of Application State.*

HC1 *Header Compression 1.*

HC2 *Header Compression 2.*

HTML5 *HyperText Markup Language version 5.*

HTTP *Hypertext Transfer Protocol.*

ICMPv6 *Internet Control Message Protocol version 6.*

IETF *Internet Engineering Task Force.*

IoT *Internet of Things.*

IP *Internet Protocol.*

IPHC *IP Header Compression.*

IPsec *Internet Protocol security.*

IPv4 *Internet Protocol version 4.*

IPv6 *Internet Protocol version 6.*

ISM *Industrial, Scientific and Medical.*

ISP *Internet Service Provider.*

JSON *JavaScript Object Notation.*

LLC *Logical Link Control.*

LQI *Link Quality Indicator.*

LR-WPAN *Low-Rate Wireless Personal Area Network.*

MAC *Medium Access Control.*

MP *Measurement Point.*

MP2P *Multipeer-to-Peer.*

MS *Measurement Stream.*

MTU *Maximum Transmission Unit.*

NAT *Network Address Translation.*

NC *Node Confirmation.*

ND *Neighbor Discovery.*

NHC *Next Header Compression.*

NR *Node Registration.*

OML *OMF Measurement Library.*

OMSP *OML Measurement Stream Protocol.*

OSI *Open System Interconnection.*

P2MP *Peer-to-Multipeer.*

P2P *Peer-to-peer.*

PAN *Personal Area Network.*

RA *Router Advertesiment.*

RAM *Random Access Memory.*

RDC *Radio Duty Cycling.*

REST *Representational State Transfer.*

RFC *Request For Comments.*

RFD *Reduced Function Device.*

ROLL *Routing Over Low Power and Lossy Networks.*

RPL *Routing Protocol for Low-Power and Lossy Networks.*

RS *Router Solicitation.*

SLIP *Serial Line Internet Protocol.*

TCP *Transmission Control Protocol.*

Telnet *Teletype Network.*

TLS *Transport Layer Security.*

TTL *Time-to-Live.*

UDP *User Datagram Protocol.*

URI *Uniform Resource Identifier.*

WPAN *Wireless Personal Area Network.*

WSAN *Wireless Sensor and Actuator Networks.*

WSN *Wireless Sensor Networks.*

XML *eXtensible Markup Language.*

Capítulo 1

Introducción

La evolución de las tecnologías inalámbricas y el avance en la nanotecnología han permitido la integración del mundo físico con el mundo digital, de tal manera que prácticamente cualquier objeto que nos imaginemos tenga la capacidad de proporcionar información acerca del entorno que le rodea a través del medio inalámbrico, impulsando al desarrollo o a la mejora de múltiples aplicaciones.

En este sentido, los sistemas de monitorización y control remoto permiten recoger información del entorno y realizar un seguimiento de la evolución de su estado habilitando, por ejemplo, la gestión inteligente de infraestructuras mediante la automatización de tareas. Los usos de este tipo de sistemas son muy diversos; entre ellos se encuentran soluciones para implantar la agricultura de precisión mediante el control de riego o de plagas.

Por otra parte, cada vez es más conocido el concepto de redes de área corporal o *Body Area Network* (BAN). En esencia, consiste en el uso de dispositivos sensóricos e inalámbricos de reducido tamaño implantados en el cuerpo de las personas. Una de las aplicaciones principales de esta tecnología está enfocada al ámbito de la medicina dando soporte a la asistencia médica mediante el control continuo de los pacientes.

Es evidente que el interés por las redes de sensores inalámbricas o *Wireless Sensor Networks* (WSN) ha ido creciendo durante los últimos años, motivada en gran medida por la facilidad, flexibilidad y bajo coste de sus despliegues. Las WSN están formadas por un conjunto de dispositivos inalámbricos sencillos, conocidos como *motes*, con capacidades para sensar y actuar en función de una configuración establecida según su área de aplicación (agricultura, domótica, medicina, etc.). En la mayoría de los casos, los datos obtenidos son analizados de forma local para tomar decisiones, que afectarán al comportamiento del propio sistema que controla la red y que posteriormente se exportarán hacia una red exterior. Las WSN además permiten por tanto

la interacción desde redes externas, como Internet, bien de forma directa o a través de nodos intermedios o *gateways*.

El despliegue masivo de este tipo de redes con dispositivos de bajo coste interconectados entre sí que permiten el control y actuación sobre diferentes entidades ha dado lugar a lo que se conoce como la Internet de las Cosas o *Internet of Things* (IoT). Todos estos nodos estarán conectados a la red y por tanto cualquiera podrá comunicarse con ella. El concepto abarca desde sencillos sensores o actuadores para manejar electrodomésticos, hasta complejas infraestructuras públicas de alumbrado o sistemas industriales, pasando por coches y casas inteligentes, permitiendo a las personas un ambiente más inteligente y confortable.

Aunque el paradigma de la IoT plantea que cualquier dispositivo estará conectado, las propias capacidades de estos nodos y la escasez de direcciones *Internet Protocol version 4* (IPv4) ha hecho inviable hacer realidad completamente esta aproximación. Como soluciones intermedias, se han desarrollado metodologías de despliegue alternativas siguiendo arquitecturas basadas en *clusters* o centralizadas. Sin embargo, es el despliegue de redes basadas en *Internet Protocol version 6* (IPv6)[1] el que parece que definitivamente ha abierto la vía hacia una IoT donde se permita la conectividad directa entre cualquier dispositivo, que además estará unívocamente identificado. No obstante, es necesario adaptar los protocolos y soluciones propietarias intermediarias para habilitarla. Para ello, el *Internet Engineering Task Force* (IETF) ya ha hecho los primeros movimientos desarrollando nuevos estándares que permiten el uso de IPv6 en las WSN.

1.1. Motivación

Con la integración de la pila de protocolos *Transmission Control Protocol* (TCP)/*Internet Protocol* (IP) en los dispositivos sensores y actuadores, las WSN se beneficiarían de las ventajas de un modelo de interconexión estándar, cuya globalidad e interoperabilidad permitiría la conectividad extremo a extremo. No obstante, hay que tener en cuenta aspectos de capacidad y coste para analizar la rentabilidad de esta propuesta.

Los sensores, como dispositivos autónomos, se encuentran alimentados por baterías. Si a esta limitación en cuanto a consumo energético se le suma las reducidas capacidades de cómputo que ofrecen, queda claro que se hace necesaria una adaptación de los sistemas estándar usualmente utilizados.

En aras de lograr hacer realidad este nuevo paradigma de red, el IETF ha desarrollado una serie de estándares, entre los que destacan los enfocados al uso de IPv6 en *Wireless Personal Area Network* (WPAN). El más importante es el que describe la capa de adaptación para el transporte de paquetes IPv6 sobre redes IEEE 802.15.4, *IPv6 over Low power Wireless Personal Area Networks* (6LoWPAN)[2], que habilita la interacción directa de las redes de sensores con

el mundo IPv6. 6LoWPAN aprovecha las características de IPv6 con respecto a su predecesor, IPv4, como son un espacio de direcciones mayor y la configuración automática de direcciones, entre otras.

1.2. Objetivos

El objetivo principal de este Proyecto Fin de Carrera es la configuración y despliegue real de una WSN de bajo consumo empleando una solución abierta que soporte 6LoWPAN.

Buscando que la red resultante no quede aislada, se realizará además la interconexión de ésta tanto con una red IPv6 nativa, como con la red Internet. Todo ello permitirá acceder de forma ubicua a la información alojada en los nodos y controlarlos de forma remota. Como parte de este trabajo se comprobarán los mecanismos de comunicación y actuación que se habilitan en el estándar 6LoWPAN.

Adicionalmente, se estudiarán diversas alternativas para ofrecer interfaces estándar basados en tecnologías web para ofrecer servicios desde los propios dispositivos y gestionarlos.

1.3. Estructura de la memoria

La presente memoria está dividida en seis capítulos organizados según el orden secuencial seguido para el desarrollo de este proyecto.

Tras un primer capítulo en el que se plantea el entorno del proyecto, el segundo capítulo introduce el concepto de las WSN y las WPAN en particular. Se describen teóricamente los protocolos necesarios para establecer una comunicación entre los dispositivos. Se analizarán distintas alternativas *software* y *hardware* para la implementación de la solución planteada en el proyecto, mostrando las ventajas e inconvenientes de cada una, para posteriormente seleccionar aquellas que mejor se ajusten a las necesidades de este proyecto.

El tercer capítulo describirá el despliegue necesario para hacer realidad la red de sensores con las características planteadas a nivel de conectividad. Se estudiarán en mayor profundidad la plataforma *hardware* seleccionada y el sistema operativo a ejecutar sobre ella. Se analizarán también los protocolos de interconexión y las diferentes fases en la configuración y operativa de las redes a través de la inspección del tráfico intercambiado entre los nodos que la conforman.

El cuarto capítulo se centrará en las aplicaciones y servicios que se ofrecerán sobre la red desplegada. Se estudiarán y evaluarán diferentes alternativas existentes para el desarrollo y provisión de servicios sobre redes basadas en el estándar 6LoWPAN, como *Constrained Application Protocol* (CoAP).

En el quinto capítulo se describirá e implementará un mecanismo que permite dotar de autonomía a una red de sensores para informar de variaciones en las métricas que sus nodos controlan.

Para finalizar, el sexto capítulo recoge las conclusiones obtenidas y las líneas futuras de investigación abiertas a raíz de este trabajo.

Capítulo 2

Redes de Sensores

Hoy en día existen numerosos tipos de sensores formando parte de un gran número de sistemas y dispositivos electrónicos (sensores de temperatura en estaciones meteorológicas, acelerómetros en móviles y tabletas, sensores infrarrojos en sistemas de alarma o detección de movimiento, etc.). La mayor parte de estos sensores se limitan a funcionar como un transductor que realiza un tipo concreto de medición (una o más variables del entorno) y envía dicha información a un procesador central dentro del mismo sistema electrónico.

Sin embargo, durante los últimos años, ha aparecido una nueva generación de sensores, independientes de un sistema electrónico concreto, que incorporan en un mismo dispositivo el transductor (de la o las variables que interesan medir), la alimentación del propio dispositivo y un módulo de comunicación dotado de cierta inteligencia propia, capaz de autogestionarse e interconectarse de forma inalámbrica con otros nodos semejantes. Como resultado de su despliegue surgen las llamadas redes de sensores inalámbricos (WSN), consistentes en redes formadas por multitud de sensores individuales que intercambian información entre sí y/o con un nodo central sin necesidad de cables y mediante un protocolo de comunicación pre-establecido.

Los diferentes nodos pueden operar no solo como sensores, sino que sus capacidades de procesamiento y comunicación permiten que funcionen como actuadores, es decir, mecanismos capaces de activar procesos automatizados sobre sistemas externos bien siguiendo instrucciones remotas o a partir de decisiones tomadas de manera autónoma a partir de la información sensada (e.g. encender luces ante una disminución de la intensidad lumínica). Junto a los sensores, constituyen las redes inalambricas de sensores y actuadores o *Wireless Sensor and Actuator Networks* (WSAN).

A continuación se introducen las caracteríticas y posibilidades de las WSN, profundizando en las arquitecturas y topologías más usuales, así como en las distintas soluciones tecnológicas empleadas en su despliegue.

2.1. Conceptos generales

Una WSN o su extensión, una WSAN, es una red inalámbrica formada por un conjunto de dispositivos autónomos e inteligentes y de carácter multifuncional, con capacidades de sensado y recogida de datos, que se comunican entre sí o con una base central con el objetivo de intercambiar la información capturada por ellos mismos y tomar las acciones pertinentes sobre los sistemas bajo su inspección y control. Estas redes deben adaptarse a los cambios (posición, alimentación, etc.) puntuales en los diferentes nodos que las conforman.

En general, en una WSN se pueden encontrar los siguientes elementos:

- Sensores: dispositivos capaces de medir las condiciones y la información deseada del lugar o del medio en el que se encuentran.
- Actuadores: dispositivos capaces de modificar su propio comportamiento o el de terceros a partir de instrucciones de configuración.
- Nodos: dispositivos que recogen la información recibida por los sensores y la transmiten a otros nodos o la estación base.
- Pasarelas o *gateways*: elementos que conectan los nodos de la red de sensores a una red externa o red troncal.
- Estación base: elemento que recoge todos los datos provenientes de la red de sensores. Suele tratarse de un ordenador común o un elemento que lo sustituya.

Usualmente se emplea el término mota o *mote* para referirse a los nodos que conforman la red de sensores. Este dispositivo integra, al menos, sensores, un microcontrolador, un conversor analógico a digital o *Analog-to-Digital Converter* (ADC) y un chip de comunicación radio.

2.2. Arquitectura funcional de un sensor

La arquitectura de un elemento sensor o actuador se puede dividir en tres bloques funcionales, tal como se refleja en la Figura 2.1:

- Bloque sensor/actuador: define la funcionalidad del dispositivo, como sensor (medición de un parámetro) o como actuador (mecanismo activador/interventor).
- Bloque de procesado: conjunto de elementos que dotan al dispositivo de inteligencia de computación y toma de decisiones.

- Bloque de comunicación: envía y recibe mensajes hacia y desde los nodos y repetidores vecinos.

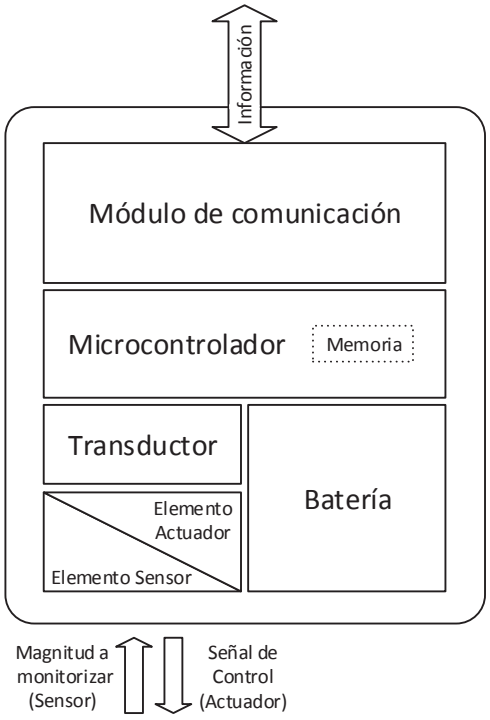


FIGURA 2.1: Esquema genérico de un nodo sensor o actuador

Se pueden establecer una serie de características que se dan en los nodos:

- Suelen estar limitados por el consumo energético al estar usualmente alimentados por baterías, así como en memoria y capacidad de proceso para ahorrar costes.
- Son autónomos y operan de forma independiente.
- Tienen una alta probabilidad de fallo, debido a las condiciones a las que se exponen y a su bajo coste.
- Deben incorporar medidas que permitan su adaptación al entorno.

El tamaño de un nodo puede variar y su coste oscila entre cientos de euros a unos pocos céntimos dependiendo de la complejidad requerida. El tamaño y el consumo energético ($<$50mW) en los nodos sensores son las principales razones por las que disponen de unos recursos limitados en cuanto a memoria ($<$1MByte), velocidad de cálculo ($<$500MHz) y ancho de banda de comunicaciones ($<$250Kbps).

Al tener un microcontrolador interno, dentro de estos nodos se puede realizar un preprocesamiento de los datos con el objetivo de reducir el tráfico circulante por la red o proporcionar un valor añadido desde la propia fuente.

2.3. Arquitecturas de red

Tomando como elementos de la red de sensores los enumerados anteriormente, podemos distinguir dos tipos de arquitectura: centralizada y distribuida. Seguidamente se detallan las características de cada una de ellas.

2.3.1. Arquitectura centralizada

En este tipo de red (Figura 2.2), los nodos se encargan de recoger información y enviarla a la pasarela más cercana si es que la hubiera quien redigirá el tráfico hacia la estación base. En esta disposición, los nodos pasarela, al actuar como concentradores, además de poder provocar un cierta lentitud en las comunicaciones de datos por la sobrecarga en la que incurren, también tienen un mayor consumo energético, lo que disminuye el tiempo de vida o autonomía de los nodos.

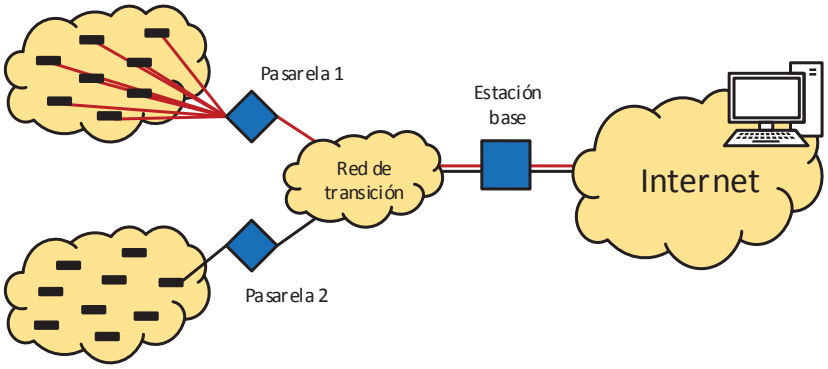


FIGURA 2.2: Esquema de arquitectura centralizada

2.3.2. Arquitectura distribuida

Los nodos se comunicarán con sus nodos vecinos, cooperando y obteniendo una respuesta única que será enviada al coordinador del *cluster*, que se encargará de comunicar a la estación base (Figura 2.3).

2.4. Topologías de red

Las redes de sensores deben de auto-organizarse y auto-mantenerse para que de esta forma se reduzcan los costes totales y se facilite su despliegue y uso. En función de los requerimientos de la aplicación existen múltiples formas desde el punto de vista lógico en las que disponer los enlaces entre los nodos. En la Figura 2.4 se muestran las topologías más comunes.

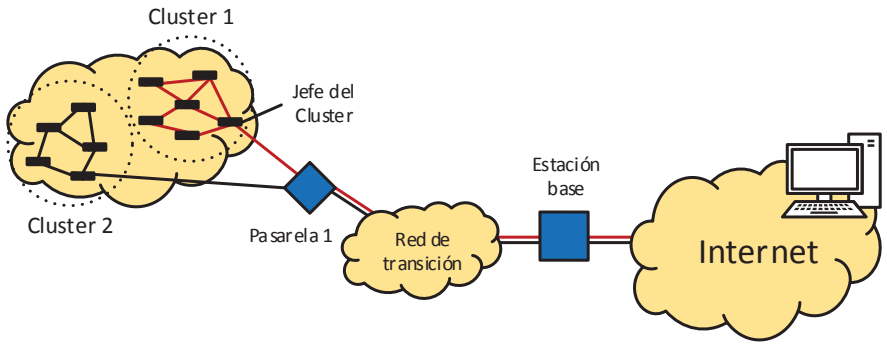


FIGURA 2.3: Esquema de arquitectura distribuida

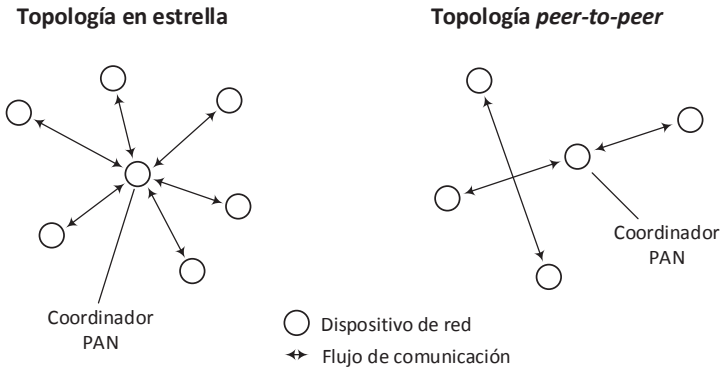


FIGURA 2.4: Topología de red en estrella y *peer-to-peer*

2.4.1. Estrella

En este tipo de configuración es necesario un coordinador de red por cada grupo de nodos, hacia el cual éstos dirigen su comunicación. Cada dispositivo establece una comunicación bidireccional con el nodo maestro. Dicho coordinador es el encargo de iniciar, terminar o redirigir la comunicación a través de la red. Además, realiza la propia función que tenga asignada (e.g. actuar como nodo central de recogida de datos).

2.4.2. *Peer-to-peer*

En la topología *Peer-to-peer* (P2P), cada dispositivo puede comunicarse con otros de la misma red siempre que estén a su alcance. Conforman redes *ad-hoc*, sin necesidad de infraestructura física preestablecida, ni de un control central que coordine su actividad.

Las redes P2P incluyen capacidades para auto-organizarse. Los nodos que las conforman pueden trabajar como emisores y receptores, estableciendo caminos de comunicación entre nodos sin visibilidad directa y modificarlos si alguno de los nodos incluidos en la ruta, etc. Estas

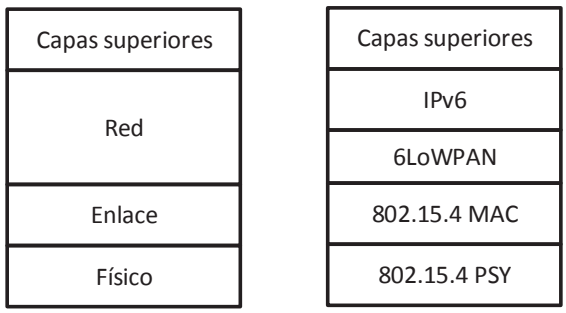
funcionalidades estarán disponibles en función del protocolo de comunicaciones que se emplee entre los nodos.

Con esta topología de red es posible crear estructuras más complejas, como redes malladas, en las que no existe una jerarquía entre nodos.

2.5. Protocolos de comunicación

Los dispositivos de una red de sensores deben seguir unas normas para gestionar sus diálogos en los intercambios de información. En las redes de sensores desplegadas actualmente, es habitual encontrarse con protocolos propietarios que no siguen normas estándar y que por tanto hace que los nodos que los emplean no sean capaces de comunicarse con dispositivos fuera de su ecosistema, como por ejemplo, los disponibles en la red Internet.

A continuación se procede al estudio del modelo de protocolos de red necesario para establecer una comunicación extremo a extremo a nivel de red sobre redes *Low-Rate Wireless Personal Area Network* (LR-WPAN). Partiendo de los niveles inferiores según el modelo *Open System Interconnection* (OSI), se ascenderá hasta el nivel de red donde se describirá en profundidad la solución 6LoWPAN con la que proporcionar soporte IPv6 en redes de sensores. En la Figura 2.5 se muestra la pila de protocolos empleada en una red basada en 6LoWPAN y la equivalencia con el modelo OSI.



(A) Arq. OSI (B) Arq. 6LoWPAN

FIGURA 2.5: Pila de protocolos de 6LoWPAN y su equivalencia en OSI

2.5.1. IEEE 802.15.4

El estándar IEEE 802.15.4 define la capa de nivel físico y parte de la capa de enlace o *Medium Access Control* (MAC) para WPAN con bajas tasas de transmisión de datos, LR-WPAN. Aunque la primera versión del estándar data de 2003, la versión actual [3] se aprobó en 2006.

Nace de la necesidad de tener una norma que defina la comunicación en una red de múltiples nodos, teniendo en cuenta los requisitos de baja tasa de transferencia de datos, poca complejidad y reducido gasto energético. Además los mecanismos de comunicación que se definan deben hacer uso de las bandas libres *Industrial, Scientific and Medical* (ISM) y presentar un corto alcance.

2.5.1.1. Capa física

Dentro del estándar, en este nivel se definen las normativas necesarias para la transmisión a través del medio inalámbrico. Se utiliza la técnica *Direct Sequence Spread Spectrum* (DSSS) para modular la información antes de ser enviada a través del medio. Básicamente por cada bit de información se genera un patrón de bits redundante (secuencia de Barker) que se empleará en la posterior modulación [4]. Esto se traduce en la necesidad de una mayor anchura espectral y una densidad espectral de potencia baja para cada señal, lo que le hace inmune a interferencias en la banda de uso.

Una característica de las redes IEEE 802.15.4 es que operan haciendo uso de bandas no licenciadas. En la Tabla 2.1 se resumen las bandas definidas, junto con el número de canales asignados y velocidad de transmisión según la región geográfica. La elección del canal en cada banda es decisión del coordinador de la red.

Banda	868 MHz	915 MHz	2.4 GHz
Región	Europa, Japón	EEUU	Internacional
Canales	1	10	16
Ancho de banda	20 kbps	40 kbps	250 kbps

Tabla 2.1: Bandas frecuenciales de trabajo en IEEE 802.15.4

Las tareas que realiza la capa física en IEEE 802.15.4 se pueden resumir en:

- Activación y desactivación del transceptor de radio.
- Detección del nivel de energía en el canal de trabajo.
- Ofrecer un indicador de calidad del enlace *Link Quality Indicator* (LQI) de los paquetes recibidos, basándose en su nivel de potencia.
- Realizar la valoración de canal libre *Clear Channel Assessment* (CCA), utilizado por la capa MAC para el algoritmo *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA).
- Elección del canal frecuencial de trabajo.
- Transmisión radio a través del medio físico.

2.5.1.2. Capa de enlace

Este nivel es el responsable de la transferencia de información dentro de la red. Utilizando los servicios que le ofrece la capa física, debe ser el encargado de la comunicación de tramas entre los distintos nodos de la LR-WPAN.

Al igual que todos los estándares IEEE 802.x, la capa de enlace de IEEE 802.15.4 se apoya en el estándar 802.2 [5], también conocido como *Logical Link Control* (LLC), responsable del control de enlace lógico. La subcapa LLC maneja el control de errores, el control de flujo, el entramado, el control de diálogo y el direccionamiento de la capa MAC de IEEE 802.15.4.

IEEE 802.15.4 trata de minimizar la complejidad, aspecto que queda reflejado en que únicamente utiliza 21 primitivas de servicio. Si esta cifra se compara con las empleadas en otras tecnologías como puede ser el caso de Blueetooth con 121 [6], se aprecia esta característica, lo que le convierte en una tecnología que no necesita gran potencia *hardware* para su implementación.

Se distinguen distintos tipos de tramas 802.15.4: de datos y de gestión o control. Para este proyecto solo tienen interés las de datos, que son sobre las que encapsularán los paquetes 6LoWPAN, sin necesidad de modificar el nivel de enlace. En la Figura 2.6 se muestra el formato general de una trama de datos IEEE 802.15.4, cuyos campos son:

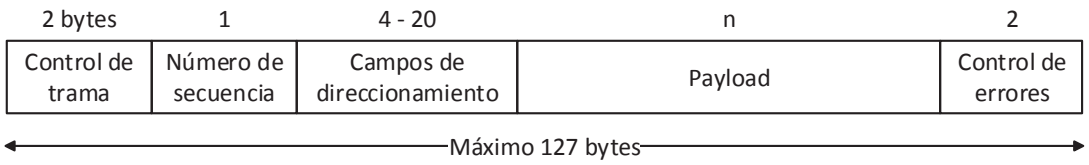


Figura 2.6: Formato de trama de datos IEEE 802.15.4

- **Control de trama:** Indica que se trata de una trama de datos, además de añadir información adicional sobre el formato del resto de campos, como el de direccionamiento. Asimismo indica si requiere reconocimiento y si utiliza seguridad a nivel MAC.
- **Numero de secuencia:** Identificador de la trama que se utiliza para los reconocimientos.
- **Campos de direccionamiento:** Indica las direcciones origen y destino además de los identificadores de la *Personal Area Network* (PAN) ID en el formato indicado en el campo de control.
- **Carga útil (*Payload*):** Datos de información a nivel MAC variables que no deben exceder los 127 bytes del total de la trama.
- **Control de errores:** Cola que se utiliza para verificar la integridad de la trama MAC.

Cada elemento de la red dispone de un identificador único de 64 bits (*Extended Unique Identifier - 64* (EUI-64)) para distinguirlo de los demás nodos. Sin embargo, resulta habitual emplear identificadores de 16 bits para reducir el tamaño de los campos de direccionamiento de la cabecera de la trama.

IEEE 802.15.4 utiliza el mecanismo CSMA/CA para evitar colisiones entre las tramas que puedan estar siendo enviadas en el mismo canal por otros nodos. Cuando se quiere transmitir un mensaje, la capa MAC solicita a la capa física que realice una valoración del canal libre CCA; si se detecta que el medio está libre se transmite; de lo contrario, si el medio está siendo utilizado, la capa MAC espera un periodo de tiempo aleatorio y reinicia el proceso.

Por otra parte, siempre que un nodo reciba una trama a nivel MAC, debe responder con un reconocimiento (ACK) notificando al origen que ésta se recibió con éxito. Para el envío de esta trama no se usa el mecanismo de CSMA/CA.

A las capacidades anteriores hay que añadir que IEEE 802.15.4 incorpora un control de errores para validar la integridad de los datos, así como otros mecanismos de seguridad basados en el cifrado de bloques a través del algoritmo *Advanced Encryption Standard* (AES) [7]. En función del nivel de cifrado (AES-CCM-32, AES-CCM-64 o AES-CCM-128) se tendrá una sobrecarga en la cabecera MAC de 9, 13 o 21 bytes respectivamente. El estudio del algoritmo se aleja de los objetivos de este trabajo.

En resumen, las tareas que realiza la capa MAC en IEEE 802.15.4 son las siguientes:

- Generación de señales de sincronización y asociación/desasociación.
- Acceder al canal mediante el mecanismo CSMA/CA.
- Comprobar la integridad de la trama.
- Ofrecer mecanismos de seguridad basados en el algoritmo AES.
- Enviar tramas de confirmación a nivel de enlace.
- Administrar la asociación y desasociación de nodos en la red.

2.5.1.3. Elementos de red

Se definen dos tipos de dispositivos en la red según sus limitaciones *hardware* y *software*: los dispositivos de funcionalidad completa o *Full Function Device* (FFD), capaces de hacer uso de todos los servicios que ofrece IEEE 802.15.4, y los dispositivos de funcionalidad reducida o *Reduced Function Device* (RFD), que no tienen implementado de forma completa el estándar.

Además, los nodos de la red pueden ser clasificados según el rol que cumplan:

- Coordinador PAN: Se encarga de asignar un identificador a la red (PAN ID), encontrar el canal de trabajo adecuado, asignarse una dirección corta, administrar la asociación de nuevos nodos y, según la topología, retransmitir mensajes entre nodos. Debe ser siempre un FFD.
- *Router*: Aparte del coordinador PAN, también puede existir un elemento que se encargue de administrar las solicitudes de asociación a la red y retransmitir mensajes entre nodos. También debe ser un FFD.
- Dispositivo: Es un nodo cualquiera de la red que no cumple las funciones de coordinador o *router*. Es el único que no tiene la obligación de ser FFD, luego puede ser RFD.

2.5.2. IPv6

El protocolo de red IPv6 [1] es la última versión de IP diseñada para reemplazar a IPv4 [8]. La limitación que está presentando éste en los últimos años frente al incremento exhaustivo del número de direcciones de red necesarias en Internet fue el principal motivo del desarrollo de un nuevo protocolo capaz de solventar lo que esperaba ser un problema. Asimismo, IPv6 ha sido diseñado con el fin de mejorar algunos aspectos que no se tuvieron en cuenta en IPv4, como la posibilidad de usar configuración automática de direcciones sin necesidad emplear el protocolo *Dynamic Host Configuration Protocol* (DHCP), entre otros.

En los siguientes apartados se profundiza en los aspectos de IPv6 más vinculados a 6LoWPAN, esto es, formato del paquete, direccionamiento y protocolo de descubrimiento de vecinos.

2.5.2.1. Formato del paquete

Un paquete IPv6 tiene el formato mostrado en la Figura 2.7. Consta de una cabecera principal obligatoria como la que se muestra en la Figura 2.8 seguida de múltiples cabeceras de extensión opcionales con un formato como el de la Figura 2.9.

Los campos que forman la cabecera principal son los siguientes:

- **Versión:** Para el caso de IPv6 toma el valor 0110 (6).
- **Clase de tráfico:** Es utilizado para que los equipos generadores de tráfico y/o los enrutadores puedan distinguir entre distintas clases de tráfico.

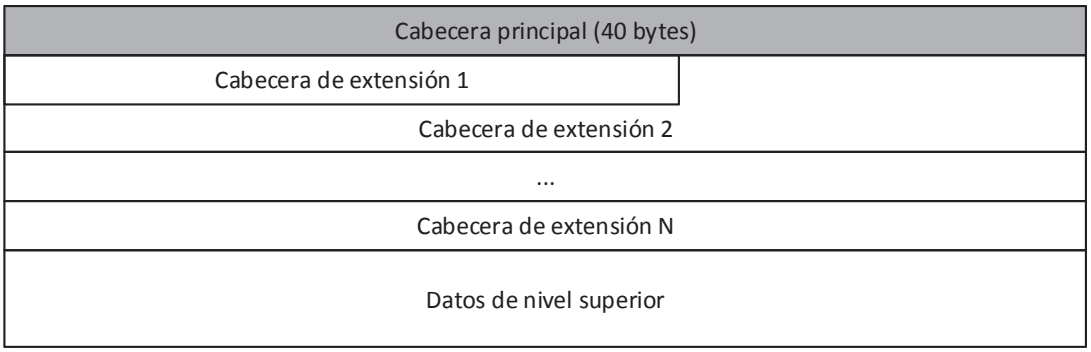


FIGURA 2.7: Formato de paquete IPv6

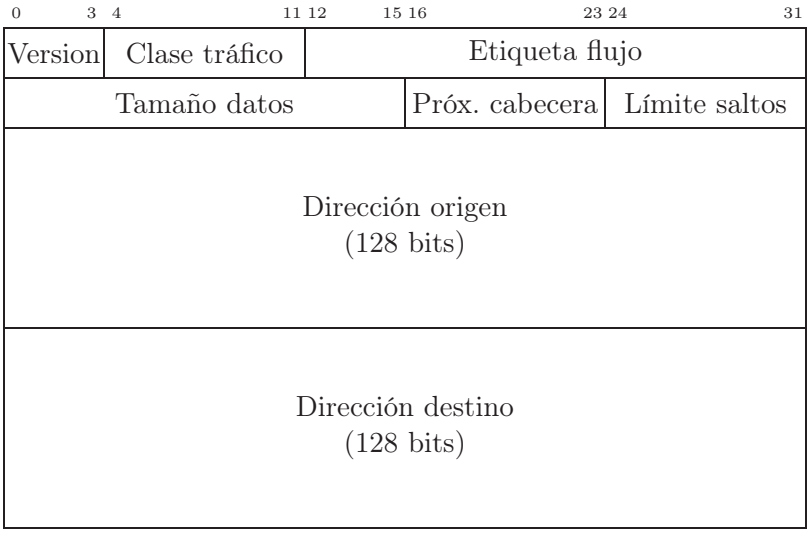


FIGURA 2.8: Formato cabecera principal de IPv6

- **Etiqueta de flujo:** Un flujo es una secuencia de datagramas. Se le da un identificador al flujo para que todos los *routers* traten a todos los datagramas de igual forma. Permite al origen y al destino establecer pseudoconexiones con propiedades particulares.
- **Tamaño de los datos:** El tamaño del datagrama IPv6 sin contar la cabecera.
- **Próxima cabecera:** Identifica el contenido del encabezado siguiente, permitiendo añadir cabeceras IPv6 de extensión.
- **Límite de saltos:** Contador que se decrementa cada vez que el datagrama sea reenviado a través de un nodo. Similar al *Time-to-Live* (TTL) de IPv4.

2.5.2.2. Direccionamiento

IPv6 presenta una capacidad de direccionamiento muy superior a su predecesor IPv4. Así frente a los 32 bits disponibles en IPv4, en IPv6 se definen direcciones de 128 bits. Esto permite

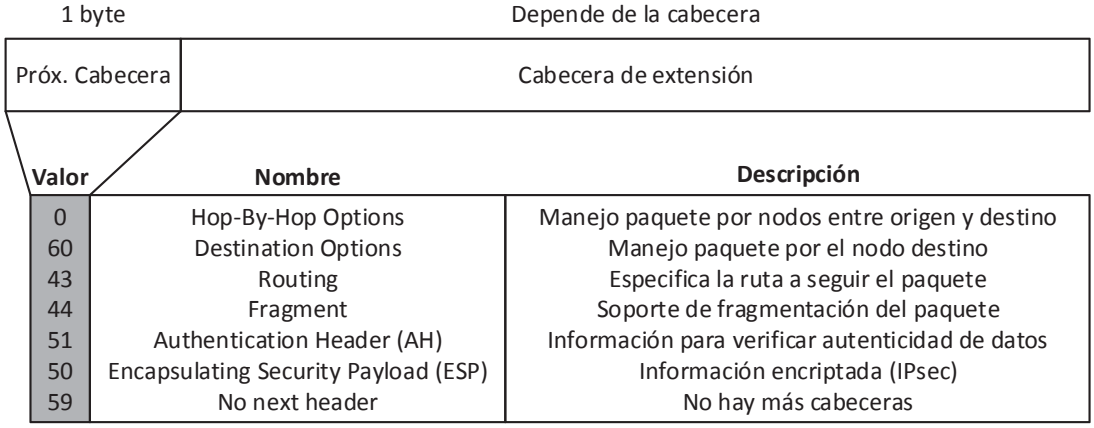


FIGURA 2.9: Formato cabecera de extensión de IPv6

un gran abanico de posibilidades entre las que se encuentra el poder dotar a casi cualquier dispositivo con una o más direcciones (por ejemplo, una por interfaz de comunicaciones) propias e inalterables a lo largo de su vida útil, eliminando la necesidad de soluciones como *Network Address Translation* (NAT) [9] empleadas en redes con direccionamiento IPv4. En la Tabla 2.2 se muestra un resumen de los distintos tipos de direcciones IPv6.

Tipo	**Prefijo binario**	**Notación IPv6**
Loopback	00...1 (128 bits)	::1/128
Multicast	1111 1111	FF00::/8
Link-Local Unicast	1111 1110 10	FE80::/10
Global Unicast	El resto	El resto

TABLA 2.2: Tipos de direcciones IPv6

Señalar que dentro del rango de direcciones *Unicast*, las *Link-Local* son usadas dentro de una misma red con conectividad a nivel de enlace, mientras que las *Global* se usan con conectividad a nivel de red, esto es, entre distintas subredes IPv6.

Además, IPv6 aporta la funcionalidad de auto-configuración del direccionamiento mediante mecanismos *stateless* [10], que permiten la construcción automática de direcciones a partir del identificador MAC del dispositivo y el prefijo de red. Los primeros 64 bits de la dirección IPv6 reciben el nombre de prefijo de red, mientras que los 64 bits restantes se obtienen a partir del identificador de la interfaz. El prefijo de red (en el caso de direcciones *unicast*) es distribuido por el *router* de borde durante el proceso de inicialización de la red. Por otra parte, el identificador de interfaz se obtiene a partir de la dirección MAC del dispositivo. En el caso de usar la versión EUI-64 se toma directamente, mientras que si se usa el direccionamiento corto, deberá rellenarse con bits adicionales con el fin de obtener un identificador de interfaz de 64 bits.

De igual manera que en IPv4, es posible utilizar direccionamiento manual, sin auto-configuración, que en muchos casos puede ser útil para no saturar la red con los mensajes de información del prefijo.

2.5.2.3. ICMPv6

El protocolo *Internet Control Message Protocol version 6* (ICMPv6) [11] es parte de la arquitectura de IPv6. Es el equivalente a ICMP de IPv4 y sirve para detectar errores encontrados con la interpretación de paquetes de red, además de usarse para el diagnóstico y análisis del estado de la red. En la Figura 2.10 se muestra el formato genérico de un paquete.

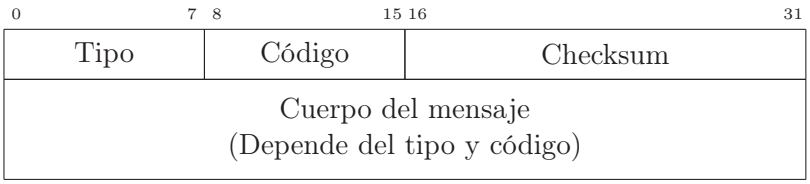


FIGURA 2.10: Formato paquete ICMPv6

Los campos de los mensajes ICMPv6 son los siguientes:

- **Tipo:** Indica si se trata de un mensaje de error o informativo.
- **Código:** Informa del tipo de mensaje ICMPv6
- ***Checksum*:** Detecta errores en los mensajes ICMPv6

2.5.2.4. Neighbor Discovery

IPv6 utiliza el protocolo de descubrimiento de vecinos *Neighbor Discovery* (ND) [12] que, además de ser el equivalente al protocolo *Address Resolution Protocol* (ARP) de IPv4 permitiendo conocer las direcciones de los nodos vecinos, agrupa diversas funcionalidades de IPv6, como la auto-configuración de direcciones. Consiste en un mecanismo por el cual un nodo que se acaba de incorporar a la red descubre la presencia de otros nodos a nivel de enlace a través de mensajes ICMPv6.

Se distinguen hasta 5 tipos de paquetes ICMPv6 definidos en ND:

- **Solicitud de *router***: Se utiliza para el descubrimiento de los *router* de la red local
- **Difusión de *router***: Los *router* se anuncian periódicamente o cada vez que reciben una solicitud de *router*. Además, sirve para informar de parámetros de la red, como por ejemplo el prefijo.

- **Solicitud de vecino**: Lo usan los nodos para determinar la dirección MAC de sus vecinos. También es empleado para asegurarse que un vecino es alcanzable, o detectar direcciones duplicadas.
- **Difusión de vecino**: Es utilizado por los nodos para responder a las solicitudes de vecino, aunque también se usa para indicar cambios de direcciones a nivel MAC.
- **Redirección**: Los nodos generan este paquete para informar a los *router* de que existe una ruta mejor para llegar a un determinado destino.

2.5.3. 6LoWPAN

El estándar 6LoWPAN surge con el objetivo de aportar una solución al transporte de paquetes IPv6 sobre tramas IEEE 802.15.4. Proporciona mecanismos de encapsulación y compresión de cabeceras para la transmisión de paquetes. El grupo de trabajo del IETF encargado de su desarrollo recibe el nombre de 6lowpan [13]. En la Tabla 2.3 se resume el estado actual del protocolo enumerando los distintos documentos realizados por dicho grupo de trabajo. Además, en la Figura 2.11 se muestra un diagrama con los mismos. Por otro lado, existen también otros grupos del IETF que paralelamente a 6lowpan, se encargan de cuestiones específicas, como es el caso del grupo de trabajo *Routing Over Low Power and Lossy Networks* (ROLL) encargado del protocolo de enrutamiento *Routing Protocol for Low-Power and Lossy Networks* (RPL).

Documento	**Título**	**Fecha**
RFC 4919	*IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*	08/2007
RFC 4944	*Transmission of IPv6 Packets over IEEE 802.15.4 Networks*	09/2007
RFC 6282	*Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*	09/2011
RFC 6568	*Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*	04/2012
RFC 6606	*Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing*	05/2012
RFC 6775	*Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)*	11/2012
ipsec-01	*Compression of IPsec AH and ESP Headers for Constrained Environments*	09/2013

Tabla 2.3: Estado del estándar 6LoWPAN

Las ventajas del uso de 6LoWPAN respecto a las soluciones tradicionales y propietarias, incluidas en el *Request For Comments* (RFC) 4919 [14], se resumen en:

- El uso de IP permite tener una infraestructura de red existente.

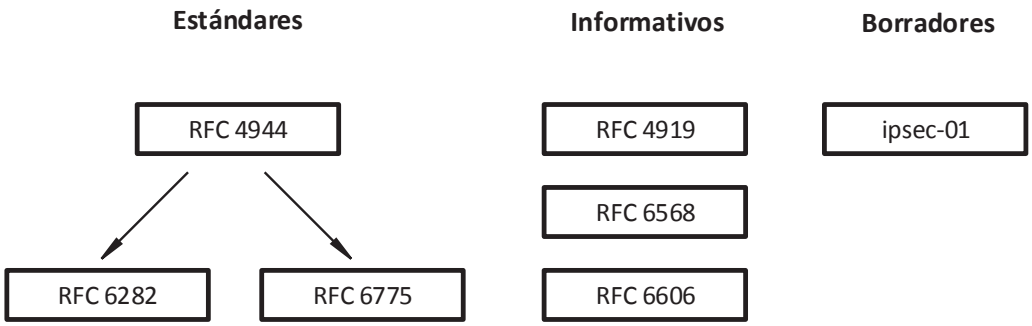


FIGURA 2.11: Diagrama del estado de 6LoWPAN

- Las tecnologías basadas en IP son conocidas y ya están en funcionamiento.
- Es un estándar abierto.
- Existen herramientas para el diagnóstico y gestión de redes IP.
- Los dispositivos IP pueden conectarse directamente a redes IP sin la necesidad de entidades intermedias, como el uso de *gateways* o *proxies*, permitiendo el uso de conexiones *end-to-end*.

A continuación se profundiza en el estándar 6LoWPAN dada relevancia en el desarrollo de este proyecto. Se describen las características principales del protocolo en cuestión en aspectos tales como las arquitecturas de red, el direccionamiento o la capa de adaptación necesaria para la encapsulación de los paquetes, explicando cómo se lleva a cabo la fragmentación y la compresión de cabeceras. Además, se explica el descubrimiento de vecinos en 6LoWPAN y las diferentes posibilidades de encaminamiento soportadas. Por último, se introducen las recomendaciones del estándar sobre seguridad.

2.5.3.1. Dispositivos y Arquitecturas

Las redes LoWPAN están formadas por dispositivos que toman distintos roles:

- El *edge router* o *router* de borde es el encargado de encaminar el tráfico entre la red IPv6 y la LoWPAN, siendo el responsable de la interacción entre los protocolos IPv6 y 6LoWPAN a través de la compresión y descompresión de las cabeceras. Además, emplea una versión modificada del protocolo de descubrimiento de vecinos ND de IPv6 para la gestión y mantenimiento de la red.
- Los *routers* son nodos con capacidad de encaminamiento, esto es, pueden enrutar paquetes 6LoWPAN hacia el resto de nodos que estén a su alcance, por lo que deben estar siempre activos.

- Los *hosts* son los dispositivos más sencillos de la red. Su misión es meramente la de nodo actuador o sensor. No requieren estar siempre activos, pues pueden utilizar técnicas de ahorro energético para extender la duración de sus baterías.

Por otra parte, existen distintos tipos de arquitecturas de red (ver Figura 2.12):

- LoWPAN Simple: solo tiene un *router* de borde.
- LoWPAN Extendida: existen varios *router* de borde compartiendo un enlace común. Su aplicación es habitual cuando se tienen muchos nodos en la red.
- LoWPAN *ad-hoc*: la red no requiere de un *router* de borde ya que no hay enrutamiento hacia fuera de la LoWPAN.

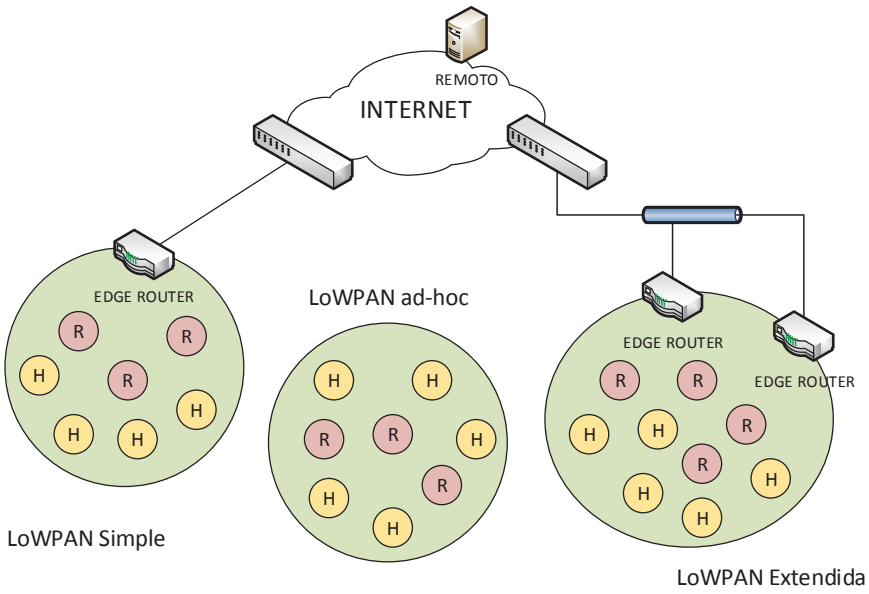


FIGURA 2.12: Arquitecturas LoWPAN

2.5.3.2. Formato de paquete

Un paquete 6LoWPAN sigue el formato descrito en la Figura 2.13. El estándar permite encapsular paquetes de red IPv6 sobre tramas de enlace IEEE 802.15.4. Para ello, se define la capa de adaptación que realiza las modificaciones y añade la información de cabeceras adicionales para llevar a cabo el proceso de encapsulación, además de ofrecer mecanismos, como el soporte *mesh-under*, fragmentación o compresión de cabeceras de capas superiores.

Las cabeceras de adaptación comienzan con un campo de 8 bits que recibe el nombre de *dispatch byte* e indica el tipo de cabecera del que se trata (ver Figura 2.14).

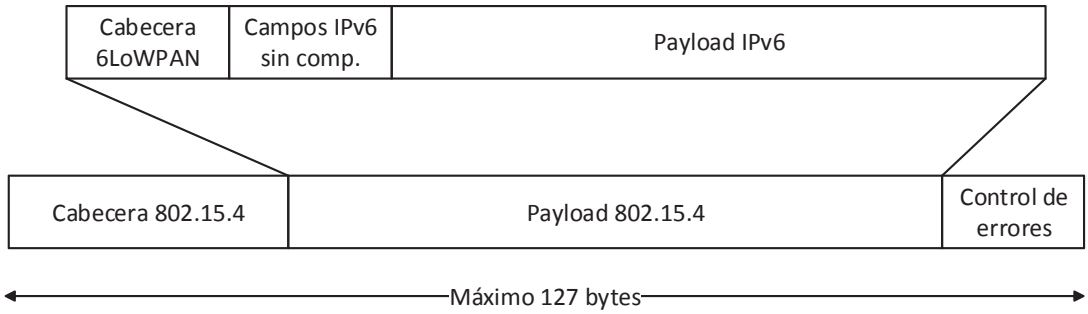


FIGURA 2.13: Formato de paquete 6LoWPAN

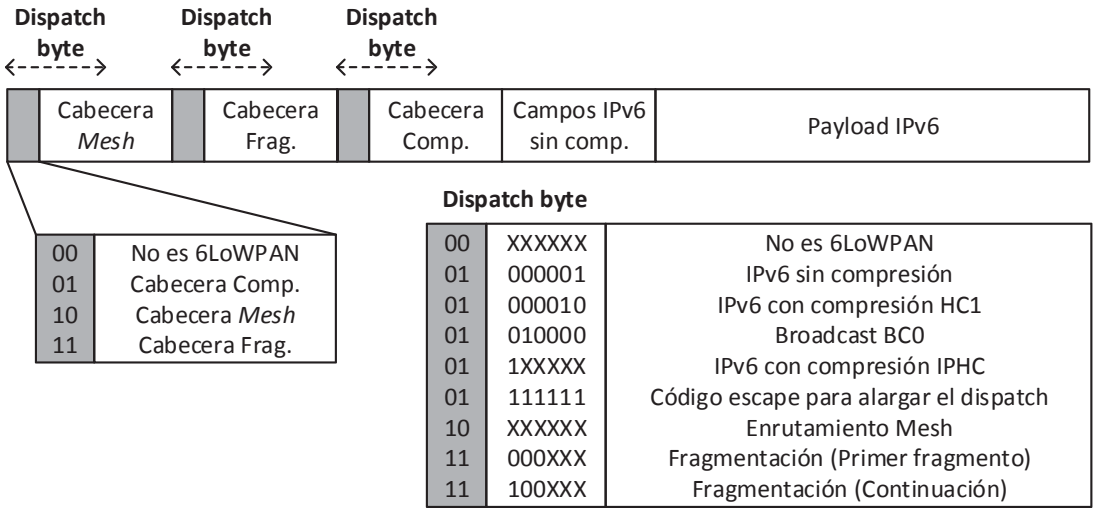


FIGURA 2.14: Dispatch Byte

2.5.3.3. Direccionamiento mallado o *mesh addressing*

Cuando la comunicación se realiza entre nodos de la misma red LoWPAN, puede ser interesante que la propia capa de adaptación sea la encargada del encaminamiento. En este caso se añade una cabecera 6LoWPAN que recibe el nombre de *mesh header* y que incluye las direcciones de nivel de enlace origen/destino y el número de saltos. Los nodos serán los encargados de formar la trama de nivel enlace a través de sus tablas de rutas con el propósito de que llegue al nodo destino en un número de saltos menor al exigido. En la Figura 2.15 se muestra el formato de la cabecera. Un estudio más detallado sobre *mesh-under* en el apartado de Encaminamiento 2.5.3.7.

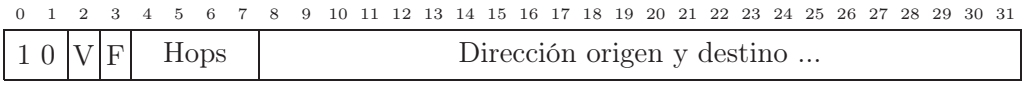


FIGURA 2.15: Formato cabecera Mesh

A continuación se analizan los campos en detalle:

- Bits V y F: Indican el tipo de direcciones (64 o 16 bits) que se usan para el origen y el destino respectivamente.
- Hops: Campo que se decrementa en cada salto. Si llega a 0 se descarta.
- Dirección origen y destino: Direcciones de nivel de enlace del origen y destino final.

Mediante la incorporación de la cabecera de *broadcast* BC0 seguida de la cabecera *mesh-under* se consigue dotar de capacidades de difusión dentro de la red, cuyo formato se muestra en la Figura 2.16.

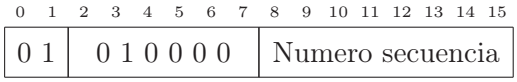


FIGURA 2.16: Formato cabecera BC0

El campo número de secuencia será incrementado por el autor del mensaje *broadcast* cada vez que mande un paquete.

2.5.3.4. Fragmentación y reensamblado

Uno de los objetivos esenciales de 6LoWPAN se basa incorporar los mecanismos necesarios para soportar la transmisión de tramas de nivel de enlace IPv6 con una *Maximum Transmission Unit* (MTU) mínima de 1280 bytes. El tamaño de una trama IEEE 802.15.4 de 127 bytes es notablemente inferior y por tanto no puede sustentar el tamaño mínimo fijado por IPv6. Es por esto, que todos los paquetes 6LoWPAN que superen el tamaño máximo de la carga útil de una trama IEEE 802.15.4, deberán ser fragmentados antes de su transmisión. En las Figuras 2.17 y 2.18 se muestran la estructura de trama y los formatos de cabecera definidos en 6LoWPAN para implementar satisfactoriamente la fragmentación.

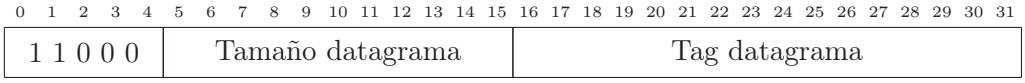


FIGURA 2.17: Formato cabecera de primer fragmento

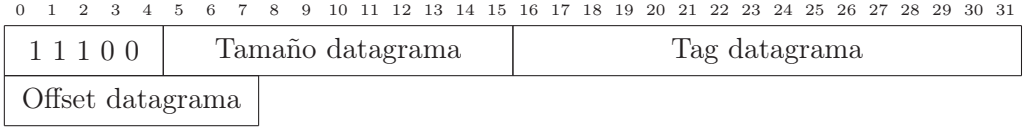


FIGURA 2.18: Formato cabecera de continuación fragmentos

Para optimizar el proceso de reensamblaje del paquete, se incluye el tamaño total del paquete 6LoWPAN en todos los fragmentos junto con el desplazamiento (*offset*), expresado en múltiplos

de 8 bytes, indicando la sección del paquete que se incluye. A modo de identificador de paquete, todos los fragmentos incluyen una misma etiqueta.

2.5.3.5. Compresión de cabeceras

La encapsulación directa de un paquete IPv6 sobre una trama IEEE 802.15.4 no presenta un rendimiento óptimo, pues la relación entre carga útil y cabeceras es muy baja. En el caso más general, representado en la Figura 2.19 se tiene:

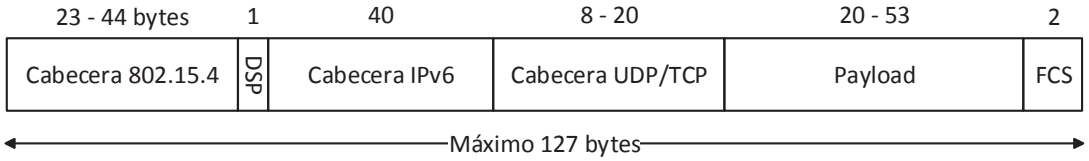


FIGURA 2.19: Encapsulación sin compresión en trama IEEE 802.15.4

- La cabecera MAC ocupa 23 bytes con direccionamiento EUI-64. Además, si se utilizan mecanismos de seguridad a nivel de enlace como el AES-CCM-128 (21 bytes), el tamaño de la cabecera aumenta hasta los 44 bytes. A esto hay que añadirle el campo *Frame Check Sequence* (FCS) de control de errores de 2 bytes.
- El *dispatch byte* (DSP) en este caso indica que se trata de un paquete de IPv6 sin compresión.
- La cabecera de red IPv6 tiene un tamaño fijo (sin cabeceras de extensión) de 40 bytes.
- Los protocolos de transporte *User Datagram Protocol* (UDP)/TCP ocupan 8 y 20 bytes respectivamente.

Teniendo en cuenta que el tamaño de la trama a nivel de enlace es de 127 bytes, se puede conocer cual es el *payload* a nivel de transporte, esto es, los bytes útiles que se mandan las aplicaciones. En la Tabla 2.4 se presenta la eficiencia resultante de la transmisión de un paquete TCP y UDP sobre 6LoWPAN.

	UDP		**TCP**	
	# Bytes	**Efic. (%)**	**# Bytes**	**Efic. (%)**
Sin seguridad	53	41.7	41	32.3
AES-CCM-128	32	25.2	20	15.7

TABLA 2.4: *Payload* a nivel de transporte sin compresión

Resulta evidente que IPv6 no está diseñado para trabajar en redes IEEE 802.15.4. En el mejor de los casos (sin seguridad a nivel de enlace y usando UDP como protocolo de transporte) solo el 41.7 % de los bytes transmitidos son información de aplicación. Es por esto, que 6LoWPAN añade mecanismos de compresión para los paquetes IPv6. Esencialmente, se basan en la compresión de la cabecera de red IPv6 y la de transporte UDP. Para el protocolo TCP el estándar no define ninguna solución de compresión debido a que no es habitual utilizar TCP en redes LoWPAN.

Actualmente existen dos formatos de compresión de cabeceras en 6LoWPAN. El primero descrito en el RFC 4944 [2] recibe el nombre de *Header Compression 1* (HC1)-*Header Compression 2* (HC2) y es un mecanismo obsoleto a fecha de hoy. Aunque las nuevas implementaciones de 6LoWPAN no deberían usar este mecanismo para la compresión, si que puede implementarse para descomprimir paquetes basadas en la versión antigua del estándar. El segundo formato de compresión de cabeceras se define en el RFC 6282 [15] y se conoce como *IP Header Compression* (IPHC)-*Next Header Compression* (NHC). Este mecanismo presenta ventajas respecto a la primera versión del estándar, la más importante conocida como compresión de direcciones basada en contexto que se explica más adelante.

Compresión HC1-HC2

El primer método de compresión del estándar se basa esencialmente en la eliminación de redundancia de la cabecera IPv6 (HC1) apoyándose en la información que ofrece la cabecera MAC. Además ofrece un mecanismo de compresión (HC2) para la cabecera UDP. En la Figura 2.20 se muestra el formato general de compresión utilizado.

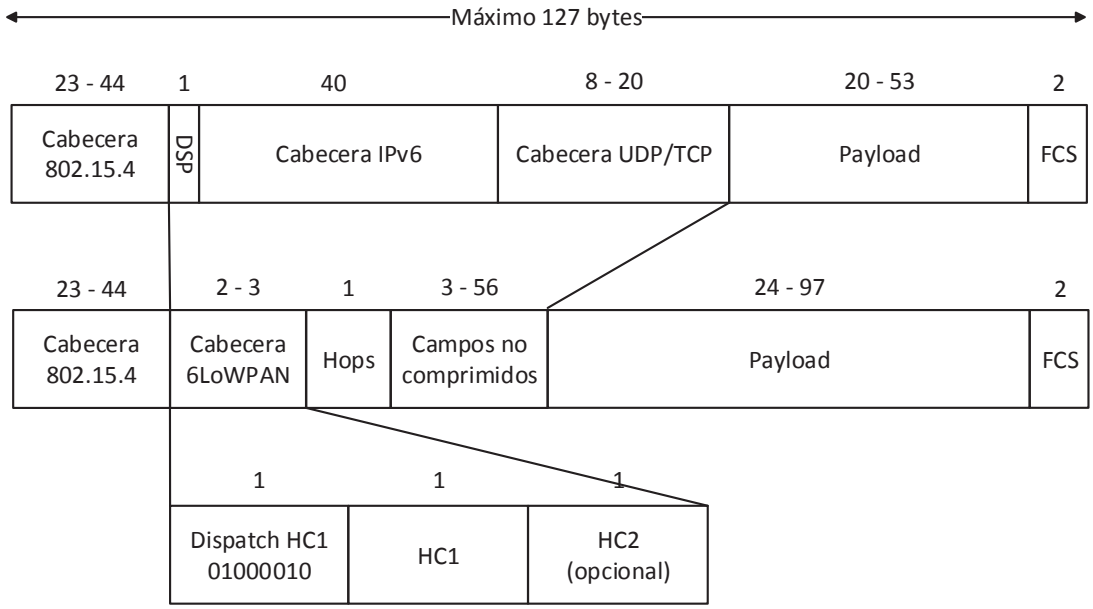


Figura 2.20: Compresión de cabeceras HC1-HC2

Los campos que forman la cabecera de IPv6, salvo el número de saltos, pueden ser eliminados, ya que son irrelevantes (clase de tráfico, etiqueta de flujo, próxima cabecera, etc.) o al menos no son estrictamente necesarios. En algunos casos, los campos asociados al direccionamiento también se pueden comprimir, por ejemplo, dentro de una misma red basta con conocer las direcciones MAC que identifican unívocamente los nodos. Para informar de los procesos de compresión empleados es necesario añadir una cabecera HC1. En la Figura 2.21 se muestra el formato. Los campos que forman la cabecera son los siguientes:

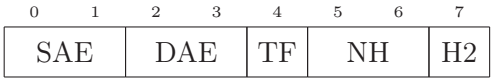


FIGURA 2.21: Formato cabecera HC1

- SAE (*Source Address Encoding*): Estos dos bits indican el tipo de compresión de la dirección IPv6 de la fuente. El primero indica si se suprime el prefijo de red y el segundo el identificador del dispositivo.
- DAE (*Destination Address Encoding*): Similar al SAE pero para la dirección del destino.
- TF (*Traffic Class and Flow Label*): Este bit indica si se eliminan los campos Clase de tráfico y Etiqueta de flujo de la cabecera IPv6.
- NH (*Next Header*): Dos bits que informan del tipo de cabecera que viene por encima de IPv6. (00 desconocida, 01 UDP, 10 ICMPv6, 11 TCP).
- H2: Este bit indica que se va a utilizar compresión de cabecera HC2 para el transporte de datos UDP. En este caso el campo NH debe contener 01 (UDP).

Por otra parte, como se ha señalado, la compresión HC2 permite reducir la cabecera UDP. De igual manera, es necesario añadir una cabecera con la información asociada a la compresión. En la Figura 2.22 se muestra el formato de dicha cabecera que está formada por los siguientes campos:

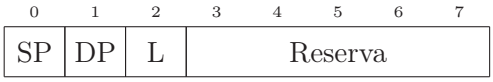


FIGURA 2.22: Formato cabecera HC2

- SP (*Source Port*): Indica si el puerto de la fuente de la cabecera UDP se comprime (16 bits a 4 bits).
- DP (*Destination Port*): Similar al bit SP pero con el puerto destino.

- L (*Length*): Indica si se suprime el campo de longitud de la cabecera UDP. Este valor se puede obtener a partir del de la cabecera IPv6
- Reserva: Estos bits no tienen uso actualmente.

Los campos no comprimidos irán tras el contador de saltos. Los correspondientes a IPv6 seguirán el orden de la cabecera HC1, mientras que los correspondientes a UDP (en caso de utilizarlo) seguirán el orden de la cabecera UDP.

En conclusión, el mecanismo de compresión de cabeceras HC1-HC2 permite en gran medida aumentar el tamaño de información útil a base de eliminar información redundante. En la Tabla 2.5 se muestra el *payload* de transporte y la eficiencia de byte para el caso de una compresión máxima de la cabecera IPv6 y de la cabecera UDP.

	UDP		**TCP**	
	# Bytes	**Efic. (%)**	**# Bytes**	**Efic. (%)**
Sin seguridad	97	76.4	79	62.2
AES-CCM-128	76	59.8	58	45.7

Tabla 2.5: *Payload* a nivel de transporte con compresión HC1-HC2

Es evidente que la compresión de cabecera HC1-HC2 incrementa notablemente la eficiencia. En el mejor de los casos (usando UDP y sin seguridad a nivel de enlace) se consigue elevarla hasta un 75 %, significativamente superior al 41.7 % que se tenía sin aplicar compresión. Esto significa que de cada 4 bytes que se transmitan, 3 de ellos van a ser información útil a nivel de transporte. Sin embargo, gran parte de este incremento es debido a la eliminación de las direcciones IPv6. En el momento de que un dispositivo 6LoWPAN quiera comunicarse con un nodo externo a su propia red (algo habitual en este tipo de redes) no es suficiente con conocer la dirección MAC del nodo, pues deberá saber cual el prefijo de red para enrutar el paquete. En aras de optimizar la compresión de los prefijos de red en estos casos, surge el concepto de compresión basada en contexto, que es la esencia de la compresión de cabeceras IPHC-NHC, el mecanismo sustituto de HC1-HC2 y actual del estándar.

Compresión basada en contexto: IPHC-NHC

El mecanismo de compresión de cabeceras IPHC-NHC definido en el RFC 6282 [15] reemplaza al HC1-HC2 incluido en el RFC 4944 [2]. Aunque ambos pueden funcionar de forma paralela, el estándar recomienda que todas las implementaciones nuevas usen el formato de compresión IPHC-NHC para la compresión y transmisión de paquetes y solo el HC1-HC2 para la descompresión, favoreciendo la interoperabilidad y el traspaso hacia el nuevo mecanismo.

Para implementar el proceso de compresión, los últimos 5 bits del campo del *dispatch byte* se aprovechan para la propia cabecera IPHC. Asimismo, a la agregación de distintas cabeceras le continúan los campos no comprimidos de la misma, a diferencia de HC1-HC2 que juntaba todas las cabeceras y a continuación todos los campos no comprimidos. El formato general se muestra en la Figura 2.23.

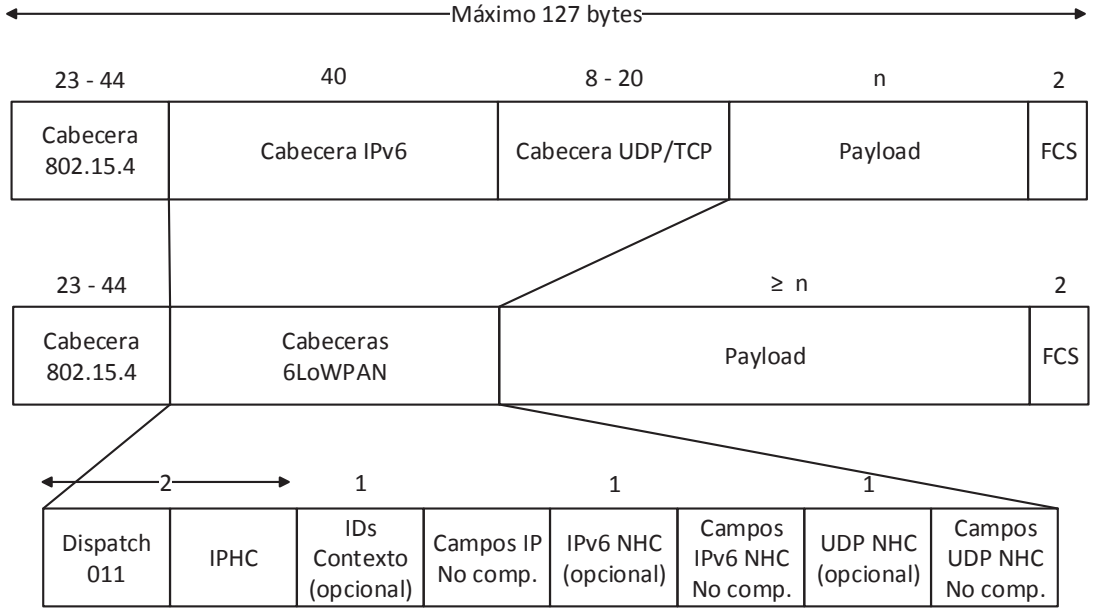


Figura 2.23: Compresión de cabeceras IPHC-NHC

La cabecera IPHC es la encargada de contener la información de cómo se lleva a cabo la compresión de la cabecera IPv6. El formato de la cabecera se muestra en la Figura 2.24.

0	1	2	3	4	5	6	7
0	1	1	TF		NH	HLIM	
CID	SAC	SAM		M	DAC	DAM	

Figura 2.24: Formato cabecera IPHC

- TF (*Trafic Class, Flow Label*): Indican si se suprime el campo Clase de tráfico y/o Etiqueta de flujo de la cabecera IPv6.
- NH (*Next Header*): Indica si se suprime el campo NH de la cabecera IPv6 y usar NHC.
- HLIM (*Hop Limit*): Indica si el campo Hops de la cabecera IPv6 se comprime además de limitar el contador a un valor máximo de 1, 64 o 255.
- CID (*Context Identifier Extension*): Informa sobre el uso opcional del campo de extensión de identificador de contexto.

- SAC (*Source Address Compression*): Indica si la dirección IPv6 de la fuente va a usar compresión *stateful* (basada en contexto) o compresión *stateless*.
- SAM (*Source Address Mode*): En función del bit comentado anteriormente, se tienen 4 modos de compresión para cada tipo (stateless o stateful). Las direcciones IPv6 pueden no comprimirse, comprimirse a 64 bits, 16 bits, o comprimirse completamente.
- M (*Multicast Compression*): Indica si la dirección IPv6 destino es *multicast*.
- DAC (*Destination Address Compression*): Similar al SAC pero para el destino.
- DAM (*Destination Address Mode*): Similar al SAM pero para el destino. Si el bit M está activado se sigue una compresión idónea (48, 32 u 8 bits) para direcciones *multicast*.

La cabecera opcional de extensión de identificador de contexto proporciona información acerca del contexto que se usa para la compresión del prefijo de red de las direcciones IPv6. El formato se muestra en la Figura 2.25.

FIGURA 2.25: Formato cabecera IDs Contexto

- SCI (*Source Context Identifier*): Identificador del prefijo que se usa para la compresión basada en contexto de la dirección IPv6. Hasta 16 posibles contextos.
- DCI (*Destination Context Identifier*): Similar al SCI pero para el destino.

Los campos de IP no comprimidos aparecen en el mismo orden que la cabecera IPv6.

La cabecera opcional IPv6 NHC proporciona mecanismos de extensión de IPv6. Para ello bit NH de la cabecera IPHC debe estar activado. El formato se muestra en la Figura 2.26.

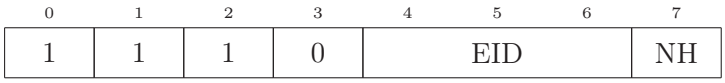


FIGURA 2.26: Formato cabecera IDs Contexto

- EID (*Extension Header ID*): Indica el tipo de información añadida que va en el campo sin compresión de IPv6 NHC: opciones de enrutamiento, fragmento, salto, movilidad...
- NH (*Next Header*): Indica si la próxima cabecera se comprime usando NHC.

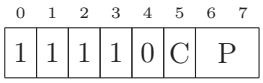


FIGURA 2.27: Formato cabecera UDP NHC

Por último, la cabecera UDP NHC permite la compresión de la cabecera UDP. El formato de la cabecera se muestra en la Figura 2.27.

- C (*Checksum*): Indica si el campo de *Checksum* de la cabecera UDP se comprime.
- P (*Ports*): Indica si los puertos origen/destino se comprimen de 16 bits a 4 bits, de forma idéntica a como se hacia en la compresión HC2.
- A diferencia con la compresión HC2, el campo longitud debe ser siempre comprimido.

En conclusión, el mecanismo de compresión de cabeceras IPHC-NHC logra una mejora significativa desde el punto de vista de la compresión de direcciones IPv6 cuando se quieren establecer comunicaciones con el exterior de una red LoWPAN. En la Tabla 2.6 se muestra el *payload* de transporte y la eficiencia de byte para el caso de una compresión máxima de la cabecera IPv6 y de la cabecera UDP.

		UDP		**TCP**	
		# Bytes	**Efic. (%)**	**# Bytes**	**Efic. (%)**
Sin seguridad	*Stateless*	98	77.2	79	62.2
	Stateful	97	76.4	78	61.4
AES-CCM-128	*Stateless*	77	60.6	58	45.7
	Stateful	76	59.8	57	44.9

TABLA 2.6: *Payload* a nivel de transporte con compresión IPHC-NHC

2.5.3.6. Neighbor Discovery

El protocolo ND utilizado en IPv6 no resulta eficiente para redes 6LoWPAN. Es por esto que 6LoWPAN define en el RFC 6775 [16] una versión modificada denominada 6LoWPAN-ND.

Durante el proceso de inicialización o *bootstrapping* (ver Figura 2.28), el *router* de borde es el encargado de distribuir el prefijo de red que el resto de nodos deberán usar para generar sus direcciones IPv6. Para ello un *host* que quiera formar parte de una red LoWPAN debe solicitar al *router* información de la red, entre ella el prefijo de red, mediante un mensaje *Router Solicitation* (RS) que será respondido por otro mensaje *Router Advertesiment* (RA). A continuación, el *host* solicita registrarse en el *router* de borde mediante un mensaje *Node Registration* (NR) en el que

se indicará la dirección de red con la que quiere identificarse, bien generada por el método de auto-configuración o manualmente. Cuando el *router* de borde reciba el mensaje llevará a cabo un mecanismo de comprobación de direcciones duplicadas, y en caso de que no exista ninguna y apruebe la incorporación del nuevo nodo a la red, responderá con un mensaje *Node Confirmation* (NC).

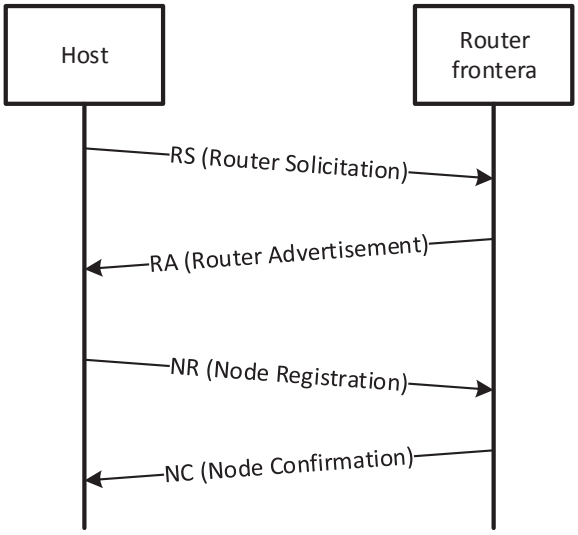


FIGURA 2.28: *Bootstrapping* 6LoWPAN-ND

En situaciones en las que un *host* no es capaz de alcanzar al *router* de borde, los *router* de la red, que son los únicos nodos con capacidad para el encaminamiento, actúan como intermediarios en la comunicación, colaborando también en la propagación del prefijo de red. En determinadas implementaciones, los *router* deben además almacenar temporalmente la información destinada a los *host*, ya que es posible que estos se encuentren dormidos.

En la operativa del protocolo ND destaca también la eliminación de direcciones basadas en *multicast* con el fin de reducir ahorro energético de los dispositivos y requerir que las interacciones *host-router* siempre sean iniciadas por los *host* para facilitar el estado de dormido de los dispositivos.

2.5.3.7. Encaminamiento

El encaminamiento en 6LoWPAN es uno de los puntos más delicados debido a las limitaciones en términos de consumo y procesamiento que tienen los nodos de la red. Existen diferentes protocolos que intentan solventar estos problemas con el fin de adaptarse a este tipo de redes. En función de la capa en la que se lleve a cabo el encaminamiento se distingue:

- **Mesh-Under (L2):** Es la capa de adaptación la encargada de gestionar el encaminamiento a través de las direcciones de enlace. Para ello 6LoWPAN hace uso de la cabecera *Mesh*

explicada anteriormente. El nodo que envía el paquete escribe en la cabecera de adaptación su dirección MAC y la dirección MAC del dispositivo final. Sin embargo, es la capa MAC la que contiene la dirección del propio nodo y la del próximo salto. Cuando la trama de enlace llega a un dispositivo que no es el nodo final, actualiza las direcciones de la cabecera MAC y decrementa el número de saltos de la cabecera *Mesh* antes de encaminar hacia el próximo nodo. Este tipo de encaminamiento tiene el inconveniente de que cuando se produce fragmentación y uno de los fragmentos se pierde, se debe retransmitir el paquete IP desde el nodo principal, aumentando drásticamente el gasto energético de la red. Además, *mesh-Under* no permite comunicar la LoWPAN con el exterior.

- **Router-Over (L3):** En este caso es la capa de red la encargada del encaminamiento, de tal manera que cada salto a nivel MAC se considera un salto a nivel IPv6. Cuando se produce fragmentación a nivel IPv6, los fragmentos son enviados al siguiente salto utilizando la información de las tablas de rutas. La capa de adaptación del siguiente salto se encarga de comprobar que los fragmentos recibidos son correctos y en tal caso, vuelve a juntarlos para enviarlos a la capa de red. En caso de que un fragmento se pierda, se retransmite desde el nodo anterior, a diferencia del encaminamiento *mesh-under*. Sin embargo, *router-over* tiene un alto consumo energético debido a los dominios *multicast* de los enlaces IPv6.

Ambos tipos de encaminamiento no resultan apropiados para redes LoWPAN. Es por esto, que el IETF y el grupo de trabajo ROLL trabaja en el protocolo de encaminamiento RPL [17] con el fin de definir un protocolo de enrutamiento eficiente para este tipo de redes. Tras previamente comparar protocolos ya existentes como *Ad-Hoc On Demand Distance Vector* (AODV) o *Dynamic MANET On-demand* (DYMO) y no obtener resultados satisfactorios, optaron por crear este nuevo protocolo, que fue publicado por primera vez en agosto de 2009.

La esencia del protocolo de enrutamiento RPL se basa en el modelo multisalto, en el que un nodo que transmite a la estación base, lo hace reenviando sus datos a uno de sus vecinos hasta que llega al destino. Está diseñado para redes inalámbricas que se implementan en un entorno con pérdidas donde es probable que se pierdan mensajes. El objetivo principal de RPL es proporcionar caminos de enrutamiento eficientes para tres tipos de comunicaciones: multipunto a punto, *Multipeer-to-Peer* (MP2P), punto a multipunto, *Peer-to-Multipeer* (P2MP) y punto a punto, P2P. Para llevar a cabo la comunicación, RPL distingue hasta tres tipos de mensajes que se encapsulan en paquetes ICMPv6 y son los siguientes:

- *DODAG Information Object* (DIO): Los envían los nodos RPL para informar sobre el estado del *Destination Oriented DAG* (DODAG). Sirve para que un nodo conozca a sus vecinos y sus respectivos *ranks*.

- *Destination Advertisement Object* (DAO): Se usan para propagar información de direccionamiento hacia los nodos RPL con rango superior, alimentando las tablas de rutas de los nodos padres.
- *DODAG Information Solicitation* (DIS): Se usan para solicitar mensajes DIO y descubrir los DODAG disponibles.

En conclusión, RPL es un protocolo que proporciona una configuración eficaz de rutas en entornos de redes de sensores, incluso en situaciones en que la topología de la red cambie. Está plenamente orientado para este tipo de escenarios, siendo a día de hoy una de las soluciones más utilizadas de forma paralela a 6LoWPAN.

2.5.3.8. Seguridad

El estándar propone diferentes mecanismos de seguridad para cada una de las capas que emplea. En la capa de enlace se puede usar mecanismos de encriptación basados en el cifrado de bloques AES, como se comentó en la descripción de IEEE 802.15.4. A nivel de red puede emplearse el protocolo *Internet Protocol security* (IPsec) (versión borrador optimizada para 6LoWPAN) [18] y así garantizar la seguridad extremo a extremo. Para el transporte se recomienda utilizar protocolos basados en *Transport Layer Security* (TLS), como es el caso de la versión adaptada a UDP *Datagram Transport Layer Security* (DTLS). En cualquier caso, los mecanismos de seguridad que propone 6LoWPAN son aún un tema de debate dentro del IETF.

2.6. Plataformas de implementación

En la actualidad existen desarrollos y soluciones que permiten el despliegue de redes de sensores basadas en 6LoWPAN. Desde un punto de vista *software*, es interesante un estudio de las distintas implementaciones disponibles para conocer las diferencias o limitaciones respecto al estándar original. En la Tabla 2.7 se muestra un resumen de las soluciones de código abierto más conocidas junto con las características mas críticas.

El sistema operativo Contiki OS [19], con licencia *Berkeley Software Distribution* (BSD) y programado en C y C++, utiliza la pila de protocolos uIP6, que además de ser una implementación que trae consigo protocolos como TCP o ND, cuenta con el logo IPv6 Ready [20] que certifica su robustez. En cuanto a la implementación de 6LoWPAN, que recibe el nombre de SICSLOWPAN, ofrece soporte para la fragmentación y compresión de cabeceras. Además, aunque este sistema operativo no soporta encaminamiento *mesh-under*, tiene implementado una versión estable del protocolo de enrutamiento RPL.

Sistema Operativo	**Contiki OS**	**Tiny OS**	**FreeRTOS**
Licencia	BSD	BSD	GPL
Lenguaje	C, C++	nesC	C
Pila TCP/IP (IPv6)	**uIP6**	**BLIP**	**NanoStack**
Auto-configuración	SI	SI	SI
UDP	SI	SI	SI
TCP	SI	Prototipo	NO
ICMP	SI	SI	SI
ND	SI	Limitado	NO
6LoWPAN	**SICSLOWPAN**	**b6lowpan**	**NanoStack**
Fragmentación	SI	SI	SI
Compresión	HC-06	HC-01	HC-01
Mesh-under	NO	SI	SI
RPL	SI	SI	NO

Tabla 2.7: Implementaciones *software* de 6LoWPAN

Tiny OS [21], también con licencia BSD pero escrito en nesC, se basa en la pila de comunicaciones BLIP. A pesar de que esta implementación tiene una solución reducida de los protocolos TCP y ND, el soporte del estándar 6LoWPAN permite el encaminamiento *mesh-under*, además del enrutamiento RPL.

Por otra parte, FreeRTOS [22] utiliza licencia *General Public License* (GPL). Desarrollado en el lenguaje de programación C y apoyándose en la pila NanoStack, permite el soporte IPv6 pero con algunas limitaciones respecto a los protocolos TCP y ND. Aun así, la implementación de 6LoWPAN trae consigo los mecanismos de compresión de cabeceras y encaminamiento *mesh-under*.

Capítulo 3

Despliegue de la red

Tras el análisis del estado del arte y conocer las diferentes opciones que existen para la implementación de las redes de sensores basadas en el estándar 6LoWPAN, a continuación se describe el escenario planteado para el conjunto de este proyecto, seleccionando a partir de las opciones *software* analizadas, la que mejor se adecue a los requerimientos extraídos de dicho escenario. Además, se realiza el diseño del escenario en diferentes fases aumentando la complejidad del sistema, desde el despliegue de una sencilla red LoWPAN hasta la interconexión de cualquier nodo de ella con la red Internet.

3.1. Descripción del escenario

Como se ha comentado anteriormente, el objetivo de este proyecto es validar las posibilidades de ofrecer servicios a través de una red de sensores 6LoWPAN permitiendo el acceso remoto a los diferentes nodos que la conforman. En este sentido el primer paso a acometer es el diseño y configuración de la red que soportará esos servicios y proporcionar los medios necesarios para establecer comunicaciones extremo a extremo entre cualquier nodo de la LoWPAN y cualquier dispositivo de la red Internet.

Si bien el número de nodos de la red puede ser muy elevado, en el marco de este proyecto se ha planteado un escenario simple donde la red de sensores está formada por un número reducido de nodos (dos en concreto) y la red de acceso se emulará por el mínimo de nodos necesarios (LoWPAN simple). El diagrama de la Figura 3.1 representa el escenario planteado, donde la mota 1 hace las funciones de *host* y la mota 2 de *router*, realizando además de las tareas habituales de nodo sensor, el encaminamiento para permitir el multisalto dentro de la red. Un tercer nodo funcionará de *router* de borde o *edge router* permitiendo la interconexión de la red LoWPAN con el exterior. Toda vez la plataforma se encuentre operativa, el funcionamiento incluirá el acceso a nivel de red entre el nodo 1 y cualquier dispositivo conectado a la red Internet.

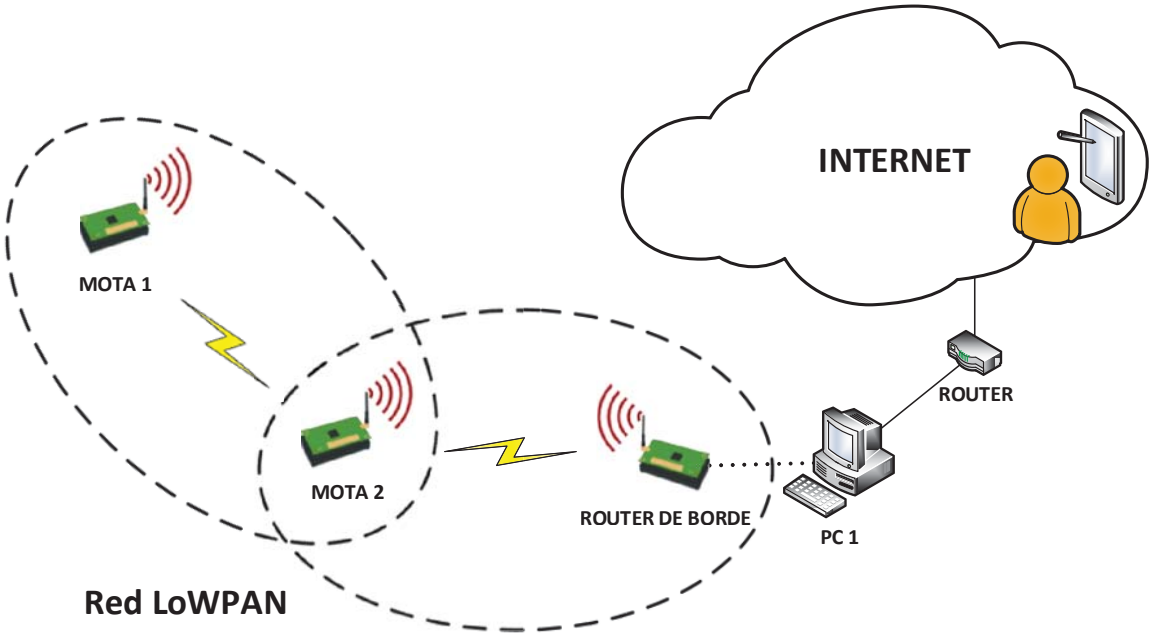


FIGURA 3.1: Descripción general del escenario

En este capítulo como se ha insistido, únicamente se validará la conectividad a nivel de enlace y red entre los dispositivos, dejando para secciones posteriores la descripción de la operativa a nivel de aplicación y servicio.

3.2. Plataforma de desarrollo

A la vista del estudio realizado en el capítulo anterior, seguidamente se describen las plataformas tanto *hardware* como *software* seleccionadas por considerarlas las más adecuadas para la realización de este proyecto. De dicho estudio se ha observado que existen numerosas similitudes entre todas las soluciones *hardware*, si bien Contiki ofrece una implementación más completa de la pila de protocolos TCP/IP. Dado que en este proyecto se plantea la evaluación de un entorno de red 6LoWPAN, éste ha sido el principal motivo por el que se selecciona Contiki como el sistema operativo más adecuado.

Considerando Contiki frente a otras alternativas, se analizan a continuación las plataformas *hardware* que lo soportan (ver Tabla 3.1). Existe un amplio abanico de opciones. De entre todas, se ha seleccionado la plataforma `mbxxx` que utiliza un microcontrolador de ST con un chip STM32W, el cual soporta multitarea. El kit de desarrollo STM32W108C incluye equipos con este microcontrolador, además de sensores de humedad, temperatura, etc. Todo ello hace posible la implementación de los desarrollos previstos para este proyecto.

MCU/SoC	Radio	Plataforma
RL78	ADF7023	EVAL-ADF7023DB1
TI CC2538	Integrado	cc2538dk
TI MSP430x	TI CC2420	exp5438, z1
TI MSP430x	TI CC2520	wismote
Atmel AVR	Atmel RF230	avr-raven, avr-rcb, avr-zigbit
Atmel AVR	TI CC2420	micaz
Freescale MC1322x	Integrado	redbee-dev, redbee-econotag
ST STM32w	Integrado	mb851, mbxxx
TI MSP430	TI CC2420	sky, jcreate, sentilla-usb
TI MSP430	TI CC1020	msb430
TI MSP430	RFM TR1001	esb
Atmel Atmega128	Integrado	avr-atmega128rfa
pic32mx795f412l	mrf24j40	seed-eye
TI CC2530	Integrado	cc2530dk
RC2300/RC2301	Integrado	sensinode

Tabla 3.1: *Hardware* soportado por Contiki
[23]

3.2.1. *Software* - Contiki OS 2.7

Contiki OS es un sistema operativo *open source*, portable y multitarea [24]. Está escrito en lenguaje C y utiliza el concepto de *Protothread* [1] [25], lo que lo caracteriza con un uso muy reducido de la memoria del microprocesador en comparación con otras arquitecturas multitarea. Está enfocado para usarse en sistemas sencillos y empotrados sobre microcontroladores, como pueden ser los nodos de una red de sensores.

La idea de este sistema operativo es tener varios hilos de ejecución en paralelo, los cuales esperan ciertas condiciones para activarse. Los hilos al activarse realizan sus tareas pertinentes y al terminarlas quedan nuevamente en espera hasta que otro evento los active. Los eventos son los que se encargan de activar a los hilos. Estos se guardan en una cola circular y se procesan por orden cronológico. Cada evento generado tiene un hilo asociado para despertar, al que le puede enviar información cuando lo despierta.

La distribución de Contiki empleada incluye los ficheros del kernel de sistema, los diferentes microcontrolodores soportados y las adaptaciones de las diferentes librerías según la plataforma *hardware* que se emplee. Adicionalmente incluye diversas utilidades e implementaciones de protocolos para operar a nivel de aplicación (CoAP, *File Transfer Protocol* (FTP), *Teletype Network* (Telnet), etc.) en las que el usuario podrá apoyarse para el desarrollo de sus soluciones. A todo ello, hay que unir las herramientas necesarias para la compilación tanto del propio sistema operativo como de los programas desarrollados por el usuario.

[1] Metodología de programación basada en macros que se utiliza en microprocesadores con bajos recursos de forma similar a un sistema de tareas pero sin utilizar memoria *Random Access Memory* (RAM) para cada uno de los hilos que se ejecutan, con las limitaciones que ello pueda tener.

Contiki proporciona el simulador de redes Contiki, conocido como COOJA, que permite desarrollar, ejecutar y validar el funcionamiento de aplicaciones en las distintas plataformas sin tener que disponer del *hardware* necesario.

De forma básica, la configuración del sistema operativo a las necesidades se realiza fijando un conjunto de directivas que dependen de la plataforma. Entre los parámetros de configuración de red empleados se encuentran las directivas para fijar el PAN ID, uso de IPv6, tipo de compresión de 6LoWPAN, uso de TCP, etc. Al compilar la imagen que se cargará en la plataforma *hardware*, la modificación de estos parámetros modificará el comportamiento del sistema final.

3.2.1.1. Arquitecturas de red

Contiki OS ofrece dos pilas de comunicaciones para redes de sensores. Por una parte, la pila Rime [26] se trata de un *stack* modular estructurado en capas muy simples con el fin de conseguir encabezados con el menor tamaño posible. Rime es una pila de comunicación ligera diseñada para radios de baja potencia que ofrece una amplia gama de primitivas. Por otra parte, y relevante para los objetivos del proyecto, Contiki también tiene soporte para una pila de comunicaciones TCP/IP, que recibe el nombre de uIP [27]. Además, la adaptación de uIP a uIP6 le convierte en el único sistema operativo embebido con pila de comunicaciones TCP/IP robusta con certificado IPv6 Ready Logo.

El sistema operativo divide la pila de comunicaciones en una serie de capas y protocolos tal y como se muestra en la Figura 3.2, donde los módulos sombreados son los que han sido habilitados para el desarrollo de este proyecto. A continuación se profundiza en la pila de protocolos que hace posible el despliegue de una red de sensores basadas en el estándar 6LoWPAN, dejando de lado el resto de posibilidades que ofrece el sistema operativo, como por ejemplo el pila Rime.

Capa Radio La capa de radio se encarga de la transmisión y recepción de tramas a través del medio inalámbrico por medio de los *drivers* de la plataforma especificada. Para el caso del kit STM32W108C el *driver* empleado es *stm32w_radio_driver*, que opera en la banda de 2.4GHz bajo el estándar IEEE 802.15.4.

Capa Framer Esta capa se encarga de dar formato al paquete bajo el estándar IEEE 802.15.4 en caso que fuese necesario a través del protocolo *framer_802154*. En caso contrario, se podría ignorar utilizando *framer_nullmac*, aunque no será posible si se desea un escenario 6LoWPAN.

Capa RDC En las redes de baja potencia, el transceptor radio es recomendable que se apague siempre que sea posible con el fin de ahorrar energía e incrementar la autonomía de las baterías.

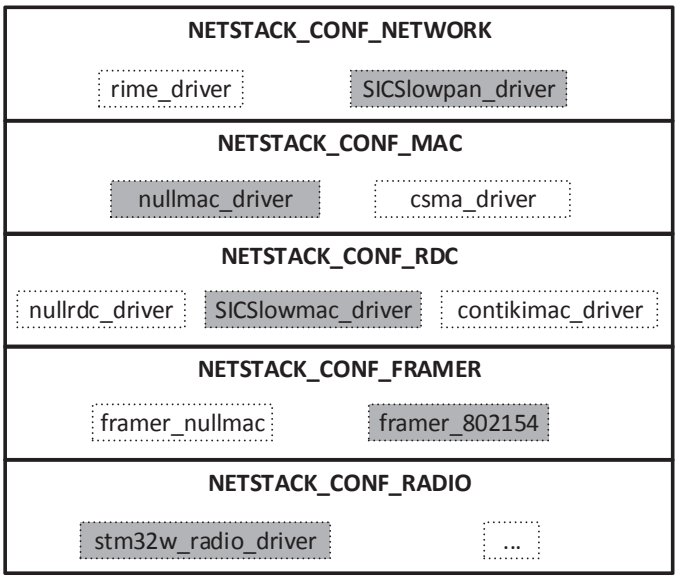


FIGURA 3.2: Pila de protocolos de Contiki

En Contiki esto se consigue por medio de la capa *Radio Duty Cycling* (RDC). Sin embargo, no todas las radios son compatibles con este *driver*. En el desarrollo del proyecto se ha seleccionado para esta capa el *driver SICSlowmac_ driver* que no utiliza ningún protocolo RDC para llevar a cabo el dormido de los dispositivos.

Capa MAC La capa MAC de Contiki es responsable de evitar las colisiones en el medio inalámbrico y retransmitir las tramas en caso de existir colisiones. Contiki proporciona dos tipos de *driver*: uno que implementa el mecanismo CSMA/CA [28] y otro NullMAC. La principal diferencia entre ellos es que NullMAC se basa en el protocolo Aloha [28] para la transmisión de tramas y en caso de colisionar lo intenta transmitir más adelante. En cambio, utilizando CSMA/CA cada nodo anuncia su intención de transmitir antes de hacerlo para así evitar colisiones. De forma análoga a lo que pasa con la capa RDC, no todas las radios son compatibles con el *driver*, luego no siempre es posible utilizar CSMA/CA. Debido al bajo número de nodos que se espera tener en el desarrollo del proyecto se ha decidido utilizar el *driver nullmac_ driver*, sin necesidad de emplear el mecanismo CSMA/CA.

Capa de red En esta capa se selecciona el protocolo a nivel de red que se desea emplear. Se ha comentado anteriormente que Contiki ofrece dos tipos de arquitecturas, Rime o uIP. La implementación de 6LoWPAN que incorpora uIP recibe el nombre de SICSLOWPAN, tal y como se muestra en la Figura 3.3. Siguiendo las directrices fijadas por la especificación RFC 4944 [2], soporta compresión de cabeceras IPHC hasta la versión HC-06 descrita en el RFC 6282 [15], lo que le convierte en una implementación estable del estándar. Sin embargo, no soporta el mecanismo *mesh-under* ni el direccionamiento corto de 16 bits de IEEE 802.15.4. En cualquier

caso, Contiki tiene implementado el protocolo de enrutamiento RPL [29], enfocado esencialmente para el mundo de las redes LoWPAN.

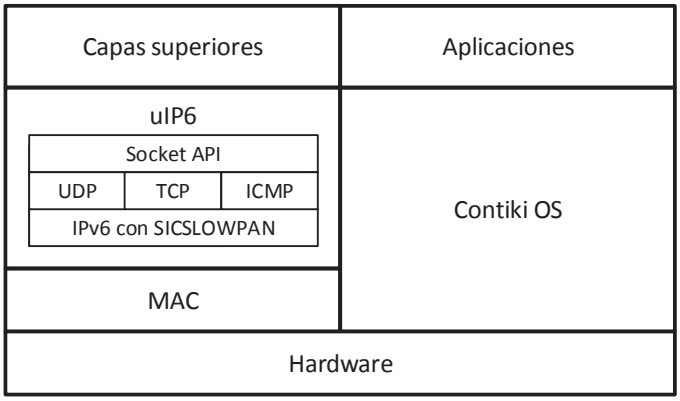


FIGURA 3.3: Stack de 6LoWPAN de Contiki

Tomando como referencia las características descritas y que además la implementación actualmente disponible se encuentra en versión estable de desarrollo, se considera como la opción más adecuada para poder cumplir los objetivos del presente proyecto.

3.2.2. *Hardware* - STM32W108C

Como se ha comentado anteriormente, se ha adquirido el kit de desarrollo STM32W108C como solución *hardware* sobre la que se lleve a cabo el desarrollo de todo el proyecto. El contenido del kit, que se muestra en la Figura 3.4, es el siguiente:

- Dos placas de aplicación STM32W108 (MB851)
- Un pincho USB STM32W108 (MB951)
- Un J-Link Lite JTAG Flash programmer y debugger
- Un cable mini USB
- Cuatro pilas AAA

Las plataformas *hardware* autónomas son MB851, cuyas funcionalidades se recogen en la Tabla 3.2. Destacar que incluyen sensores de temperatura ambiente y acelerómetro (sobre los 3 ejes), que serán de utilidad para validar los desarrollos de este proyecto con aplicaciones de sensado reales. Por otra parte, el nodo USB utiliza la plataforma MB951 que, aunque dispone de similares capacidades de cómputo y comunicaciones IEEE 802.15.4, no incluye los sensores de la MB851 y tiene un menor número de interfaces de entrada/salida. En la Tabla 3.3 se resumen sus especificaciones.

FIGURA 3.4: Contenido del STM32W108C-SK

Funcionalidad	**Etiqueta**	**Chip / Tipo**	**Pin**
Botón	RST1	-	-
Botón	S1	-	PB3
EEPROM	U4	M24C64MW1 128 KBytes	PA1, PA2
Led	D1	Visible	PA5
Led	D3	Visible	PA0
Led	D5	Infrarrojo	PB4
Acelerómetro	U2	LIS302	PA1, PA2
Sensor Temperatura	U3	STLM20	PA4

TABLA 3.2: Funcionalidades plataforma MB851

Funcionalidad	**Etiqueta**	**Chip / Tipo**	**Pin**
Botón	S1	-	PA3
Led	D1	Visible	PA5
Led	D3	Visible	PA0

TABLA 3.3: Funcionalidades plataforma MB951

3.3. Configuración base

Antes de proceder a configurar la red de sensores corriendo sobre la plataforma descrita, es necesario familiarizarse con los procedimientos de configuración y programación del sistema

operativo seleccionado. Seguidamente se describen los pasos fundamentales para poder ejecutar un sencillo ejemplo, pero que sirven como toma de contacto con el entorno *hardware* y *software* y facilitarán el establecimiento de una metodología para los desarrollos y pruebas posteriores.

Como se ha analizado, Contiki OS se basa en el lenguaje de programación C. Dado que las máquinas en la que se va a desarrollar y ejecutar los aplicativos no comparten la misma arquitectura *hardware* (el desarrollo se realiza sobre arquitectura Intel y la ejecución se lleva a cabo en el microprocesador de STM con arquitectura ARM) se requiere hacer uso de un compilador cruzado. El propio entorno de desarrollo del sistema operativo Contiki incorpora las utilidades necesarias para realizar esta tarea. Para hacer más sencillo los procesos de programación, compilación, instalación y prueba, la distribución de Contiki proporciona una máquina virtual (Instant Contiki) con todo lo necesario. Sin embargo, se recomienda utilizar el entorno de desarrollo Contiki de forma nativa para optimizar el rendimiento del propio entorno de desarrollo.

Configurado el entorno de desarrollo, se procede a realizar un sencillo programa que permita validar el funcionamiento de todo el proceso. En el Cuadro de Texto 3.1 se muestra un extracto del código de dicho programa, consistente en la ejecución de dos procesos que monitorizan la temperatura (`mideTemperatura`) y gestionan la iluminación de un led (`parpadear`).

```
1 PROCESS(mideTemperatura, "PROCESO MIDE TEMPERATURA"); // Definicion de procesos
2 PROCESS(parpadear, "PROCESO PARPADEAR LEDS");
3 AUTOSTART_PROCESSES(&mideTemperatura, &parpadear); // Lanzamiento de procesos
4 PROCESS_THREAD(parpadear, ev, data) {   // Define el cuerpo de un proceso
5   static struct etimer timer; // Variable local del proceso
6   PROCESS_BEGIN();  // Siempre al principio en la definicion de un proceso
7     etimer_set(&timer, 5 * CLOCK_CONF_SECOND); // Timer de 5 segundos
8     printf("Comenzando el proceso Parpadear LEDS\n");
9     while(1) {
10      PROCESS_WAIT_EVENT();
11      if(ev == PROCESS_EVENT_TIMER) { // Espera a que suceda un evento por timer
12        leds_blink();
13        printf("Los leds acaban de parpadear...\n");
14        etimer_reset(&timer);
15      }
16    }
17  PROCESS_END();  // Siempre al final en la definicion de un proceso
18 }
19 PROCESS_THREAD(mideTemperatura, ev, data) {
20   ...
21 }
```

Cuadro de Texto 3.1: Primer ejemplo en Contiki

La definición de los procesos se realiza mediante el procedimiento `PROCESS()` y el lanzamiento con `AUTOSTART_PROCESSES()`. El cuerpo de cada hilo se define mediante la macro `PROCESS_THREAD(nombre, ev, data)` y debe comenzar con un `PROCESS_BEGIN()` y terminar

con un `PROCESS_END()`. Ambos hilos están a la espera de un evento, en concreto la expiración de un temporizador. Dicho evento despierta los hilos: por un lado `parpadear` ejecuta la función `leds_blink()` que se encarga de encender y apagar los leds de la placa, mientras que `mideTemperatura` transmite a través del puerto serie el valor recogido por el sensor de temperatura ambiente.

Definido el comportamiento de la aplicación y generado el código, resta compilar la aplicación para la plataforma *hardware* y la arquitectura específica que se emplee, en este caso STM32W. Para la plataforma MB851 dicho proceso se realizada empleando el comando del Cuadro de Texto 3.2.

```
$ make TARGET=mbxxx STM32W_CPUREV=CC
```

Cuadro de Texto 3.2: Compilación de la aplicación para la plataforma MB851

En el caso de emplear la plataforma MB951 las opciones varían, siendo el comando a ejecutar el que se muestra en el Cuadro de Texto 3.3.

```
$ make TARGET=mbxxx STM32W_CPUREV=xB
```

Cuadro de Texto 3.3: Compilación de la aplicación para la plataforma MB951

La manera de operar de Contiki sobre cualquier plataforma, implica reprogramar completamente la plataforma *hardware*. Para ello es necesario cargar o (*flashear*) la imagen completa del sistema con la aplicación que se quiera ejecutar. Por tanto los siguientes pasos incluyen la generación del fichero binario y el proceso de carga. En el Cuadro de Texto 3.4 se muestra el comando utilizado para el formateo del fichero binario antes de ser cargado en la placa y en el Cuadro de Texto 3.5 para llevar a cabo el proceso de carga en la plataforma *hardware*.

```
$ arm-none-eabi-objcopy -O binary personal1.mbxxx personal1.bin
```

Cuadro de Texto 3.4: Generación del fichero binario de la aplicación

```
$ sudo ../../tools/stm32w/stm32w_flasher/py_files/stm32w_flasher.py -f -r personal1.
  bin -p /dev/ttyACM0;
INFO: STM32W flasher utility version 2.0.1b2-20140506 for Linux
INFO: Programming user flash
INFO: Erasing pages from 0 to 28...done
INFO: Programming 29012/29012
INFO: Done
INFO: Resetting device
INFO: Done
```

Cuadro de Texto 3.5: Carga del fichero binario

Una vez se ha reprogramado satisfactoriamente la placa, la aplicación se ejecuta automáticamente tras alimentar y reiniciar el sistema. Para comprobar el correcto funcionamiento del

aplicativo se establece una conexión a través del puerto serie con la placa que habilita la transferencia de datos de *debug* con la aplicación residente, tal y como se muestra en el Cuadro de Texto 3.6.

```
$ sudo ../../tools/stm32w/serialdump-linux -b115200 -d10000 /dev/ttyACM0
Starting Contiki 2.7 on MB851 D
************************************
Board name = MB851 D
************************************
Number of leds on the board: 2
Number of buttons on the board: 1
Led D1 (port:A, bit:5)
Led D3 (port:A, bit:0)
Button S1 (port:B, bit:3)
MEMS = LIS302
Temp sensor = STLM20, port:A, pin:4, div:2, adcFix:Yes
EEProm:Yes
PC i/f:STM32F
Power Amplifier:No

nullmac sicslowmac, channel check rate 1000 Hz
802.15.4 PAN ID 0x1234, EUI-64:00:80:e1:03:00:01:34:10, radio channel 16
Comenzando el proceso mideTemperatura
Comenzando el proceso Parpadear LEDS
Los leds acaban de parpadear...
Temp: 31.9 grados cent.
Los leds acaban de parpadear...
Los leds acaban de parpadear...
Temp: 32.1 grados cent.
       ...
```

Cuadro de Texto 3.6: Lectura del puerto serie de la placa

Toda vez que se dispone de un entorno operativo de instalación, configuración y testeo de aplicaciones en Contiki, se procede seguidamente a trabajar con las interfaces de comunicación inalámbricos y generar una red entre los nodos disponibles. A continuación se describe la evolución en la configuración y despliegue de la red de sensores.

3.4. Comunicación básica entre nodos

El proceso de establecimiento de comunicación entre nodos requiere configurar los parámetros de red de cada uno de ellos para que compartan mismo identificador de red, canal y protocolo de direccionamiento y enrutamiento. De forma genérica ofrece la posibilidad de fijar estos parámetros a través de un fichero de configuración `contiki-conf.h`, que será común para todos los nodos de la red. Algunos de los parámetros fijados se muestran a continuación:

- Canal de RF: 16

- PAN ID: 0x1234
- Mecanismo CSMA/CA: Desactivado
- Protocolo IPv6: Activado
- Tipo de compresión 6LoWPAN: SICSLOWPAN_COMPRESSION_HC06
- Protocolo de enrutamiento RPL: Activado
- Protocolos UDP/TCP: Solo activado para las aplicaciones que lo requieran

Configurados los nodos se procede a realizar un aplicativo que permita el intercambio de información entre ellos. En la Figura 3.5 se muestra el escenario considerado, que consiste en una red LoWPAN formada por dos nodos que se comunicarán en modo *ad-hoc* empleando 6LoWPAN. Uno de los nodos operará como servidor UDP a la espera de peticiones generadas por el otro nodo, que actuará de cliente. La aplicación se desarrolla empleando el *Application Programming Interface* (API) de comunicaciones que proporciona Contiki y que permite entre otros la creación de *sockets* UDP y el intercambio de datagramas.

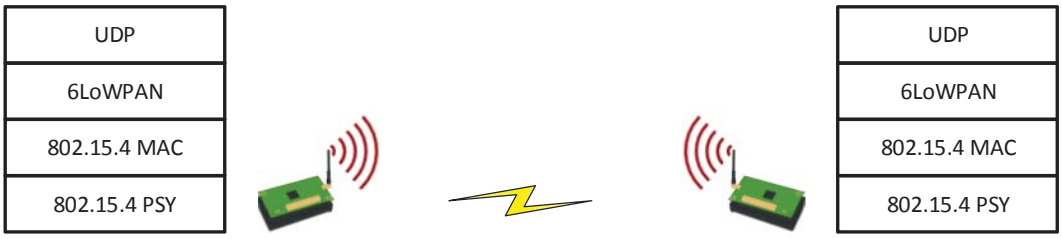


Figura 3.5: Escenario de comunicación entre nodos

El funcionamiento del aplicativo se describe a continuación. Tanto el servidor como el cliente UDP realizan la asignación de direcciones mediante el mecanismo de autoconfiguración a partir del prefijo de red y el identificador MAC. A continuación, el servidor UDP se pone a la escucha en el puerto 3000 esperando peticiones de un cliente. El cliente tiene un temporizador cada 15 segundos que dispara el envío de un datagrama UDP con un mensaje tal que así: *"Hola (número) desde el cliente!"*. Cuando el servidor lo recibe, responde al cliente con un datagrama UDP de este tipo: *"Hola (número) desde el servidor!"*. Programados los nodos con la aplicacion, en los Cuadros de Texto 3.7 y 3.8 se observan los mensajes de depuración.

```
1 $ sudo ../../tools/stm32w/serialdump-lux -b115200 -d10000 /dev/ttyACM2
2             ...
3 Servidor ha recibido: 'Hola 1 desde el cliente!'
4 Respuesta del servidor: Hola 1 desde el servidor!
5 Servidor ha recibido: 'Hola 2 desde el cliente!'
6             ...
```

Cuadro de Texto 3.7: Puerto serie del servidor UDP

```
$ sudo ../../tools/stm32w/serialdump-lux -b115200 -d10000 /dev/ttyACM1
            ...
Peticion del cliente: Hola 1 desde el cliente!
Cliente ha recibido: 'Hola 1 desde el servidor!'
Peticion del cliente: Hola 2 desde el cliente!
            ...
```

Cuadro de Texto 3.8: Puerto serie del cliente UDP

Aunque a nivel de aplicación la operativa es la esperada, el interés en este proyecto es comprender cómo se transmiten los datos y si el formato se ajusta a lo esperado. Para ello se hace necesario emplear un analizador de protocolos que opere sobre redes IEEE 802.15.4. Entre el conjunto de aplicaciones disponibles en la suite de Contiki se encuentra una que cargada en el sistema permite la captura de paquetes 6LoWPAN. La aplicación establece un modo monitor para la interfaz inalámbrica y se encarga de escuchar los mensajes que circulan por el canal inalámbrico compartido, volcando las tramas al puerto serie conectado al PC, permitiendo visualizar el contenido en el analizador de red *Wireshark*.

En la Figura 3.6 se muestra una captura de paquete 6LoWPAN. Se observa que, tal como se había establecido en la configuración del aplicativo, para la compresión de la cabecera IPv6 se utiliza IPHC con todos los campos comprimidos, incluido el direccionamiento, pues al tratarse de una comunicación dentro de la red LoWPAN, pueden utilizarse las direcciones de capa de enlace para generar las direcciones *link-local* de red (*stateless*). Además, se utiliza compresión de cabecera UDP NHC que permite comprimir el campo longitud.

3.5. Interconexión de la LoWPAN con red externa IPv6

Logrado el primer objetivo de establecer una red de interconexión entre nodos sensores, se desea ahora extender el rango de acción, pasando de una red aislada basada en una arquitectura *ad-hoc* a una red LoWPAN con capacidades de interacción con una red externa IPv6.

Para ello es necesario que uno de los nodos haga de *router* de borde y se encargue de implementar la capa de adaptación, como se explicó en 2.5.3.1. En la Figura 3.7 se muestra el escenario descrito.

Como *router* de borde se va a usar el aplicativo proporcionado en Contiki, `rpl-border-router`. La plataforma del *router* de borde tiene dos interfaces, una para la red LoWPAN y otra para conectar con el mundo exterior. La interfaz LoWPAN se implementa usando la plataforma MB951 tal y como se muestra en la Figura 3.8, en la que se ejecuta el programa `rpl-border-router`. Este aplicativo hace uso del protocolo *Serial Line Internet Protocol* (SLIP) [30] con una ligera modificación propia de los desarrolladores de Contiki para adaptarlo a IPv6. De esta manera, los

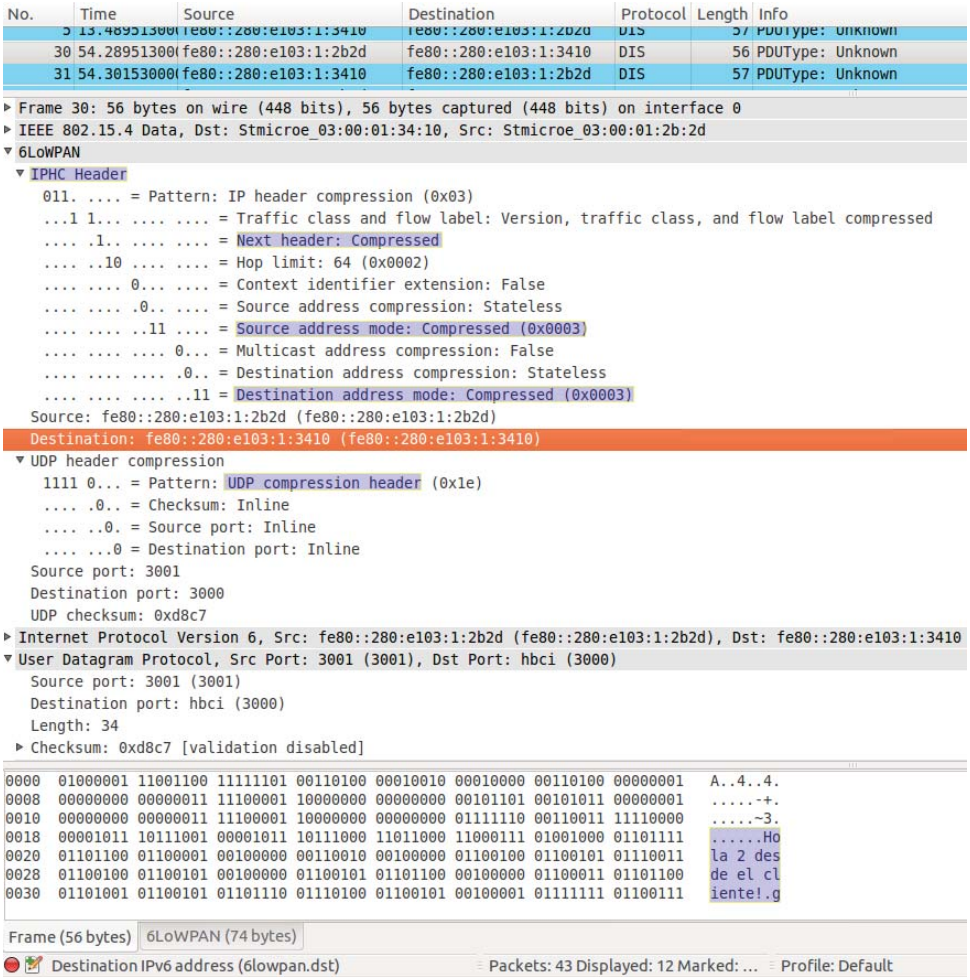


FIGURA 3.6: Captura de comunicación entre nodos

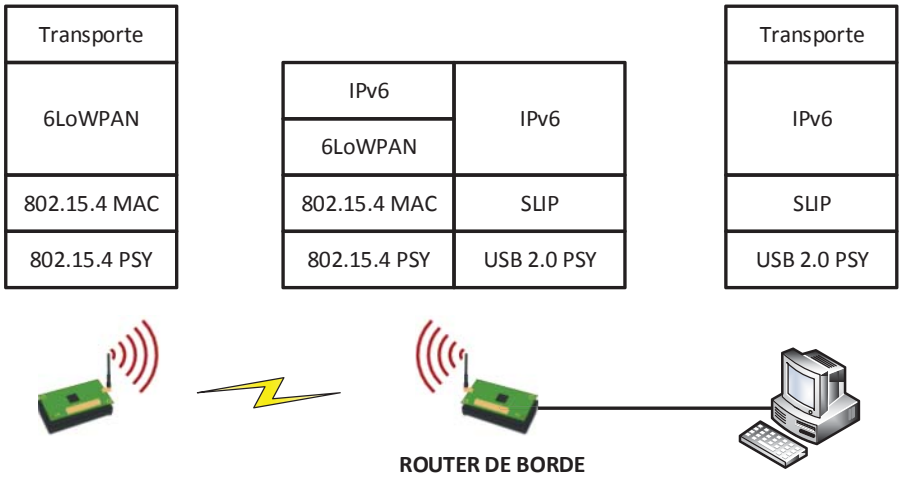


FIGURA 3.7: Escenario Interconexión de la LoWPAN con IPv6

paquetes recibidos por la interfaz LoWPAN se encapsulan en un túnel serie cuyo extremo opuesto es el ordenador. En él se emplea la herramienta `tunslip6` para establecer la conexión con el túnel serie y generar una interfaz virtual IPv6 que habilite la comunicación ethernet estándar. La operativa es similar para la transmisión de paquetes hacia la red LoWPAN.

FIGURA 3.8: Conexión border router al PC

Así pues, una vez instalado el programa en la placa `rpl-border-router` se ejecuta en el PC la aplicación *tunslip6*, indicando como parámetro el dispositivo y la dirección de red que se quiere asignar a la máquina, tal y como se muestra en 3.9.

```
$ sudo ./tunslip6 -s /dev/ttyACM1 aaaa::1/64
********SLIP started on ``/dev/ttyACM1''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC  direccionHW
  00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          Direc. inet:127.0.1.1  P-t-P:127.0.1.1  Masc:255.255.255.255
          Direccion inet6: fe80::1/64 Alcance:Enlace
          Direccion inet6: aaaa::1/64 Alcance:Global
          ACTIVO PUNTO A PUNTO FUNCIONANDO NOARP MULTICAST  MTU:1500  Metrica:1
          Paquetes RX:0 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:0 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:500
          Bytes RX:0 (0.0 B)  TX bytes:0 (0.0 B)

*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::280:e103:1:387f
 fe80::280:e103:1:387f
```

CUADRO DE TEXTO 3.9: Ejecución de tunslip6

Para validar la conectividad entre la parte IPv6 nativa del *router* de borde y la red LoWPAN se utiliza la aplicación `ping6`. Las pruebas realizadas ofrecieron un resultado satisfactorio, si bien la latencia de la transmisión es muy superior (35 milisegundos) respecto a la que se observa para transmisiones en una red de area local inalámbrica IEEE 802.11 o cableada (inferior al milisegundo).

Como en el caso anterior, adicionalmente se procede al análisis de los paquetes 6LoWPAN intercambiados tanto en la interfaz IPv6 (`tun0`) como en el medio inalámbrico IEEE 802.15.4, haciendo uso del capturador de tráfico. En la Figura 3.9 se muestra una petición `ping6` a la entrada del *router* de borde, mientras que en la Figura 3.10 y 3.11 se muestran los fragmentos tras ser redirigida dentro de la red LoWPAN. Ambos fragmentos contienen una cabecera con la información necesaria para el reensamblaje del paquete completo.

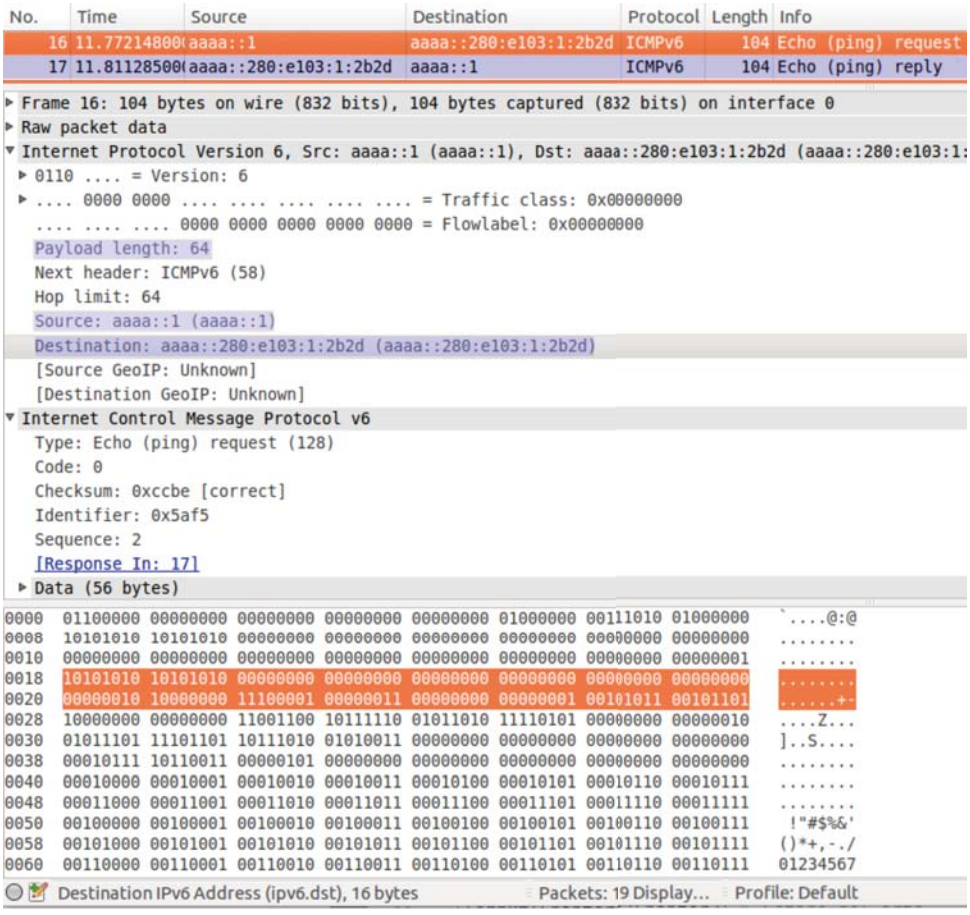


FIGURA 3.9: Petición `ping6` en el interfaz tun0

Además, es interesante insistir que se sigue usando la compresión de cabecera IPHC, pero, a diferencia de la comunicación *ad-hoc*, el campo CIE está habilitado, incorporando un nuevo byte con la información de los identificadores de contexto que son usados para generar los prefijos de red de la direcciones globales IPv6 (*stateful*). En el caso de la fuente es necesario la transmisión

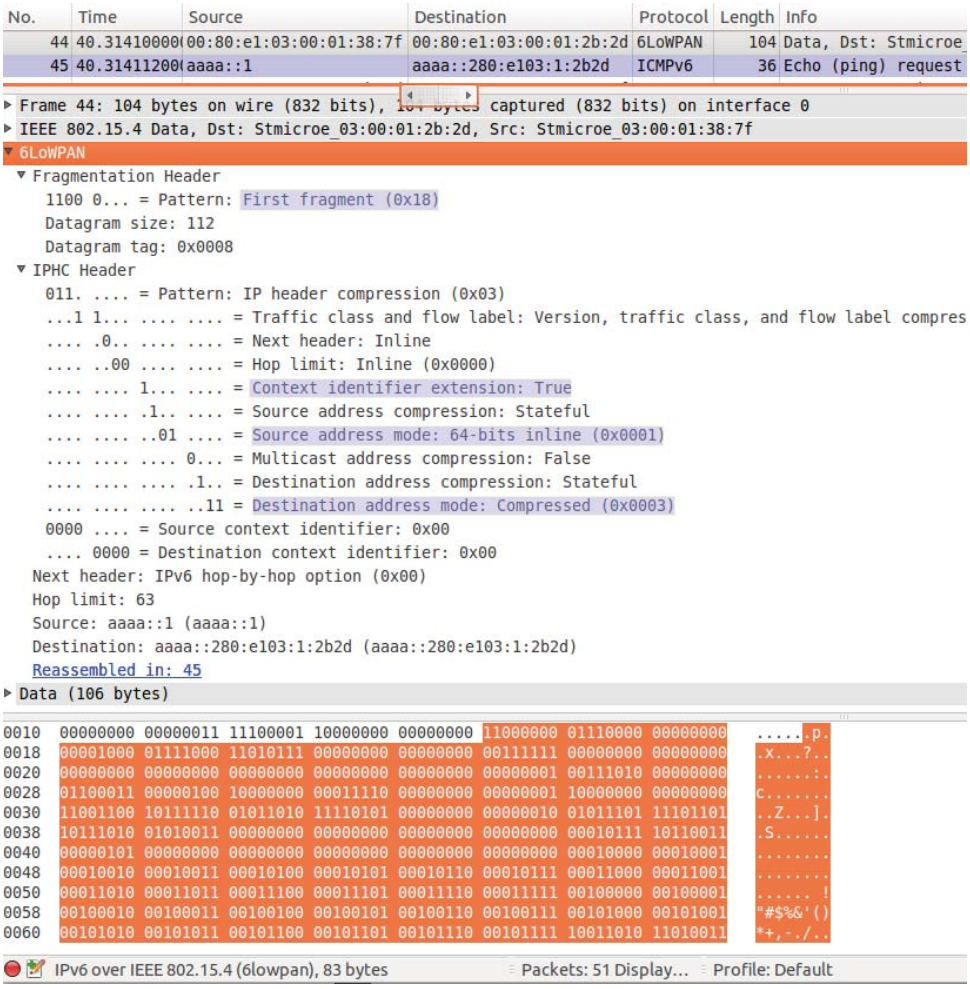


FIGURA 3.10: Petición `ping6` en la red LoWPAN (Fragmento 1)

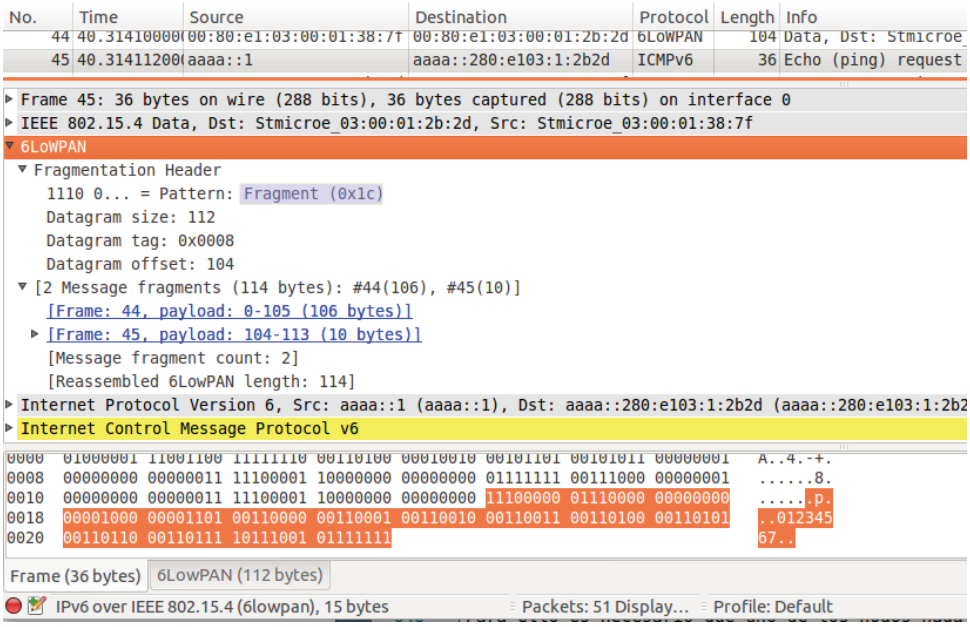


FIGURA 3.11: Petición `ping6` en la red LoWPAN (Fragmento 2)

del *Host ID*; sin embargo, en el destino, se puede suprimir completamente utilizando la dirección MAC para generar el identificador de red.

3.5.1. Comunicación con enrutamiento multisalto

Como parte de las etapas de construcción del entorno final que se usará en las siguientes fases de este proyecto, seguidamente se extienden las funcionalidades de la red para soportar comunicaciones multisalto, tanto dentro de la propia LoWPAN como en la red IPv6. El escenario se muestra en la Figura 3.12. Para llevar a cabo el enrutamiento multisalto en la red LoWPAN, la pila de protocolos implementada en Contiki utiliza el protocolo de enrutamiento RPL [29]. Para habilitar esta funcionalidad debe compilarse las aplicaciones activando la directiva *UIP_CONF_IPV6_RPL*. Además, los nodos que quieran hacer las funciones de *router* también necesitarán indicarlo a través de la directiva *UIP_CONF_IPV6_ROUTER*.

Por otra parte, para conseguir multisalto en la red IPv6 es preciso habilitar la redirección (*forwarding*) de paquetes en los equipos que se desea que realicen el encaminamiento, esto es en PC1. Esto se consigue mediante el comando del Cuadro de Texto 3.10. Además, será necesario configurar las tablas de rutas para que el tráfico se redirija hacia los nodos deseados. En este sentido el equipo remoto (PC2) tendrá una entrada en su tabla de rutas que indicará el camino debe seguir el tráfico generado hacia la red LoWPAN. Para nuestro caso, bastará con añadir en PC2 la entrada a la tabla de rutas, tal y como se muestra en el Cuadro de Texto 3.11.

```
1 $ sudo sysctl -w net.ipv6.conf.all.forwarding = 1
```

Cuadro de Texto 3.10: Configuración del forwarding para IPv6

```
1 $ sudo ip -6 route add aaaa::/64 via bbbb::1 dev eth0
```

Cuadro de Texto 3.11: Configuración de la tabla de rutas de PC2

Con objeto de validar la configuración, se emplea nuevamente la aplicación `ping6`, en este caso trasmitiendo desde PC2 hacia las distintas interfaces de las motas LoWPAN. Además, el aplicativo `rpl-border-router` tiene un proceso añadido que actúa como servidor web y permite mostrar por medio de un navegador cuales son sus nodos vecinos y su tabla de rutas. De esta manera, accediendo a la *Uniform Resource Identifier* (URI) `http://[aaaa::280:e103:1:387f]` se muestra el contenido del Cuadro de Texto 3.12

```
1 Neighbors
2 fe80::280:e103:1:3410
3 Routes
4 aaaa::180:e103:1:3410/128 (via fe80::280:e103:1:3410) 16711245s
5 aaaa::280:e103:1:2b2d/128 (via fe80::280:e103:1:3410) 16711252s
```

Cuadro de Texto 3.12: Tabla de rutas del router de borde

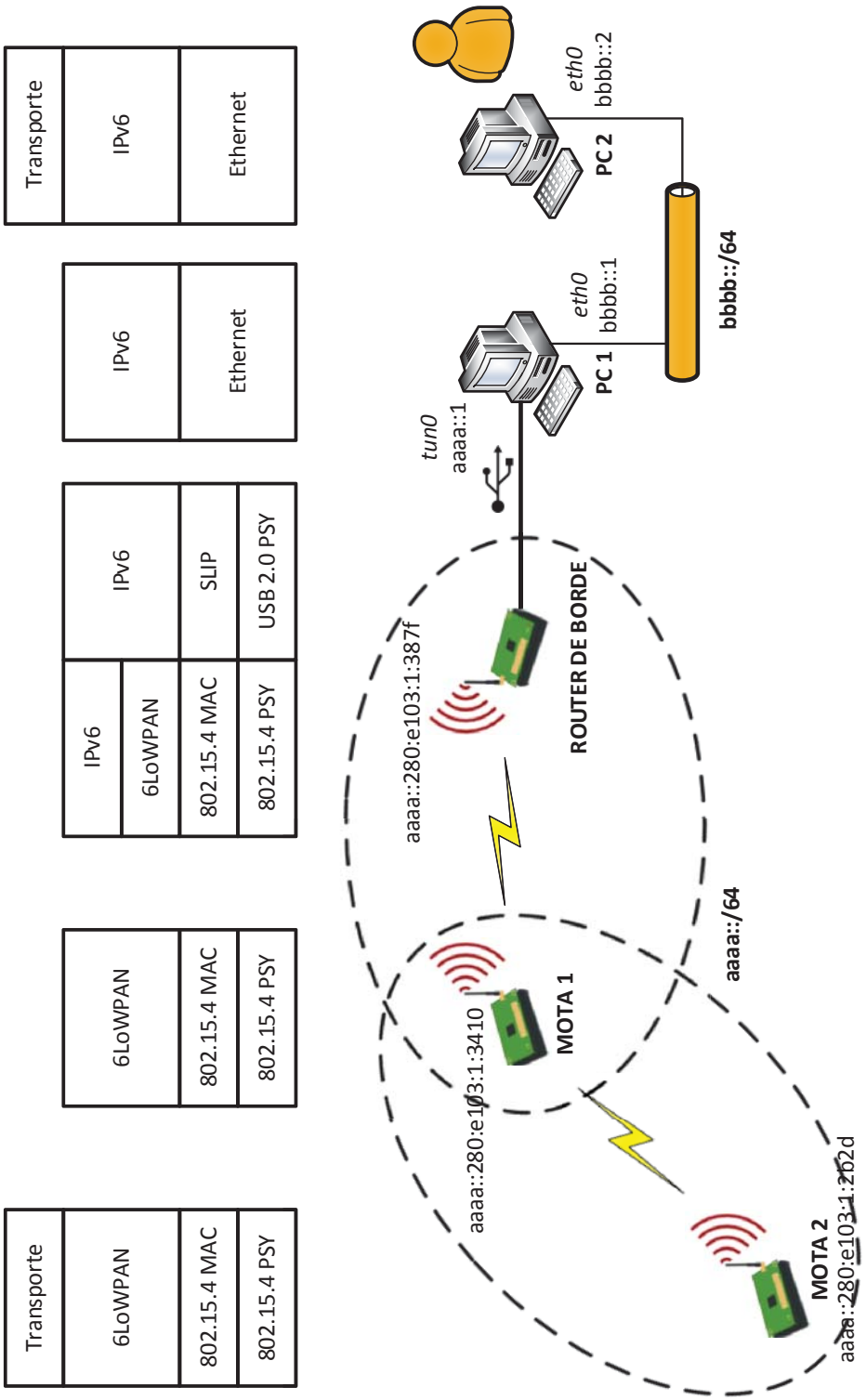


FIGURA 3.12: Escenario Multisalto

En el Cuadro de Texto 3.13 se puede observar el resultado de utilizar la herramienta `ping6` desde PC2 hacia los distintos nodos de la red LoWPAN: *router* de borde (enrutamiento directo), MOTA1 (1 salto) y MOTA2 (2 saltos).

```
$ ping6 aaaa::280:e103:1:387f
PING aaaa::280:e103:1:387f(aaaa::280:e103:1:387f) 56 data bytes
64 bytes from aaaa::280:e103:1:387f: icmp_seq=1 ttl=63 time=20.4 ms
64 bytes from aaaa::280:e103:1:387f: icmp_seq=2 ttl=63 time=23.0 ms
64 bytes from aaaa::280:e103:1:387f: icmp_seq=3 ttl=63 time=25.9 ms
          ...
$ ping6 aaaa::280:e103:1:3410
PING aaaa::280:e103:1:3410(aaaa::280:e103:1:3410) 56 data bytes
64 bytes from aaaa::280:e103:1:3410: icmp_seq=1 ttl=62 time=37.6 ms
64 bytes from aaaa::280:e103:1:3410: icmp_seq=2 ttl=62 time=39.9 ms
64 bytes from aaaa::280:e103:1:3410: icmp_seq=3 ttl=62 time=37.0 ms
          ...
$ ping6 aaaa::280:e103:1:2b2d
PING aaaa::280:e103:1:2b2d(aaaa::280:e103:1:2b2d) 56 data bytes
64 bytes from aaaa::280:e103:1:2b2d: icmp_seq=1 ttl=61 time=58.1 ms
64 bytes from aaaa::280:e103:1:2b2d: icmp_seq=2 ttl=61 time=55.0 ms
64 bytes from aaaa::280:e103:1:2b2d: icmp_seq=3 ttl=61 time=58.0 ms
```

Cuadro de Texto 3.13: Respuesta al aplicativo ping6 desde PC2

Se observa como se consigue el multisalto en la red LoWPAN, además de la redirección de paquetes en la red IPv6. A efectos prácticos, ya existe conectividad IPv6 entre un *host* remoto y cualquier mota de la red LoWPAN.

3.5.2. Comunicación con una red IPv4 - Servicio Tunnel Broker

A pesar de la antigüedad del estándar IPv6, aún son pocos, si no ninguno, los *Internet Service Provider* (ISP) a nivel nacional que ofrecen este servicio a todos sus clientes, al menos, en igual medida que lo hacen con IPv4. Es por esto, que en muchos casos se ofrecen soluciones basadas en *tunnelling*, que a efectos prácticos, permiten acceder y ofrecer servicios IPv6 desde una red IPv4.

Para permitir el acceso de la red generada hasta el momento a la red Internet es necesario emplear alguna de estas opciones. De esta manera, se pretende que el cliente remoto que interactúe con la red LoWPAN pueda ser cualquier ordenador de Internet con la pila de protocolos IPv6 activada. En este caso, se ha optado por utilizar el servicio gratuito IPv6 Tunnel Broker [31]. En la Figura 3.13 se muestra este escenario.

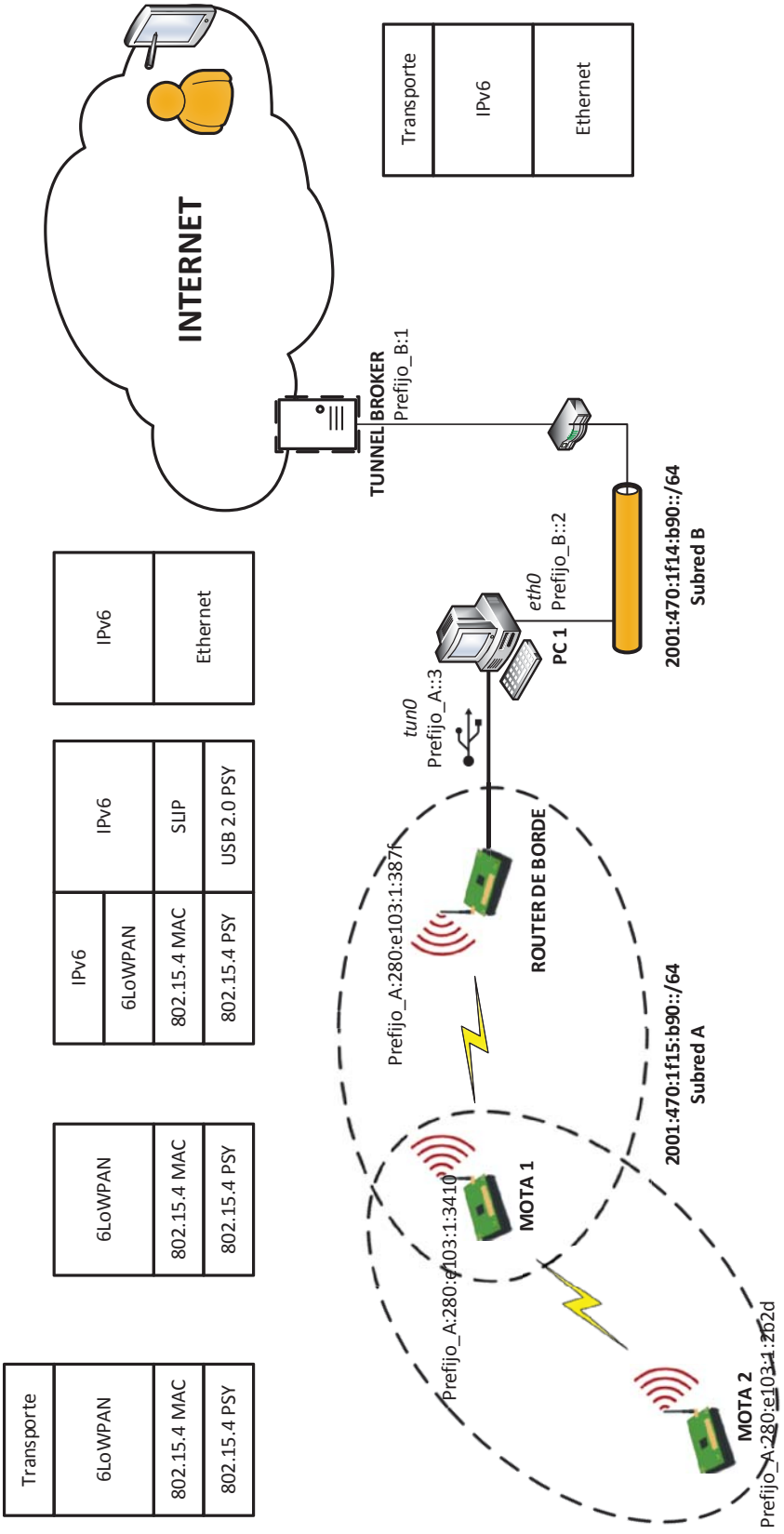


FIGURA 3.13: Escenario Tunnel Broker

La configuración del túnel se realiza mediante una aplicación provista por el proveedor con el que solo añadiendo la dirección IPv4 del cliente (PC1) al que se quiera ofrecer servicio IPv6, el aplicativo se encarga de gestionar la solicitud y asignar una dirección IPv6 al cliente así como una nueva subred IPv6 del tipo /64, tal y como se observa en la Figura 3.14. En este caso la subred IPv6 asignada ha sido la 2001:470:1f15:b90::/64.

Adicionalmente, es necesario habilitar una nueva interfaz de red en el cliente del servicio de tunnel (PC1). Esta interfaz creará un túnel IPv6-sobre-IPv4. En el Cuadro de Texto 3.14 se incluye la configuración para el sistema sistema operativo empleado, que consiste en la modificación del fichero /etc/network/interfaces. Los parámetros de establecimiento del túnel incluyen la dirección IPv4 de los extremos y las direcciones IPv6 (local y pasarela de acceso predeterminada) asociadas a la nueva interfaz *tunnel* que se generará en el equipo local.

```
auto he-ipv6
iface he-ipv6 inet6 v4tunnel
        address 2001:470:1f14:b90::2
        netmask 64
        endpoint 216.66.84.46
        local 84.127.153.191
        ttl 255
        gateway 2001:470:1f14:b90::1
```

Cuadro de Texto 3.14: Configuración del interfaz de red para habilitar el servicio de tunnel

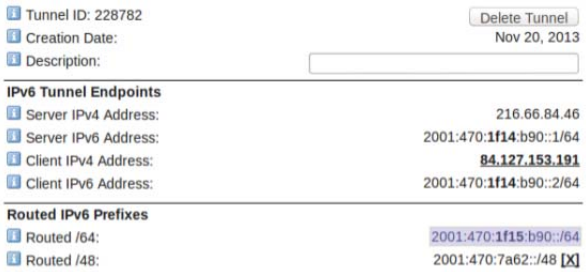


Figura 3.14: Detalles del tunel

Una vez habilitado el túnel, en el Cuadro de Texto 3.15 se puede observar como la nueva interfaz se ajusta a los parametros fijados.

```
$ ifconfig he-ipv6
he-ipv6   Link encap:IPv6-en-IPv4
          Direccion inet6: 2001:470:1f14:b90::2/64 Alcance:Global
          Direccion inet6: fe80::547f:99bf/128 Alcance:Enlace
          ACTIVO PUNTO A PUNTO FUNCIONANDO NOARP  MTU:1480  Metrica:1
          Paquetes RX:5 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:5 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:0
          Bytes RX:520 (520.0 B)  TX bytes:520 (520.0 B)
```

Cuadro de Texto 3.15: Comprobación del interfaz de red

Por otra parte, se ha de tener en cuenta que a la hora de ejecutar la aplicación *tunslip6*, se deberá indicar como parámetro una dirección de red que tenga como prefijo la que el túnel ofrece, tal y como se muestra en 3.16.

```
$ sudo ./tunslip6 -s /dev/ttyACM0 2001:470:1f15:b90::3/64
```

Cuadro de Texto 3.16: Ejecución de tunslip6 usando el servicio de tunnel

En el Cuadro de Texto 3.17 se muestran las tablas de rutas resultantes tras la configuración del servicio. Por una parte, la tabla de rutas correspondiente al protocolo IPv4 tiene como salida la red Internet por medio de la interfaz `eth0`, encargándose de como se lleva a cabo el servicio de túnel. Por otra parte, la tabla de rutas IPv6 encaminará todo el tráfico IPv6 a través de la interfaz *he-ipv6*, salvo aquellos paquetes que pertenezcan a la red LoWPAN, que en ese caso, lo hará a través de la interfaz `tun0`.

```
$ route -4
Tabla de rutas IP del nucleo
Destino         Pasarela        Genmask         Indic Metric  Interfaz
default         84.127.152.1.dy 0.0.0.0         UG    0       eth0
84.127.152.0    *               255.255.240.0   U     1       eth0
link-local      *               255.255.0.0     U     1000    eth0

$ route -6
Tabla de ruteado IPV6 del nucleo
Destination                Next Hop               Flag  Met   If
2001:470:1f14:b90::1/128   ::                     U     1024  he-ipv6
2001:470:1f14:b90::/64     ::                     Un    256   he-ipv6
2001:470:1f15:b90::/64     ::                     U     256   tun0
::/0                       2001:470:1f14:b90::1   UG    1024  he-ipv6
```

Cuadro de Texto 3.17: Tablas de rutas usando el servicio de tunnel

Para validar el correcto funcionamiento de la configuración realizada se lleva a cabo el acceso al servicio proporcionado por `http://test-ipv6.com` que se encarga de determinar las posibilidades de acceso a una red IPv6 de un equipo. En la Figura 3.15 se observa como detecta tanto la dirección IPv6 como IPv4 del equipo e informa de la existencia de un túnel activo que permite la comunicación a través de ambas redes.

Para comprobar la conectividad IPv6 desde cualquier *host* remoto de Internet y las motas de la red LoWPAN, se ha utilizado un dispositivo móvil conectado vía 3G. Como dicha conexión solo ofrece servicio IPv4 ha sido necesario el uso de un *proxy* IPv4 a IPv6 habilitado por el servicio ofrecido por `http://www.ipv6proxy.net/` que permite realizar una petición web IPv6 a la dirección de la mota. Dado que este *proxy* sólo permite el acceso a servicios web, en el nodo final se ha desplegado un servicio web, a través del cual evaluar la correcta configuración y el acceso. El resultado se observa en la Figura 3.16.

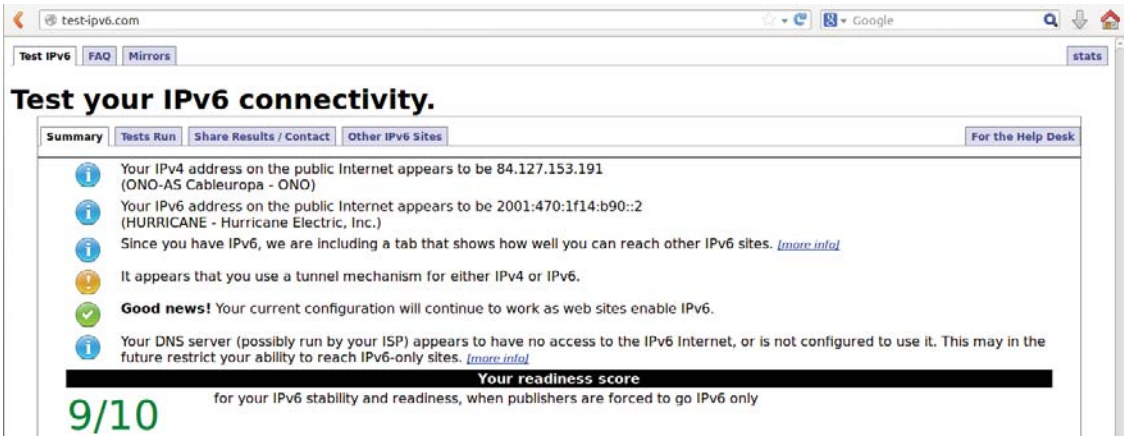


FIGURA 3.15: Testeo del tunneling

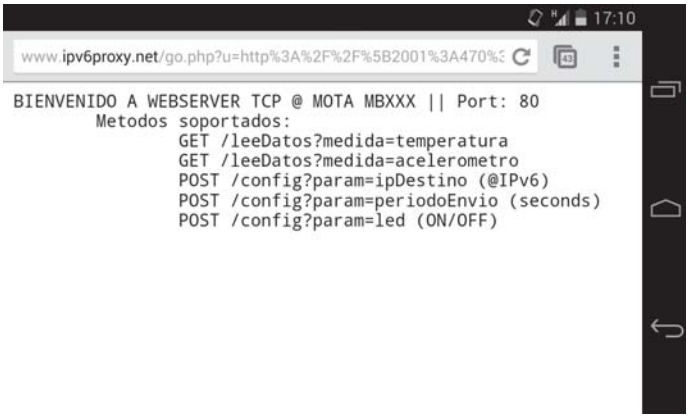


FIGURA 3.16: Petición HTTP desde terminal móvil

Como resultado de los trabajos descritos en este capítulo se ha conseguido proporcionar conectividad IPv6 extremo a extremo entre un nodo conectado a la red Internet y la red LoWPAN, en la que se han dispuesto una serie de servicios de monitorización cuyo detalle se describe en el siguiente capítulo.

Capítulo 4

Servicios de monitorización

Tras realizar la configuración y el despliegue de una red 6LoWPAN con capacidades de interconexión con el exterior, en este capítulo se afronta el desarrollo de los servicios que se ejecutarán sobre ella. En este sentido, si en el capítulo anterior se hizo hincapié en las capas de enlace y red, éste se centra en el nivel de aplicación. A continuación se presenta primeramente una de las tecnologías más usadas para habilitar el acceso y manipulación de servicios basados en recursos, *Representational State Transfer* (REST) [32], cuya operativa y rendimiento se evaluará operando sobre protocolos como *Hypertext Transfer Protocol* (HTTP) o CoAP.

4.1. Introducción a REST

Definido en el año 2000 por Roy Fielding en su tesis doctoral, la arquitectura REST es hoy en día, el modelo arquitectónico que explica como debe funcionar lo que todo el mundo conoce como web. En esencia, REST es un estilo que intenta representar cómo se comporta una aplicación hipermedia distribuida. Para ello hace uso del concepto de recurso definido como cualquier objeto que tenga identidad, es decir, que pueda ser identificado de alguna forma, como puede ser URI [33].

Basado en una arquitectura cliente-servidor, la comunicación entre ambos debe ser auto-contenida (*stateless*), esto es, el servidor no debe guardar ningún tipo de sesión con el cliente. Además, la representación de recursos debe incluir enlaces a otros relacionados (*Hypermedia as the Engine of Application State* (HATEOAS)).

Los recursos en REST han de seguir un conjunto de operaciones bien definidas, que se equiparan a las operaciones *Create, Read, Update and Delete* (CRUD) [34] en bases de datos, cuyas primitivas a emplear dependerán del protocolo sobre el que se gestionen los recursos, ya sea HTTP o CoAP. La funcionalidad de las operaciones CRUD en REST es la siguiente:

- `CREATE`: Crea un nuevo recurso
- `READ`: Lee una representación del estado de un recurso
- `UPDATE`: Actualiza el estado de un recurso existente en el servidor
- `DELETE`: Elimina un recurso del servidor

En cualquier caso, no es objetivo del proyecto en sí un estudio exhaustivo de dicha arquitectura, más bien la implementación práctica de aplicaciones que usen este mecanismo. En el siguiente apartado se resumen las características generales que se han tenido en cuenta para el desarrollo de las aplicaciones basadas en servicio REST.

4.2. Diseño de aplicacion RESTful

A la hora de definir una aplicación RESTful, el primer paso es identificar los recursos que va a ofrecer el sistema. En nuestro caso, se pretende exportar la información que es capaz de gestionar cada uno de los nodos sensores (temperatura, acelerómetro, etc.) y configurar el comportamiento de estos de forma remota.

Así inicialmente se definirán dos recursos que por medio de una operacion de tipo `READ` permitirán conocer los datos sensados de temperatura y acelerómetro en cada momento. Por otra parte, existirán otros dos recursos mediante los cuales se podrán actualizar por medio de una operación `UPDATE` el valor de dos variables (`ipDestino` y `periodoEnvio`) que almacenarán la información necesaría para un aplicativo de monitorización automático en el que los nodos actuarán como clientes enviando los datos sensados a un servidor remoto, y cuya descripción se muestra en detalle en la sección 5.2. Por último, se definirá un recurso que permitirá el control de iluminación de un diodo led mediante una operación de tipo `UPDATE`. A partir de esta información se plantean los identificadores de los recursos, evitando nombrarlos como acciones.

Por otro lado, dadas las características de las redes de sensores y considerando que los servicios se ejecutarán extremo a extremo, es importante seleccionar un tipo de representación de la información que se adapte y no incluya datos redundantes. En este sentido, se puede emplear mecanismos de representación estandarizados como *eXtensible Markup Language* (XML) o *JavaScript Object Notation* (JSON), pero para simplificar aún más el flujo de datos se ha optado por usar una representación propietaria en texto plano.

En la Tabla 4.1 se muestran los identificadores de recursos soportados y el modelo de datos empleado.

URI-Request	Operación	Representación del contenido
/index	READ	Resume recursos disponibles
/datos?medida=temperatura	READ	t(Decimas de grado)
/datos?medida=acelerometro	READ	ax(num);ay(num);az(num);
/config?param=ipDestino	UPDATE	ip=@IPv6
/config?param=periodoEnvio	UPDATE	pE=num
/config?param=led	UPDATE	ls=on/off

Tabla 4.1: Resumen de recursos basados en REST

4.3. Servicio REST sobre HTTP

Basado en la arquitectura cliente-servidor bajo el protocolo de transporte TCP, HTTP se ha convertido en el protocolo de transferencia de hipertexto por excelencia para ofrecer servicios web.

El funcionamiento es sencillo, tal como se refleja en la Figura 4.1: un cliente HTTP envía un mensaje de petición a un servidor, el cual, después de realizar algún tipo de procesado, devuelve un mensaje de respuesta que contiene un código de estado y el resultado de la petición. En el siguiente apartado se va a estudiar de forma resumida los distintos tipos de peticiones y respuestas que existen en HTTP.

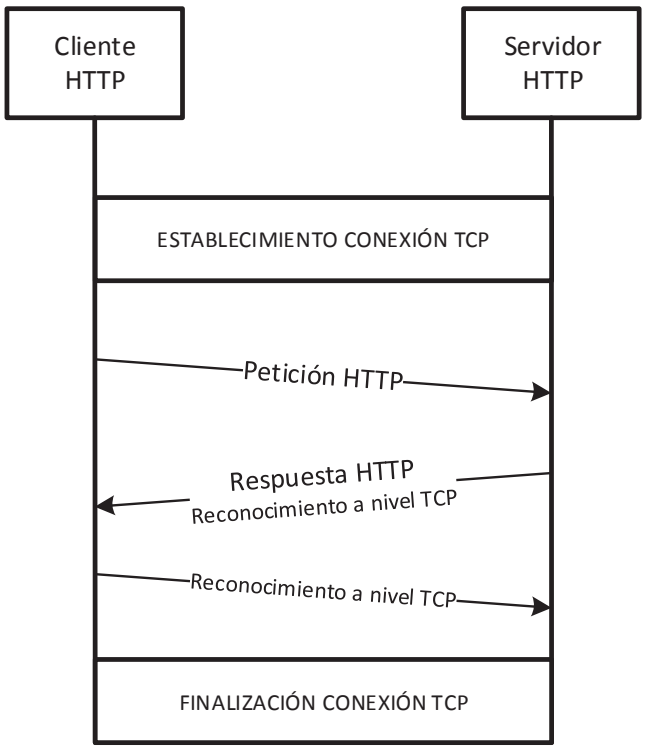


Figura 4.1: Diagrama de comunicación HTTP

4.3.1. Formato

Una petición HTTP está formada por una primera línea de petición, cero o más líneas de cabecera, y opcionalmente, el cuerpo del mensaje. La línea de petición consta de tres partes separadas por espacios, en el que se indica el nombre del método a utilizar, la ruta local del recurso solicitado y la versión de HTTP.

El método define la semántica de la petición a realizar. Los más comunes son:

- `GET`: Obtiene la representación de un recurso.
- `PUT`: Crea un recurso en la dirección especificada en la URI. Si existe, lo reemplaza.
- `POST`: Envía datos a una URI para que el recurso lo procese. Estos datos pueden servir para crear un nuevo recurso.
- `DELETE`: Desvincula un recurso de su URI.

Asimismo, se definen distintas líneas de cabecera que pueden ser usadas tras la línea de petición. Estas cabeceras añaden meta-información adicional acerca del cuerpo del mensaje o del propio cliente en sí. Las más comunes son:

- **Accept:** Informa al servidor del tipo de representación de contenido que soporta.
- **Accept-Language:** Informa al servidor del lenguaje del contenido.
- **Authorization:** Credenciales para que el servidor autentique al cliente.
- **Connection:** Tipo de conexión que el cliente desea.
- **Content-Length:** Longitud del cuerpo del mensaje.
- **Content-Type:** Tipo de formato del cuerpo del mensaje.
- **Connection:** Tipo de conexión que el cliente desea.
- **Host:** Identifica al host objetivo de la petición.
- **User-Agent:** Informa al servidor sobre el agente de usuario que hace la petición.

Los mensajes de respuesta HTTP se componen de un código de estado, algunas cabeceras y el cuerpo del mensaje. El código de estado está compuesto por 3 dígitos y representa el resultado del intento realizado por el servidor para entender y satisfacer la petición del cliente. El primer dígito del código de estado define el tipo de respuesta. En la Tabla 4.2 se muestra el significado y los códigos de estado más comunes.

Código	Semántica	Descripción
1xx	Informativo	-
2xx	Éxito	-
200	OK	Respuesta estándar para peticiones correctas
201	Creado	Recurso creado satisfactoriamente
202	Aceptado	Petición aceptada pero aún sin completar
3xx	Redirecciones	-
4xx	Errores de cliente	-
400	Solicitud incorrecta	Sintaxis incorrecta en la solicitud
401	No autorizado	Credenciales de autorización incorrectas
403	Prohibido	Solicitud válida, pero el servidor la deniega
404	No encontrado	El recurso no se encuentra
405	Método no permitido	El recurso existe, pero no soporta dicho método
5xx	Errores de servidor	-

Tabla 4.2: Resumen de códigos de estado HTTP

4.3.2. Implementación

Seguidamente se describe el proceso de desarrollo y despliegue de un servicio REST sobre HTTP ofrecido por los nodos de la red LoWPAN. En la Figura 4.2 se muestra el escenario planteado y la pila de protocolos en cada uno de los nodos, lo que permite comprender la funcionalidad de cada uno de los nodos y la encapsulación que se realiza.

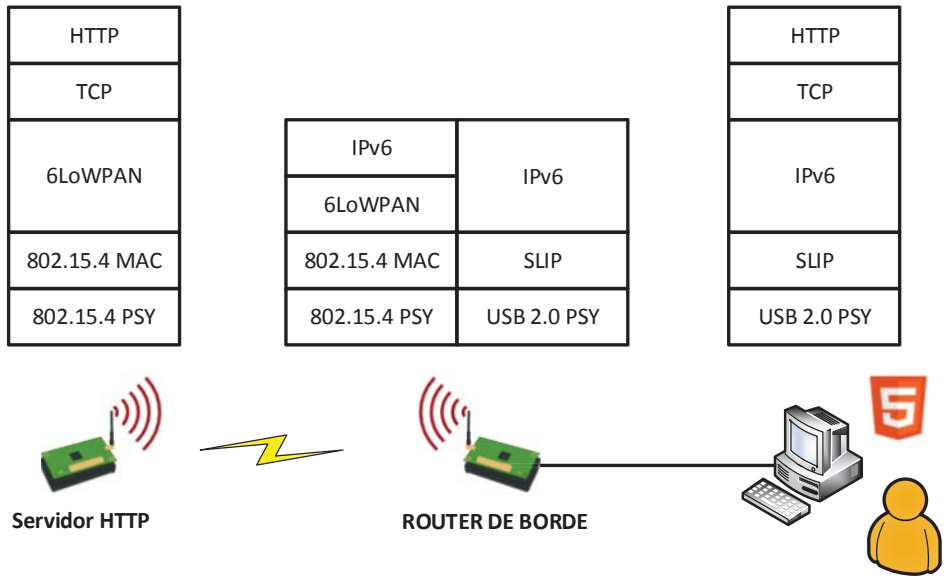


Figura 4.2: Escenario HTTP

El API de comunicaciones de Contiki no incluye soporte estable para el desarrollo de servicios HTTP, lo que ha obligado a realizar una implementación propia del protocolo HTTP. A partir de las funciones del API de comunicaciones para operar sobre *sockets* y empleando *protosockets*, se ha llevado a cabo dicho trabajo. Si bien no se ha implementado toda la operativa del protocolo

HTTP, sí se ha implementado el conjunto de primitivas necesaria para dar respuesta al servicio propuesto. En el desarrollo de la implementación se ha restringido dentro del propio código el acceso únicamente a los recursos que se definen, siendo cualquier otro considerado como erroneo. En este caso, el acceso a nuevos recursos supone el desarrollo y recompilación de la imagen del sistema.

El funcionamiento de la implementación (ver Figura 4.3) es el siguiente:

- El servidor se pone a la escucha en el puerto 80 mediante *tcp_ listen(UIP_ HTONS(80))*.
- Si se establece una conexión TCP con algún cliente, se inicializa un *Protosocket* a partir del cual se mandará la información cliente-servidor y viceversa.
- El servidor lee del *Protosocket* hasta un salto de línea y parsea lo leido comparándolo con las distintas líneas de petición que soporta. En caso de no coincidir con ninguna, el servidor responderá con una respuesta HTTP con código de estado 404 (No encontrado).
- Si existe y el método usado ha sido un `GET`, devolverá en una respuesta HTTP con código 200 (OK) la información del recurso solicitado.
- Si existe y el método usado ha sido un `PUT`, modificará algún parámetro de configuración del servidor, además de devolver una respuesta con código de estado 200 (OK).
- Cuando el servidor envía la respuesta HTTP al cliente, cierra el *Protosocket* de la conexión TCP.

4.3.3. Validación

Para validar el correcto funcionamiento del servicio, se ha desarrollado un cliente web basado en tecnología *HyperText Markup Language version 5* (HTML5) bajo el framework jQuery Mobile [35], que facilita el diseño responsivo. De esta manera, es posible gestionar los recursos definidos anteriormente bajo una aplicación web de una manera sencilla y accesible desde cualquier dispositivo móvil. El aspecto del cliente se puede apreciar en la Figura 4.4.

Para poder llevar a cabo las peticiones *Asynchronous JavaScript And XML* (AJAX) desde JavaScript [36] ha sido necesario habilitar *Cross-origin resource sharing* (CORS) [37] en el nodo que hace de servidor HTTP, permitiendo así la comunicación entre diferentes dominios. Esto se consigue procesando la petición CORS recibida por parte del cliente y respondiendo con una serie de cabeceras que validen el origen y los métodos HTTP.

Con objeto de analizar en mayor profundidad el comportamiento de la red 6LoWPAN se ha estudiado el flujo de mensajes tanto dentro de la sección LoWPAN como en el interfaz `tun0`.

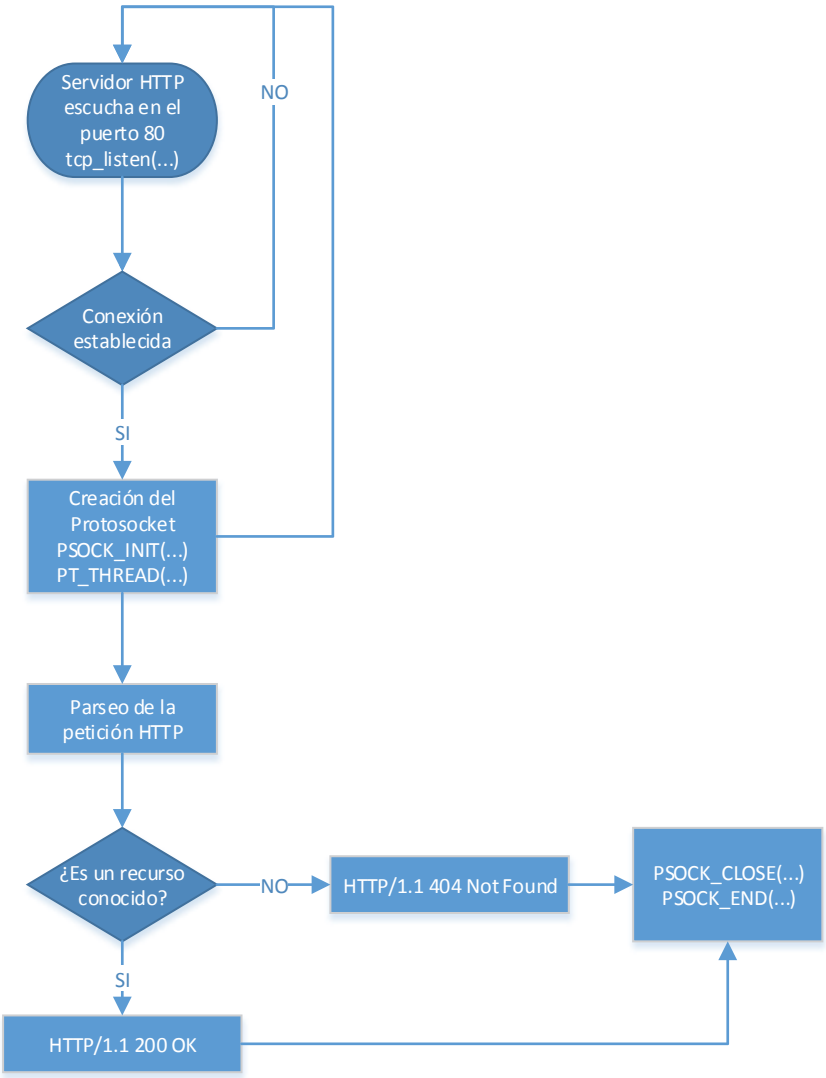


FIGURA 4.3: Diagrama de flujo del servidor HTTP

FIGURA 4.4: Validación de HTTP con tecnología HTML5

En la Figura 4.5 se muestra como es una petición `PUT` de HTTP antes de pasar por la capa de adaptación de 6LoWPAN. Asimismo, en la Figura 4.6 se observa como es este mismo mensaje ya dentro de la LoWPAN, siendo necesario el mecanismo de fragmentación. Este mecanismo hace que el rendimiento dentro de la red, fuera a parte de la propia velocidad inherente al medio, sea peor en comparación con la red tradicional IPv6.

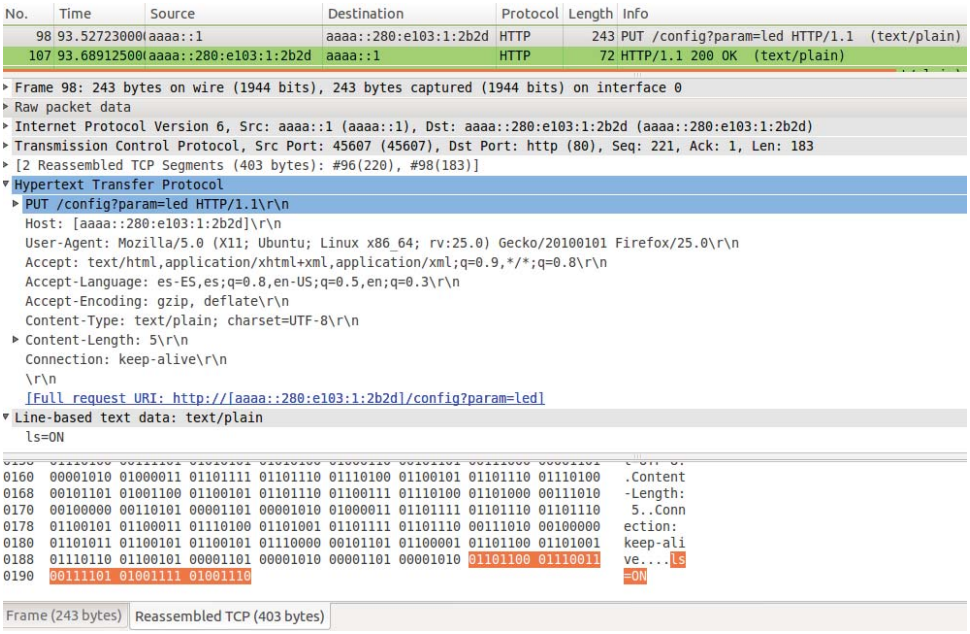


FIGURA 4.5: Petición HTTP en el interfaz tun0

Por tanto, se observa como el servidor HTTP responde de la forma esperada a la petición del cliente. De forma análoga, aunque no se muestre en esta memoria, se ha comprobado el acceso al resto de recursos, siendo el resultado el esperado. Por tanto, se puede decir que los nodos de la red LoWPAN ya tienen servicio web a través de HTTP. Sin embargo, el uso de HTTP sobre este tipo de redes no resulta del todo aconsejable debido al tamaño elevado de la cabecera.

Por otra parte, el uso de TCP sobre los canales radio inalámbricos resulta ineficiente [38], siendo en muchos casos necesaria la búsqueda de soluciones para solventar el problema [39]. Este tipo de redes tiene un alto porcentaje de probabilidad de pérdida, hasta tal forma que en el caso de las redes de sensores, el uso de HTTP-REST sobre TCP/IP resulta desaconsejable. TCP es un protocolo orientado a conexión y, por tanto, necesita mensajes de establecimiento y terminación (*Three-wayhandshake*). Además es un protocolo de transporte fiable, que realiza retransmisiones siempre que se pierde cualquiera de los datagramas.

Por este motivo, el IETF ha definido CoAP, un nuevo protocolo de capa de aplicación para entornos LoWPAN que conserva el modelo REST pero utiliza UDP como capa de transporte, además de formato binario para reducir el tamaño de las cabeceras.

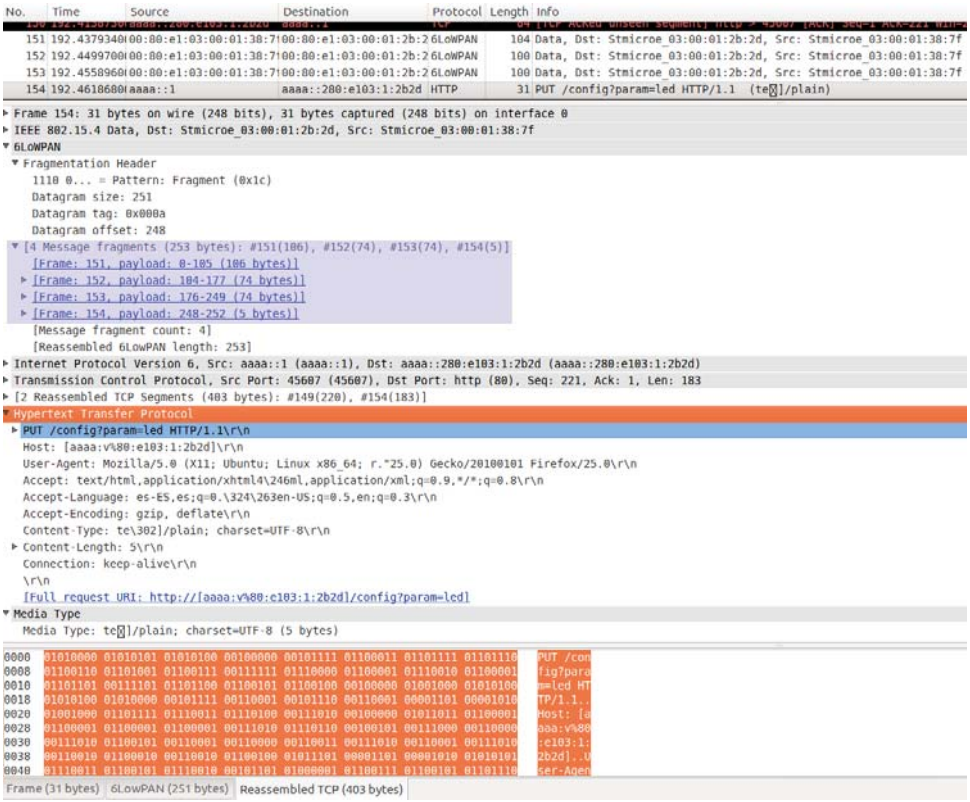


FIGURA 4.6: Petición HTTP en el interfaz LoWPAN

4.4. Servicio REST sobre CoAP

Con un formato de mensajes alineado con HTTP, pero sobre una capa de transporte no orientada a la conexión, como es UDP, surge el protocolo de aplicación CoAP [40], protocolo que hereda directamente de HTTP los métodos CRUD para la gestión de los recursos: `GET`, `POST`, `PUT` y `DELETE`.

A diferencia de HTTP y tratando de solventar las deficiencias de la capa de transporte que emplea, CoAP distingue mensajes confirmados y no confirmados. Los mensajes confirmados requieren un reconocimiento por parte del receptor, mientras que existe la posibilidad de utilizar mensajes no confirmados con el fin de evitar la saturación de la red. En la Figura 4.7 se muestra un sencillo diagrama de una comunicación CoAP.

Además, CoAP codifica los mensajes en formato binario en lugar del formato texto empleado en HTTP, lo que supone una reducción significativa de la longitud de las cabeceras y del propio mensaje, permitiendo un mejor aprovechamiento del ancho de banda disponible. Es por esta razón que CoAP está enfocado a redes operando sobre canales con bajo ancho de banda.

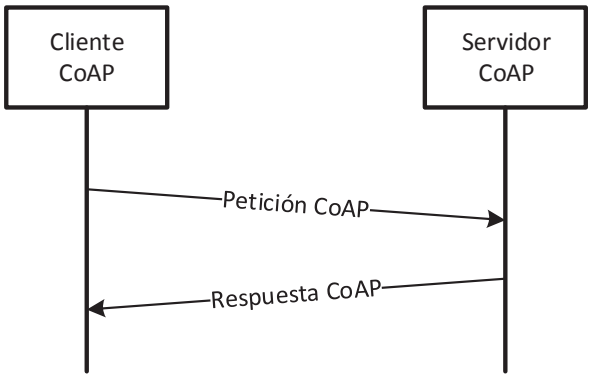


FIGURA 4.7: Diagrama de comunicación CoAP

4.4.1. Formato

En la Figura 4.8 se muestra el formato de la cabecera de un mensaje CoAP. Como se ha comentado anteriormente, esta cabecera se codifica en formato binario con los campos que se detallan a continuación:

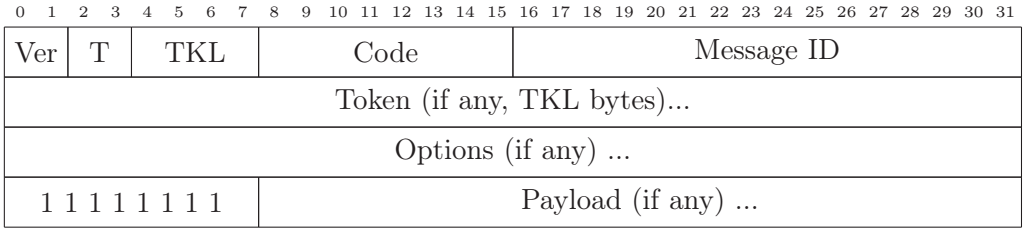


FIGURA 4.8: Formato cabecera CoAP

- **Ver (Version):** Indica la versión de CoAP que se esta usando.
- **T (Type):** Informa del tipo de mensaje.
- **TKL (Token Length):** Indica la longitud del campo opcional Token.
- **Code:** Codificación binaria de los distintos métodos(petición) o códigos de estado(respuesta).
- **Message ID:** Identificador del mensaje usado para la confirmación o detección de duplas.
- **Token:** Identificador que genera el cliente para identificarse al servidor.
- **Options:** Información equivalente a los metadatos de HTTP. La URI también va en este campo.
- **Payload:** Carga útil a nivel de CoAP.

CoAP define 4 tipos de mensajes, el cual debe ser indicado en el campo *Type* de la cabecera. De esta manera, es posible dar funcionalidad de reconocimiento a los mensajes a pesar de funcionar sobre un protocolo no orientado a la conexión como es UDP, siendo la capa CoAP la encargadada de asegurar que los mensajes lleguen al destino. En la Tabla 4.3 se muestra este conjunto de mensajes.

Por otra parte, el campo *Code* mantiene un alineamiento directo de los métodos y códigos de estado que se tienen en HTTP y en CoAP. De esta manera, CoAP utiliza un sistema de codificación binario para identificador a cada uno de ellos.

Mensaje	**Descripción**
CON	Peticion para mensajes confirmados
NON	Petición y respuesta para mensajes no confirmados
ACK	Reconocimiento de un mensaje CON
RST	Respuesta a un mensaje CON aún sin procesar

Tabla 4.3: Tipos de mensaje CoAP

Asimismo, CoAP presenta un mecanismo de descubrimiento de recursos [41] que permite saber cuales son los recursos disponibles en el servidor por medio de una petición que sigue un formato como el que se muestra en la Figura 4.9

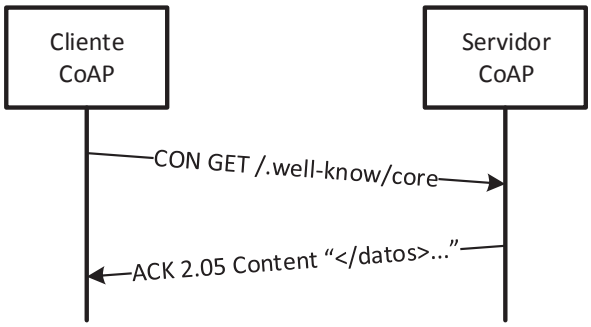


Figura 4.9: Descubrimiento de recursos en CoAP

4.4.2. Implementación

En este caso, se pretende que los nodos de la red LoWPAN ofrezcan servicio web basado en CoAP tal y como se muestra en la Figura 4.10. Como se puede observar, la operativa del escenario planteado no difiere de la descrita para el caso de HTTP; es más, la única diferencia a nivel de protocolos reside en la sustitución de HTTP por CoAP, por lo que sólo será necesario redefinir la codificación del servicio en su parte servidor y cliente.

Contiki dispone de una implementación estable del estándar que recibe el nombre de Erbium [42] desarrollada por Matthias Kovatsch, que actualmente soporta hasta la versión 13 del

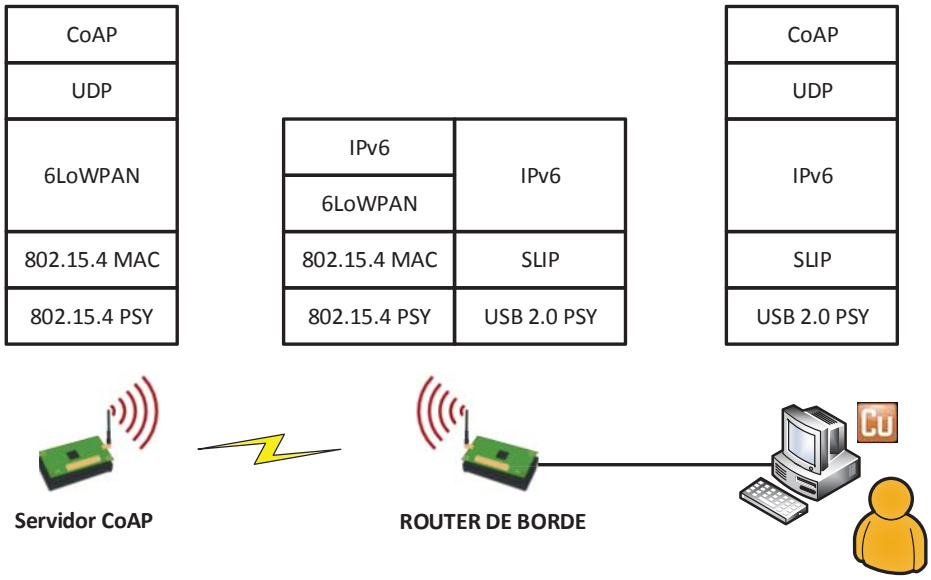


FIGURA 4.10: Escenario CoAP

draft y que será la que se emplee para facilitar el desarrollo del servicio. El API Erbium permite abstraer la capa de transporte a la hora de desarrollar un servicio REST, lo que garantiza que la aplicación cumpla las recomendaciones RESTful.

El proceso seguido para desplegar los servicios REST sobre CoAP ha buscado definir una estructura de recursos similar a la empleada en el servicio sobre HTTP. De esta forma gran parte de las funcionalidades desarrolladas podrán ser reutilizadas. El funcionamiento de la aplicación es el siguiente:

- Se define el puerto CoAP, por defecto 5683.
- Se definen los recursos a través de la macro `RESOURCE`.
- Se realiza la función manejadora para cada recurso definido.
- Se inicializa la máquina REST mediante `rest_init_engine()`
- Se activan los recursos a través de la función `rest_activate_resource()`.

4.4.3. Validación

En este apartado se muestra el resultado de la implementación citada anteriormente. Para testearla es necesario utilizar un cliente CoAP. Para ello, se ha utilizado un plugin de Mozilla Firefox que recibe el nombre de Copper [43] implementado por el mismo desarrollador de Erbium. En la Figura 4.11 se muestra la interfaz de usuario del plugin tras haber hecho descubrimiento de recursos sobre contra el servidor CoAP.

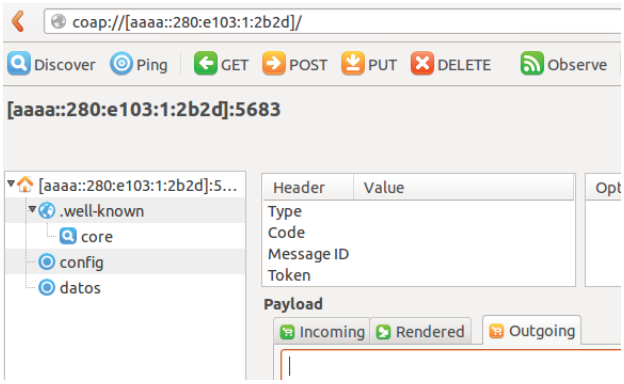


FIGURA 4.11: Aspecto del cliente Cupper

En primer lugar es interesante comprobar el funcionamiento del mecanismo de descubrimiento de recursos disponibles en el servidor. En la Figura 4.12 se observa cómo es la petición en la interfaz 6LoWPAN y en la Figura 4.13 su reconocimiento con la información asociada de los recursos disponibles del servidor. Por otra parte, aunque no se muestre en la captura, es interesante destacar que en este caso el descubrimiento global de todos los recursos del servidor se realiza por medio de tres solicitudes `GET` de tipo `CON` con sus respectivos reconocimientos.

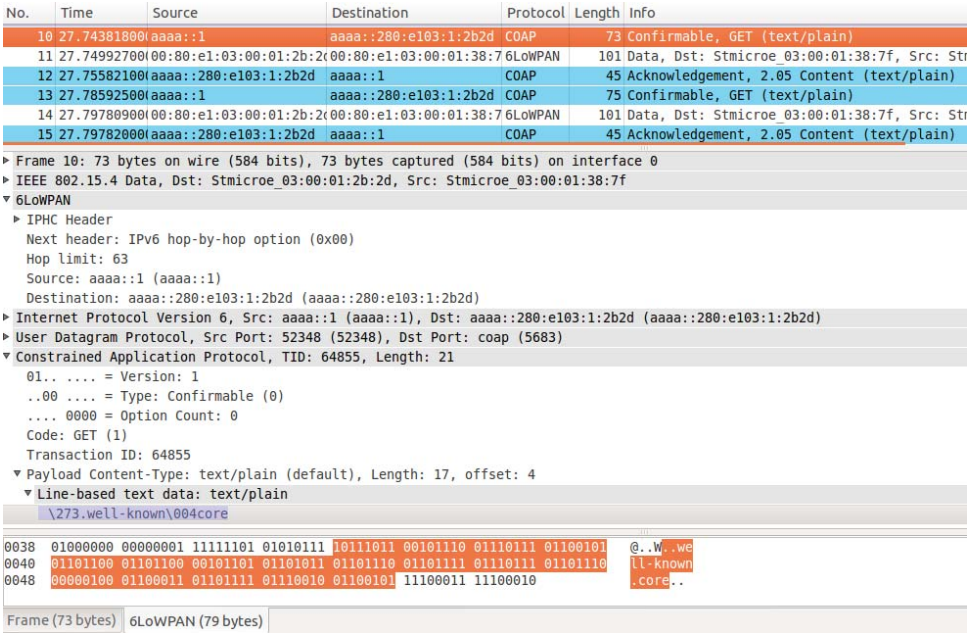


FIGURA 4.12: Peticion CoAP /.well-know/core

De igual manera que se hizo con HTTP, se procede al estudio de cómo se comportan los mensajes CoAP en las interfaces LoWPAN y `tun0`. Para ello, se procede a analizar por medio de Wireshark el mismo recurso que se estudió para el caso de HTTP, esto es, el recurso del led. En la Figura 4.14 y 4.15 se muestra como es la petición para acceder al recurso y la respuesta a ella respectivamente.

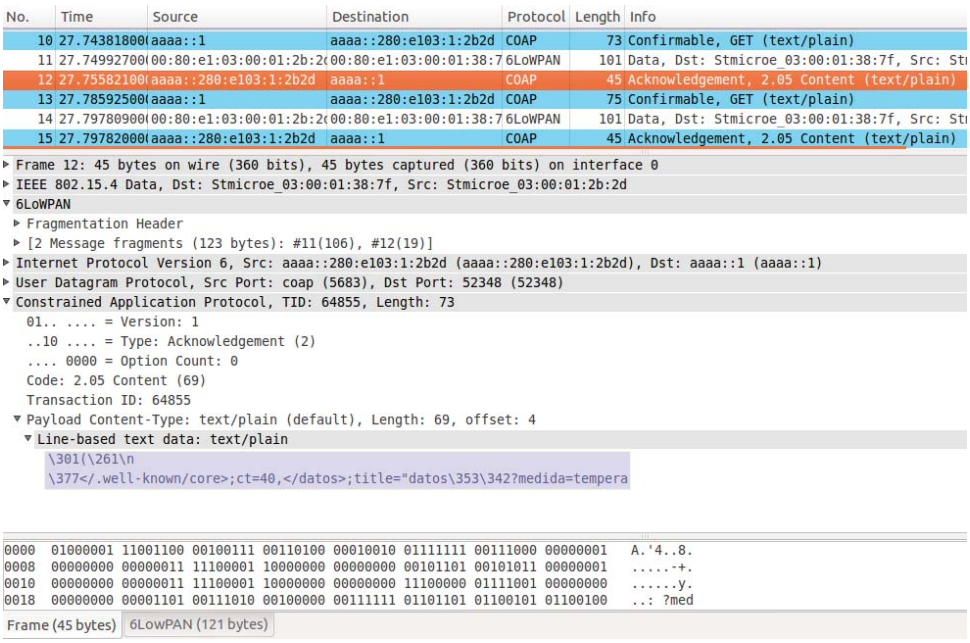


FIGURA 4.13: Reconocimiento CoAP /.well-know/core

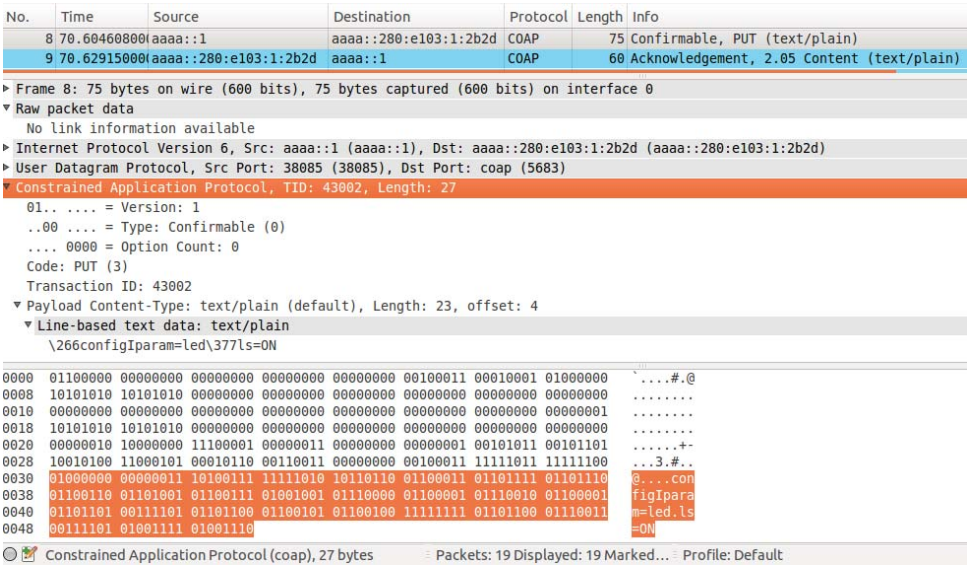


FIGURA 4.14: Petición CoAP en el interfaz tun0

A diferencia de HTTP, en este caso no es necesaria la fragmentación a nivel 6LoWPAN, pues el tamaño de la cabecera CoAP es más pequeña, lo que se traduce en un tamaño de mensaje menor para la recolección del mismo flujo de datos, 79 bytes frente a los 243 bytes que se tenían con el recurso en HTTP, consiguiendo una reducción del 67 %. Por otra parte, aunque CoAP no utilice un protocolo de transporte orientado a conexión como es TCP, permite tener la funcionalidad de reconomientos a nivel CoAP, lo que le convierte en el protocolo por excelencia para operar con 6LoWPAN.

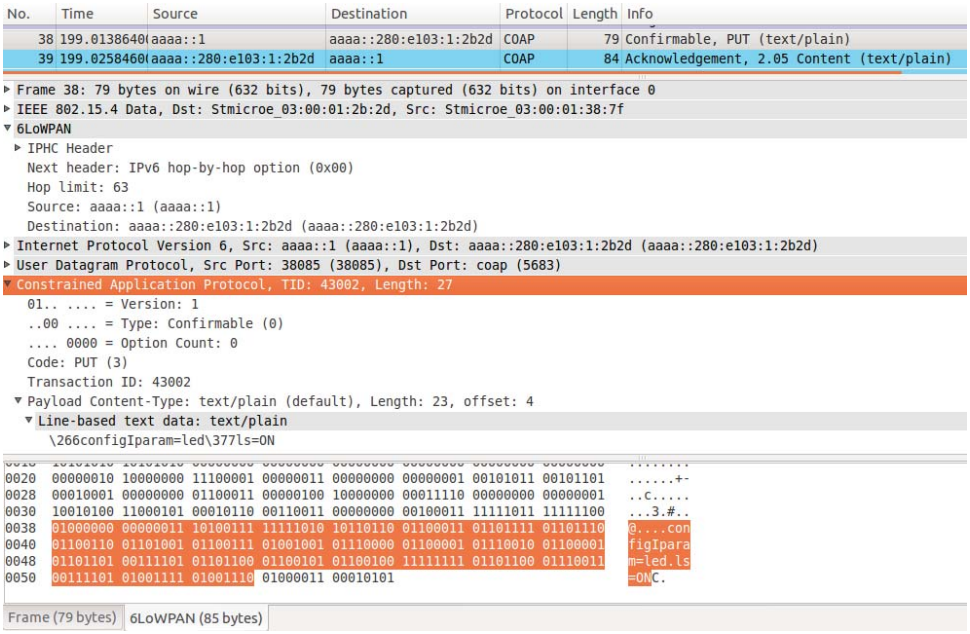


FIGURA 4.15: Petición CoAP en el interfaz LoWPAN

4.5. Interoperabilidad HTTP/CoAP

Aunque la solución de usar CoAP como protocolo de transferencia de hipertexto para la red LoWPAN es en gran medida más ventajosa que HTTP, a día de hoy, pocos (si no ninguno) son los navegadores web que entienden el protocolo.

Las similitudes existentes entre CoAP y HTTP y el hecho de que ambos se planteen para un uso muy similar han empujado al desarrollo de *proxies* intermedios que realicen la traducción HTTP/CoAP y viceversa, permitiendo acceder desde HTTP a los nodos que operan empleando CoAP. Uno de los más conocidos ha sido desarrollado por Francesco Corazza [44] y forma parte del framework de Californium [45]. En la Figura 4.16 se muestra el escenario utilizando el *proxy*.

El funcionamiento de dicho *proxy* considera que la petición CoAP se incorpora como un parámetro a la petición HTTP que se lanza al proxy, conformando una dirección URI tal y como se muestra en la Tabla 4.4.

URL HTTP del proxy			URL CoAP del nodo		
@IPv6	Puerto	Recurso	@IPv6	Puerto	Recurso
[localhost]	:8080	/proxy/	[aaaa::280:e103:1:2b2d]	:5683	/config?param=ipDestino

TABLA 4.4: Petición CoAP a través del proxy

El *proxy* se encarga de extraer la información y remitir la petición CoAP al nodo correspondiente.

En la Figura 4.17 se muestra como es una petición HTTP antes del *proxy* y en la Figura 4.18 tras ser traducida a CoAP. Es interesante observar como el verbo `POST` de HTTP se sigue

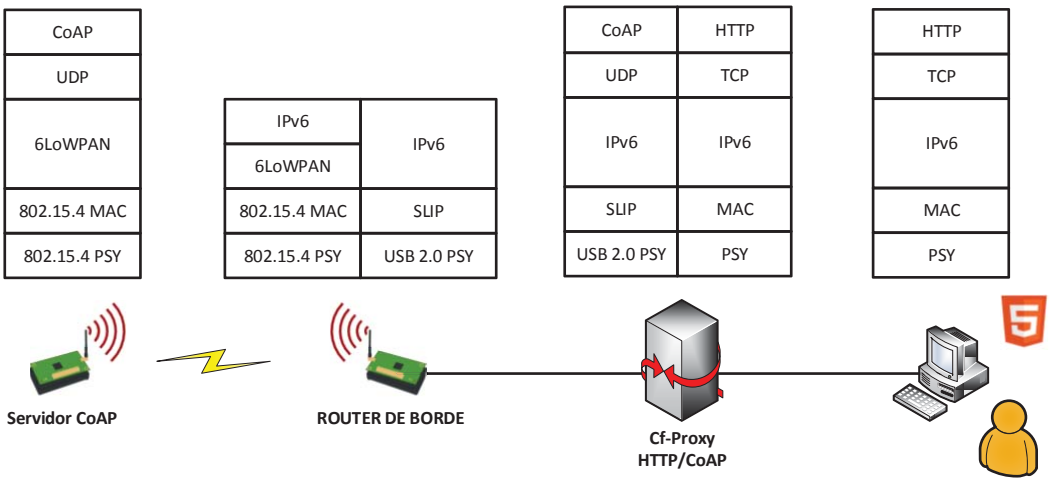


FIGURA 4.16: Escenario Proxy HTTP/CoAP

manteniendo en CoAP a través del campo *Code* con codificación binaria. Además, el mensaje CoAP se traduce como un mensaje de tipo `CON` garantizando la llegaba de los datos como sucede con HTTP. La respuesta CoAP del nodo LoWPAN y su traducción a HTTP se muestra en la Figura 4.19 y 4.20. De forma análoga, se observa la traducción alineada del campo *Code*.

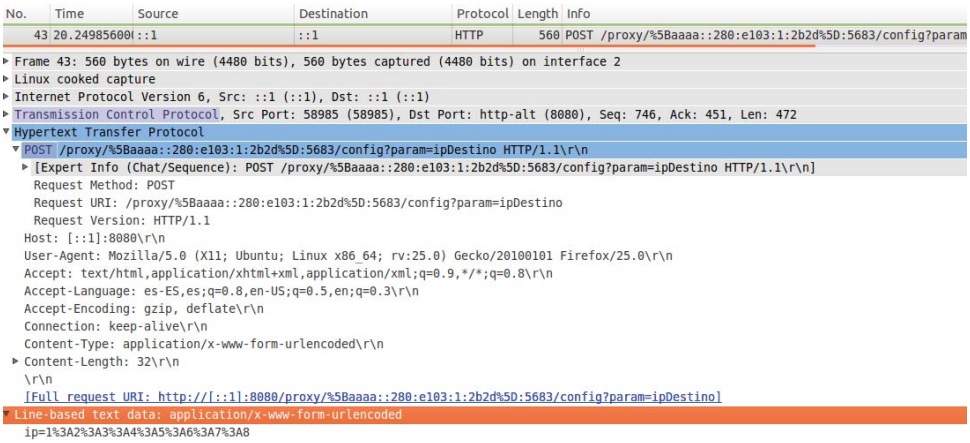


FIGURA 4.17: Petición HTTP utilizando Cf-proxy

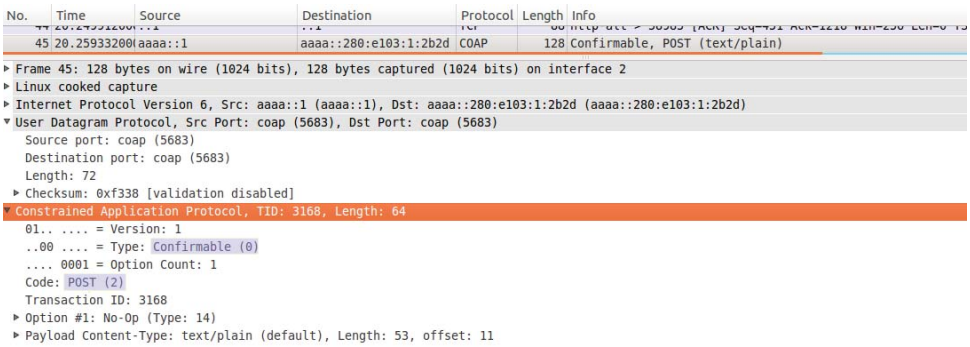


FIGURA 4.18: Petición CoAP utilizando Cf-proxy

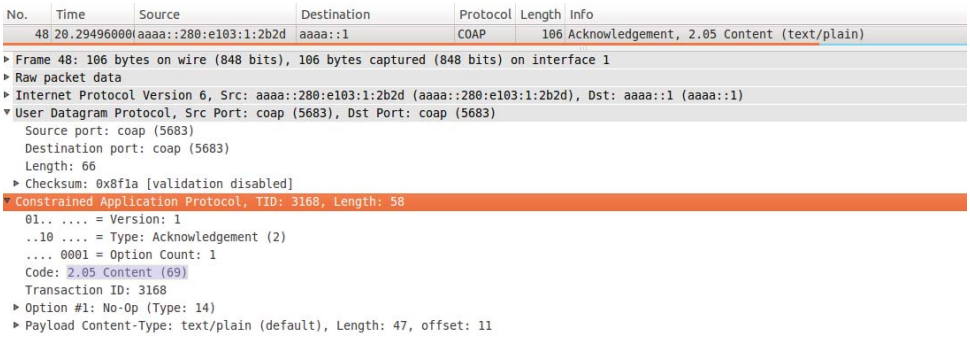


FIGURA 4.19: Respuesta CoAP utilizando Cf-proxy

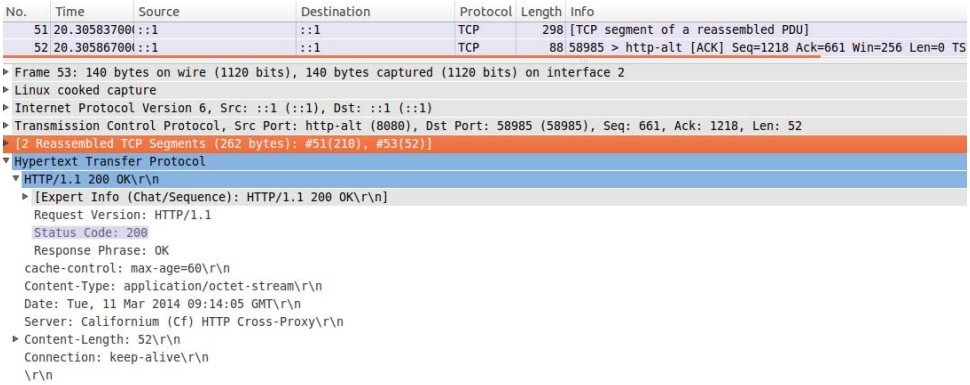


FIGURA 4.20: Respuesta HTTP utilizando Cf-proxy

Por tanto, de esta manera se consigue que la red LoWPAN se aproveche de los beneficios que CoAP ofrece. Básicamente esto se traduce en una optimización a nivel de byte dentro de la LoWPAN logrando evitar la fragmentación siempre que sea posible, cosa que con HTTP no es viable, pues el tamaño de los datos en este caso es muy superior (580 bytes frente a 128 bytes). Todo esto es posible sin necesidad de que un cliente remoto conozca el protocolo CoAP. En cualquier caso, se ha de tener en cuenta que se pierde la esencia de la conectividad extremo a extremo, pues no es el cliente el que se comunica con el propio nodo, si no el proxy, pero puede ser una solución intermedia hasta que CoAP sea interpretado de facto por los navegadores web.

Capítulo 5

Diseño de Servicios OML

La herramienta de instrumentación *OMF Measurement Library* (OML) [46] permite a los programadores desarrollar aplicaciones para la toma y gestión de medidas dentro de una red. De este manera, utilizando OML es posible centralizar los datos de medición recogidos por múltiples clientes en un único nodo central. Este último, deberá actuar como servidor recogiendo las inyecciones de los clientes y almacenándolas en una base de datos. En este capítulo se pretende que los nodos de la red LoWPAN actúen como clientes OML y sea un servidor externo a la red, el que recoja y almacene los datos inyectados por los nodos.

5.1. Conceptos teóricos

En primer lugar resulta imprescindible definir los conceptos básicos que forman parte de la arquitectura de OML, cuya descripción se muestra en la Figura 5.1:

- *Measurement Point* (MP): Punto de medida encargado del sensado de los datos
- *Measurement Stream* (MS): Corriente de medicción mediante la cual se envían los datos sensados desde los MP hacia los puntos de recogida
- *Schema*: Esquema o patrón sintáctico que siguen los datos al ser enviados desde un MP a través de un MS

El protocolo de aplicación encargado de describir la comunicación entre dispositivos recibe el nombre de *OML Measurement Stream Protocol* (OMSP) [47]. En la Figura 5.2 se muestra la pila de protocolos correspondiente y en la Figura 5.3 un diagrama de una comunicación basada en OML.

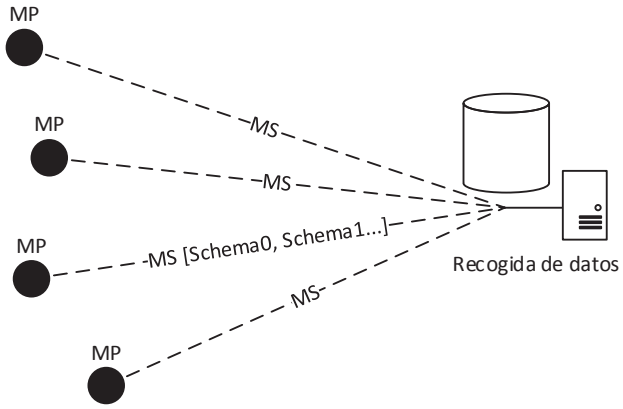


FIGURA 5.1: Arquitectura de OML

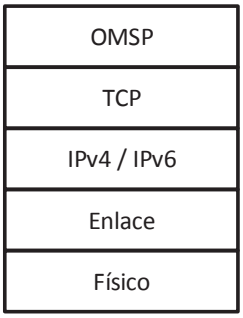


FIGURA 5.2: Pila de protocolos de OML

Apoyándose en TCP como protocolo de transporte, el cliente OML establece una conexión con el servidor remoto a través de la cual envia un conjunto de cabeceras con el formato *(key:value)* que describen la configuración del punto de inyección junto con la definición de los esquemas de los puntos de medida o MP a los que inyectar datos a posteriori a través de una corriente de medición o MS.

OMSP soporta dos modelos de codificación de datos, binario o texto, el cual será indicado durante la fase de intercambio de la configuración entre el MP y el servidor. Todos los parámetros de configuración establecidos al inicio de la sesión se usarán durante toda la conexión OML y no podrán ser modificados. Sin embargo, a partir de la versión OMSP v4 es posible la inyección de nuevos MP a través del esquema 0 (_experiment_metadata) sin necesidad de reiniciar el proceso y tener que abrir una nueva conexión.

Una vez establecida la configuración OML solo es necesaria la inyección de datos en el flujo TCP siguiendo el patrón definido previamente. Es interesante observar que en caso de que la conexión TCP se rompa, va a ser necesario el envío de la cabecera OML de nuevo, pero los datos van a seguir referenciados al mismo identificador de nodo, y a efectos de servicio, van a seguir almacenándose en la tabla generada en la conexión anterior.

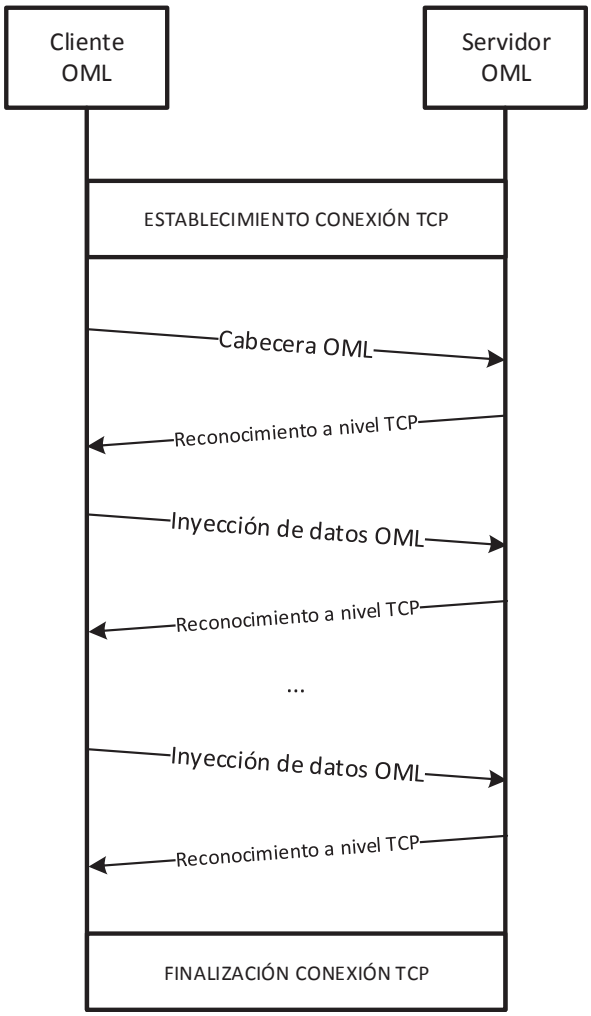


FIGURA 5.3: Diagrama de comunicación OML

El formato de la cabecera OMSP es el siguiente:

- **protocol:** Indica la versión del protocolo OMSP a utilizar (1-4).
- **domain:** String que identifica el nombre del experimento.
- **start-time:** Fecha local del cliente (*Timestamp* en segundos) al enviar la cabecera. El servidor lo utiliza para reajustar las marcas temporales de las medidas almacenadas.
- **sender-id:** String que identifica la fuente de la medida.
- **app-name:** String que identifica el nombre de la aplicación.
- **schema:** Describe el patrón que siguen los datos para enviarse a través de un MS.
- **content:** Formato de codificación de los datos (*binary*/*text*).

5.2. Implementación

Conocida la operativa de OML y la estructura del protocolo OMSP, a continuación se describe el proceso que se ha seguido para que los nodos de la red LoWPAN actúen como clientes OML. En la Figura 5.4 se muestra el escenario, donde a cada mota se le ha añadido la pila de protocolos OMSP. Haciendo uso de la librería *Protosockets* de Contiki se implementan interfaces de configuración, siguiendo los procesos descritos en el apartado anterior, y el propio envío de información según OML. Tras el estudio del comportamiento de los protocolos HTTP y CoAP en las redes 6LoWPAN se decide utilizar CoAP frente al HTTP para el despliegue de los recuros de configuración con el fin de optimizar el rendimiento de la red. Dichos recursos podrán definir un comportamiento síncrono o asíncrono en la inyección de los datos. Por su parte la inyección de los datos en OML se realiza sobre HTTP, siguiendo las recomendaciones, y empleando el modo texto con el fin de facilitar la programación y el posterior análisis aunque no sea la opción más óptima.

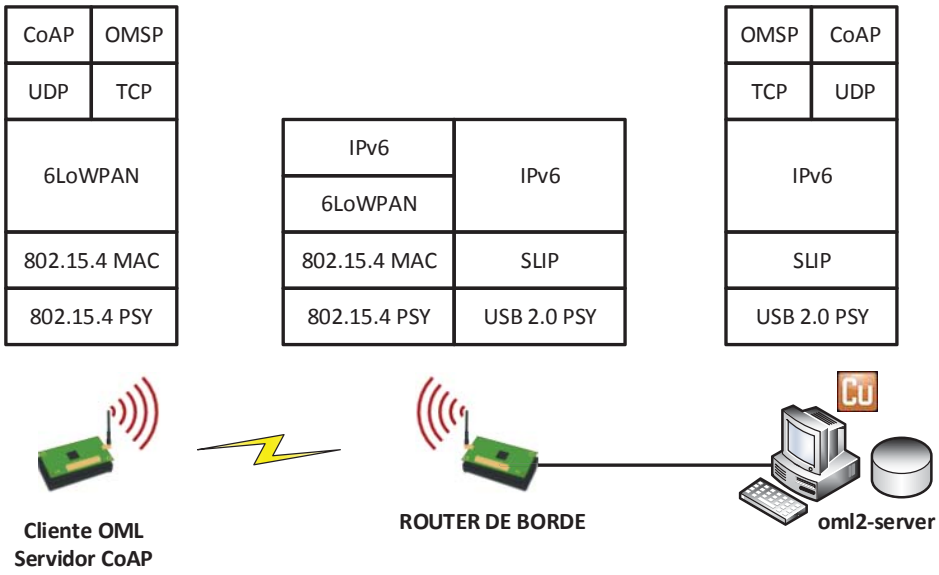


FIGURA 5.4: Escenario OML

El funcionamiento de la implementación del cliente OML (ver Figura 5.5) es el siguiente:

- El cliente establece una conexión TCP con el servidor OMSP en el puerto por defecto (3003).
- Si se establece la conexión, se inicializa un *Protosocket* a partir del cual se mandará la información.
- El cliente prepara la cabecera OML y la envía al destino.

- La variable `Run` determinará si el cliente OML está preparado para la inyección de datos. Mientras sea igual a 1, el cliente OML permanecerá activo en tres posibles modos de operación: apagado, asíncrono o periódico. El modo de operación lo determinará la variable `Mode`, que podrá ser configurada de forma remota a través del servicio REST con una petición `PUT` al recurso `/config?param=periodoEnvio`, tal y como se explicó en 4.2.

  En el modo apagado el cliente OML mantiene activa su conexión pero no realiza ninguna inyección de datos.

  En el modo asíncrono se inyectan datos cuando la diferencia absoluta de temperatura entre la última inyección y el nuevo valor sensado es mayor de 10 décimas de grado (valor establecido mediante las interfaces de configuración).

  En el modo síncrono se realiza inyección de datos cada `Mode` segundos.

- Por último, podrá configurarse de forma remota la dirección IPv6 del servidor OML al que conectarse. De esta manera, si se accede al recurso `/config?param=ipDestino` a través del método `PUT`, la variable `Run` pasará a tomar el valor 0, cerrando así la conexión TCP y configurando la nueva dirección a la cual lanzar la nueva petición de conexión.

5.3. Resultados

Seguidamente se muestra el resultado de la implementación citada anteriormente. Para comprobar el funcionamiento del cliente OML, se ha utilizado el aplicativo `oml2-server` [1] como servidor OML, que almacena las inyecciones en una base de datos SQLite3. La elección de este tipo de base de datos ha sido realizada con el fin de simplificar la solución del sistema, pues OML permite la inyección de información en otros tipos de bases de datos.

Por otra parte, aunque se haya decidido realizar la codificación de datos en modo texto tal y como se comentó anteriormente, se ha realizado una evaluación con codificación binaria sin observar grandes diferencias, aunque se han descubierto ciertos *bugs* que se han sido reportado a los desarrolladores, como un tamaño insuficiente para representar los identificadores y alguna deficiencia en la definición de los tipos de datos.

El modo de operación para la comprobación del cliente OML ha sido el siguiente:

- Configuración por defecto contra el servidor OML remoto.
- Inyección de datos por defecto en modo síncrono `Mode = 5`.
- Configuración por medio de una petición CoAP en modo apagado `Mode = 0`.

[1] "`http://oml.mytestbed.net/doc/oml/2.11/oml2-server.1.html`"

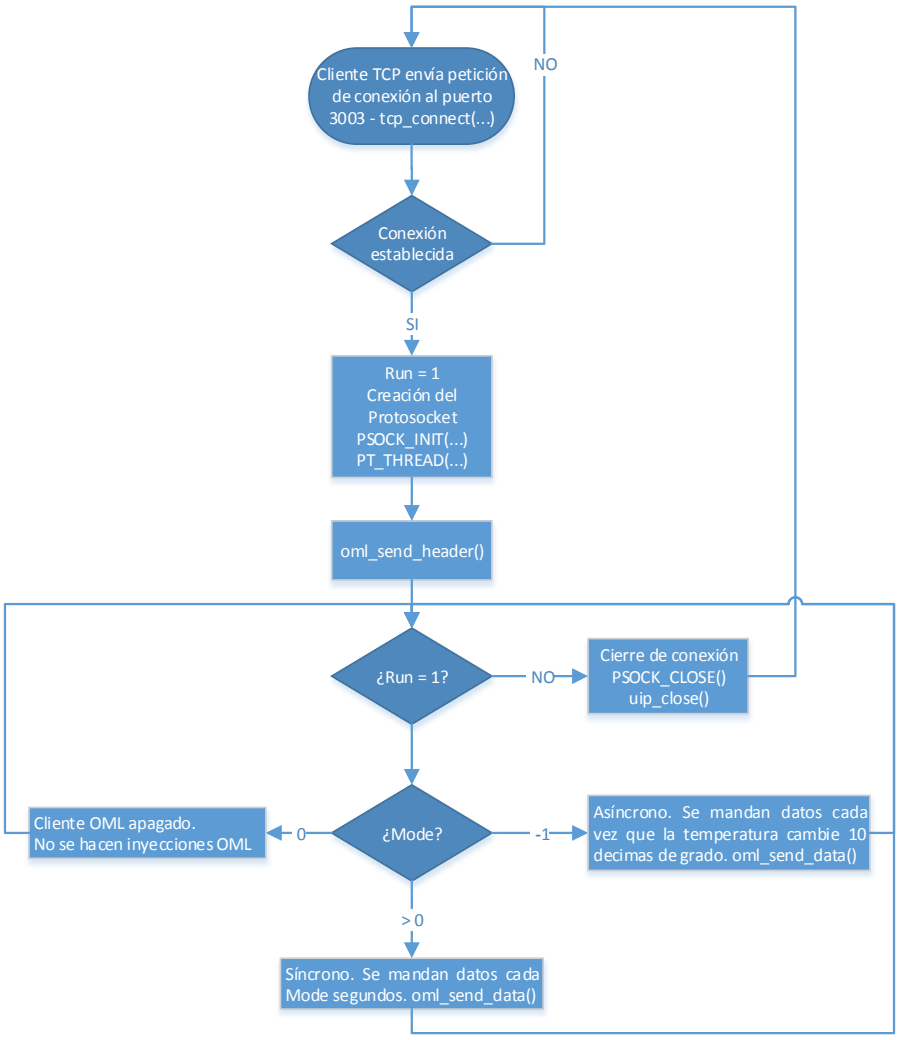


FIGURA 5.5: Diagrama de flujo del cliente OML

- Al cabo de un cierto intervalo de tiempo se vuelve a configurar en modo síncrono `Mode = 30`.
- Configuración por medio de una petición CoAP la dirección IPv6 del servidor remoto a otro valor con el fin de cerrar la conexión TCP.

En el Cuadro de Texto 5.1 se muestra el resultado del mensaje OML generado por las motas y en la Figura 5.6 se presenta el contenido de la base de datos. Se observa como los intervalos de tiempo de las inyecciones de datos corresponden con los esperados en función del modo de operación en cada momento.

Siguiendo la misma idea que en capítulos anteriores, se ha estudiado el comportamiento de los paquetes dentro de la red LoWPAN, teniendo especial interés indicar que una inyección de

datos con codificación en modo texto ocupa 85 bytes, lo que implica no ser necesaria la fragmentación de los paquetes 6LoWPAN. Por tanto, desde un punto de vista de eficiencia de byte OML puede ser válido para una red de este tipo, aunque se ha de remarcar que TCP no es en la práctica un protocolo apropiado para una LoWPAN, ya que el mantener una conexión activa de forma constante puede reducir drásticamente la autonomía de las baterías.

```
protocol: 4
domain: EXPERIMENTO
start-time: 9
sender-id: 1
app-name:
schema: 0 _experiment_metadata subject:string key:string value:string
schema: 1 APLICACION_MEDIDAS Temperatura:long Ax:long Ay:long Az:long
content: text
28.1.0.322.-288.72.972
34.1.1.321.-288.90.918
    ...
```

Cuadro de Texto 5.1: Mensaje OML

	oml_tuple_id	oml_sender_id	oml_seq	oml_ts_client	oml_ts_server	Temperatura	Ax	Ay	Az
1	1	1	0	0	1409857629	322	-288	72	972
2	2	1	1	34	1409857634	321	-288	90	918
3	3	1	2	39	1409857640	322	-324	72	936
4	4	1	3	45	1409857645	322	-270	72	918
5	5	1	4	50	1409857651	321	-288	72	972
6	6	1	5	56	1409857656	322	-306	90	972
7	7	1	6	61	1409857662	323	-306	54	954
8	8	1	7	67	1409857667	322	-306	108	918
9	9	1	8	72	1409857673	322	-306	72	972
10	10	1	9	118	1409857718	322	-270	72	918
11	11	1	10	123	1409857724	322	-306	72	972
12	12	1	11	129	1409857729	323	-306	90	972
13	13	1	12	134	1409857735	323	-288	72	918
14	14	1	13	140	1409857740	323	-270	72	954
15	15	1	14	170	1409857771	322	-288	72	954
16	16	1	15	201	1409857801	323	-324	90	918

Figura 5.6: Base de datos OML

En conclusión, OML permite un mecanismo de inyección y recolección de datos en un servidor remoto sin necesidad de gran complejidad en el cliente. De esta manera, es posible centralizar los datos de una red con múltiples nodos en un solo servidor, y que sea éste el que sirva la información en los diversos formatos estándar de represetanción a los clientes que deseen acceder a ellos. Este servidor podrá realizar un preprocesado de los datos infiriendo y exportando información de valor añadido. Adicionalmente, señalar que una solución basada en OML sobre UDP y CoAP sería una solución quizás más acertada para la mejora global del rendimiento del sistema.

Capítulo 6

Conclusiones y Líneas Futuras

En este último capítulo se recogen las principales conclusiones fruto de la realización de este proyecto. Además, se incluyen las posibles líneas futuras de trabajo que han quedado abiertas.

6.1. Conclusiones

El principal objetivo planteado en este proyecto fue el despliegue real de una red de sensores de bajo consumo con soporte 6LoWPAN. Con este fin, ha sido necesario el estudio del estándar, así como el análisis de las distintas soluciones existentes para la realización del escenario en cuestión.

A diferencia de otras redes de sensores desplegadas hoy en día, como por ejemplo el caso de SmartSantander [48], la solución propuesta en este trabajo permite la comunicación extremo a extremo sin necesidad de un *proxy* intermedio que se encargue de la traducción y encaminamiento del tráfico hacia/desde los distintos nodos de la red.

A pesar del lento despliegue que está teniendo IPv6 en el mundo Internet, el número de dispositivos que demandan este servicio cada vez es mayor, promoviendo así que la transición al nuevo protocolo de red sea ya una realidad inminente. Con la solución que se propone en este trabajo, los nodos de la red son dispositivos con dirección IPv6 propia y cuentan a priori con conectividad extremo a extremo con cualquier sistema de la red Internet.

En este sentido, el presente proyecto no se ha restringido a habilitar una isla 6LoWPAN, sino que se han realizado las configuraciones necesarias no solo para permitir la comunicación de esta subred con otras redes IPv6 de forma nativa, sino también con dispositivos conectados a Internet mediante IPv4.

Para dar soporte a la solución planteada, de entre las diferentes alternativas analizadas se ha seleccionado *Contiki* como sistema operativo que ejecutaran los dispositivos sensores. Si bien la plataforma *Contiki* habilita un conjunto de herramientas que facilitan el despliegue de diferentes servicios, el constante proceso de actualización y mejora de ella, resultado de ser un plataforma de código abierto, ha obligado a la continua reinstalación de los equipos empleados para incorpar las nuevas funcionalidades disponibles, la mayoría de ellas enfocadas a dotar de robustez y fiabilidad.

Configurada y establecida la red de sensores, con el propósito de ofrecer servicios sobre los nodos de la red desplegada, se han estudiado e implementado distintas alternativas que permitan el acceso a recursos que ofrecen los nodos, así como un sistema de recolección y almacenamiento de datos. En términos de eficiencia, la solución REST basada en CoAP presenta mayor rendimiento frente a la tradicional empleando HTTP. Sin embargo, los navegadores actuales no tienen implementado el protocolo CoAP de facto, y es necesario el uso de *plugins* que conozcan el protocolo.

Adicionalmente, se ha incluido el soporte para la gestión de las medidas de las métricas disponibles en los sensores mediante el despliegue de OML sobre *Contiki*, desarrollo este novedoso para éste sistema operativo. Así se ha evaluado el comportamiento de OML sobre una red 6LoWPAN, con resultado satisfactorio, lo que permitirá integrar en un futuro cualquier nodo sensor en dicho sistema.

Señalar que la realización de este trabajo ha servido de apoyo en la docencia de la asignatura Redes No Convencionales del 4º curso del Grado Ingeniería de Tecnologías de Telecomunicación impartida en la Universidad de Cantabria, al facilitar la confección de una práctica docente de dicha asignatura que acercara los conceptos de 6LoWPAN más allá de la propia teoría del protocolo.

Personalmente, el desarrollo de este proyecto ha permitido ampliar los conocimientos telemáticos adquiridos durante la carrera, desde el estudio completo de la pila de comunicaciones de la red Internet y del estándar 6LoWPAN, hasta el desarrollo de un despliegue de una red de sensores real e implementación de los servicios de la misma.

6.2. Líneas futuras

Este proyecto abre distintas posibles vías para la mejora y continuación de nuevos trabajos. De forma natural, la ampliación del número de nodos de la red no solo en cuanto a sensores sino también en cuanto a la inclusión de múltiples *routers* de borde permitiría validar la escalabilidad de la solución al tiempo que habilitaría un tested en el que poder evaluar diferentes soluciones de encaminamiento, de servicios, etc.

Sería igualmente deseable que esos nuevos nodos incluyeran heterogeneidad tanto a nivel *software* como *hardware*, lo que se podría suponer un despliegue en unas condiciones equiparables a las de una plataforma en producción. En este sentido, la plataforma disponible fruto del trabajo en el proyecto SmartSantander se erige como un potencial entorno de despliegue, pues no solo se cumplirían las condiciones de un despliegue en producción, sino que además permitiría evaluar la viabilidad de múltiples servicios.

Por otro lado, la comparativa e inclusión de la solución 6LoWPAN en SmartSantander posibilitaría también un análisis del consumo de los sensores, no solo desde el punto de vista de uso del canal (ya realizado a lo largo de este proyecto) sino desde el punto de vista energético. Por tanto, la evaluación de las distintas soluciones desarrolladas y observar la diferencia energética entre HTTP y CoAP resulta de interés.

Dada la importancia que tienen las políticas con las que se exportan las medidas a la red, extendiendo tradicionales en el acceso a la información a través de servicios de alto nivel, se sugiere el incorporar mecanismos de seguridad al sistema [49] y analizar las mejoras y la influencia en el rendimiento global.

Por otro lado, a lo largo de este proyecto se ha presentado OML como una solución para la gestión y almacenamiento de los datos recogidos. En la memoria se ha sugerido que la sustitución de HTTP sobre TCP por CoAP sobre UDP podría incrementar el rendimiento del sistema. Es por esto que como línea futura de trabajo también se plantea realizar esta adaptación y compararla con el procedimiento desplegado en este proyecto.

Por último, y considerando la importancia de los servicios desplegados sobre las redes de sensores, se plantea, por un lado el empleo de formatos estándar en la transferencia de datos sobre 6LoWPAN y su evaluación y por otro el desarrollo de aplicaciones móviles que permitan la gestión remota de los nodos y sus recursos.

Bibliografía

[1] Steve Deering and Robert Hinden. Internet protocol, version 6 (ipv6) specification, 1998. URL `http://www.ietf.org/rfc/rfc2460.txt`.

[2] Gabriel Montenegro, Nandakishore Kushalnagar, J Hui, and D Culler. Transmission of ipv6 packets over ieee 802.15. 4 networks, 2007. URL `http://www.ietf.org/rfc/rfc4944.txt`.

[3] IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Technical report, 2006. URL `http://dx.doi.org/10.1109/ieeestd.2006.232110`.

[4] Christian Daniel Arévalo Anchundia, Jonnathan Jair Rojas Urbano, and Robin Álvarez Rueda. Material didáctico para el estudio y simulación de cdma: Aplicación a comunicaciones móviles. 2012.

[5] IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Net- works — Specific Requirements — Part 2: Logical Link Control. Technical report, 1998.

[6] SIG Bluetooth. Specification of the bluetooth system, version 1.1. *http://www. bluetooth. com*, 2001.

[7] Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting smart objects with ip: The next internet*. Morgan Kaufmann, 2010.

[8] Jon Postel. Internet protocol. September 1981. URL `http://www.ietf.org/rfc/rfc791.txt`.

[9] Kjeld Egevang and Paul Francis. The ip network address translator (nat). Technical report, RFC 1631, may, 1994.

[10] Susan Thomson. Ipv6 stateless address autoconfiguration. 1998. URL `http://www.ietf.org/rfc/rfc2462.txt`.

[11] Alex Conta and Mukesh Gupta. Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. 2006. URL `http://www.ietf.org/rfc/rfc2463.txt`.

[12] T Narten, E Nordmark, and W Simpson. Neighbor discovery for ip version 6 (ipv6). September 2007. URL `http://www.ietf.org/rfc/rfc4861.txt`.

[13] 6lowpan WG. IPv6 over Low power WPAN (6lowpan) . `https://datatracker.ietf.org/wg/6lowpan/`, 2008. [Online; último acceso: Noviembre-2014].

[14] N Kushalnagar, G Montenegro, and C Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, August 2007. URL `http://www.ietf.org/rfc/rfc4919.txt`.

[15] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks, September 2011. URL `http://www.ietf.org/rfc/rfc6282.txt`.

[16] Z Shelby, S Chakrabarti, E Nordmark, and C Bormann. Neighbor discovery optimization for ipv6 over low-power wireless personal area networks (6lowpans). Technical report, RFC 6775, November, 2012.

[17] T Winter, ABP Thubert, T Clausen, J Hui, R Kelsey, P Levis, K Pister, R Struik, and J Vasseur. Rpl: Ipv6 routing protocol for low power and lossy networks,(rfc 6550). *IETF ROLL WG, Tech. Rep*, 2012.

[18] Shahid Raza, Simon Duquennoy, and Goran Selander. Compression of ipsec ah and esp headers for constrained environments. 2013.

[19] Contiki OS. URL `http://www.contiki-os.org`.

[20] IPv6 Forum. IPv6 Ready Logo Site. `https://ipv6ready.org`. [Online; último acceso: Noviembre-2014].

[21] Tiny OS. URL `http://www.tinyos.net`.

[22] FreeRTOS. URL `http://www.freertos.org`.

[23] Contiki OS. Contiki Hardware. `http://www.contiki-os.org/hardware.html`. [Online; último acceso: Noviembre-2014].

[24] Adam Dunkels, Oliver Schmidt, Niclas Finne, Joakim Eriksson, Fredrik Österlind, Nicolas Tsiftes, and Mathilde Durvy. The contiki os: The operating system for the internet of things, 2011.

[25] Adam Dunkels, Oliver Schmidt, and Thiemo Voigt. Using protothreads for sensor node programming. In *Proceedings of the REALWSN*, volume 5, 2005.

[26] The Rime communication stack. URL `http://contiki.sourceforge.net/docs/2.6/a01798.html#_details`.

[27] Adam Dunkels. The uip embedded tcp/ip stack. *The uIP*, 1, 2006.

[28] M Ajmone Marsan, D Roffinella, and A Murru. Aloha and csma protocols for multichannel broadcast networks. In *Proc. 1985 Canadia˜ Communications and E˜ efgll Conference,(October 1982)*, 1982.

[29] Nicolas Tsiftes, Joakim Eriksson, and Adam Dunkels. Low-power wireless ipv6 routing with contikirpl. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 406–407. ACM, 2010.

[30] JL Romkey. Rfc 1055: Nonstandard for transmission of ip datagrams over serial lines: Slip. *Chapter 11 Networking*, 1988.

[31] Hurricane Electric. Hurricane Electric Free IPv6 Tunnel Broker. `https://tunnelbroker.net/`. [Online; último acceso: Noviembre-2014].

[32] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[33] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Rfc 3986: Uniform resource identifier (uri): Generic syntax. *The Internet Society*, 2005.

[34] Joseph W Yoder, Ralph E Johnson, and Quince D Wilson. Connecting business objects to relational databases. *Urbana*, 51:61801, 2005.

[35] Jon D Reid. *jQuery Mobile*. O'Reilly Germany, 2012.

[36] David Flanagan. *JavaScript: the definitive guide*. .ºO'Reilly Media, Inc.", 2002.

[37] William Waites. Cors check - http header logs for linked open data, October 2010. URL `http://river.styx.org/ww/2010/10/corscheck`.

[38] Marta García, Ramón Agüero, Luis Muñoz, and JoséÁngel Irastorza. Smart Enhancement of TCP Performance Over Lossy WLAN Channels Through the Combined Use of Link Layer Techniques. *Wireless Personal Communications*, 37(1-2):91–103, 2006. ISSN 0929-6212. doi: 10.1007/s11277-006-0962-1. URL `http://dx.doi.org/10.1007/s11277-006-0962-1`.

[39] Roberto Sanz, Ramón Agüero, Luis Sánchez, Johnny Choque, and Luis Muñoz. Aportación a la mejora de prestaciones de los protocolos TCP/IP en entornos inalámbricos multi-salto. In *Actas de las IV Jornadas de Ingeniería Telemática, JITEL*, pages 393–400, Septiembre 2003.

[40] Z SHELBY, HARTKE K SENSINODE, et al. Cons-mned application proto col (coap), draft—ietf-core-coap-18.

[41] Zach Shelby. Constrained restful environments (core) link format. 2012.

[42] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. A low-power coap for contiki. In *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)*, Valencia, Spain, October 2011.

[43] Matthias Kovatsch. Copper (Cu) CoAP user-agent for Firefox. `http://people.inf.ethz.ch/mkovatsc/copper.php`, . [Online; último acceso: Noviembre-2014].

[44] Francesco Corazza. Web-Integrated Smart City Infrastructure. `http://www.slideshare.net/francescocorazza/webintegrated-smart-city-infrastructure`. [Online; último acceso: Noviembre-2014].

[45] Matthias Kovatsch. Californium (Cf) CoAP framework in Java. `http://people.inf.ethz.ch/mkovatsc/californium.php`, . [Online; último acceso: Noviembre-2014].

[46] Olivier Mehani, Guillaume Jourjon, Thierry Rakotoarivelo, and Max Ott. An instrumentation framework for the critical task of measurement collection in the future internet. *issues*, 24:46, 2012.

[47] OML. OML Measurement Stream Protocol (OMSP). `http://oml.mytestbed.net/doc/oml/latest/doxygen/omsp.html`. [Online; último acceso: Noviembre-2014].

[48] Luis Sanchez, Jose A Galache, Verónica Gutiérrez, Jose M Hernández, Jesús Bernat, Alex Gluhak, and Tomás García. Smartsantander: The meeting point between future internet research and experimentation and the smart cities. In *Future Network & Mobile Summit (FutureNetw), 2011*, pages 1–8. IEEE, 2011.

[49] Anass Rghioui, Mohammed Bouhorma, and Abderrahim Benslimane. Analytical study of security aspects in 6lowpan networks. In *Information and Communication Technology for the Muslim World (ICT4M), 2013 5th International Conference on*, pages 1–5. IEEE, 2013.

Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN

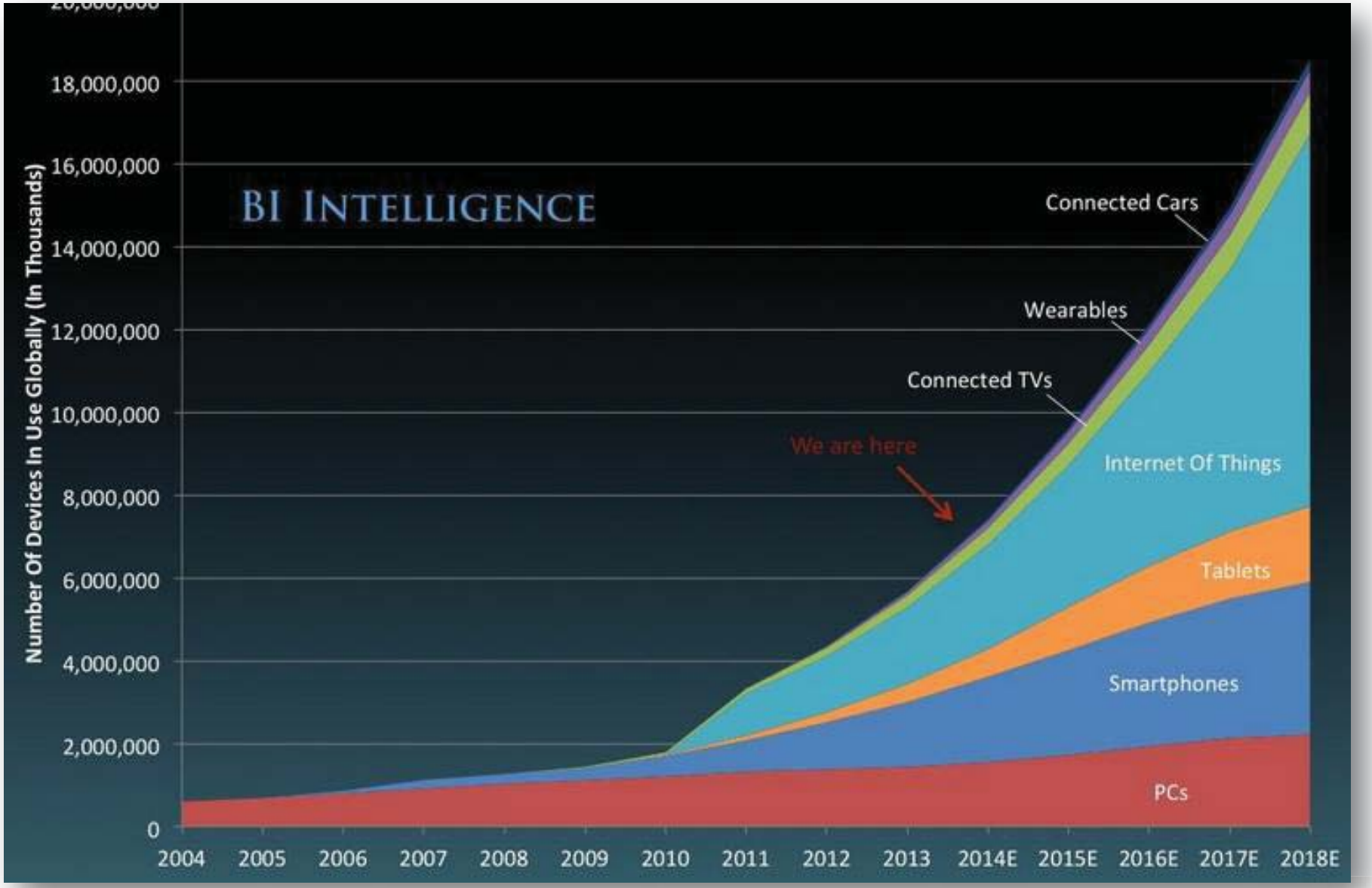
Proyecto Fin de Carrera – Ingeniería de Telecomunicación

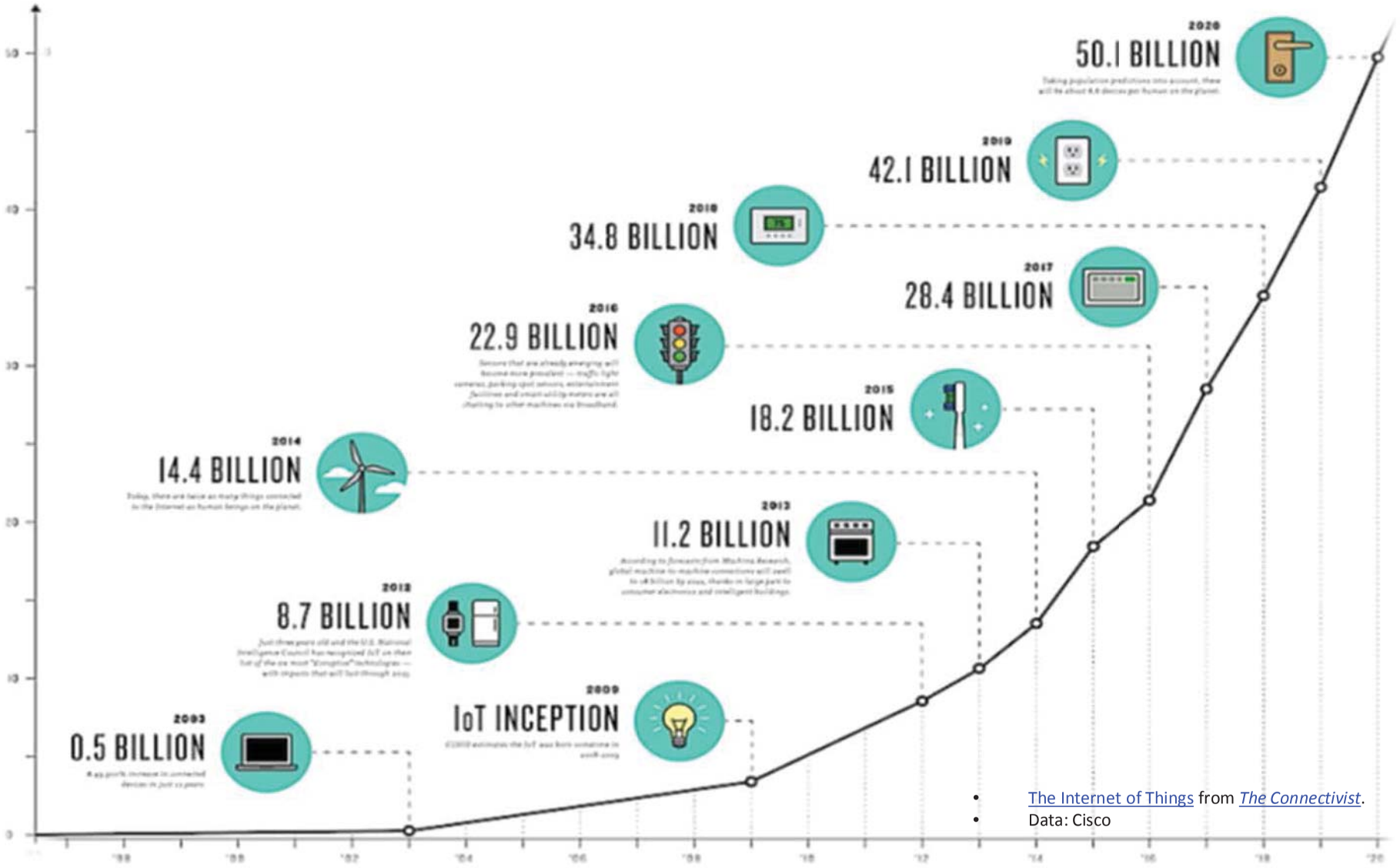
Autor: Óscar Vadillo Gutiérrez

Director: Jorge Lanza Calderón

ÍNDICE

- Introducción
- Protocolos de comunicación (6LoWPAN)
- Despliegue de la red y diseño de servicios
- Demostración
- Conclusiones y líneas futuras


BI INTELLIGENCE
Number Of Devices In Use Globally (In Thousands)
18,000,000
16,000,000
14,000,000
12,000,000
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014E
2015E
2016E
2017E
2018E
Connected Cars
Wearables
Connected TVs
We are here
Internet Of Things
Tablets
Smartphones
PCs



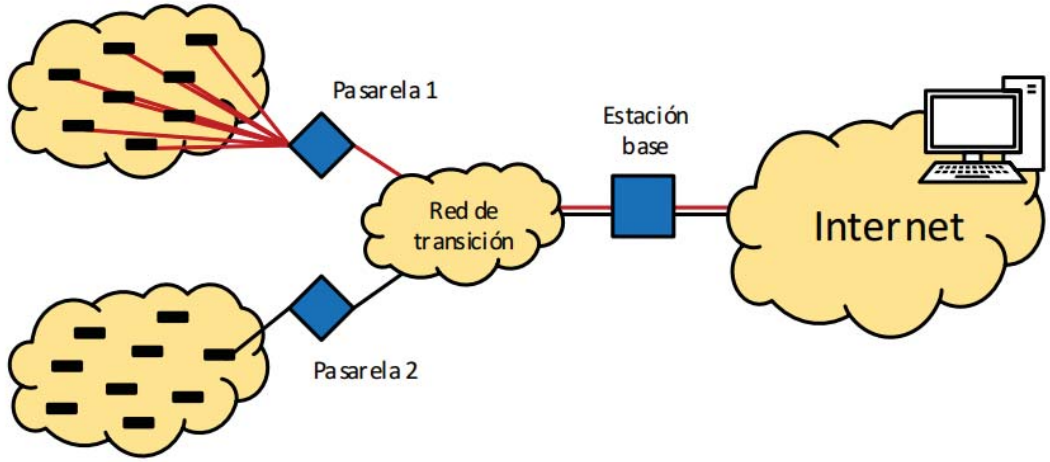
- The Internet of Things from *The Connectivist*.
- Data: Cisco

Internet of Things (IoT): Internet de las Cosas

- Conectar a la red millones de dispositivos (cosas) que hagan la vida más cómoda a las personas
- Capacidad de sensado y actuación
- Aplicaciones:
  - Agricultura
  - Salud
  - Ciudades inteligentes
  - Transporte
  - Industria
  - Domótica

Redes de sensores

- Conjunto de dispositivos autónomos e inteligentes con capacidades de sensado y comunicación inalámbrica
- Arquitectura tradicional

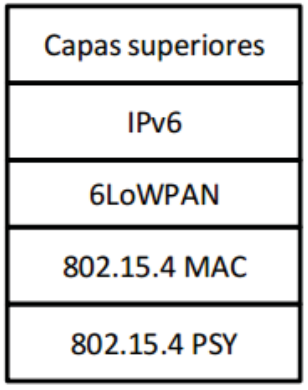


Motivación y objetivos

- Integración de los protocolos de Internet en las WSN
  - ☐ Identificación unívoca de las cosas en la red Internet
  - ☐ Direccionamiento prácticamente ilimitado
  - ☐ Estándar abierto
  - ☐ Conectividad *end-to-end*
  - ☐ Omitir el uso de protocolos propietarios y *gateways*
  - ☐ Tecnologías basadas en IP ya son conocidas
  - ☐ Fácil curva de aprendizaje
  - ☐ Infraestructura de red existente
- Es necesaria la adaptación de los protocolos de Internet

6LoWPAN

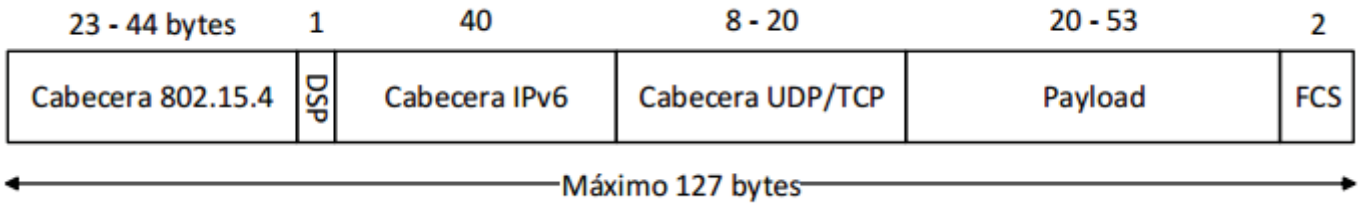
- Redes de sensores
  - IEEE 805.15.4
  - Tamaño trama < 127 bytes
- Red Internet
  - Protocolos de Internet
  - Paquetes de hasta 1280 bytes



- Capa de adaptación
  - Fragmentación y reensamblado
  - Compresión de cabeceras
    - HC1 – HC2
    - IPHC – NHC
  - Adaptación del protocolo *Neighbor Discovery*
- Optimizar el mundo IPv6 para las WSN

Compresión de cabeceras

- Paquete 6LoWPAN sin compresión

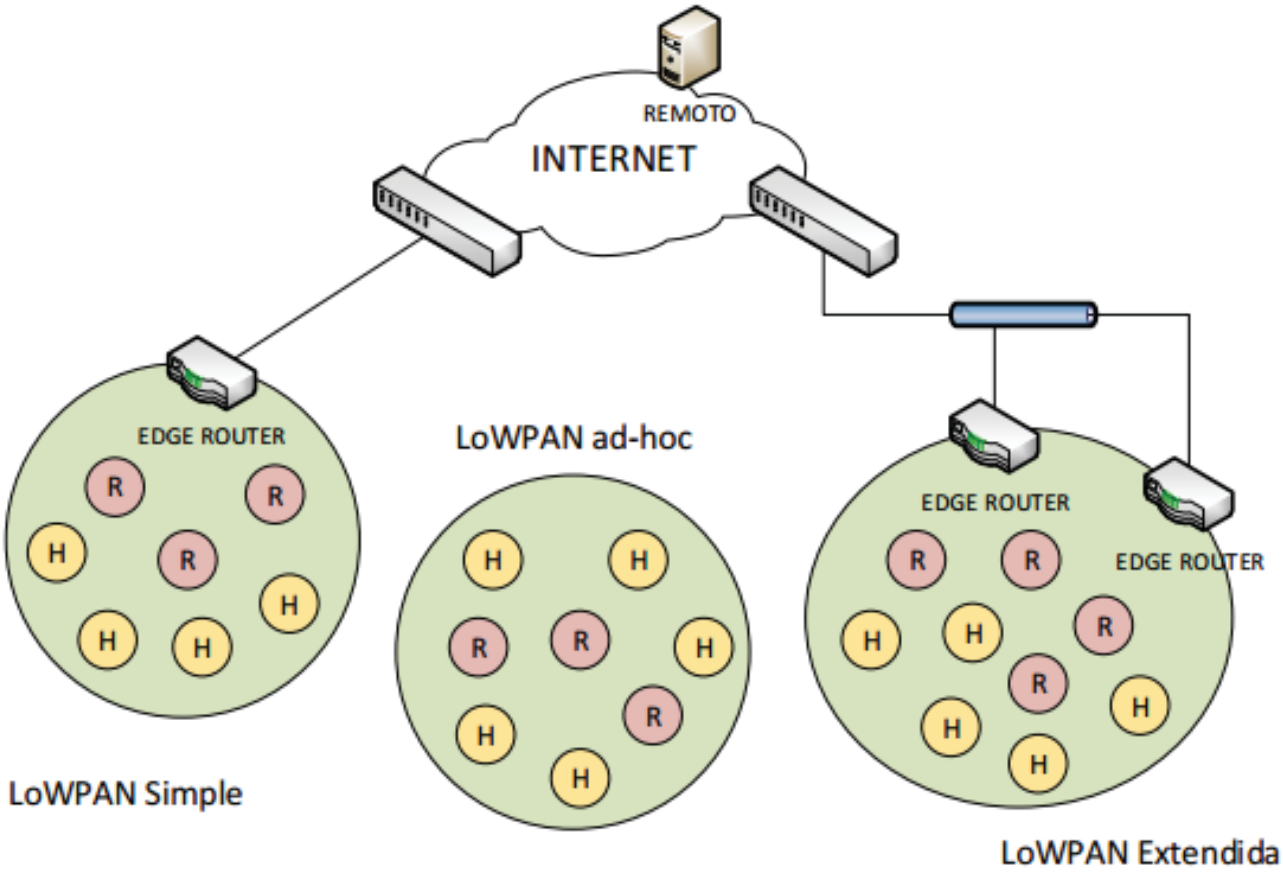


- Compresión IPHC – NHC
  - Direccionamiento IPv6
    - Comunicaciones *link-local: Stateless*
    - Comunicaciones *global: Stateful* (Contexto)

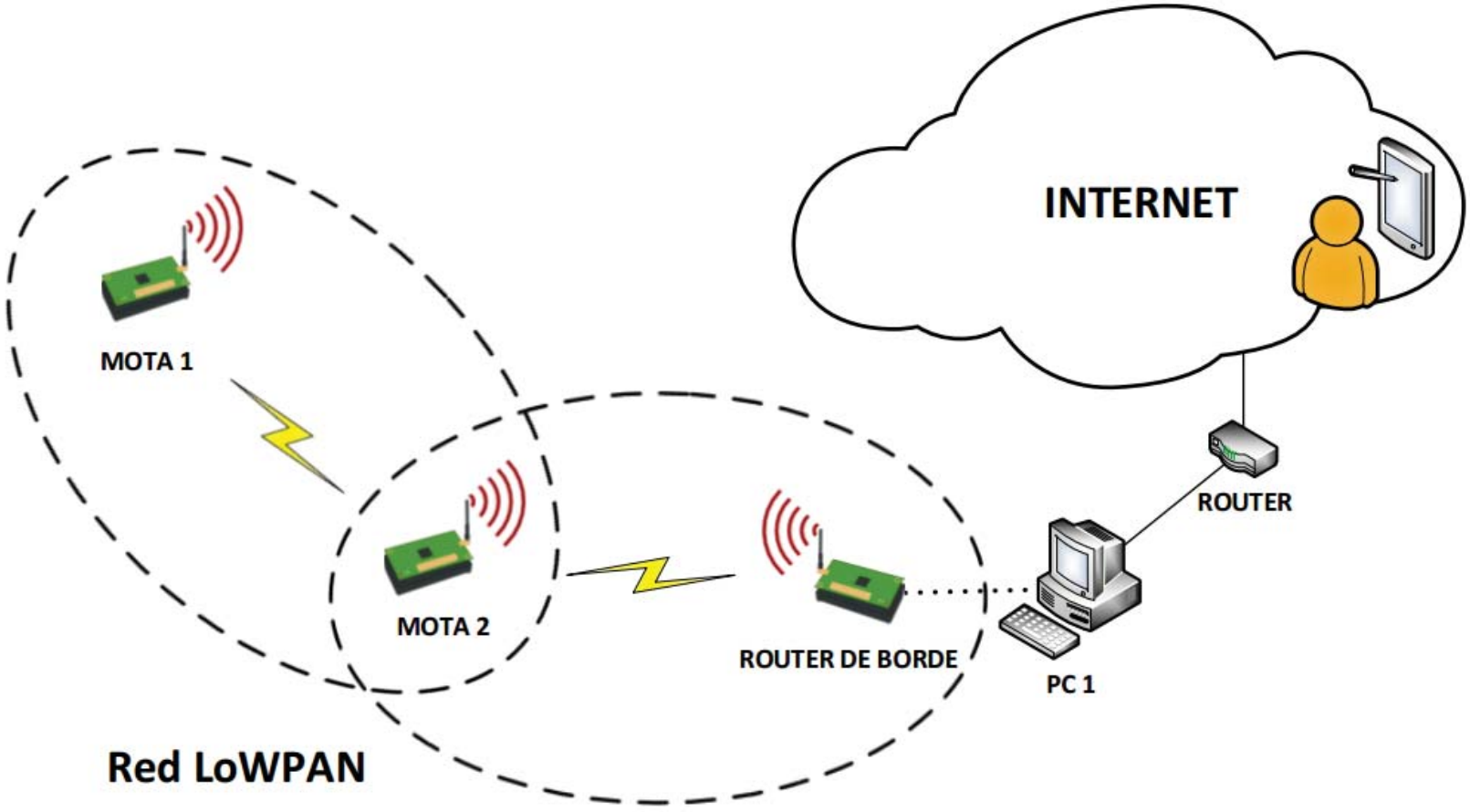
	UDP		TCP	
	# Bytes	Efic. (%)	# Bytes	Efic. (%)
Sin seguridad	53	41.7	41	32.3
AES-CCM-128	32	25.2	20	15.7

		UDP		TCP	
		# Bytes	Efic. (%)	# Bytes	Efic. (%)
Sin seguridad	*Stateless*	98	77.2	79	62.2
	Stateful	97	76.4	78	61.4
AES-CCM-128	*Stateless*	77	60.6	58	45.7
	Stateful	76	59.8	57	44.9

Arquitecturas 6LoWPAN



Objetivo



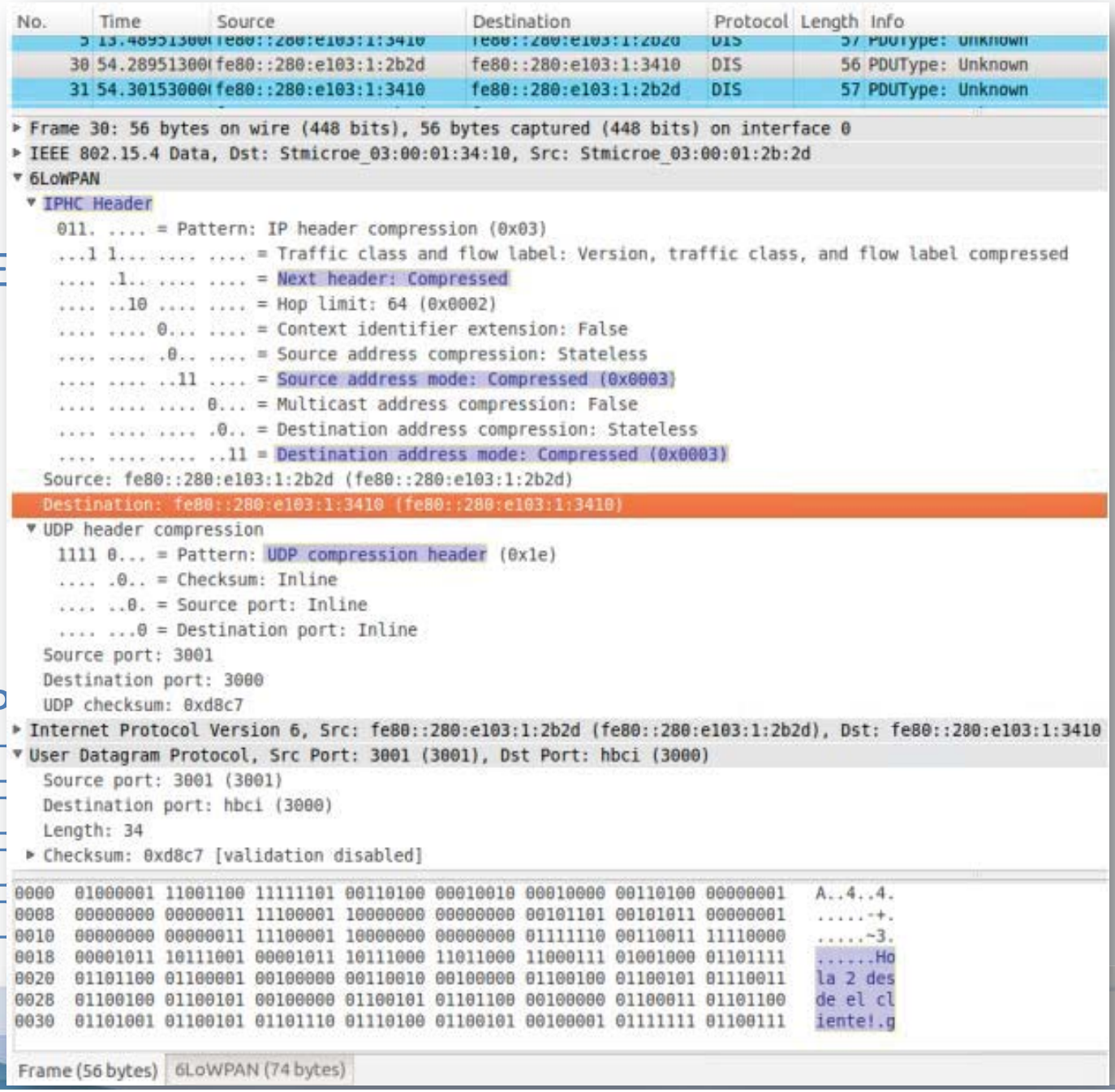
Plataformas de desarrollo

- Contiki OS
  - ☐ Robustez IPv6
    - IPv6 Ready Logo
  - ☐ Robustez 6LoWPAN
  - ☐ Continuo desarrollo

Sistema Operativo	**Contiki OS**	**Tiny OS**	**FreeRTOS**
Licencia	BSD	BSD	GPL
Lenguaje	C, C++	nesC	C
Pila TCP/IP (IPv6)	**uIP6**	**BLIP**	**NanoStack**
Auto-configuración	SI	SI	SI
UDP	SI	SI	SI
TCP	SI	Prototipo	NO
ICMP	SI	SI	SI
ND	SI	Limitado	NO
6LoWPAN	**SICSLOWPAN**	**b6lowpan**	**NanoStack**
Fragmentación	SI	SI	SI
Compresión	HC-06	HC-01	HC-01
Mesh-under	NO	SI	SI
RPL	SI	SI	NO

- Hardware
  - ☐ STM32W108C
  - ☐ Tipos de sensores
    - Temperatura
    - Acelerómetro

- E
- P


No. Time Source Destination Protocol Length Info
30 54.28951300 fe80::280:e103:1:2b2d fe80::280:e103:1:3410 DIS 56 PDUType: Unknown
31 54.30153000 fe80::280:e103:1:3410 fe80::280:e103:1:2b2d DIS 57 PDUType: Unknown
Frame 30: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
IEEE 802.15.4 Data, Dst: Stmicroe_03:00:01:34:10, Src: Stmicroe_03:00:01:2b:2d
6LoWPAN
IPHC Header
011. .... = Pattern: IP header compression (0x03)
...1 1... .... = Traffic class and flow label: Version, traffic class, and flow label compressed
.... .1.. .... = Next header: Compressed
.... ..10 .... = Hop limit: 64 (0x0002)
.... 0... .... = Context identifier extension: False
.... .0.. .... = Source address compression: Stateless
....11 .... = Source address mode: Compressed (0x0003)
.... 0... = Multicast address compression: False
.... .0.. = Destination address compression: Stateless
....11 = Destination address mode: Compressed (0x0003)
Source: fe80::280:e103:1:2b2d (fe80::280:e103:1:2b2d)
Destination: fe80::280:e103:1:3410 (fe80::280:e103:1:3410)
UDP header compression
1111 0... = Pattern: UDP compression header (0x1e)
.... .0.. = Checksum: Inline
.... ..0. = Source port: Inline
....0 = Destination port: Inline
Source port: 3001
Destination port: 3000
UDP checksum: 0xd8c7
Internet Protocol Version 6, Src: fe80::280:e103:1:2b2d (fe80::280:e103:1:2b2d), Dst: fe80::280:e103:1:3410
User Datagram Protocol, Src Port: 3001 (3001), Dst Port: hbci (3000)
Source port: 3001 (3001)
Destination port: hbci (3000)
Length: 34
Checksum: 0xd8c7 [validation disabled]
0000 01000001 11001100 11111101 00110100 00010010 00010000 00110100 00000001 A..4..4.
0008 00000000 00000011 11100001 10000000 00000000 00101101 00101011 00000001 .....-+.
0010 00000000 00000011 11100001 10000000 00000000 01111110 00110011 11110000 .....~3.
0018 00001011 10111001 00001011 10111000 11011000 11000111 01001000 01101111 ......Ho
0020 01101100 01100001 00100000 00110010 00100000 01100100 01100101 01110011 la 2 des
0028 01100100 01100101 00100000 01100101 01101100 00100000 01100011 01101100 de el cl
0030 01101001 01100101 01101110 01110100 01100101 00100001 01111111 01100111 iente!.g
Frame (56 bytes)
6LoWPAN (74 bytes)

Interconexión de

- Escenario

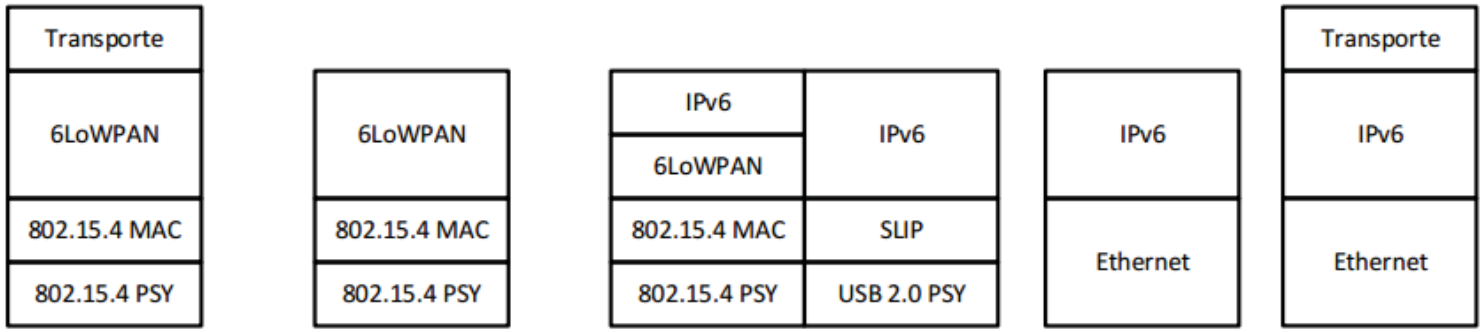
```
16 11.77214800 aaaa::1                  aaaa::280:e103:1:2b2d ICMPv6  104 Echo (ping) request
17 11.81128500 aaaa::280:e103:1:2b2d    aaaa::1               ICMPv6  104 Echo (ping) reply

▸ Frame 16: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
▸ Raw packet data
▾ Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::280:e103:1:2b2d (aaaa::280:e103:1:
  ▸ 0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 64
    Next header: ICMPv6 (58)
    Hop limit: 64
    Source: aaaa::1 (aaaa::1)
    Destination: aaaa::280:e103:1:2b2d (aaaa::280:e103:1:2b2d)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▾ Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
```

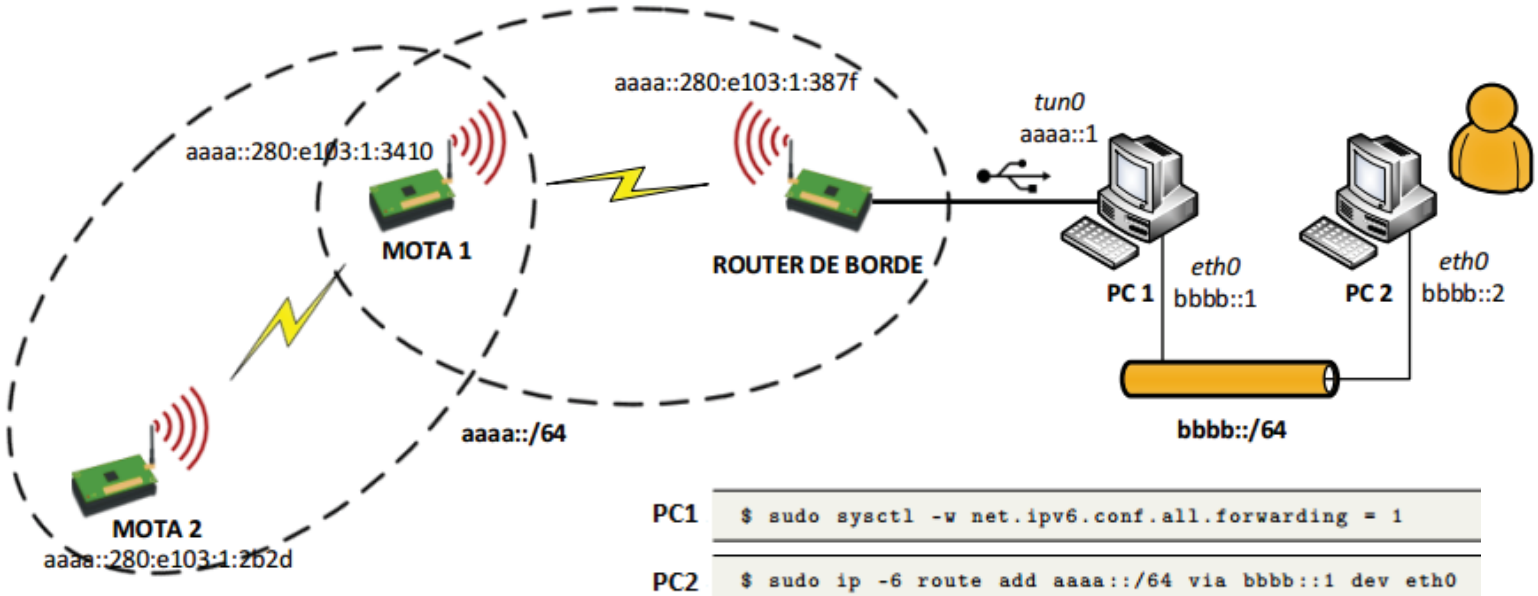
```
No.  Time          Source                 Destination            Protocol  Length  Info
  44 40.31410000   00:80:e1:03:00:01:38:7f 00:80:e1:03:00:01:2b:2d 6LoWPAN      104  Data, Dst: Stmi
  45 40.31411200   aaaa::1                aaaa::280:e103:1:2b2d  ICMPv6

▸ Frame 44: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on i
▸ IEEE 802.15.4 Data, Dst: Stmicroe_03:00:01:2b:2d, Src: Stmicroe_03:00:01:3
▾ 6LoWPAN
  ▾ Fragmentation Header
      1100 0... = Pattern: First fragment (0x18)
      Datagram size: 112
      Datagram tag: 0x0008
  ▾ IPHC Header
      011. .... = Pattern: IP header compression (0x03)
      ...1 1... .... .... = Traffic class and flow label: Version, traffic cl
      .... .0.. .... .... = Next header: Inline
      .... ..00 .... .... = Hop limit: Inline (0x0000)
      .... .... 1... .... = Context identifier extension: True
      .... .... .1.. .... = Source address compression: Stateful
      .... .... ..01 .... = Source address mode: 64-bits inline (0x0001)
      .... .... .... 0... = Multicast address compression: False
      .... .... .... .1.. = Destination address compression: Stateful
      .... .... .... ..11 = Destination address mode: Compressed (0x0003)
      0000 .... = Source context identifier: 0x00
      .... 0000 = Destination context identifier: 0x00
```

```
No.  Time          Source                 Destination            Protocol  Lengt
  44 40.31410000   00:80:e1:03:00:01:38:7f 00:80:e1:03:00:01:2b:2d 6LoWPAN    10
  45 40.31411200   aaaa::1                aaaa::280:e103:1:2b2d  ICMPv6      3

▸ Frame 45: 36 bytes on wire (288 bits), 36 bytes captured (288 bits) on interface
▸ IEEE 802.15.4 Data, Dst: Stmicroe_03:00:01:2b:2d, Src: Stmicroe_03:00:01:38:7f
▾ 6LoWPAN
  ▾ Fragmentation Header
      1110 0... = Pattern: Fragment (0x1c)
      Datagram size: 112
      Datagram tag: 0x0008
      Datagram offset: 104
  ▾ [2 Message fragments (114 bytes): #44(106), #45(10)]
      [Frame: 44, payload: 0-105 (106 bytes)]
    ▸ [Frame: 45, payload: 104-113 (10 bytes)]
      [Message fragment count: 2]
      [Reassembled 6LoWPAN length: 114]
▸ Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::280:e103:1:2b2d
```

```
/tunslip6 -s /dev/ttyACM1 aaaa::1/64
```

Enrutamiento multisalto

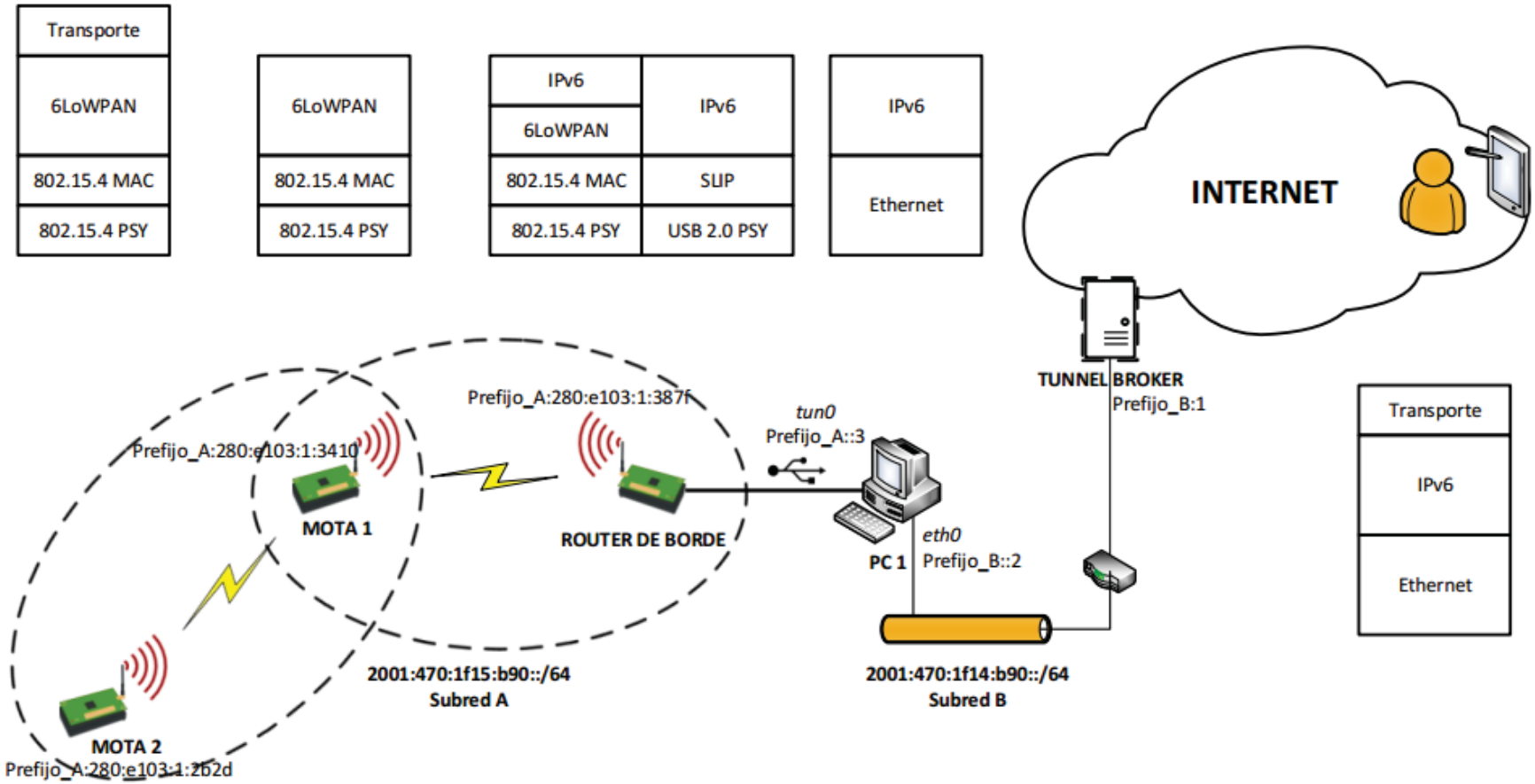




PC1
```
$ sudo sysctl -w net.ipv6.conf.all.forwarding = 1
```

PC2
```
$ sudo ip -6 route add aaaa::/64 via bbbb::1 dev eth0
```

Servicio Tunnel Broker



Diseño Servicios REST

- Definición de recursos

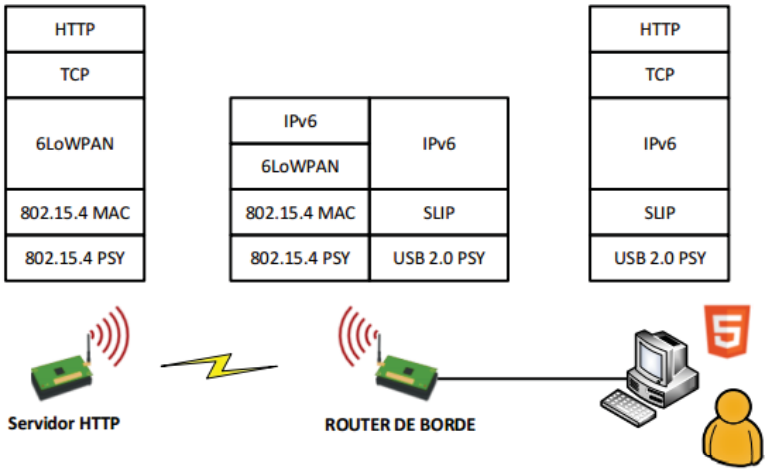
URI-Request	Operación	Representación del contenido
/index	READ	Resume recursos disponibles
/datos?medida=temperatura	READ	t(Decimas de grado)
/datos?medida=acelerometro	READ	ax(num);ay(num);az(num);
/config?param=ipDestino	UPDATE	ip=@IPv6
/config?param=periodoEnvio	UPDATE	pE=num
/config?param=led	UPDATE	ls=on/off

- Protocolos web

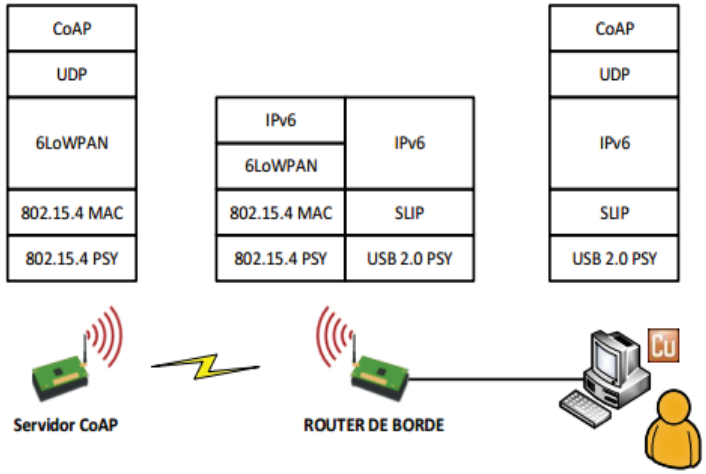
	HTTP	CoAP
Transporte	TCP	UDP
Codificación	Texto	Binaria
Desc. recursos	NO	SI
Reconocimiento	Nivel transporte	Nivel aplicación

Servicios REST

- HTTP

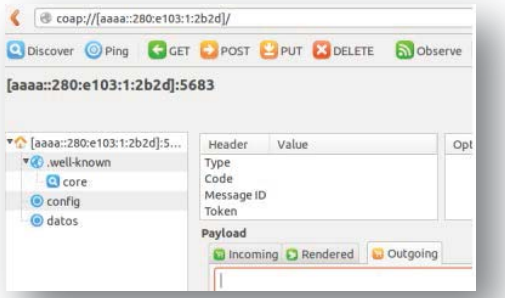


- CoAP



- Servidor:
  - Protosockets
- Cliente:
  - HTML5

- Servidor:
  - API Erbium
- Cliente:
  - Cupper



```
No.   Time          Source                Destination           Protocol Length
   98 93.52723000(  aaaa::1               aaaa::280:e103:1:2b2d HTTP     243
  107 93.68912500(  aaaa::280:e103:1:2b2d aaaa::1               HTTP      72

▸ Frame 98: 243 bytes on wire (1944 bits), 243 bytes captured (1944 bits) on int
▸ Raw packet data
▸ Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::280:e103:1:2b2
▸ Transmission Control Protocol, Src Port: 45607 (45607), Dst Port: http (80), S
▸ [2 Reassembled TCP Segments (403 bytes): #96(220), #98(183)]
▾ Hypertext Transfer Protocol
  ▸ PUT /config?param=led HTTP/1.1\r\n
    Host: [aaaa::280:e103:1:2b2d]\r\n
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0) Gecko/20100101 F
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3\r\n
    Accept-Encoding: gzip, deflate\r\n
    Content-Type: text/plain; charset=UTF-8\r\n
  ▸ Content-Length: 5\r\n
    Connection: keep-alive\r\n
    \r\n
    [Full request URI: http://[aaaa::280:e103:1:2b2d]/config?param=led]
▾ Line-based text data: text/plain
    ls=ON
```

```
▸ IEEE 802.15.4 Data, Dst: Stmicroe_03:00:01:2b:2d, Src: Stmicroe_03:00:01:38:7f
▾ 6LoWPAN
  ▾ Fragmentation Header
      1110 0... = Pattern: Fragment (0x1c)
      Datagram size: 251
      Datagram tag: 0x000a
      Datagram offset: 248
  ▾ [4 Message fragments (253 bytes): #151(106), #152(74), #153(74), #154(5)]
      [Frame: 151, payload: 0-105 (106 bytes)]
    ▸ [Frame: 152, payload: 104-177 (74 bytes)]
    ▸ [Frame: 153, payload: 176-249 (74 bytes)]
    ▸ [Frame: 154, payload: 248-252 (5 bytes)]
      [Message fragment count: 4]
      [Reassembled 6LowPAN length: 253]
▸ Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::280:e103:1:2b2d (aaaa::280
▸ Transmission Control Protocol, Src Port: 45607 (45607), Dst Port: http (80), Seq: 221, Ack
▸ [2 Reassembled TCP Segments (403 bytes): #149(220), #154(183)]
▾ Hypertext Transfer Protocol
  ▸ PUT /config?param=led HTTP/1.1\r\n
```

```
No.   Time          Source                Destination           Protocol Length Info
    8 70.60460800(  aaaa::1               aaaa::280:e103:1:2b2d COAP       75 Confi
    9 70.62915000(  aaaa::280:e103:1:2b2d aaaa::1               COAP       60 Ackno

▸ Frame 8: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface 0
▾ Raw packet data
    No link information available
▸ Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::280:e103:1:2b2d (aa
▸ User Datagram Protocol, Src Port: 38085 (38085), Dst Port: coap (5683)
▾ Constrained Application Protocol, TID: 43002, Length: 27
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 0000 = Option Count: 0
    Code: PUT (3)
    Transaction ID: 43002
  ▾ Payload Content-Type: text/plain (default), Length: 23, offset: 4
    ▾ Line-based text data: text/plain
        \266configIparam=led\377ls=ON
```

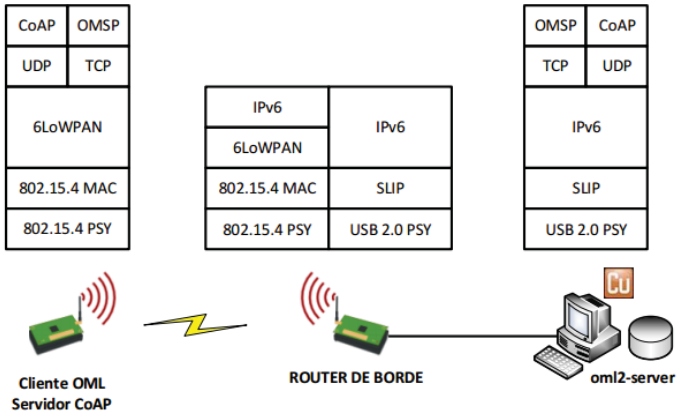
```
No.   Time          Source                Destination           Protocol Length Info
   38 199.0138640(  aaaa::1               aaaa::280:e103:1:2b2d COAP       79 Confirma
   39 199.0258460(  aaaa::280:e103:1:2b2d aaaa::1               COAP       84 Acknowle

▸ Frame 38: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0
▸ IEEE 802.15.4 Data, Dst: Stmicroe_03:00:01:2b:2d, Src: Stmicroe_03:00:01:38:7f
▾ 6LoWPAN
  ▸ IPHC Header
    Next header: IPv6 hop-by-hop option (0x00)
    Hop limit: 63
    Source: aaaa::1 (aaaa::1)
    Destination: aaaa::280:e103:1:2b2d (aaaa::280:e103:1:2b2d)
▸ Internet Protocol Version 6, Src: aaaa::1 (aaaa::1), Dst: aaaa::280:e103:1:2b2d (aaaa:
▸ User Datagram Protocol, Src Port: 38085 (38085), Dst Port: coap (5683)
▾ Constrained Application Protocol, TID: 43002, Length: 27
    01.. .... = Version: 1
    ..00 .... = Type: Confirmable (0)
    .... 0000 = Option Count: 0
    Code: PUT (3)
    Transaction ID: 43002
  ▾ Payload Content-Type: text/plain (default), Length: 23, offset: 4
    ▾ Line-based text data: text/plain
        \266configIparam=led\377ls=ON
```

CoAP es más eficiente que HTTP

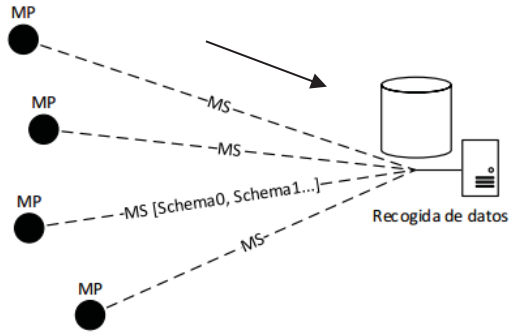
Diseño de servicios OML

- Protocolo OML Measurement Stream Protocol (OMSP)
  - Pseudo - estándar
  - Transporte: TCP
  - Codificación datos: texto/binario

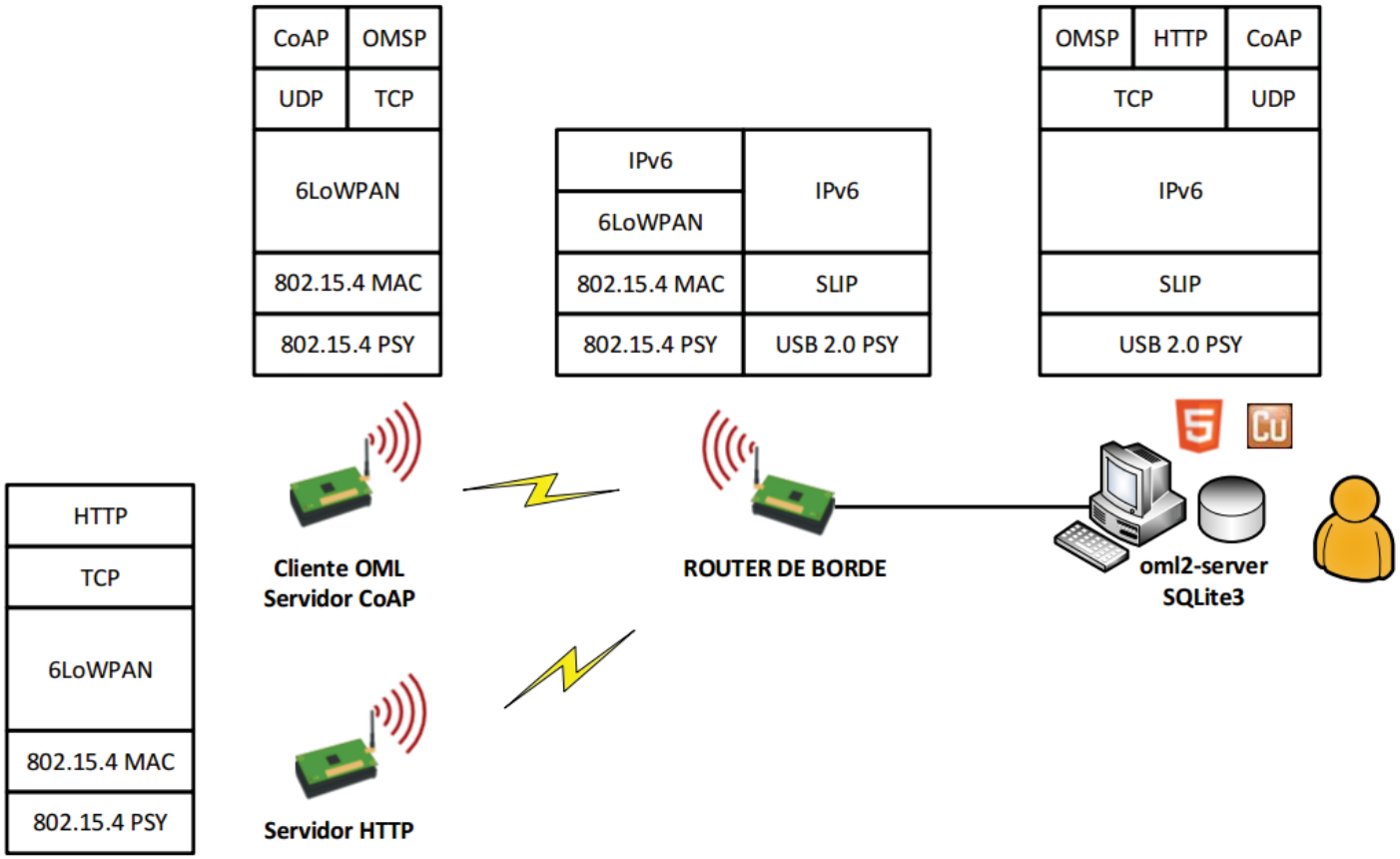
- Escenario



- Arquitectura



Demostración



```
protocol: 4
domain: EXPERIMENTO
start-time: 9
sender-id: 1
app-name:
schema: 0 _experiment_metadata subject:string key:string value:string
schema: 1 APLICACION_MEDIDAS Temperatura:long Ax:long Ay:long Az:long
content: text
28.1.0.322.-288.72.972
34.1.1.321.-288.90.918
    ...
```

	oml_tuple_id	oml_sender_id	oml_seq	oml_ts_client	oml_ts_server	Temperatura	Ax	Ay	Az
1	1	1	0	0	1409857629	322	-288	72	972
2	2	1	1	34	1409857634	321	-288	90	918
3	3	1	2	39	1409857640	322	-324	72	936
4	4	1	3	45	1409857645	322	-270	72	918
5	5	1	4	50	1409857651	321	-288	72	972
6	6	1	5	56	1409857656	322	-306	90	972
7	7	1	6	61	1409857662	323	-306	54	954
8	8	1	7	67	1409857667	322	-306	108	918
9	9	1	8	72	1409857673	322	-306	72	972
10	10	1	9	118	1409857718	322	-270	72	918
11	11	1	10	123	1409857724	322	-306	72	972
12	12	1	11	129	1409857729	323	-306	90	972
13	13	1	12	134	1409857735	323	-288	72	918
14	14	1	13	140	1409857740	323	-270	72	954
15	15	1	14	170	1409857771	322	-288	72	954
16	16	1	15	201	1409857801	323	-324	90	918

Conclusiones

- 6LoWPAN permite la conectividad extremo a extremo entre los nodos de una red de sensores e Internet
- Contiki OS ofrece una versión estable del protocolo 6LoWPAN
- Los servicios basados en *tunneling* ofrecen soluciones temporales al lento despliegue que está teniendo el protocolo IPv6
- CoAP presenta mayor rendimiento que HTTP. Sin embargo, los navegadores no lo traen implementado de facto
- Desarrollo de OML sobre una red 6LoWPAN
- Colaboración en el desarrollo de una práctica docente de la asignatura Redes No Convencionales (4º GITT)

Líneas futuras

- Escalabilidad del sistema aumentando el número de nodos (tanto *host* como *routers* de borde)
- Comprobar la heterogeneidad tanto a nivel *software* como *hardware* con soluciones ya existentes en producción (e.g SmartSantander)
- La inclusión de la solución 6LoWPAN en SmartSantander permitiría el análisis del consumo energético de los nodos
- Incorporar mecanismos de seguridad al sistema y analizar la influencia en el rendimiento global
- Adaptación del sistema basado en OML utilizando el protocolo de transporte UDP y comparar con el desarrollado en este trabajo

Provisión de servicios de la Internet de las Cosas sobre redes de sensores basadas en 6LoWPAN

Proyecto fin de carrera – Ingeniería de Telecomunicación

Autor: Óscar Vadillo Gutiérrez

Dirección: Jorge Lanza Calderón