UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Análisis y mejoras de seguridad a una aplicación prototipo en IoT
Autores: Pertini Bruno
Director: Lic. Francisco Javier Díaz y Lic. Paula Venosa
Codirector: -
Asesor profesional: Lic. Fernando López
Carrera: Licenciatura en sistemas

Resumen

En el presente trabajo de grado se analizan los requerimientos de seguridad en capa de enlace de una aplicación de IOT (Internet of Things, en castellano Internet de las Cosas) y se implementan mecanismos para resolver los mismos, de manera de garantizar autenticidad de las componentes e integridad de la información transmitida.

Se presentan dos alternativas para resolver estos requerimientos. La primera, integrada en el mismo sistema operativo sobre el cual se ejecuta la aplicación en cuestión; a pesar de resolver algunos problemas, conlleva limitaciones considerables que pueden llevar a corromper por completo la capa que se quiere securizar. Para la segunda opción se hace uso de una librería ya implementada para correr en dispositivos embebidos, que resuelve los problemas que presenta la anterior propuesta, y es fácilmente configurable en cada actor de la aplicación.

Luego de presentar la solución a adoptar, se demuestra su funcionalidad mediante el planteo de escenarios de prueba en un ambiente de simulación controlado, en donde se tienen en cuenta situaciones en las cuales los requerimientos de la aplicación a securizar no lleguen a cumplirse.

Palabras Claves

IoT, Contiki, Adaptivesec, Noncoresec, seguridad, autenticación, confidencialidad, reenvío, reinicio, mota, Cooja, clave, cifrado, MIC.

Conclusiones

Se obtuvieron los requerimientos de seguridad de una aplicación de IOT. Para resolverlos, se barajaron dos alternativas, eligiendo una y demostrando que la misma efectivamente resuelve los requerimientos de seguridad de la red en cuestión.

Se propone además una alternativa para que la solución utilizada sea capaz de securizar también la mota que cumple funciones de border router.

Trabajos Realizados

Se presentaron en primer lugar los principales conceptos y protocolos de IoT, con un enfoque descendente por las capas de su arquitectura. Se mostraron los distintos niveles de seguridad del estándar 802.15.4, detallando los servicios que cada uno implementa. Se establecieron los requerimientos de seguridad en capa de enlace de una aplicación IOT. Se presentaron dos enfoques para resolver los requerimientos planteados, descartando uno por los problemas que el mismo acarrea.

Trabajos Futuros

Tres propuestas toman protagonismo:

- La integración de un protocolo de capa de aplicación para la gestión de mensajes.
- Implementar una alternativa para que la aplicación a tratar tenga salida a Internet, asegurando a la vez a la mota que cumple funciones de border router.
- Implementar el esquema de predistribución de clave de a pares.

Fecha de la presentación: Octubre 2017

Resumen

En el presente trabajo de grado se analizan los requerimientos de seguridad de una aplicación de IOT (Internet of Things, en castellano Internet de las Cosas) y se implementan mecanismos para resolver los mismos, de manera de garantizar autenticidad de las componentes e integridad de la información transmitida.

Se acota el análisis a la capa de enlace de la aplicación. Una vez identificados los requerimientos, se presentan dos alternativas para resolverlos. La primera, la cual viene integrada en el mismo sistema operativo sobre el cual se ejecuta la aplicación en cuestión; a pesar de resolver algunos problemas, conlleva limitaciones considerables que pueden llevar a corromper por completo la capa que se quiere securizar.

Para la segunda opción se hace uso de una librería ya implementada para correr en dispositivos embebidos, que resuelve los problemas que presenta la anterior propuesta, y es fácilmente configurable en cada actor de la aplicación.

Luego de presentar la solución a adoptar, se demuestra su funcionalidad mediante el planteo de escenarios de prueba en un ambiente de simulación controlado, en donde se tienen en cuenta situaciones en las cuales los requerimientos de la aplicación a securizar no lleguen a cumplirse.

Agradecimientos

A mis papás. Sin su ayuda hubiera sido imposible enfocarse en la carrera y mucho menos terminarla.

A Paula y a Fernando por estar siempre a disposición desde que empezamos.

A la Facultad, a la que espero algún día devolverle todo lo que me dió.

A todos los profesores, ayudantes y compañeros que me dieron una mano, ya sea en prácticas,parciales o en finales.

Índice general

Introducción **9**
Objetivo 9
Motivación 9

IoT: conceptos básicos **10**
Redes de sensores inalámbricos (WSN, Wireless Sensor Network) 13
Roles en una WSN 15
RFID 16
Sistemas operativos embebidos 16
Contiki OS 16
TinyOS 19
Protocolos de IoT 19
IEEE 802.15.4 20
IPV6 25
6LoWPAN 29
RPL 32
MQTT 33
COAP 34

Seguridad en IoT **35**
Aspectos de seguridad en aplicaciones IoT 35
Seguridad en la capa de enlace 36
Esquemas de predistribución de clave 41

Enfoque utilizado en Contiki 42
Enfoque a utilizar en este trabajo 43
Estableciendo parámetros de configuración 44
Estableciendo claves de sesión 45
Asegurando frames broadcast: el protocolo EBEAP 46
Manejo de la movilidad de las motas 47

Aplicación a securizar **47**
Presentación 47
Déficits de seguridad en la capa a tratar 50

Solución propuesta **51**
Aplicando Adaptivesec a WENU 52
Escenarios de prueba en un ambiente simulado 56
Escenario 1: prueba de autenticación de las motas 57
Escenario 2: prueba del cifrado de los mensajes enviados 60
Escenario 3: reinicios de las motas y ataques anti replay 63
Implementación 69

Conclusiones **70**
Aportes 72
Trabajos futuros 72

Apéndice **73**
Tabla 1: notaciones 73
Tabla 2: información almacenada por un nodo u por cada vecino v 74

Bibliografía **75**

Índice de Figuras

Evolución de la cantidad de dispositivos conectados a IoT. 10
Arquitectura en capas de IoT. 13
Mota Zolertia Z1. 14
Ubicación de IEEE 802.15.4 en la pila de protocolos. 20
Campos en el encabezado fijo del frame de datos del protocolo IEEE 802.15.4. 21
Modos de direccionamiento en IEEE 802.15.4. 22
Frame de datos 802.15.4. 22
Frame de reconocimiento 802.15.4. 23
Frame de comando 802.15.4. 23
Frame faro 802.15.4. 23
Topología en estrella. 24
Topología par a par. 25
Cabecera IPv6 básica 26
Dirección IPv6. 29
Pilas de protocolos IP y 6LoWPAN. 30
Estructura de paquete 6LoWPAN de un primer fragmento FRAG1 y los subsecuentes fragmentos FRAGN. 32
DODAG RPL. 33
Arquitectura CoAP y MQTT. 35
AES-CTR. 38
AES-CBC-MAC. 38
AES-CCM. 38

Variantes de seguridad soportadas por 802.15.4. 39
Formato de una entrada ACL. 41
Formato de la entrada xi para el algoritmo block cipher para los tipos AES-CTR y AES-CCM. 41
Esquema de conectividad y servicios implementados en WENU. 50
Configuración inicial de Adaptivesec para su aplicación en WENU. 54
Salida de las motas dispuestas para la primer aplicación de Adaptivesec. 54
Datagrama UDP asociado a la aplicación servreg-hack. 55
Configuraciòn de Adaptivesec en el escenario 1. 58
Estado inicial de la red del escenario 1. 59
Salida estándar de las tres primeras motas en el escenario 1. 59
Segundo estado de la red del escenario 1. 61
Clave compartida de la mota cuatro, la cual difiere de la del resto. 61
Mota 4 tratando de entablar comunicaciones con la mota receptora. 61
Mensaje sin cifrar desde un nodo emisor (en este caso la mota 3) al receptor. 62
Archivo de configuración de proyecto para el escenario 2. 62
Paquete de datos medidos cifrado. 63
Formato del nonce en Adaptivesec. 63
Configuración de seguridad de capa de enlace para el escenario 3. 64
Motas receptora (1) y sensora (2) para el escenario 3. 65
Salida de las motas en lo primeros tres minutos de simulación. Escenario 3. 65
Mota 1 rechazando frames reenviados de la mota 2. 66
Configuración de seguridad de capa de enlace (Adaptivesec) para el escenario 3. 67
Salida de las motas en lo primeros tres minutos de simulación utilizando Adaptivesec. 68
Mota receptora eliminando a la mota sensora como vecino permanente. 68
Configuración final de Adaptivesec en WENU. 70

Introducción

Objetivo

Analizar los requerimientos de seguridad de una aplicación de IOT (Internet of Things, en castellano Internet de las Cosas) e implementar mecanismos para resolver los mismos de manera de garantizar autenticidad de las componentes e integridad y confidencialidad de la información transmitida.

Motivación

El surgimiento de IoT propició la aparición de nuevas tecnologías que se encargan conjuntamente de dar el soporte necesario para que este modelo se pueda mantener comunicado adecuadamente. Las mismas incluyen, entre otras cosas: protocolos para la comunicación (como IEEE 802.15.4, Bluetooth Low Energy, RPL, 6Lowpan, CoAP, MQTT); sistemas operativos embebidos (como Contiki, TinyOS, Riot-OS), que son los que se encuentran integrados en el hardware de los dispositivos y aplicaciones que hacen uso de los recursos disponibles. Las aplicaciones que encontramos en la mayoría de los casos son desarrolladas para cumplir con un propósito específico, ya sea para sensar algún atributo o característica del ambiente en donde se encuentran instaladas; o para recibir un dato determinado proveniente de otro objeto de la red, y actuar en consecuencia.
Si bien la aparición de Internet de las Cosas supone un cambio drástico en las redes existentes actuales, permitiendo así conectar todo tipo de objeto doméstico a ellas, su desarrollo no ha requerido grandes cambios en la estructura de Internet, ya que varios protocolos de la pila TCP/IP se reutilizan en IoT (IPv6,UDP,TCP,HTTP,entre otros). Al igual que en Internet, las aplicaciones, a menudo, tienen la responsabilidad de asegurar la confidencialidad, integridad y autenticidad de los datos que almacenan y envían entre las diferentes partes. En consecuencia, debe proveerse seguridad tanto en las diferentes componentes de las mismas como en la comunicación entre las mismas, teniendo en cuenta las diferentes capas de red y la sensibilidad de los datos que las mismas manejan.

IoT: conceptos básicos

IoT (“Internet of things”, Internet de las cosas en español) se define como una evolución de Internet en la cual los objetos que son utilizados en la vida cotidiana se encuentran conectados en red, enviando y recibiendo datos. También se puede definir como la interfaz entre el mundo físico y el digital, que permite obtener información de, y controlar, a los objetos mencionados anteriormente. 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 [7].

A pesar de que la definición de “cosas” ha cambiado a medida que la tecnología fue evolucionando, el objetivo principal de realizar un sensado de información sin la ayuda de la intervención humana se mantiene. Esto llevó a una evolución radical de la Internet actual en una red de objetos interconectados que no solo recolecta información del ambiente e interactúa con el mundo físico, sino que también utilice los estándares existentes de internet para proveer servicios para la transferencia de información.

Como muestra la figura 1, IoT conectará 50 billones de objetos inteligentes en el 2020, cuando la población mundial alcance los 7.600 millones de habitantes.[2]

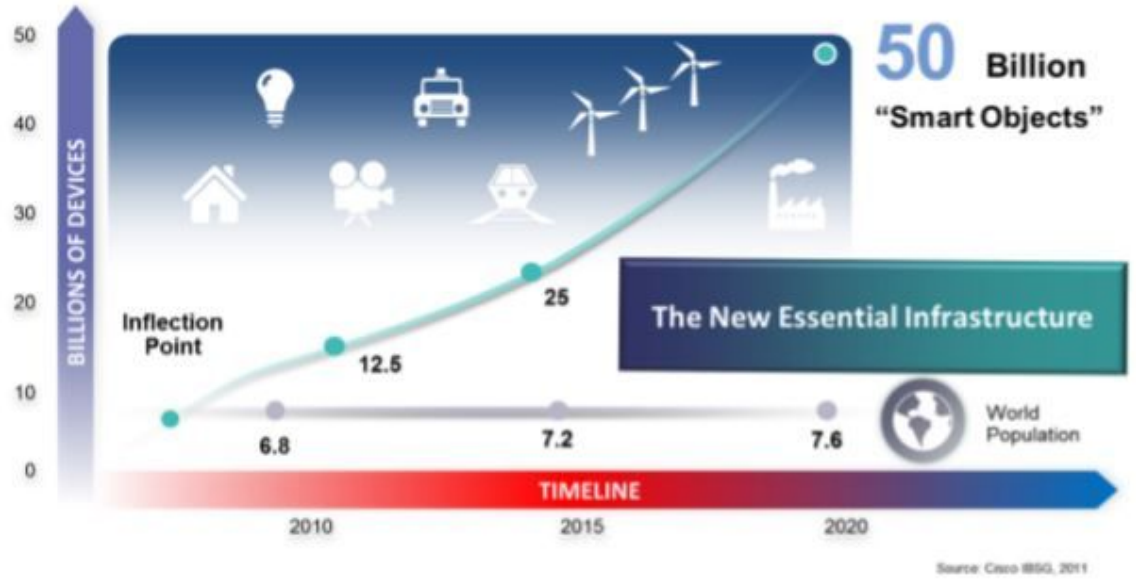


Figura 1. Evolución de la cantidad de dispositivos conectados a IoT.

Dentro del amplio rango de aplicaciones de IoT, podemos citar algunos ejemplos concretos:

- Una heladera que avise al supermercado con las cosas que se necesite y puedan ser llevadas a la casa de uno, o que avise cuando un producto está vencido.
- Conocer en tiempo real las necesidades de las plantas de la casa.
- Encender un electrodoméstico antes de llegar a casa.
- Pulseras que cuando se salga a correr informen los signos vitales.

El ámbito de aplicación de IoT no está limitado solo a la vida personal. Construir una red de sensores dentro de un ambiente en el cual se pretende sensar o medir un atributo, permite implementar IoT en diversos escenarios:

Ejemplo 1: Transporte/Logística
La sincronización contínua de información sobre el flujo de materiales, y el seguimiento y rastreo en tiempo real de objetos permite mejorar no sólo la cadena de suministro, sino también el posicionamiento global y la identificación automática de las mercancías. También gracias a IoT se aumenta la eficiencia energética y por lo tanto se disminuirá el consumo de energía.

Ejemplo 2: Casas Inteligentes
Afecta principalmente a tres aspectos: el uso de recursos (ahorro de agua y consumo de energía), seguridad (la detección de robo, incendio o la entrada no autorizada) y confort. El objetivo es lograr mejores niveles de confort mientras se disminuye el gasto total.

Ejemplo 3: Ciudades Inteligentes
Las aplicaciones IoT que podemos encontrar actualmente en las Ciudades Inteligentes o Smart Cities son variadas como control de tráfico, alumbrado inteligente, etc.

Ejemplo 4: Fábrica Inteligente
En una cadena de suministro global, las empresas pueden realizar un seguimiento de todos sus productos por medio de las etiquetas de identificación por radiofrecuencia (RFID). Como consecuencia, reducen sus gastos operativos y mejoran su productividad debido a una mayor integración con la planificación de recursos empresariales y de otros sistemas. Además, el mantenimiento de la maquinaria se ve facilitado por sensores conectados, lo que permite la monitorización en tiempo real del estado y el rendimiento de los equipos de la fábrica.

Componentes, funcionamiento y tecnologías de IoT

IoT es la nueva infraestructura esencial que conectará en 2020 cincuenta billones de objetos inteligentes. Esta se construye alrededor de una arquitectura multi-capa donde los objetos inteligentes son usados para proveer diferentes servicios a través de las cuatro capas principales detalladas en la figura 2: una capa de dispositivos, una capa de red, una capa de soporte y una capa de aplicación.
En la capa de dispositivos se pueden encontrar sensores, actuadores, dispositivos RFID y gateways usados para recolectar las mediciones de los sensores para un posterior procesamiento, mientras que la capa de red provee las capacidades de transporte y red necesarias para encaminar la información de IoT hacia diferentes lugares. La capa de soporte es una capa middleware que sirve para ocultar la complejidad de las capas más bajas a la capa de aplicación, y provee servicios genéricos y específicos, como almacenamiento en diferentes formas (sistemas de administración de bases de datos y/o sistemas cloud computing) y otros servicios.

En los inicios de la revolución de Internet, los usuarios estaban maravillados con la posibilidad de contactar gente e intercambiar información a través del mundo y a través de las zonas horarias. El próximo paso en esta revolución tecnológica es conectar objetos inanimados a una red de comunicación. Esta visión subyacente a IoT permite que la información sea accedida no sólo en cualquier momento y en cualquier lugar, sino también por cualquier cosa. Esto es posible usando redes de sensores inalámbricos y etiquetas RFID.
En lo que respecta al presente trabajo, y para una mejor comprensión, las capas explicadas en esta sección serán llamadas de igual manera que en el modelo TCP/IP. Puntualmente: la capa de aplicación mantiene su nombre; la capa de soporte de servicios y aplicaciones se llamará capa de transporte; la capa de red mantendrá su nombre y la capa de dispositivos se llamará capa de enlace.

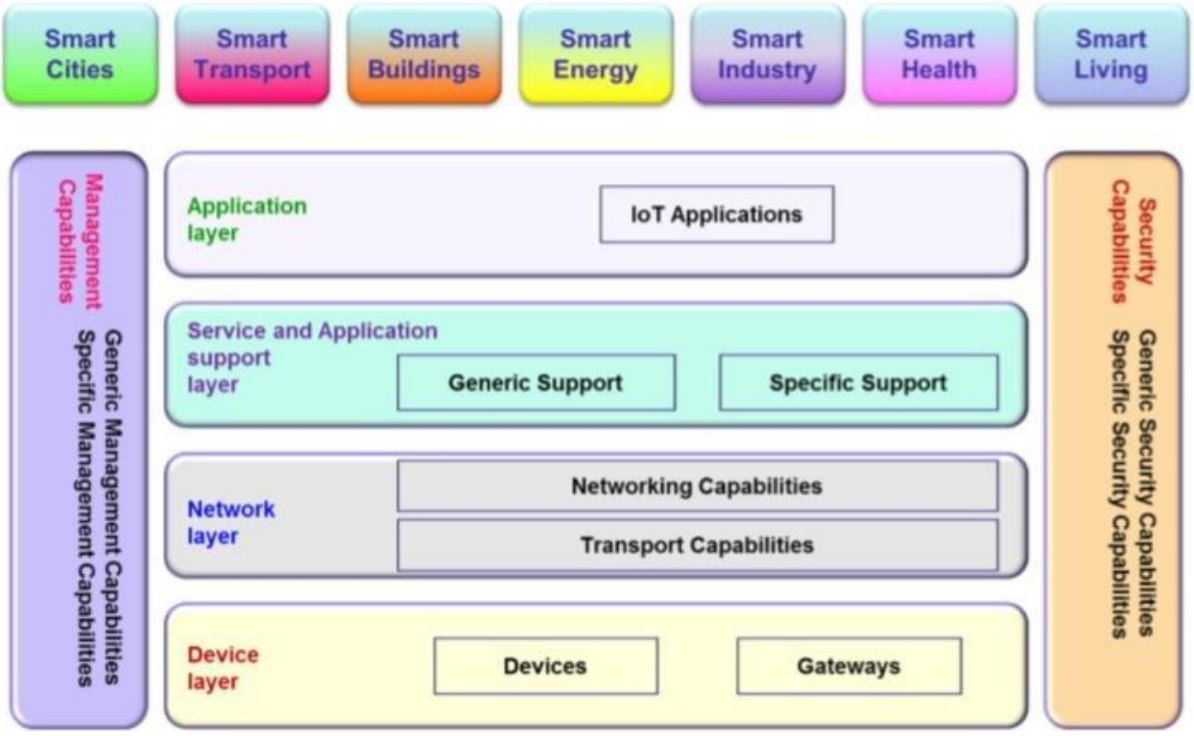


Figura 2. Arquitectura en capas de IoT.

Redes de sensores inalámbricos (WSN, Wireless Sensor Network)

También denominadas redes personales inalámbricas o redes de área personal (PAN), es una red autoconfigurable de pequeños nodos sensores (llamados motas) que se comunican entre sí mediante señales de radio, y desplegada en cantidad

para medir el mundo real. Las motas son pequeñas computadoras con una funcionalidad extremadamente básica (figura 3). Consisten de: una unidad de procesamiento con poder computacional limitado y una memoria limitada, un dispositivo de comunicación por radio, una fuente de energía y uno o más sensores. Vienen en diferentes formas y tamaños, dependiendo del uso que se les vaya a dar. Pueden ser muy pequeñas, si por ejemplo se desplegarán varias y se necesita tener un pequeño impacto visual. Además, pueden tener una batería recargable como fuente de energía. La integración de estos dispositivos minúsculos y ubicuos en los escenarios más diversos asegura un amplio rango de aplicaciones.
En una aplicación típica IoT, una WSN es diseminada en una región donde esté destinada a recolectar datos a través de sus nodos sensores. Estas redes proveen un puente entre el mundo físico y el virtual, y tienen habilidades sin precedentes para observar y entender fenómenos del mundo real de gran escala con una fina resolución espacio-temporal.

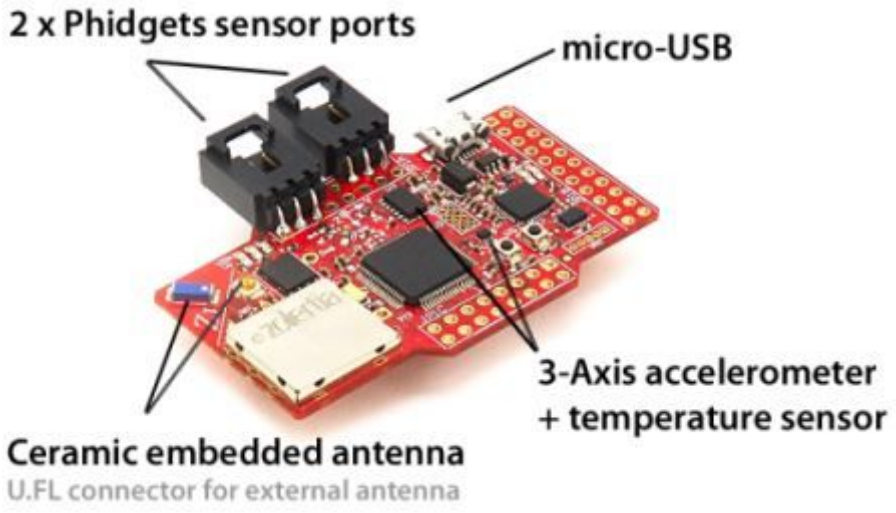


Figura 3. Mota Zolertia Z1.

Todas las motas están compuestas de cinco elementos principales:

- Procesador: la tarea de esta unidad es procesar localmente información sensada por la mota a la que pertenece e información sensada por otras motas. En el presente, los procesadores son limitados en cuanto a poder computacional, pero dada la ley de Moore [39], estos serán más pequeños, más poderosos y consumirán menos energía. El procesador puede correr en diferentes modos: el modo sueño, utilizado la mayoría del tiempo para ahorrar energía, el modo ocioso cuando pueden

llegar datos de otras motas, y el modo activo utilizado cuando se censan datos, o cuando se envían a, o reciben de, otras motas.

- Fuente de energía: las motas pueden instalarse en varios ambientes, incluyendo regiones remotas y hostiles, por lo que deben utilizar poca energía. Típicamente tienen poco almacenamiento energético, por lo que los protocolos de red deben enfatizar el ahorro de energía. También deben tener incorporados mecanismos para permitir al usuario final la opción de prolongar el tiempo de vida de la red. Además las motas pueden estar equipadas de mecanismos de generación de energía, como celdas solares, por lo que pueden dejarse desatendidas por meses o años. Las fuentes de poder más usuales son las baterías recargables, los paneles solares y capacitores.

- Memoria: utilizada para almacenar tanto programas como datos. En las motas Zolertia Z1, es posible encontrar una memoria RAM de 8KB y una memoria Flash de 92 KB [20].

- Radio: los dispositivos presentes en las WSN incluyen un radio inalámbrico de baja tasa y de corto alcance. Las tasas típicas están entre 10-100 kbps, y el rango es menor a 100 metros. La comunicación por radio usualmente es la tarea que más energía consume, por lo que es imperativo incorporar técnicas de ahorro de energía.

- Sensores: las redes de sensores consisten de varios tipos de sensores diferentes capaces de monitorear una amplia variedad de condiciones ambientales, como por ejemplo el medido de temperatura, humedad, luz, presión, niveles de ruido, aceleración, humedad del suelo, etc. Algunas aplicaciones requieren un modo de multi sensado, por lo que cada dispositivo puede tener varios sensores incorporados [2].

Roles en una WSN

1. Los nodos sensores son utilizados para sensar sus alrededores y transmitir los valores leídos a un nodo “sink”, también llamado “estación base”. Típicamente están equipados con diferentes tipos de sensores.

2. Nodos “sink” o "estaciones base". Recolectan los valores leídos de los sensores de otros nodos, y los envían hacia un gateway con el que se encuentran

directamente conectados, para un futuro procesamiento. Están equipados con capacidades mínimas de procesamiento y comunicación, pero no tienen capacidades para medir.

3. Los actuadores son dispositivos utilizados para controlar el ambiente, basándose en valores leídos por un sensor o por otras entradas. Pueden tener la misma configuración que una mota, pero equipados además con cualidades de control, por ejemplo para apagar o prender una luz.

4. Los gateways conectados a los nodos sink están conectados a una fuente de alimentación estable, dado que consumen mucha energía. Estas entidades son dispositivos de cómputo normales como laptops, notebooks, desktops o teléfonos celulares. Sin embargo, no están equipados con capacidades para medir. [2]

RFID

Es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transpondedores o tags RFID. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las tecnologías RFID se agrupan dentro de las denominadas Auto ID (*automatic identification*, o identificación automática). Son unos dispositivos pequeños, similares a una pegatina, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID [23].

Sistemas operativos embebidos

Contiki OS

Es un sistema operativo de código abierto desarrollado para usar en ordenadores de 8 bits y sistemas integrados sobre microcontroladores, incluyendo nodos de redes de sensores. Está diseñado para sistemas embebidos con escasa memoria. Una configuración típica de Contiki consta de 2 KB de RAM y 40 KB de ROM. Cuenta con un núcleo orientado a eventos sobre el cual los programas pueden ser cargados y descargados de forma dinámica en tiempo de ejecución. Los procesos en Contiki usan protothreads, un mecanismo de abstracción ideado para proporcionar un estilo de programación secuencial sobre el núcleo orientado a eventos. También soporta multithreading apropiativo opcional por proceso. La comunicación entre procesos se

realiza mediante la técnica de pasaje de mensajes, la cual está implementada mediante el sistema de eventos del núcleo. Tiene un subsistema GUI opcional, con soporte de gráficos para terminales locales, terminales virtuales en red mediante VNC o sobre Telnet. Incluye una pila ligera TCP/IP y la pila Rime, que está diseñada especialmente para comunicaciones inalámbricas de baja potencia, y cuenta con un amplio rango de primitivas de comunicación. También soporta IPv6, junto con protocolos como RPL y 6LoWPAN. Funciona en una variedad de plataformas, desde microcontroladores embebidos, como el MSP430 y el AVR, a viejas computadoras domésticas. El tamaño del código está en el orden de los kilobytes y el uso de la memoria puede configurarse para que sea de sólo unas decenas de bytes. Está escrito en el lenguaje de programación C y usa una licencia tipo BSD [21].

Procesos y protothreads

Contiki comparte el hardware entre las diferentes aplicaciones mediante el uso de procesos y protothreads. Presenta dos contextos de ejecución: cooperativo y apropiativo. El primero corresponde a una ejecución regular, donde las funciones son llamadas y ejecutadas secuencialmente en el micro controlador. El contexto apropiativo corresponde a la ejecución de interrupciones, provocadas por entrada/salida o por timers. Cuando se ejecutan en un contexto cooperativo, todas las tareas se ejecutan hasta que finalizan. Las tareas apropiativas pueden interrumpir a las cooperativas en cualquier momento.
Todos los programas en Contiki son procesos, por lo que si se quiere crear una nueva aplicación se debería crear un nuevo proceso para ejecutarla. Dado que se ejecutan en el contexto cooperativo, el scheduler de procesos de Contiki no los interrumpe, y por ende se ejecutan hasta el momento en que cede el micro controlador al siguiente proceso.
Cuando se ejecuta código apropiativo, el contexto cooperativo se detiene y la interrupción es tratada. El contexto cooperativo es retomado desde el punto donde había sido interrumpido después de la finalización de la ejecución apropiativa. En Contiki es posible gestionar la ejecución de un pedazo de código apropiativamente, lo que significa que será ejecutado sin importar qué aplicación esté corriendo. Esto es útil para implementar algoritmos que requieran un timing preciso.

Un proceso en Contiki consiste de dos partes: un bloque de control de proceso (PCB) y un thread de proceso. El bloque de control de proceso contiene información acerca de cada proceso, como el estado, un puntero al thread del proceso, y el nombre textual del proceso. Un thread del proceso contiene el código del proceso.

Un bloque de control de proceso es una estructura interna utilizada por el scheduler para invocar al thread del proceso y comenzar su ejecución. La definición de un bloque de control de proceso se realiza usando la siguiente macro:

PROCESS(nombreVariable, nombre);

- nombreVariable - El nombre de la estructura del proceso.
- nombre - La representación en string del nombre del proceso.

Un thread de proceso contiene el código de ejecución del proceso. Por lo tanto, cada proceso en Contiki puede tener solamente un thread de ejecución. Los threads de Contiki no son threads auténticos, sino que se los denomina protothreads. Estos son threads optimizados para sistemas operativos dirigidos por eventos. Se declaran usando la siguiente macro:

PROCESS_THREAD(nombre, ev, data)

- nombre - El nombre de la estructura del proceso.
- ev - Identificador de evento enviado al thread durante la invocación.
- data - Puntero a los datos pasados al thread durante la invocación.

Finalmente, un concepto importante es el de autoejecución de procesos. Es un mecanismo donde los procesos pueden comenzar automáticamente, incluso cuando el sistema es booteado, o cuando un módulo que contiene los procesos es cargado. Estos procesos se mantienen en una lista, que es usada por el módulo de autoejecución para ejecutar los procesos. Estos se ejecutan en el orden en el que aparecen en la lista.
En Contiki, el mecanismo de autoejecución es la forma más común mediante la cual los procesos de usuario son ejecutados. Para especificar los procesos que deberían autoejecutarse, el usuario debe utilizar la siguiente macro:

AUTOSTART_PROCESSES(&name1, &name2, ...)

- name1, name2, ... - Los nombres de los procesos para autoejecutarse. [42]

TinyOS

Es un sistema basado en componentes y apto para dispositivos pertenecientes a redes de sensores inalámbricos (WSNs). Está escrito en el lenguaje de programación nesC, como un conjunto de tareas colaborativas y procesos. Su desarrollo comenzó como una colaboración entre las universidades de California y Berkeley; es open-source y tiene una licencia BSD. Además, ha ido creciendo dentro de un consorcio internacional, la alianza TinyOS. Las aplicaciones de TinyOS están escritas en el lenguaje de programación nesC, un dialecto del lenguaje C optimizado para las limitaciones de memoria que presentan las redes de sensores. Sus herramientas suplementarias están principalmente escritas en Java y shell-script. Las librerías asociadas y herramientas, como el compilador de nesC, están en su mayoría escritas en C. Los programas de TinyOS se componen de componentes de software, algunos de los cuales presentan abstracciones de hardware. Los componentes se conectan unos a otros utilizando interfaces. TinyOS provee interfaces y componentes para abstracciones comunes como comunicación de paquetes, ruteo, sensado, accionamiento y almacenamiento. TinyOS es completamente no bloqueante: tiene una pila de llamada. Por lo tanto, todas las operaciones de entrada/salida que tarden más que unos pocos cientos de microsegundos son asincrónicas y tienen regreso. Para permitir que el compilador nativo optimize mediante límites a las llamadas, TinyOS hace uso de las características de nesC para relacionar estas llamadas, conocidas como eventos, estáticamente [24].

Protocolos de IoT

En esta sección se pretende brindar una explicación de los protocolos más importantes de cada capa de IoT, empezando por la capa de enlace hasta la capa de aplicación.

En primer lugar, se abordará el protocolo IEEE 802.15.4, del cual hacen uso los dispositivos IoT utilizados en el desarrollo de esta tesina para las comunicaciones en capa de enlace. Se presentarán conceptos importantes de este estándar que aparecerán a lo largo del trabajo. En cuanto a la capa de red, IPv6 será el protocolo a tratar, subrayando sus diferencias con IPv4 y su importancia para dar soporte a las redes de sensores inalámbricos. Cumpliendo el rol de una capa de adaptación, el protocolo 6LoWPAN implementa servicios como la fragmentación y el reensamblado de frames, de forma similar a como IPv6 lo hace con los paquetes; pero teniendo en cuenta las limitaciones de los dispositivos embebidos. Dadas las características del

tipo de redes que componen IoT, es necesario también un protocolo de ruteo que las tenga en cuenta. El utilizado en este trabajo será RPL. Por último, el estudio de la capa de aplicación en este trabajo engloba a dos protocolos muy utilizados en IoT, MQTT y CoAP.

IEEE 802.15.4

IEEE 802.15.4 es un estándar para comunicaciones inalámbricas. En concreto, define las capas físicas y de control de acceso al medio para las redes personales inalámbricas. Hoy en día es un estándar muy popular utilizado con una gama de aplicaciones muy amplia. Su objetivo es proveer comunicaciones de corto alcance y bajo rendimiento para dispositivos embebidos. Varios otros estándares utilizan IEEE 802.15.4 como su capa física y de enlace, incluyendo 6LoWPAN, ISA100 y ZigBee. El compartimiento del canal se consigue mediante el protocolo de múltiple acceso sensible al contexto (CSMA), y los reconocimientos son usados para proveer confiabilidad. Utiliza direccionamiento de 64 y de 16 bits con capacidades unicast y broadcast. La capa de enlace puede correr en dos modos: beaconless y beacon-enabled (en español, modo sin faro y modo con faro). El modo beaconless usa un acceso al canal puramente CSMA y opera de manera similar a IEEE 802.11, sin reservas de canal. El modo Beacon-enabled es más complejo, con una estructura superframe y la posibilidad de reservar slots de tiempo para datos críticos [8].

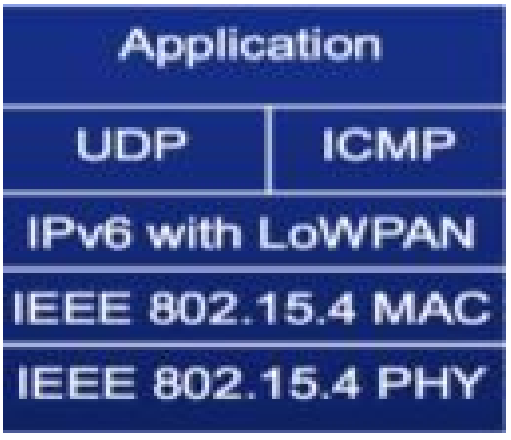


Figura 4. Ubicación de IEEE 802.15.4 en la pila de protocolos.

En una red IEEE802.15.4 pueden participar dos tipos de dispositivos: un dispositivo de función completa (FFD, full-function device) y un dispositivo de función reducida

(RFD, reduced-function device). El primero puede encaminar mensajes, en cuyo caso se le denomina coordinador (coordinador de la PAN si es el responsable de toda la red y no sólo de su entorno). El segundo está orientado a aplicaciones extremadamente simples, por ejemplo interruptores de luz, y no necesita enviar gran cantidad de datos, además sólo se asocia con un FFD a la vez. Por ende, puede implementarse usando recursos y capacidades de memoria mínimas.

Las principales tareas del coordinador son:

- Administra una lista de los dispositivos asociados.
- Aloca direcciones cortas a sus dispositivos. Cuando estos se asocian a un coordinador, solicitan una dirección de 16 bits para las comunicaciones subsecuentes entre el dispositivo y el coordinador.
- En el modo faro, regularmente transmite frames beacon (en español, mensajes faro).
- Intercambia paquetes de datos con dispositivos y con otros coordinadores.

El estándar define los siguientes tipos de frames:

- Frame de datos: encargado del transporte de los mismos. Comienza con un header de un byte que contiene un bit reservado y siete bits que representan la longitud del paquete restante, por lo que el tamaño del frame de datos entero varía de 1 a 128 bytes (incluyendo el encabezado). Luego, tres bytes se utilizan para la parte fija del encabezado de capa de enlace, seguido de partes variables, cuya presencia y longitud depende de los ajustes en la sección fija. La carga útil (o payload) sigue, posiblemente encriptada e incluyendo 4, 8 o 16 bytes adicionales para el chequeo de integridad del mensaje, seguida de un valor de 16 bits CRC (chequeo de redundancia cíclica, cyclic redundancy check) que se utiliza como la secuencia de chequeo del frame. Se presenta un resumen de los campos del encabezado fijo en la figura seis.

–	(reserved fields)
C	PAN ID compression
A	ACK request
P	Frame pending
S	Security enabled
ftype	Frame type (001 binary for data packets)
SAM	Source addressing mode
FV	Frame version (00 binary for compatibility with 2003 version, 01 binary for frames only compatible with 2006 version)
DAM	Destination addressing mode
Sequence	Sequence number (for ACK)
FCS	Frame check sequence

Figura 5. Campos en el encabezado fijo del frame de datos del protocolo IEEE 802.15.4.

El contenido de la sección denominada “direcciones” de la figura 7 depende de los valores del modo de dirección fuente, del modo de dirección destino y del flag C. Los campos SAM y DAM establecen longitudes de direcciones de 0, 16 o 64 bits. Para los últimos dos casos, la dirección está precedida de un identificador PAN de 16 bits, excepto que el flag C esté seteado:

00	Neither PAN identifier nor the address field is given
01	Reserved
10	Address field contains a 16-bit short address
11	Address field contains a 64-bit extended address

Figura 6. Modos de direccionamiento en IEEE 802.15.4.

Esto indica que los dos identificadores PAN son iguales y que por lo tanto se omite en el campo dirección fuente (SAM y DAM no pueden ser ambos cero , es decir al menos una dirección debe estar presente). Dependiendo de los valores de estos tres campos, la longitud total de la sección “direcciones” puede estar entre cuatro y veinte bytes.
Finalmente, el número de secuencia de un byte identifica el número de paquete, para los reconocimientos [11].

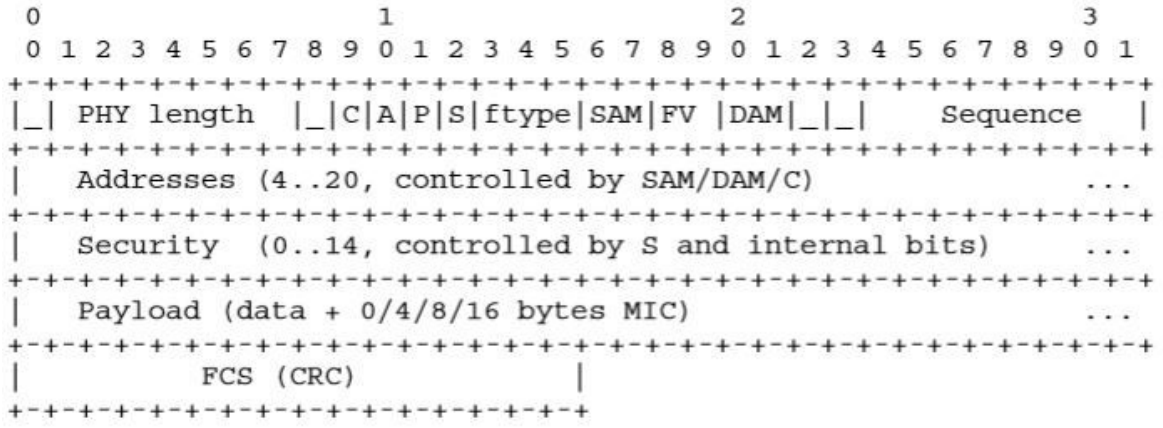


Figura 7. Frame de datos 802.15.4.

- Frame de reconocimiento: enviados por un receptor inmediatamente después de recibir exitosamente un frame de datos. El receptor lo envía sólo si el correspondiente paquete de datos no fue enviado a una dirección broadcast, y el emisor además lo solicitó. Su formato es simple: un campo flags de dos bytes similar al del paquete de datos, el número de secuencia de un byte y un CRC de dos bytes [3].

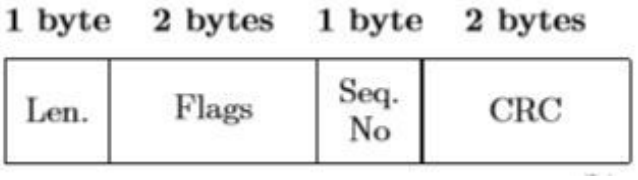


Figura 8. Frame de reconocimiento 802.15.4.

- Frame de comando: usado principalmente en el modo con faro, permite activar varios servicios de la capa MAC, tales como la asociación y desasociación de un coordinador y la gestión de las transmisiones sincronizadas. El campo ID de comando identifica al comando en cuestión que se ejecuta, mientras que el campo payload en este caso almacena al comando en sí [3].

Octets: 2	0/1	variable	variable	variable		1	variable	2/4
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	IE		Command ID	Content	FCS
				Header IEs	Payload IEs			
MHR				MAC Payload				MFR

Figura 9. Frame de comando 802.15.4.

- Frame faro: es un frame especial enviado por un coordinador de la PAN con el propósito de sincronizar con otras unidades. El modo faro ofrece ahorro de energía a partir de que las unidades pueden “dormir” hasta que son “despertadas” por los mensajes faro [3].

Octets: 2	1	4/10	variable	2	variable	variable	variable	2/4
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS Info	Pending address	Beacon Payload	FCS
MHR				MAC Payload				MFR

Figura 10. Frame faro 802.15.4.

Tipos de direcciones y topologías

Dependiendo de los requerimientos de la aplicación, una red IEEE 802.15.4 opera en cualquiera de las siguientes dos topologías: estrella o “par a par”. En la primera, la comunicación se establece entre los dispositivos y un controlador central, llamado coordinador de la PAN. Todos los dispositivos operando en la topología que sea tienen una dirección globalmente única, llamada dirección extendida, de 8 bytes. Además, a un dispositivo se le puede asignar una dirección corta de 2 bytes durante el proceso de asociación. Para las comunicaciones dentro de la PAN, el dispositivo utiliza la dirección extendida o la corta.
La topología “par a par” también presenta un coordinador de la PAN; sin embargo, difiere de la topología en estrella en cuanto a que cualquier dispositivo puede comunicarse con cualquier otro siempre y cuando cada uno esté en el rango del otro. Esto permite que se implementen formas de red más complejas, como una topología en malla. Además, esta topología permite que, a través de múltiples saltos, un mensaje pueda ser enviado de un dispositivo a cualquier otro de la red.

Cada PAN selecciona un identificador único. Este, permite las comunicaciones entre los dispositivos dentro de una red utilizando direcciones cortas, y además permite las transmisiones entre dispositivos de diferentes redes [11].

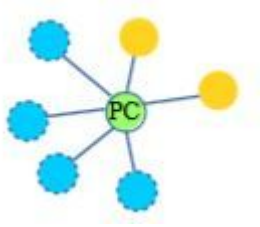


Figura 11. Topología en estrella.

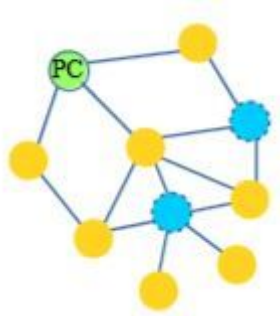


Figura 12. Topología par a par.

IPV6

A principios de la década de 1990, el Internet Engineering Task Force comenzó a desarrollar un sucesor para el protocolo IPv4. La principal motivación de esta iniciativa fue que se dieron cuenta que el espacio de direcciones de 32 bits estaba comenzando a agotarse, a causa de las nuevas subredes y nodos IP que estaban conectándose a Internet (a los que se les estaban asignando direcciones IP unicast) a una velocidad sobrecogedora. Para responder a esta necesidad, se desarrolló un nuevo protocolo IP, el protocolo IPv6. Los diseñadores de IPv6 también vieron aquí la oportunidad de ajustar y aumentar otros aspectos de IPv4, basándose en la experiencia acumulada sobre el funcionamiento de IPv4. Los cambios más importantes introducidos en IPv6 son:

1. Capacidades ampliadas de direccionamiento. IPv6 aumenta el tamaño de la dirección IP de 32 a 128 bits. De esta manera, se asegura que el mundo no se

quedará sin direcciones IP. Además, IPv6 ha introducido un nuevo tipo de dirección, denominado dirección anycast, que permite entregar un datagrama a uno cualquiera de un grupo de hosts.

2. Una cabecera de 40 bytes simplificada. Permite un procesamiento más rápido del datagrama IP.

3. Prioridad y etiquetado del flujo. IPv6 utiliza una definición bastante amplia de flujo. Los documentos RFC 1752 y RFC 2460 establecen que esto permite “etiquetar los paquetes que pertenecen a determinados flujos para los que el emisor solicita un tratamiento especial, como un servicio en tiempo real o una calidad de servicio no predeterminados”. La cabecera de IPv6 también tiene un campo de 8 bits para definir la clase de tráfico. Este campo, como el campo TOS de IPv4, se puede utilizar para dar prioridad a determinados datagramas de flujo, o se puede emplear para dar prioridad a datagramas de ciertas aplicaciones con respecto a datagramas de otras aplicaciones.

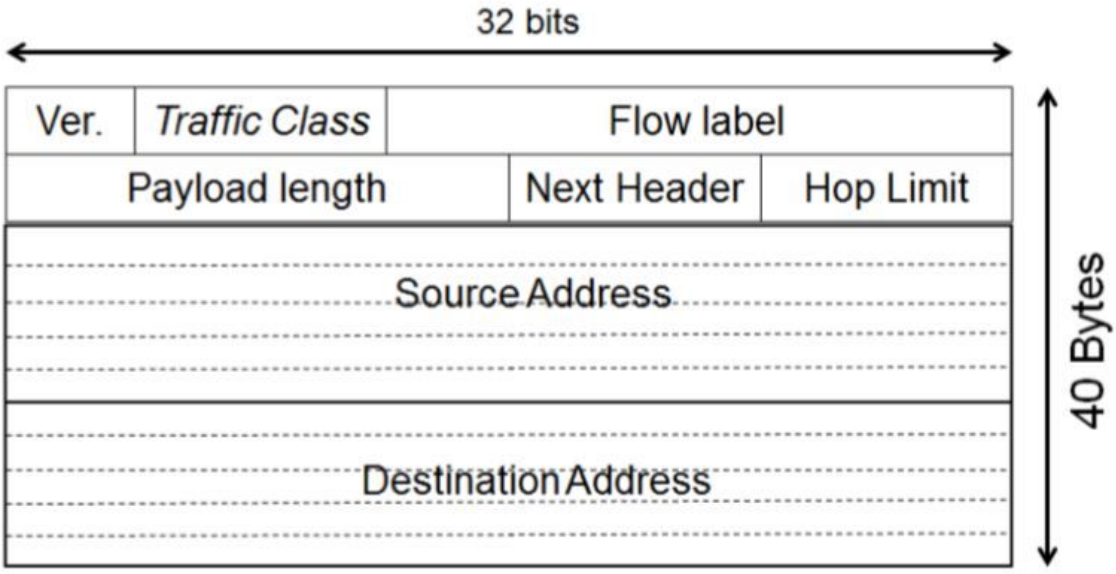


Figura 13. Cabecera IPv6 básica

La cabecera IPv6 tiene un tamaño fijo de 40 bytes, seguido de datos de capas superiores y, opcionalmente, por algunas cabeceras de extensión. Se pueden observar varios campos en la cabecera, incluyendo algunas mejoras con respecto a la cabecera IPv4:

- Se redujo el número de campos de 12 a 8.
- La cabecera tiene un tamaño fijo de 40 bytes, seguido de los 64 bits correspondientes al resto del paquete, lo que permite un procesamiento más rápido de los paquetes en los routers.
- El tamaño de las direcciones se incrementó de 32 a 128 bits.

Cada dirección IPv6 tiene 128 bits, lo que da un total de 2^{128} direcciones disponibles (aproximadamente 3.4×10^{38}, osea , 3.4 seguido de 38 ceros), mientras que IPv4 utiliza 32 bits para codificar cada una de las 2^{32} direcciones (4,294,967,296) disponibles.

El uso de direcciones de 128 bits otorga algunos beneficios:
- Provee muchas más direcciones, para satisfacer las necesidades actuales y futuras.
- Simplifica los mecanismos de autoconfiguración de direcciones.
- Una administración de direcciones más sencilla.
- Espacio para más niveles de jerarquía y para agregación de rutas.
- Habilidad para realizar IPsec punto a punto.

Las direcciones IPv6 se clasifican en las siguientes categorías:

- Unicast: usadas para enviar un paquete desde un origen hacia un solo destino.
- Multicast: usadas para enviar un paquete desde un origen hacia varios destinos.
- Anycast: usadas para enviar un paquete desde un origen hacia el destino más cercano a él.
- Reserved: grupos de direcciones para usos especiales, por ejemplo para ser usadas en documentaciones y ejemplos.
- Link-local: están siempre presentes en una interface IPv6 conectada a una red. Todas comienzan con el prefijo FE80::/10 y se utilizan para comunicarse con hosts en la misma red local. No pueden ser usadas para comunicarse con otras redes.
- ULA (Unique Local Address): comienzan con el prefijo FC00::/7. Destinadas a comunicaciones locales, no son ruteables en Internet, sólo se utilizan dentro de un ambiente más limitado.
- Global Unicast: Equivalentes a las direcciones IPv4 públicas, son únicas en todo Internet y pueden ser usadas para enviar un paquete de un destino a cualquier otro.

IPv6 posee características que facilitan ciertas cuestiones, como el direccionamiento global y la autoconfiguración de las direcciones de los hosts. Dado que IPv6 nos provee direcciones para unos cuantos cientos de años, podemos asignar una dirección unicast global prácticamente donde sea. [5]

La disponibilidad de una inmensa cantidad de direcciones ha permitido un nuevo mecanismo denominado autoconfiguración de direcciones sin estado (SLAAC, “stateless address autoconfiguration”) que no existía en IPv4. Una dirección IPv6 puede ser configurada de las siguientes formas:

- Estáticamente: la dirección IP es configurada manualmente. También se deben configurar otros parámetros, como el gateway a utilizarse en caso de enviar paquetes hacia fuera de la red.

- DHCPv6 (Dynamic Host Configuration Protocol for IPv6) [RFC3315]: se debe configurar un servidor dedicado que, luego de una breve negociación con el dispositivo, le asigne una dirección IP. DHCPv6 permite que los dispositivos IP sean configurados automáticamente, es por eso que se lo denomina autoconfiguración de direcciones con estado, ya que el servidor DHCPv6 mantiene el estado de las direcciones asignadas.

- SLAAC: autoconfiguración de direcciones sin estado [RFC4862] es un nuevo mecanismo introducido en IPv6 que permite configurar automáticamente todos los parámetros de red en un dispositivo IP, utilizando el router que le da conectividad a una red.
  La ventaja de SLAAC es que simplifica la configuración de dispositivos como sensores, cámaras o cualquier otro con bajo poder de procesamiento. No es necesario usar ninguna interfaz en el dispositivo IP para configurarlo. También simplifica la infraestructura de red necesaria para construir una red IPv6 básica, ya que se utiliza el mismo router mediante el cual se envían paquetes fuera de la red para configurar los dispositivos IP. Este router envía toda la información de configuración necesaria a sus hosts a través de un mensaje RA (Router Advertisement). Periódicamente, el router envía estos mensajes, sobre todo para facilitar que un host pueda enviar un mensaje RS (Router Solicitation) cuando su interface se conecta a la red. El router enviará inmediatamente un RA en respuesta al RS [5].

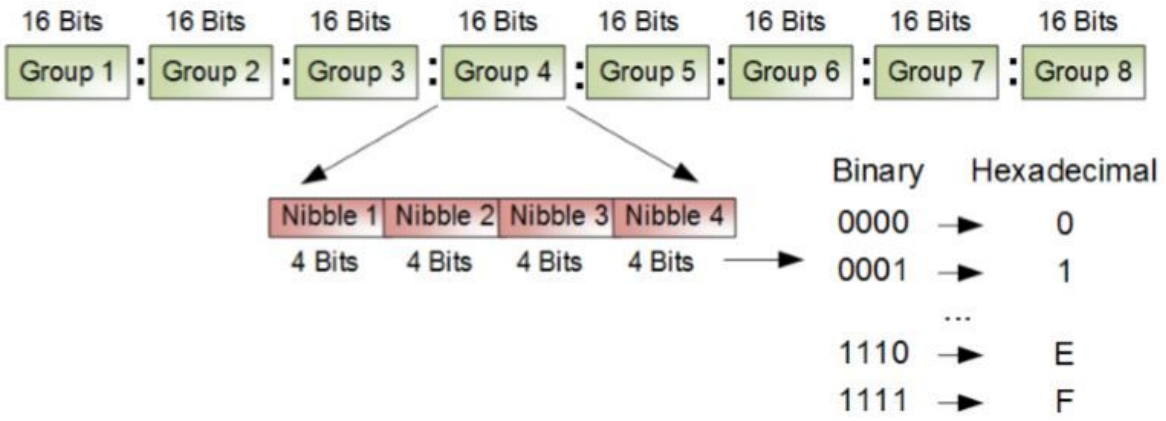


Figura 14. Dirección IPv6.

Las reglas de notación de IPv6 son:

- 8 grupos de 16 bits separados por ":".
- Notación hexadecimal.
- No sensible a mayúsculas.
- Los prefijos de red son escritos prefijo/longitud del prefijo, es decir, la longitud del prefijo indica el número de bits de la dirección que son comunes para el grupo.
- Los ceros de más a la izquierda dentro de cada grupo pueden ser eliminados.
- Uno o más grupos compuestos por todos ceros puede ser sustituído por "::". Esto puede ser hecho solo una vez.

6LoWPAN

Los estándares 6LoWPAN permiten el uso eficiente de IPv6 sobre redes de sensores inalámbricos, sobre dispositivos embebidos a través de una capa de adaptación, y la optimización de los protocolos relacionados [8]. Es una capa de adaptación para IPv6 sobre enlaces 802.15.4, cuya funciones principales son la fragmentación y reensamblado de paquetes, y la compresión de encabezados [7].

La figura 12 muestra la pila de protocolos IPv6 con 6LoWPAN en comparación con una típica pila de protocolos IP y las correspondientes cinco capas del modelo de Internet. Una simple pila de protocolos IPv6 con 6LoWPAN es casi idéntica a una pila IP, con las siguientes diferencias. En primer lugar 6LoWPAN sólo soporta IPv6,

para lo cual una pequeña capa de adaptación ha sido definida con el objetivo de optimizar IPv6 sobre IEEE 802.15.4. El protocolo de transporte más utilizado con 6LoWPAN es el protocolo UDP. El protocolo TCP no es utilizado por cuestiones de performance, eficiencia y complejidad. El protocolo de mensajes de control de Internet v6 (ICMPv6) es utilizado para el envío de mensajes ICMP echo, ICMP destino inalcanzable y mensajes de descubrimiento de vecinos.
Los protocolos de aplicación son usualmente específicos a la aplicación, a pesar de que más protocolos de aplicación estándares están apareciendo [8].

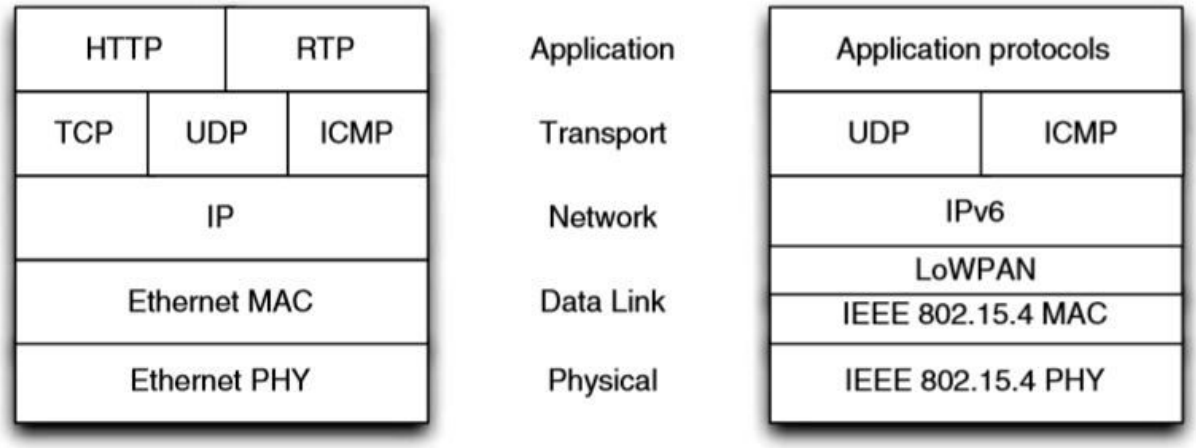


Figura 15. Pilas de protocolos IP y 6LoWPAN.

Fragmentación y reensamblado

Los paquetes IPv6 vienen en muchos tamaños diferentes. El más pequeño puede contener sólo los 40 bytes del encabezado IP. El más grande puede contener un payload de hasta 65,535 bytes y por ende un tamaño de paquete total de 65,575 bytes. Pocas subredes pueden transportar paquetes tan grandes de manera eficiente; la mayoría define una unidad de transmisión máxima (MTU) muy por debajo de esos tamaños. En un host con una sola interfaz, usualmente hay una manera de encontrar la MTU de la misma, lo que le permite a una aplicación ajustar el tamaño de los paquetes que envía. Con más de una interfaz, esta elección se vuelve menos clara. Además, la MTU puede cambiar desde un punto a otro en el camino del paquete, lo cual dificulta la elección por parte de la aplicación de un tamaño que pueda ser transportado en todos los enlaces en el camino. Para resolver esto, IPv4 requiere que cada nodo emisor (host o router) fragmente los paquetes, y que cada host receptor los reensamble. Para simplificar el encabezado IP, IPv6

eliminó los campos de fragmentación de IPv4. Si un emisor quiere usar fragmentación, debe insertar un encabezado de fragmentación separado como un encabezado de extensión. Este es el único lugar donde la fragmentación puede ocurrir. Además, la mínima MTU se incrementó considerablemente: todas las subredes IPv6 tienen que proveer una mínima MTU de 1280 bytes. Mientras que IPv6 provee su propia fragmentación para datagramas más grandes que 1280 bytes o que la MTU del enlace en cuestión, depende de la capa LowPan para llevarla a cabo. Para los enlaces incapaces de transportar paquetes de 1280 bytes, se deberán realizar las acciones adecuadas en sus capas de adaptación, con el objetivo de dividir los paquetes en frames de capa de enlace y reensamblarlos en el nodo receptor.

Cada fragmento 6LoWPAN traslada información que permite el reensamblaje, incluso para fragmentos fuera de orden. En contraste a los fragmentos IP regulares, los fragmentos 6LoWPAN sólo incluyen información del encabezado IP en el fragmento inicial de un paquete. Cuando un paquete IPv6 en un nodo emisor excede el tamaño del payload de la capa de enlace, el mecanismo de fragmentación de 6LoWPAN trata al paquete IPv6 como un único campo de datos e iterativamente segmenta este campo en fragmentos acordes al tamaño de frame máximo de la capa de enlace. Cada fragmento incluye un encabezado de tamaño fijo. El espacio restante del frame de capa de enlace es llenado iterativamente con el contenido del paquete IPv6 (Figura 16). Este proceso implica que sólo el primer fragmento contenga información de ruteo punto a punto. Los fragmentos 6LoWPAN contienen una etiqueta de datagrama que es incluída en cada encabezado de fragmento y es única por cada emisor y paquete fragmentado. Por lo tanto, esta permite que un nodo receptor busque información de ruteo para todos los fragmentos pertenecientes a un paquete fragmentado, luego de que el primer fragmento ha sido recibido. Cada fragmento también contiene información que permite el reensamblado de paquetes fragmentados en un nodo receptor. El tamaño de datagrama del paquete no fragmentado sirve para que un nodo receptor reserve espacio de buffer para el reensamblado del paquete entero. El offset de datagrama indica la posición del payload dentro del paquete IPv6 original. [8]

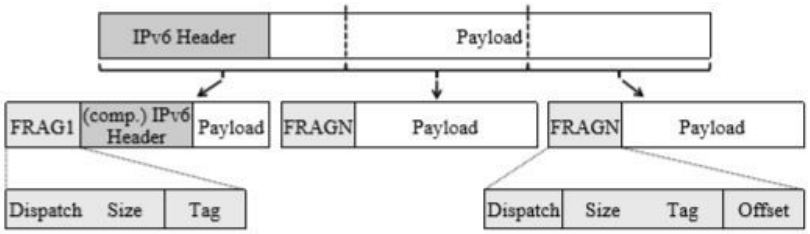


Figura 16. Estructura de paquete 6LoWPAN de un primer fragmento FRAG1 y los subsecuentes fragmentos FRAGN.

RPL

Es un protocolo de ruteo independiente del enlace basado en IPv6, para nodos de recursos limitados. Fue creado para soportar requerimientos de ruteo mínimos, mediante la construcción de una topología robusta sobre los enlaces con pérdidas. Soporta modelos de tráfico simples y complejos, como multipunto a punto, punto a multipunto y punto a punto. Un grafo acíclico dirigido orientado al destino (DODAG) representa el núcleo de RPL, que muestra un diagrama de enrutamiento de nodos. Cada nodo en el DODAG (figura 17) es consciente de sus padres, pero no tiene información sobre sus hijos. Además, RPL mantiene por lo menos un path a la raíz para cada nodo, y un padre preferido para alcanzar el path a la raíz más rápido. Para mantener la topología y la infomación de ruteo actualizadas, RPL utiliza cuatro tipos de mensajes de control. El más importante es el objeto de información DODAG (DIO) que es usado para mantener el nivel actual del nodo, determinar la distancia de cada nodo con respecto a la raíz en función de algunas métricas específicas, y para elegir el path preferido. El otro tipo de mensaje es el objeto de aviso de destino (DAO). RPL provee soporte para tráfico ascendente así como descendente, usando mensajes DAO mediante los cuales envía información de destino hacia los padres seleccionados. El tercer mensaje es solicitud de información DODAG (DIS), el cual es usado para adquirir mensajes DIO desde un nodo adyacente accesible. El último tipo de mensaje es reconocimiento DAO (DAO-ACk) que es una respuesta a un mensaje DAO, y es enviado por un nodo destinatario DAO, como un padre DAO o la raíz del DODAG.

Un DODAG comienza a formarse cuando la raíz en primer lugar envía su localización a través del mensaje DIO hacia todos los niveles de la red. En cada uno, los routers destino registran el path padre y los paths participantes para cada nodo. A su vez, propagan sus mensajes DIO, y de esa manera el DODAG se construye

gradualmente. Cuando el DODAG es construído, el padre preferido obtenido por un router se mantiene como el path por defecto hacia la raíz. La raíz también puede almacenar los prefijos destino obtenidos por los DIOs de otros routers en sus mensajes DIO para tener rutas ascendentes. Para soportar rutas descendentes, los routers deberían emitir y propagar mensajes DAO a la raíz.
Los routers RPL trabajan bajo uno de dos modos de operación: modo de almacenamiento o modo de no almacenamiento. En este último, los mensajes de rutas RPL se dirigen hacia los niveles más bajos basándose en el enrutamiento de origen IP, mientras que en el modo de almacenamiento estos mensajes se envían basándose en las direcciones IPv6 destino. [10]

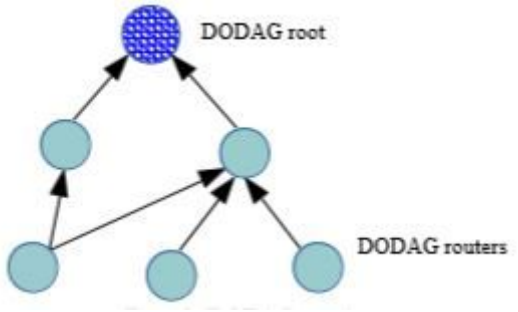


Figura 17. DODAG RPL.

MQTT

Es un protocolo de mensajería introducido en 1999 y estandarizado en 2013. Tiene como objetivo conectar dispositivos embebidos y redes con aplicaciones y un middleware. Utiliza el patrón publicar/suscribir para proveer flexibilidad de transición y simplicidad de implementación, como se muestra en la figura 18. Además, es apropiado para dispositivos con limitaciones de recursos que utilicen enlaces no confiables o de bajo ancho de banda. Está construído en la cima de la pila TCP/IP. Entrega mensajes mediante tres niveles de calidad de servicio. Existen dos especificaciones principales para MQTT: MQTT v3.1 y MQTT-SN (formalmente conocida como MQTT-S) V1.2. La última fue definida específicamente para redes de sensores. Las especificaciones proveen tres elementos: semánticas de la conexión, ruteo y punto final.
MQTT simplemente consiste de tres componentes, suscriptor, publicador y el broker. Un dispositivo interesado se registra como un suscriptor en topics específicos para ser informado por el broker cuando los publicadores publican tópicos de interés. El publicador actúa como un generador de datos de interés. Después de eso, el publicador transmite la información a las entidades interesadas (suscriptores)

mediante el broker. Además, el broker logra seguridad comprobando la autorización de los publicadores y los suscriptores. Numerosas aplicaciones usan MQTT, como el cuidado de la salud, monitoreo, medición de energía, y notificaciones de Facebook. Por ende, el protocolo MQTT representa un protocolo de mensajería ideal para IoT y comunicaciones M2M. [10]

COAP

Las aplicaciones web actuales en su gran mayoría requieren del Protocolo HTTP para acceder a la información y realizar actualizaciones; sin embargo, no es el más idóneo para trabajar con redes de sensores inalámbricos. HTTP está basado en REST, un estilo de arquitectura que proporciona la información disponible en la web mediante recursos identificados por los URI (figura 18). Los clientes acceden a estos recursos alojados en servidores y controlados en forma de petición-respuesta síncrona. Los recursos en la web a menudo contienen enlaces a otros recursos creando una web distribuida entre los puntos finales de Internet, lo que la convierte en una arquitectura altamente escalable y flexible. Estos conceptos fundamentales web se describen habitualmente como REST. En términos generales, la web se compone de tres tecnologías: HTML, HTTP/REST y URI. HTTP es por sí mismo un protocolo muy potente, pero tiene un alto costo en el tamaño del código de una implementación (superando la capacidad de memoria de los dispositivos embebidos) y también en los requerimientos de recursos de red. Ante esta situación, el grupo de trabajo CoRE (Constrained RESTful Environments) ha propuesto el protocolo CoAP (Constrained Application Protocol), un protocolo de aplicación de transferencia web para funcionar con recursos muy limitados. A diferencia de los protocolos basados en HTTP, CoAP opera sobre UDP y define una muy simple capa de mensajes para retransmitir los paquetes perdidos en lugar de usar el complejo control de congestión usado en el estándar TCP. En su capa de mensaje se definen cuatro métodos de petición: GET, POST, PUT y DELETE y sus códigos de respuesta son parecidos a los del HTTP (como el 404, de “no encontrado”). Por otro lado, CoAP puede ser fácilmente traducido a HTTP para promover la integración de los nodos embebidos y la web que conocemos. La pila de CoAP usa 6LoWPAN en dispositivos embebidos que necesitan comunicarse con los servicios basados en Internet, y su principal beneficio se alcanza cuando se interconecta con HTTP y se aprovecha la estructura REST que involucra las comunicaciones con el cliente, los proxies, gateways y servidores. Hay librerías de CoAP para C, C++, Java y Python [40].

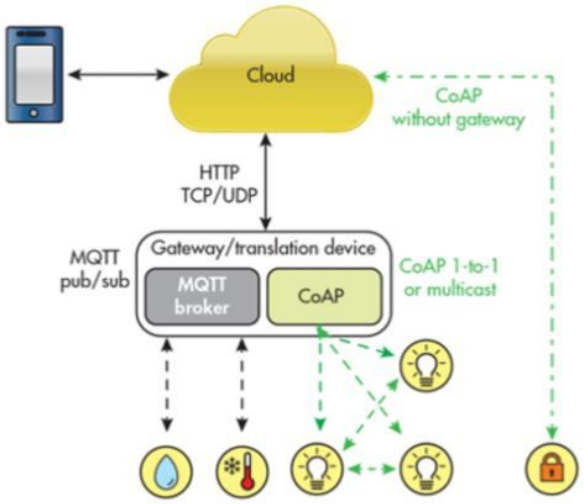


Figura 18. Arquitectura CoAP y MQTT.

Seguridad en IoT

Aspectos de seguridad en aplicaciones IoT

Un sistema/aplicación IoT, al igual que cualquier otro tipo de aplicación, no puede ser considerado seguro a no ser que cumpla ciertos objetivos. Estos pueden ser agrupados en tres categorías:

Confidencialidad. Los datos no pueden ser leídos por intrusos, es decir, se mantienen secretos excepto para los participantes autorizados en el flujo de información. Esto puede conseguirse haciendo la información ininteligible mediante cifrado criptográfico.

Integridad. Los datos no pueden ser alterados por partes no autorizadas, es decir, los datos que son recibidos por una parte autorizada son idénticos a los que fueron enviados por otra parte. En el mundo digital, la integridad se consigue añadiendo chequeos criptográficos a los mensajes. Un objetivo de seguridad relacionado es la autenticación: un mensaje se originó en un emisor que efectivamente es quien dice ser. Por lo tanto, un control de integridad de mensaje es con frecuencia lo mismo que la autenticación del mensaje.

Disponibilidad. El sistema no está sujeto a ataques de denegación de servicio. Nótese que cualquier sistema inalámbrico puede sufrir de interferencia en el nivel de radio; sin embargo, esa interferencia debería ser fácil de localizar y eliminar. Por otra parte, interferencias mucho más peligrosas pueden nacer desde emisores que no pueden ser fácilmente localizables y controlables.

Una vez definidos los objetivos de seguridad, es necesario entender el modelo de amenaza: lo que el atacante es capaz de hacer para vulnerar estos objetivos. Además, hay que tener en cuenta qué ventajas o beneficios obtendría el atacante en caso de impedir que se cumpla algún objetivo. Esto puede tener una implicancia en la cantidad de recursos que el atacante puede tener a disposición. El modelo de amenaza para sistemas inalámbricos como 6LoWPAN no difiere mucho del modelo general asumido por protocolos de seguridad de Internet: se asume que el atacante tiene el control total sobre el canal de comunicaciones. El atacante puede leer, borrar y cambiar cualquier mensaje, así como inyectar mensajes nuevos. De nuevo, sin soporte criptográfico no hay manera de proteger a los mensajes de ser leídos por partes no autorizadas, o de detectar cualquier manipulación.
El modelo de amenaza de Internet asume que los sistemas finales no han sido comprometidos. Sin embargo, el tamaño pequeño y la naturaleza distribuida de los nodos crea una amenaza significativa: es relativamente fácil obtener acceso físico y controlar al menos un nodo de la red. [3]

Seguridad en la capa de enlace

La implementación de la capa de enlace en Contiki engloba a tres diferentes capas: Framer, RDC (Radio Duty Cycle) y MAC (Medium Access Control).

- Capa MAC: Contiki implementa dos drivers MAC: CSMA y NullMAC. CSMA (Carrier-Sense Multiple Access) recibe los paquetes de la capa RDC, y utiliza a la misma para transmitir paquetes. Si la capa RDC detecta que el medio está ocupado, la capa MAC debe retransmitir el paquete más tarde. El protocolo CSMA mantiene una lista de los paquetes enviados a cada uno de los vecinos y calcula estadísticas tales como el número de retransmisiones, colisiones, etc. NullMAC sólo llama a las funciones RDC apropiadas, no realiza ninguna tarea adicional.

- Capa RDC: la capa de ciclo de servicio de radio (RDC) maneja el período de descanso de los nodos. Esta capa decide cuándo los paquetes serán

transmitidos y asegura que los nodos estén despiertos cuando van a recibir paquetes.
Contiki implementa los siguientes drivers RDC: Contikimac , xmac , lpp , nullrdc y sicslowmac. El más usado es ContikiMAC. NullRDC es un driver que nunca apaga el radio. Los drivers RDC intentan mantener el radio apagado la mayor cantidad de tiempo que sea posible, chequeando periódicamente si hay actividad en el medio inalámbrico. Cuando se detecta actividad, el radio se mantiene encendido para comprobar si debe recibir el paquete o si puede descansar y de esa forma ahorrar energía. La tasa de chequeo del canal está dada en Hz, especificando el número de veces que se chequea el canal por segundo. Un paquete generalmente debe ser retransmitido hasta que el receptor esté activo y lo reciba. Esto incrementa el consumo de energía del emisor y el tráfico de radio, pero los ahorros de energía en el receptor compensan esto. Una alternativa para optimizar la capa RDC es activar la “optimización de fase”, que retarda el envío de paquetes hasta el momento en que se estima que el receptor se despertará. Sin embargo, esto requiere una buena sincronización entre el emisor y el receptor.

- Capa framer: es un conjunto de funciones para encapsular los datos a ser transmitidos, y para parsear los datos recibidos. Los drivers que implementa Contiki son framer-802154 y framer-nullmac. [2]

La seguridad de 802.15.4 es manejada en la capa de control de acceso al medio (también llamada capa MAC). La aplicación especifica sus requerimientos de seguridad estableciendo los parámetros de control apropiados en la pila de radio. Si la aplicación no setea ningún parámetro, entonces la seguridad no se activa por defecto.
Una aplicación puede elegir el tipo de protección que se proporciona para los datos transmitidos. Cada una ofrece un conjunto diferente de propiedades de seguridad y garantías, y diferentes formatos de paquetes. La especificación 802.15.4 define ocho tipos de seguridad diferentes, mostrados en la figura 22. Se puede clasificar a estos tipos de seguridad en cuanto a las propiedades que ofrecen: ninguna, sólo cifrado (AES-CTR), sólo autenticación (AES-CBC-MAC), y cifrado y autenticación (AES-CCM). Cada categoría que soporta autenticación tiene tres variantes, dependiendo del tamaño del MAC (también llamado MIC) que ofrece. Cada variante tiene su propio nombre. El MAC puede ser de 4, 8, o 16 bytes de longitud. Cuanto más largo sea el MAC, más baja es la probabilidad de un atacante de adivinarlo. Por ejemplo, con un MAC de 8 bytes, tiene 2^{64} combinaciones posibles para adivinar el código. La desventaja, es que se tiene un tamaño de paquete más grande para

incrementar la protección contra ataques de autenticación. Adicionalmente, el receptor puede elegir también protección contra reenvíos. Los diseñadores de radio no tienen que implementar todas las variantes. La especificación sólo requiere que los chips de radio provean soporte para la variante sin seguridad y la AES-CCM-64. [3] [11]

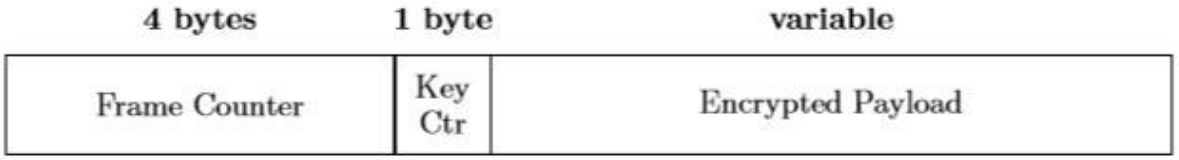


Figura 19. AES-CTR.

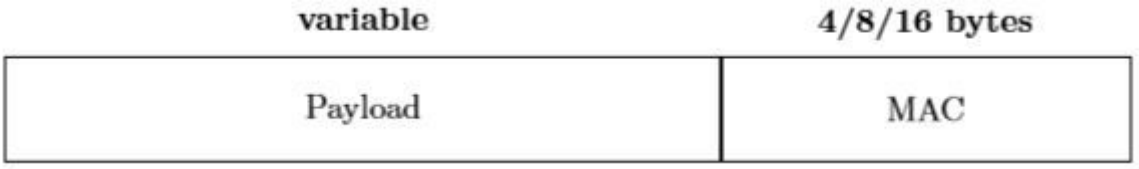


Figura 20. AES-CBC-MAC.

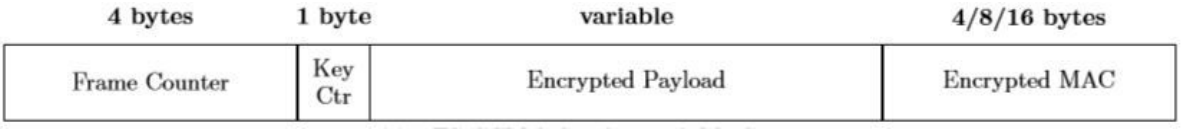


Figura 21. AES-CCM.

Name	Description
Null	No security
AES-CTR	Encryption only, CTR Mode
AES-CBC-MAC-128	128 bit MAC
AES-CBC-MAC-64	64 bit MAC
AES-CBC-MAC-32	32 bit MAC
AES-CCM-128	Encryption & 128 bit MAC
AES-CCM-64	Encryption & 64 bit MAC
AES-CCM-32	Encryption & 32 bit MAC

Figura 22. Variantes de seguridad soportadas por 802.15.4.

Una aplicación indica sus preferencias de seguridad en base a las direcciones origen y destino. Los chips de radio 802.15.4 tienen una lista de control de acceso (ACL) que controla qué tipo de seguridad y qué información de clave usar. Los dispositivos compatibles soportan hasta 255 entradas en su ACL. Cada entrada contiene una dirección 802.15.4, un identificador de tipo de seguridad, y material de seguridad, como se muestra en la figura 23. El material de seguridad es necesario para ejecutar el tipo de seguridad. Consiste de la llave de cifrado y, para los tipos de seguridad que provean encriptación, del nonce (o vector de inicialización) que debe ser preservado durante las diferentes invocaciones de cifrado de paquetes. Cuando se utiliza protección contra reenvíos, también se incluye una "marca de agua" del identificador del paquete más recientemente recibido.
Como parte de la interfaz para enviar paquetes, la aplicación debe especificar un valor booleano indicando si la seguridad está activada. Si no se solicita seguridad, se envía el paquete sin ningún agregado. Si se activa la seguridad, la capa MAC busca la dirección destino en su tabla ACL. Si hay una entrada que coincida, el tipo de seguridad, la clave y el nonce especificados en esa entrada ACL son usados para cifrar y/o autenticar el paquete saliente, y se setean acordemente los flags. Si no se encuentra la dirección destino en la tabla ACL, se utiliza una entrada ACL por defecto; que es similar a las demás entradas excepto que machea con todas las direcciones destino. Si la entrada por defecto es vacía y la aplicación solicitó seguridad, la capa MAC devuelve un código de error.
En la recepción de un paquete, la capa MAC consulta el campo flags para determinar si es un paquete securizado. Si no se utilizó seguridad, se pasa el paquete a las capas superiores. De otra manera, se emplea un proceso similar para encontrar la entrada ACL apropiada, pero esta vez basándose en la dirección origen. Luego se aplica el tipo de seguridad, clave y contador de reenvío apropiados al paquete entrante, arrojando como resultado un mensaje de error si no se localizó una entrada ACL. [8] [11]

A continuación se provee más detalle sobre las categorías de los tipos de seguridad de 802.15.4:

- Null: este es el tipo de seguridad más simple. Su inclusión es obligatoria en todos los chips de radio. No posee ningún material de seguridad. No ofrece ninguna garantía de seguridad.

- AES-CTR: provee confidencialidad mediante el algoritmo AES block cipher con modo contador [19]. Para encriptar datos con el modo contador, el emisor divide el paquete en bloques de 16 bytes $p_1,...,p_n$ y calcula $c_i = p_i \oplus E_k(x_i)$. Cada bloque

de 16 bytes utiliza su propio contador variable, denominado x_i. El receptor recupera el paquete original calculando $p_i = c_i \oplus E_k(x_i)$. Claramente, el receptor necesita el valor del contador x_i para reconstruir p_i. El contador x_i, conocido como nonce, se compone de: un campo estático flags, la dirección del emisor, y 3 contadores separados: un contador de frame de 4 bytes que identifica el paquete, un contador de clave de 1 byte, y un contador de 2 bytes que enumera los bloques de 16 bytes dentro del paquete, como se muestra en la figura 24. El contador de frame es mantenido por el hardware del radio. El emisor lo incrementa luego de cifrar cada paquete. Cuando alcanza su máximo valor, el radio devuelve un código de error y no se permiten más cifrados. El contador de clave está bajo el control de la aplicación. Puede incrementarse si el contador del frame llega a su máximo valor. El requisito es que el nonce nunca debe repetirse dentro del tiempo de vida de una clave, y el rol de los contadores de frame y de clave es prevenir el reuso del nonce. El contador de bloque de 2 bytes asegura que cada bloque utilizará un nonce diferente; el emisor no necesita incluirlo en el paquete, dado que el receptor puede inferir su valor para cada bloque. En resumen, el emisor incluye el contador de frame, el contador de clave y el contenido encriptado dentro del campo datos, como muestra la figura 19 .[3]

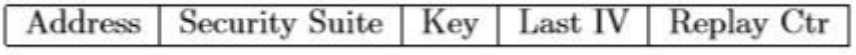


Figura 23. Formato de una entrada ACL.

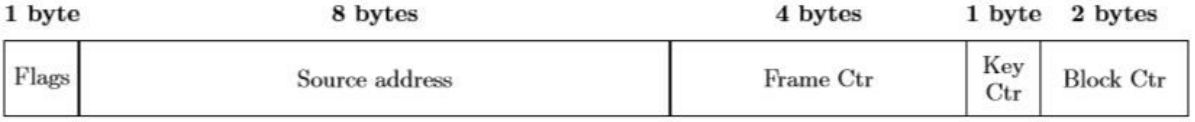


Figura 24. Formato de la entrada x_i para el algoritmo block cipher para los tipos AES-CTR y AES-CCM.

- AES-CBC-MAC: provee protección de integridad usando el algoritmo CBC-MAC. El emisor puede calcular un MAC de 4, 8 o 16 bytes, dando lugar a tres variantes AES-CBC-MAC. El MAC sólo puede ser calculado por partes que

tengan una clave simétrica. Protege a los encabezados de los paquetes así como al contenido de los mismos. El emisor agrega los datos en texto plano junto con el MAC, como muestra la figura 20. El receptor verifica el MAC calculándolo y comparándolo con el valor incluído en el paquete.[14]

- AES-CCM: utiliza CCM para encriptación y autenticación. En general, primero aplica protección de integridad sobre el encabezado y la carga útil de datos usando CBC-MAC y luego cifra la carga útil y el MAC usando AES-CTR. Como tal, AES-CCM incluye los campos para las operaciones de autenticación y cifrado: un MAC, y los contadores de frame y de clave. Estos campos tienen la misma función que se explicó anteriormente. Se muestra el formato de paquete de AES-CCM en la figura 21.[3]

Opcionalmente un receptor puede activar la protección contra reenvíos. Esto incluye a AES-CTR y a todas las variantes de AES-CCM. Los receptores utilizan los contadores de frame y de clave como un valor de 5 bytes, el contador de reenvío, con el contador de clave ocupando el byte más significativo de este valor. El receptor compara el contador de reenvío del paquete entrante con el valor más alto almacenado en la entrada ACL. Si el paquete entrante tiene un contador de reenvío más grande que el almacenado, el paquete es aceptado y el nuevo contador de reenvío almacenado. Si el paquete entrante tiene un valor más bajo, se rechaza y se notifica de esto a la aplicación. Este contador de reenvío sirve a un propósito lógicamente diferente que el del nonce, el cual es usado para confidencialidad [1].

Esquemas de predistribución de clave

Para conseguir seguridad en las WSNs, es importante poder encriptar y autenticar los mensajes enviados entre los nodos de la misma. Las claves para encriptar y autenticar deben ser acordadas entre los nodos. Debido a las limitaciones de recursos propias de este tipo de redes, lograr ese acuerdo no es una cuestión trivial. Muchos esquemas de acuerdo de clave utilizados en las redes generales, tales como Diffie-Hellman y los basados en clave pública [5], no son apropiados para redes de sensores inalámbricos.
Por lo tanto, el problema que surge es cómo establecer claves privadas entre los nodos que se comunicarán. Hay tres tipos de esquemas de acuerdo de clave: servidor de confianza, auto-ejecución, y pre-distribución de clave. El esquema de servidor de confianza depende de un servidor para el acuerdo de clave entre nodos. Este tipo de esquema es inapropiado para las WSNs porque no hay una infraestructura de confianza en dichas redes. El esquema de auto-ejecución

depende de la criptografía asimétrica, por ejemplo mediante certificados de clave pública. Sin embargo, el estrecho límite de capacidad de cómputo y de recursos hace que no sea conveniente el uso de algoritmos de clave pública como Diffie-Hellman o RSA. El tercer tipo de acuerdo de clave es la pre-distribución de claves, donde la información sobre las mismas es distribuida entre todos los nodos antes del despliegue de la red. Si se conoce qué nodos están en el mismo rango antes de la instalación de la red, las claves pueden ser decididas a priori.
Dos métodos sencillos para el problema de distribución de clave son el esquema de toda la red y el esquema de pares. En el primero, cada nodo se carga con una clave k. Una vez desplegada la red, cada nodo usa la clave k (es la misma para cada nodo) para securizar sus comunicaciones. A pesar de que este esquema es extremadamente eficiente, es muy propenso a amenazas, ya que un nodo que se vea comprometido puede dejar al descubierto la clave de toda la red.
En el esquema de pares, cada nodo es pre cargado con n-1 claves, siendo n el número de vecinos de ese nodo, cada una de las cuales es compartida con otro nodo para establecer comunicaciones seguras. A diferencia del esquema anterior, este no pone en compromiso a toda la red si la clave de un nodo es afectada, ya que para comunicarse con los demás tendrá otras claves diferentes. Por otro lado, el overhead producido por el almacenamiento de las claves en cada nodo es lineal al número de nodos de la red y por ende inaceptable [38].

Enfoque utilizado en Contiki

Cualquier aplicación desarrollada en este sistema operativo puede elegir en tiempo de compilación los servicios de seguridad que usará en la capa de enlace. Esto se realiza seteando algunos parámetros y flags de la aplicación en el archivo de configuración de proyecto de Contiki (generalmente llamado *project-conf.h*), como se muestra en la siguiente imágen:

```
#undef LLSEC802154_CONF_ENABLED
#define LLSEC802154_CONF_ENABLED         1
#undef NETSTACK_CONF_FRAMER
#define NETSTACK_CONF_FRAMER             noncoresec_framer
#undef NETSTACK_CONF_LLSEC
#define NETSTACK_CONF_LLSEC              noncoresec_driver
#undef NONCORESEC_CONF_SEC_LVL
#define NONCORESEC_CONF_SEC_LVL          1
```

```
#define NONCORESEC_CONF_KEY { 0x00 , 0x01 , 0x02 , 0x03 , \
                              0x04 , 0x05 , 0x06 , 0x07 , \
                              0x08 , 0x09 , 0x0A , 0x0B , \
                              0x0C , 0x0D , 0x0E , 0x0F }
```

Se destacan dos parámetros. El primero, NONCORESEC_CONF_SEC_LVL, es el utilizado para indicar el nivel de seguridad 802.15.4 (los que se describieron anteriormente) que usará la aplicación. El segundo, NONCORESEC_CONF_KEY, es la clave de red utilizada para hacer uso de los servicios de seguridad especificados por el parámetro anterior.
Se puede modificar esta configuración en cualquier momento a través de sus parámetros, en función de los servicios que precise la aplicación. Cabe destacar que el último parámetro (es decir, la clave) responde al enfoque utilizado en Contiki en el cual se utiliza un esquema de pre distribución de clave compartida en toda la red.
Actualmente Contiki utiliza una implementación de seguridad permeable a ser comprometida, denominada noncoresec. A pesar de soportar las ocho variantes de seguridad de 802.15.4 y de estar implementada y subida al repositorio oficial, esta implementación presenta graves falencias: descarta frames inyectados y reenviados añadiendo un MIC (código de integridad del mensaje) y un contador de frame incremental a cada frame saliente. Almacenar un contador por cada vecino no es una alternativa aceptable, ya que la memoria en los dispositivos 802.15.4 está severamente limitada. Por lo tanto, en redes con gran número de nodos móviles algunos datos anti replay tienen que ser llevados a la memoria no volátil. Dado que la única memoria no volátil en la mayoría de los dispositivos 802.15.4 es la memoria flash, este procedimiento es muy costoso en cuanto a energía y tiempo.
Los reinicios ocurren en Contiki cuando se atasca un proceso, o cuando se cambian las baterías. Después de un reinicio, toda la información anti replay que se encontraba en la RAM se pierde. Por lo tanto, para prevenir ataques anti replay posteriores a reinicios, la información anti replay tiene que ser almacenada entre cada uno. Dos problemas asoman si un contador de frame de un nodo se reutiliza después de un reinicio. Por un lado, causaría un reuso del nonce y, por el otro, los vecinos interpretarían a los frames del nodo reiniciado como frames reenviados [17].

Enfoque a utilizar en este trabajo

Dados los inconvenientes de la implementación de la seguridad en capa de enlace de Contiki, presentados en la sección anterior, en el presente trabajo se utiliza una implementación de seguridad en capa de enlace para redes 802.15.4 denominada Adaptivesec [33]. Ofrece los mismos servicios de seguridad que especifica el

estándar 802.15.4, con las siguientes ventajas: resistencia a los reinicios de los nodos, adaptabilidad con respecto a la movilidad de los mismos, y la posibilidad de elegir un esquema de predistribución de claves que se adapte a las necesidades de la red.
Construído sobre la implementación APKES [17], Adaptivesec deja que su esquema de predistribución de claves sea intercambiable. El mismo lo provee de claves secretas. A partir de éstas, establece claves de sesión entre pares y, en caso de ser necesarias, claves de sesión de grupo. Estas últimas pueden servir para securizar frames unicast y broadcast, mientras que las primeras sirven sólo para frames unicast.
Como se explicó anteriormente, para sobrellevar los reinicios de las motas la seguridad de 802.15.4 normalmente requiere que las mismas almacenen su contador de frame en la memoria no volátil. Sin embargo, la única memoria no volátil en la mayoría de los dispositivos 802.15.4 es la memoria flash [34], la cual conlleva un significativo gasto energético. Se verá a continuación que el establecimiento de claves de sesión de Adaptivesec libera a los nodos de tener que almacenar la información anti reenvío y los contadores de frame en la memoria no volátil.
Siguiendo el enfoque del estándar 802.15.4, un frame entrante es aceptado sí y sólo si tiene un MIC (código de integridad del mensaje) auténtico, y si su contador de frame es más grande que el último aceptado proveniente del mismo emisor (en el resto del trabajo esto equivaldrá a decir que el frame entrante “es reciente”). El cifrado del payload de los frames enviados y la generación de los MICs se realizan con una versión ajustada del algoritmo Counter with CBC-MIC (CCM), que utiliza AES-128 para el cifrado en bloque.
El esquema de predistribución de claves integrado provee a Adaptivesec de claves secretas. A partir de éstas, se establecen claves de sesión entre pares y, en caso de ser necesarias, claves de sesión de grupo. En las siguientes secciones se pretende explicar y detallar el funcionamiento de Adaptivesec.

Estableciendo parámetros de configuración

Cualquier nodo que corra Adaptivesec debe contar con la siguiente información:

- Cuando se solicita la clave compartida para un nodo *v*, Adaptivesec introduce el identificador PAN (PAN_v) y la dirección de *v* (identificada como ID_v) en el esquema de predistribución de claves que se utilize.

- Además, el nodo debe ser cargado con material de clave. El mismo es específico al esquema de predistribución de claves que se utilize. Adaptivesec

soporta el esquema de claves entre pares y el esquema de clave compartida en toda la red. Por defecto, Adaptivesec utiliza este último, y en este trabajo se seguirá el mismo enfoque[1].

[1] *Se toma la decisión de adoptar este enfoque porque el esquema de claves entre pares aún no está probado en las motas Zolertia z1, que son las utilizadas en este trabajo.*

Estableciendo claves de sesión

Adaptivesec utiliza un saludo de tres vías para entablar claves de sesión. Los tres mensajes son enviados como tres frames de comandos 802.15.4; denominados HELLO, HELLOACK, y ACK. En contraste a los frames de datos, los frames de comandos no transportan tráfico de capas superiores.

El saludo básico de tres vías es mostrado en la figura 25. En la misma, un nodo *u* establece una clave de sesión entre pares con un vecino *v*, siguiendo el proceso que se describe a continuación. Inicialmente, *u* genera un número aleatorio R_u y envía un frame broadcast con un mensaje HELLO que contiene a R_u. Una vez recibido, *v* solicita la clave $K_{v,u}$ a su esquema de predistribución, y también genera un número aleatorio R_v. Luego, *v* obtiene la clave de sesión entre pares $K'_{v,u}$ realizando AES-128($K_{v,u}$, R_u || R_v). Por último, *v* envía un HELLOACK hacia *u* incluyendo a R_v . El HELLOACK es autenticado añadiendo un MIC que es generado con $K'_{v,u}$. Una vez recibido, *u* chequea si el MIC añadido es auténtico obteniendo la clave de sesión entre pares $K'_{u,v}$ de forma análoga a *v*. Si el MIC es auténtico, *u* envía un ACK a *v.* El ACK también contiene un MIC que es generado en este caso con $K'_{u,v}$. Cuando lo recibe, *v* chequea si el mismo es auténtico. Si lo es, *u* y *v* concordaron su clave de sesión entre pares $K'_{v,u}= K'_{u,v}$. Eventualmente, *u* cambia R_u para preparar el broadcasting del siguiente HELLO. Durante este saludo, *u* y *v* pueden intercambiar sus respectivas claves de sesión de grupo $K_{u,*}$ y $K_{v,*}$ en los ACKs y los HELLOACKs respectivamente. Estas se cifran utilizando $K'_{u,v}$ y $K'_{v,u}$, y son utilizadas para cifrar frames unicast y broadcast [2].

Adaptivesec distingue entre vecinos tentativos y permanentes. Mientras que los vecinos tentativos son creados cuando se recibe un HELLO de una mota, los permanentes se crean cuando se reciben ACKs y HELLOACKs. La diferencia principal entre ambos radica en que los frames de datos solo pueden ser enviados desde y hacia vecinos permanentes.

Establecer claves de sesión con nuevos vecinos requiere hacer un broadcasting de un comando HELLO. Por lo tanto, el desafío es transmitir la menor cantidad de

HELLOs posible, y a la vez adaptarse a los cambios de la red. Adaptivesec utiliza Trickle para lograr esto [18], un algoritmo para diseminar información en una red de sensores inalámbricos.

[2] Si se ejecuta Adaptivesec con un esquema de predistribución de clave compartida, la clave de sesión de grupo será la clave compartida tomada del esquema (la cual será la misma en todos los nodos).

$$
\begin{aligned}
& u : \text{Generate } R_u \in \{0,1\}^{64} \text{ randomly} \\
u \rightarrow * & : \langle \texttt{HELLO}, PAN_u, ID_u, R_u, C_u \rangle \\
& v : \text{Request shared secret } K_{v,u} \text{ from the plugged-in key predistribution scheme} \\
& v : \text{Generate } R_v \in \{0,1\}^{64} \text{ randomly} \\
& v : K'_{v,u} = \text{AES-128}(K_{v,u}, R_u \| R_v) \in \{0,1\}^{128} \\
v \rightarrow u & : \langle \texttt{HELLOACK}, PAN_u, ID_u, PAN_v, ID_v, R_v, (\{K_{v,*}\}), [(I_{u,v}), C_v | I_{u,v}, C_{v,u}, C_{v,*}], P_u \rangle \\
& u : \text{Request shared secret } K_{u,v} \text{ from the plugged-in key predistribution scheme} \\
& u : K'_{u,v} = \text{AES-128}(K_{u,v}, R_u \| R_v) \in \{0,1\}^{128} \\
u \rightarrow v & : \langle \texttt{ACK}, PAN_v, ID_v, PAN_u, ID_u, (\{K_{u,*}\}), [(I_{v,u}), C_u | I_{v,u}, C_{u,v}, C_{u,*}] \rangle \\
& u : \text{Generate } R_u \in \{0,1\}^{64} \text{ randomly}
\end{aligned}
$$

Figura 25. Saludo básico de tres vías en Adaptivesec entre dos nodos *u* y *v*.

Asegurando frames broadcast: el protocolo EBEAP

Contrariamente al enfoque utilizado en la implementación de la seguridad de 802.15.4 en Contiki donde los frames broadcast son autenticados con claves que son compartidas entre todos los nodos vecinos; se propone una solución alternativa desarrollada en un protocolo llamado EBEAP [17] (easy broadcast encryption and authentication protocol). Se provee a continuación una visión general del protocolo:

Si un nodo *u* quiere enviar un frame broadcast *f* de manera segura, entonces *u* agrega un contador de frame a *f* para obtener *f'* y enviar dos frames broadcast. El primero contiene MICs m_0 $||m_1||$ ... sobre *f'* para cada vecino permanente de *u* v_0, v_1, ... Por lo tanto, m_i se genera usando la clave de sesión entre *u* y v_i. Una vez recibido, el vecino v_i extrae su MIC correspondiente m_i y lo almacena en un buffer. Para esto, v_i tiene que tener su índice $i = I_{vi,u}$ en la lista de vecinos de *u*. Estos índices son distribuidos por Adaptivesec. El segundo frame broadcast es *f'*. Una vez que lo recibe, v_i genera un MIC sobre *f'* utilizando la clave de sesión entre v_i y *u*. Si el MIC está almacenado y si *f'* es reciente, *f'* es aceptado. Bajo demanda, EBEAP también puede encriptar el payload de *f'* usando una clave de sesión de grupo.

Dependiendo de cuál sea la configuración elegida, Adaptivesec asegura los frames unicast y broadcast de diferente maneras. En una, asegura a ambos usando claves

de sesión de grupo. En otra, asegura a los frames unicast con claves de sesión entre pares y los broadcast con el protocolo EBEAP. En este trabajo se adopta la primer configuración.

Manejo de la movilidad de las motas

En cuanto a la movilidad de las motas participantes, los vecinos pueden irse de rango, además de que nuevos vecinos pueden aparecer. A continuación, se detalla cómo Adaptivesec se adapta a esos cambios borrando vecinos permanentes que hayan desaparecido y enviando mensajes HELLO para descubrir nuevos.
Cuando un vecino permanente desaparece, Adaptivesec chequea si está todavía en rango intercambiando dos frames de comando. Puntualmente, un nodo *u* envía un UPDATE autenticado a *v*. Una vez recibido un UPDATE auténtico y reciente, *v* responde con un UPDATEACK autenticado. Finalmente, como *u* recibe el UPDATEACK auténtico y reciente de *v*, *u* prolonga el tiempo de vida de *v*. Si en cambio no se recibe ningún UPDATEACK auténtico después de reenviar el UPDATE, *u* borra a *v*. Además, cuando *v* recibe un UPDATE auténtico de *u*, *v* prolonga el tiempo de vida de *u también*. Esta prolongación también se realiza implícitamente cuando se recibe cualquier frame auténtico y reciente, lo que reduce las interacciones explícitas UPDATE/UPDATEACK.

Aplicación a securizar

Presentación

El proyecto WENU, desarrollado por el Laboratorio de Investigación en Nuevas Tecnologías Informáticas (LINTI) de la Universidad Nacional de La Plata, se enfoca en controlar el uso innecesario de aires acondicionados y estufas en edificios públicos. Para esto es necesario desplegar varios componentes: un sensor de temperatura por cada ambiente a controlar, un punto de acceso a Internet y un servidor para el monitoreo de datos. Opcionalmente se puede usar una App móvil para monitorear y tomar decisiones. La misma permite observar rápidamente el estado de los sensores de un edificio y opcionalmente enviar señales para apagar dispositivos eléctricos de climatización. La aplicación muestra cada sensor posicionado en su ubicación física en un mapa y, destacando por medio de colores, la información sensada en cada habitación. Además actualiza los datos en pantalla de forma periódica comunicándose con un servidor que concentra los datos. Esta

aplicación se encuentra dirigida a administradores de edificios y cualquier responsable a cargo del ahorro energético de la institución.
Se definieron parámetros para evaluar el estado de cada habitación, chequeando los datos de temperatura y movimiento. Dichos parámetros son comparados con los datos obtenidos por los sensores para reflejar los colores correspondientes de los mismos. En función de la diferencia que se encuentre entre ambos se decide mostrar los datos con el color correspondiente. Se definió el color verde como estado normal y el rojo como advertencia cuando se encuentra en un estado que no es el esperado.
Como se mencionó anteriormente, WenuApp accede al estado de los sensores y también permite almacenar configuraciones en un servidor centralizado, actualmente el servicio que se ha estado utilizando es InfluxDB [29].
Con el fin de simplificar las comunicaciones con la aplicación móvil, el mismo grupo de trabajo desarrolló una api REST propia, WenuAPI, que fue diseñada para actuar como intermediario con InfluxDB, el cual se seguirá utilizando para almacenar la información enviada por los sensores.
Las motas se comunican utilizando el protocolo MQTT, por lo que el proyecto utiliza Mosquitto [26] como broker MQTT y Huginn [30] como puente para almacenar los datos recibidos por el broker en la base de datos InfluxDB. Huginn también es utilizado para verificar periódicamente si es necesario realizar alguna acción en una mota, y en tal caso publicar un mensaje por MQTT para que la mota la realice. Por otro lado, Huginn tiene la flexibilidad de poder ser utilizado para automatizar otras tareas y es fácilmente configurable por un administrador.

El sistema requiere una serie de sensores para funcionar: un sensor de temperatura, un sensor de movimiento y, opcionalmente, un sensor de corriente para controlar si un dispositivo de climatización eléctrico se encuentra encendido.

Como parte de la primer etapa de desarrollo del proyecto WENU, el equipo de trabajo dispuso del escenario que se describe a continuación:

Se pusieron a funcionar 2 motas con sensores Zolertia Z1 [20], una de ellas cuenta solamente con un sensor de temperatura y representa los datos de control que servirán para determinar si un aula u oficina está a una temperatura más alta o baja de la esperada. Esta mota idealmente debe ser ubicada en un pasillo u otro lugar abierto, y próximo a las dependencias a medir, que además no tenga calefacción ni aire acondicionado. La segunda mota está configurada para ser instalada en un aula u oficina y medir además de la temperatura, corriente (para determinar si un

equipamiento eléctrico se encuentra conectado) y movimientos para determinar si la dependencia está ocupada.
Una tercer mota cumple funciones como “border router” para proveer conectividad IPv6 al resto.
En la figura 26 se puede ver el esquema de conectividad y servicios implementados (con la salvedad que actualmente hay solamente 2 motas ejecutando mqtt-sensores.c).

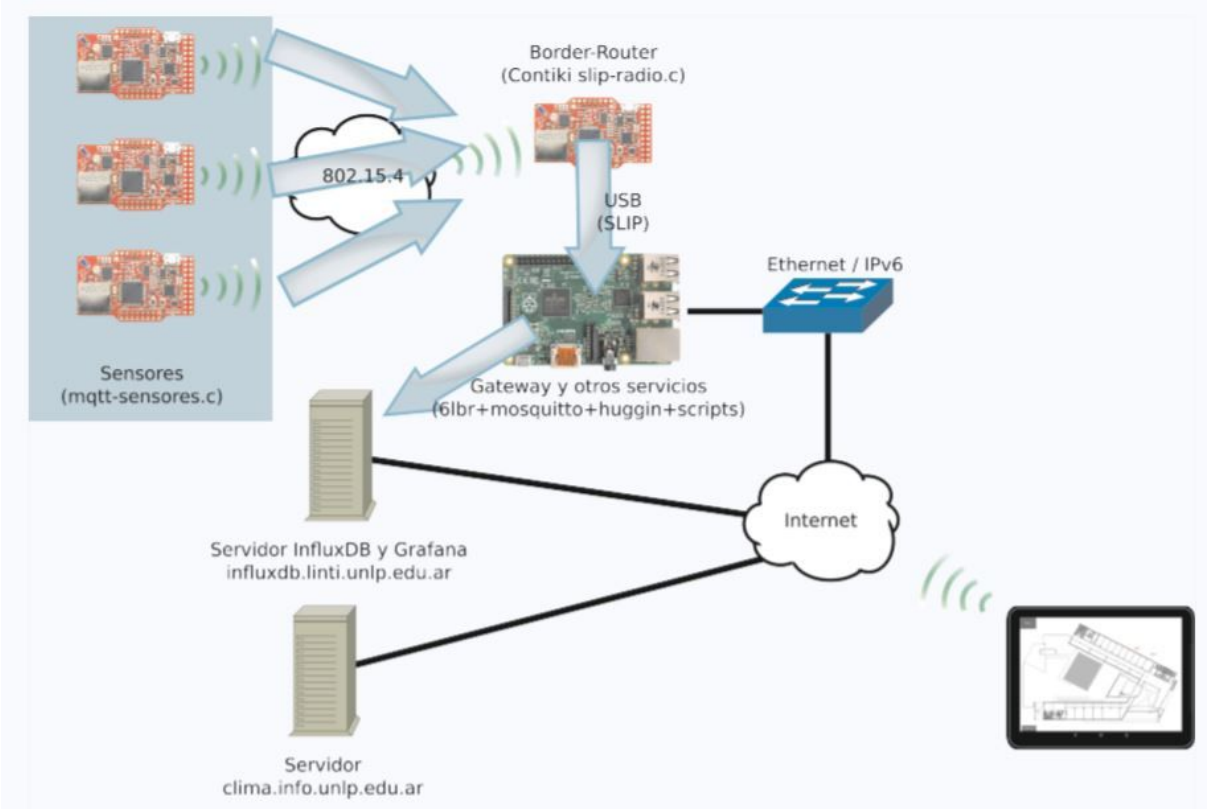


Figura 26. Esquema de conectividad y servicios implementados en WENU.

Las motas cuentan con conectividad IPv6 sobre 802.15.4, en capa de aplicación se utiliza el protocolo MQTT enviando como payload de los mensajes un objeto JSON [31] que codifica el identificador de la mota, la temperatura media, la corriente y un valor 1 o 0 que indica si hay movimientos o no. Por ejemplo:

```
{
    “mote_id”: “linti_oficina_x”,
    “temperature”: 20,
```

“current”: 0,
“movement”:1
}

A su vez , el protocolo 6LoWPAN será el encargado de adaptar entre el mundo estándar IPv6 y el protocolo de comunicación inalámbrica 802.15.4.

Como se indicó anteriormente, se puede apreciar dos tipos de motas:

- Mota de control: tiene una de las configuraciones más sencillas, solamente tiene conectado un sensor de temperatura y humedad SHT25, al operar entre 2.1v y 3.6v es posible usar este sensor alimentando la mota solamente con 2 pilas AA. Solamente envía datos de temperatura, cargando los campos “current” y “movement” del mensaje JSON generado en cero.

- Motas para aulas/oficinas: estas motas (por el momento hay una en funcionamiento) contarían con 3 sensores, un sensor de humedad SHT25, un sensor de movimiento infrarrojo pasivo HC-SR501 y un sensor de corriente efecto hall Acs712. El propósito de estas motas es medir la temperatura, el uso de aparatos eléctricos y la presencia de personas en cada oficina/aula.

Déficits de seguridad en la capa a tratar

Como se explicó anteriormente, este trabajo de investigación está enfocado en el estudio de la seguridad en la capa de enlace de la aplicación presentada en la sección anterior. Para obtener un panorama general, basta solo con mirar el archivo de configuración de proyecto de la aplicación mqtt-sensores:

project-conf.h | reports.h | mqtt-sensores.c

```
#ifdef NETSTACK_CONF_RDC
#undef NETSTACK_CONF_RDC
#endif
#define NETSTACK_CONF_RDC nullrdc_driver
// Si esta comentado usa CSMA
/*
#ifdef NETSTACK_CONF_MAC
#undef  NETSTACK_CONF_MAC
#endif
#define NETSTACK_CONF_MAC   nullmac_driver
*/
```

Se puede apreciar que, en lo que concierne a seguridad de capa 2, ningún parámetro se encuentra seteado y, por ende, la aplicación se encuentra totalmente desprotegida para las comunicaciones en dicha capa. Un posible atacante podría simplemente conseguir datos de la red objetivo mediante técnicas de descubrimiento; obtener otra mota; configurar la mota obtenida con RPL e IPv6 para que se una a la red, y mediante técnicas de sniffing monitorear todo el tráfico entre las motas y el border router.
Pensando un escenario más nocivo, la mota atacante podría enviar datos de mediciones erróneos, pudiendo provocar acciones inapropiadas para el sistema (de calefacción o de ventilación) que se esté controlando.

Por último, la mota atacante podría enviar una cantidad desmesurada de frames a cualquier otra de la red, pudiendo provocar un reinicio en ella que altere el funcionamiento normal del sistema (es decir, un ataque de denegación de servicio).
Para solventar esto, se puede optar por la solución ofrecida por Contiki para implementar seguridad en capa de enlace [32]. Sin embargo, como se demostró antes, se va a descartar esta opción por todos los riesgos que acarrea.
En la próxima sección, se explica el proceso mediante el cual se logró que Adaptivesec se aplique en el proyecto WENU.

Solución propuesta

En este capítulo se intenta describir de manera concisa cómo se logró aplicar el enfoque presentado en este trabajo para resolver los requerimientos de seguridad de la aplicación WENU. Una vez comprendido esto, se presentarán escenarios de prueba en un ambiente de simulación de Contiki, llamado Cooja [2]. Cooja permite

simulaciones de redes de motas a pequeña y gran escala, y es una herramienta sumamente útil para el desarrollo en Contiki ya que le permite a los desarrolladores probar su código antes de ejecutarlo en el hardware final, y sin necesidad de tener que disponer de las motas fisicas para hacerlo.

Las motas corren una versión modificada de Contiki (en particular la rama de desarrollo de Adaptivesec), para las motas sensoras se desarrolló ‘mqtt-sensores.c’ usando la librería Contiki-mqtt [35], mientras que la mota que cumple funciones de border router corre otra versión modificada de Contiki llamada 6lbr [36]. Esta es una implementación de border-router que es fácilmente configurable y puede soportar varios tipos de topología de red.

Aplicando Adaptivesec a WENU

Con el objetivo de embeber Adaptivesec en WENU y ejecutar la aplicación resultante con eficacia, se llevó durante el presente trabajo de grado un enfoque incremental en el cual se probó la correcta ejecución de la librería de seguridad en los siguientes flujos de comunicaciones:

A. Entre las motas.

B. Entre las motas y el border router.

Para la prueba del primer punto, se prescindió del uso del border router, dejando también para la siguiente fase el uso de MQTT (ya que la mota que corre el border router también ejecuta el broker). Se dispuso de dos motas que sensen los datos correspondientes y se los envíen, imprimiendo en salida estándar los datos recibidos. Se enviaron mensajes multicast, dejando para un paso posterior la búsqueda de la dirección IP del border router. Por otra parte, Adaptivesec se configura en el nivel 1, proveyendo solamente servicio de autenticación de las motas con un MIC de 4 bytes; y se optó (temporalmente) por UDP para el transporte de los mensajes de las motas, dado su bajo overhead de memoria y procesamiento, lo cual es esencial en dispositivos IoT.

El objetivo de esta etapa es probar la ejecución de WENU con una configuración mínima de Adaptivesec:

```
/* Cifrado adaptivesec */
#if WITH_LLSEC==1
  #undef ADAPTIVESEC_CONF_UNICAST_SEC_LVL
  #define ADAPTIVESEC_CONF_UNICAST_SEC_LVL 1
  #undef ADAPTIVESEC_CONF_BROADCAST_SEC_LVL
  #define ADAPTIVESEC_CONF_BROADCAST_SEC_LVL 1

  #undef LLSEC802154_CONF_USES_AUX_HEADER
  #define LLSEC802154_CONF_USES_AUX_HEADER 1

  #include "net/llsec/adaptivesec/noncoresec-autoconf.h"

  #define SINGLE_CONF_KEY "rR0lUtQ5UC"

  #undef CSPRNG_CONF_SEEDER
  #define CSPRNG_CONF_SEEDER iq_seeder

  #define AES_128_CONF aes_128_driver

#endif

#undef NBR_TABLE_CONF_MAX_NEIGHBORS
#define NBR_TABLE_CONF_MAX_NEIGHBORS 4
```

Figura 27. Configuración inicial de Adaptivesec para su aplicación en WENU.

Se presenta a continuación la salida que arroja cada mota, luego de la ejecución del programa (el cual está presente en mqtt-sensores.c y mqtt-sensores.z1 en su versión compilada):

```
00:06.201  ID:2  {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
00:06.203  ID:2  e::40a:0:0:0:0:0
00:06.205  ID:2  Publicando cada 5 segundos
00:06.366  ID:1  {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
00:06.367  ID:1  e::40a:0:0:0:0:0
00:06.369  ID:1  Publicando cada 5 segundos
00:06.406  ID:1  Recibi: {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
00:06.760  ID:2  Recibi: {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
00:11.209  ID:2  {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":97}
00:11.211  ID:2  e::40a:0:0:0:0:0
00:11.213  ID:2  Publicando cada 5 segundos
00:11.383  ID:1  {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":159}
00:11.384  ID:1  e::40a:0:0:0:0:0
00:11.386  ID:1  Publicando cada 5 segundos
00:11.423  ID:1  Recibi: {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":97}
00:11.687  ID:2  Recibi: {"mote_id":"linti_servidores","temperature":65490.65451,"current":0,"movement":0,"voltage":159}
```

Figura 28. Salida de las motas dispuestas para la primer aplicación de Adaptivesec.

Luego de probar WENU con una configuración mínima de Adaptivesec, se plantean las siguientes modificaciones:

Las dos motas establecidas anteriormente dejan de imprimir los datos recibidos. Se agrega una tercer mota, la cual no sensa ningún dato, sólo se limita a imprimir en su

salida estándar las mediciones que recibe. Dado que el flujo de mensajes será desde las motas sensoras hacia la receptora, se envían las mediciones en mensajes unicast, dando lugar al problema de cómo obtener la ip de la mota que actuará como receptora de las mediciones. Para esto, se hace uso de una aplicación llamada servreg-hack, la cual se asimila a un pequeño resolver DNS y viene integrada en Contiki, pudiéndose invocar agregando las siguientes líneas en nuestra aplicación:

```
3  APPS=servreg-hack
```

En el makefile

```
12  #include "servreg-hack.h"
```

En las motas

La mota receptora, en un proceso aparte y respetando el intervalo establecido en un timer, asocia su dirección IP con un número de servicio:

```
100  servreg_hack_register(SERVICE_ID, ipaddr);
```

Lo que provoca el envío de mensajes broadcast cada cierta cantidad de segundos (los que se pre establezcan) desde ella informando su dirección IP:

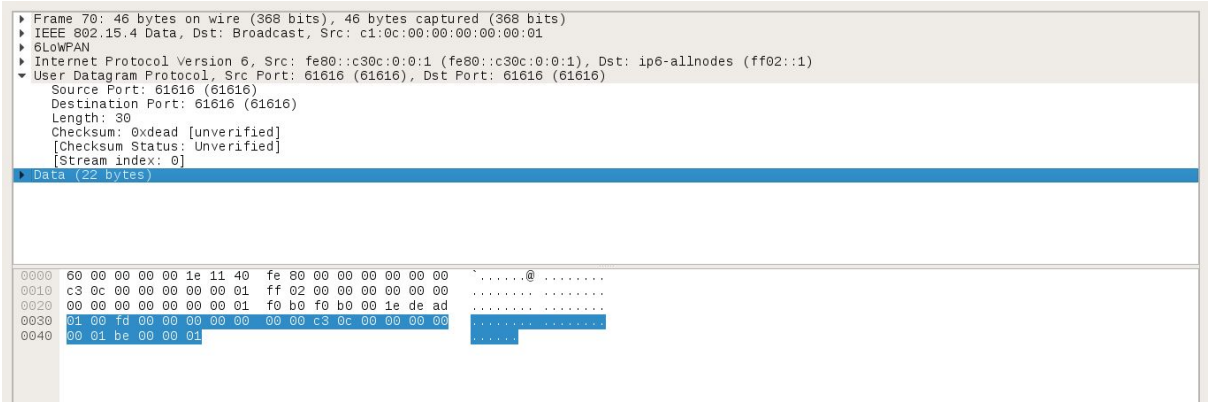


Figura 29. Datagrama UDP asociado a la aplicación servreg-hack. En el campo datos, se incluye la dirección IP de la mota que registra el servicio (en este caso, la mota que recibe las mediciones)

En cuanto a las motas que buscan la dirección IP, lo hacen con la siguiente instrucción:

```
137    addr = servreg_hack_lookup(SERVICE_ID);
```

Por último, se ajusta Adaptivesec en el nivel 3, es decir, se provee sólo servicio de autenticación para las motas con un MIC de 16 bytes. Sólo se modifica en el archivo de configuración de proyecto la siguiente línea:

```
3    #undef ADAPTIVESEC_CONF_UNICAST_SEC_LVL
4    #define ADAPTIVESEC_CONF_UNICAST_SEC_LVL 3
```

Puede observarse a continuación que la aplicación se sigue ejecutando de manera correcta, enviando en este caso dos motas las mediciones correspondientes (motas 2 y 3), e informando la última (mota 1) todo lo que recibe:

```
04:01.303  ID:1   Recibi: {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
04:01.898  ID:3   {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
04:01.900  ID:3   fd00::c30c:0:0:3
04:01.902  ID:3   Publicando cada 10 segundos
04:02.054  ID:1   Recibi: {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":0}
04:11.156  ID:2   {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":244}
04:11.158  ID:2   fd00::c30c:0:0:2
04:11.160  ID:2   Publicando cada 10 segundos
04:11.302  ID:1   Recibi: {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":244}
04:11.913  ID:3   {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":250}
04:11.915  ID:3   fd00::c30c:0:0:3
04:11.917  ID:3   Publicando cada 10 segundos
04:12.054  ID:1   Recibi: {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":250}
04:21.172  ID:2   {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":253}
04:21.174  ID:2   fd00::c30c:0:0:2
04:21.176  ID:2   Publicando cada 10 segundos
04:21.303  ID:1   Recibi: {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":253}
04:21.929  ID:3   {"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":84}
04:21.931  ID:3   fd00::c30c:0:0:3
04:21.933  ID:3   Publicando cada 10 segundos
```

Estas modificaciones a la aplicación pueden verse en los archivos mqtt-sensores-2.c y mqtt-sensores-2.z1 en su versión compilada.

Ya probadas con éxito las comunicaciones entre las motas que corren WENU, el siguiente paso es la integración de la mota (en el proyecto WENU es una mota Raspberry pi) que cumple funciones de border router. A tal fin, y manteniendo el objetivo primordial de securizar las comunicaciones dentro de la PAN, el primer paso fue intentar compilar 6lbr junto con Adaptivesec, exactamente con la última configuración utilizada en esta sección (es decir, implementando el nivel 3 del estándar de seguridad de 802.15.4). Luego de varios intentos y modificaciones a la configuración de Adaptivesec, no se pudo concretar que 6lbr compile de manera

exitosa. Consultando a uno de los autores del proyecto 6lbr [37], se pudo saber que la integración de Adaptivesec dentro de 6lbr no puede ser posible hasta que la librería esté integrada dentro del repositorio oficial de Contiki.

Dado este nuevo obstáculo para cumplir los objetivos de seguridad pautados en este trabajo, fue necesaria la modificación de la arquitectura de la aplicación, de manera tal que se siga cumpliendo con la finalidad del proyecto WENU, pero a la vez aprovechando todas las ventajas que en materia de seguridad en capa de enlace aporta Adaptivesec. Se mantuvieron los tipos de motas de WENU (es decir, las motas sensoras y las de control) pero no se hizo uso de una mota como border router. En su reemplazo se estableció una mota que informa por salida estándar las mediciones que recibe, además de mostrar de dónde provienen las mismas.

Se deja para una segunda etapa del proyecto WENU la salida de la PAN a Internet, planteada de la siguiente manera: la mota receptora de las mediciones se encontrará conectada por USB a una PC de escritorio, a la cual le enviará las mediciones recibidas, en formato JSON. Finalmente, esta PC cumplirá el rol de gateway recibiendo estos mensajes con un script en Python y reenviandolos al broker MQTT, por lo que no sólo que se mantiene la comunicación con el mundo exterior, sino que también se garantiza el funcionamiento de Adaptivesec en la PAN (ya que la mota receptora de las mediciones, como se demostró, lo ejecuta correctamente).

Esta nueva disposición puede observarse en los archivos fuente sensora.c (implementa la funcionalidad de la mota encargada de las mediciones, midiendo sólo temperatura si se tratara de una mota de control) y receptora.c (la mota que recibe las mediciones).
Ya integrado Adaptivesec en WENU, el siguiente paso en el presente trabajo de grado es plantear diversos escenarios donde quede demostrado el correcto funcionamiento de la librería de seguridad, además de ahondar en ciertos detalles de la misma.

Escenarios de prueba en un ambiente simulado

Comprobada ya la correcta integración de Adaptivesec en la aplicación sujeta a estudio, en esta sección se plantean distintas situaciones en donde la red que corre WENU se ve comprometida ante una amenaza. Se muestra un resumen de cada escenario, junto con una explicación de cómo se evita la amenaza en cuestión.

Escenario 1: prueba de autenticación de las motas

Se despliegan tres motas en la red, donde dos cumplen el rol de sensar las mediciones correspondientes y enviarlas a la tercera, que simplemente imprime en la salida estándar los mensajes que recibe.

Se pueden apreciar dos estados de la red. En el primero se ven las tres motas comunicándose sin ningún inconveniente. En el segundo, se agrega una nueva mota que mide y envía mensajes a la mota receptora. Sin embargo, a diferencia de las otras motas, esta última tiene cargada en su esquema de predistribución una clave distinta a la que tienen las demás, por lo que le resulta imposible establecer comunicaciones con el resto de la red.

Se configura Adaptivesec en el nivel de seguridad 3. El mismo está especificado en la línea cuatro del archivo de configuración de proyecto de Contiki:

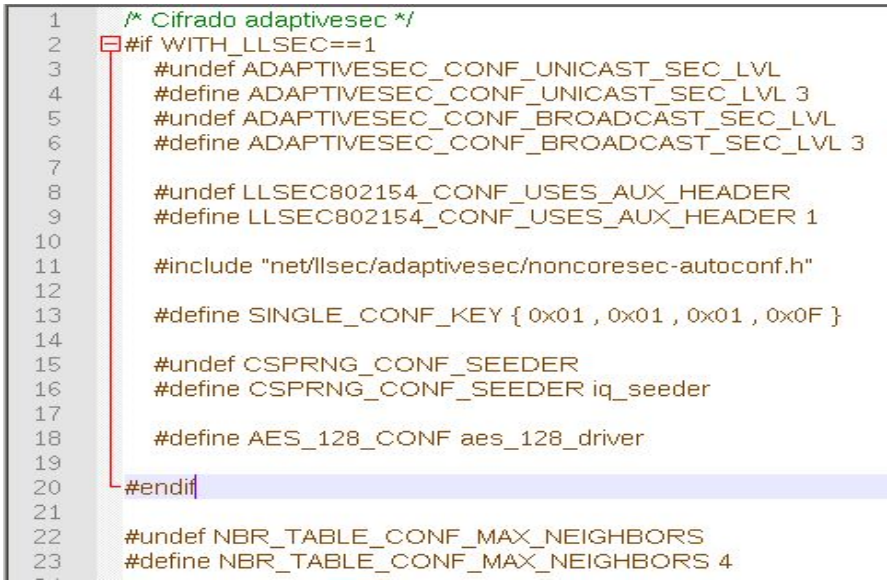


Figura 30. Configuraciòn de Adaptivesec en el escenario 1.

En la línea 13 está especificada la clave que se carga en el esquema de pre-distribución de cada mota. Cabe recordar que el utilizado en este trabajo es el esquema de predistribución de clave compartida, donde todos los nodos de la red son configurados con la misma clave para implementar la seguridad de 802.15.4. Esta configuración se carga en las tres primeras motas de la red:

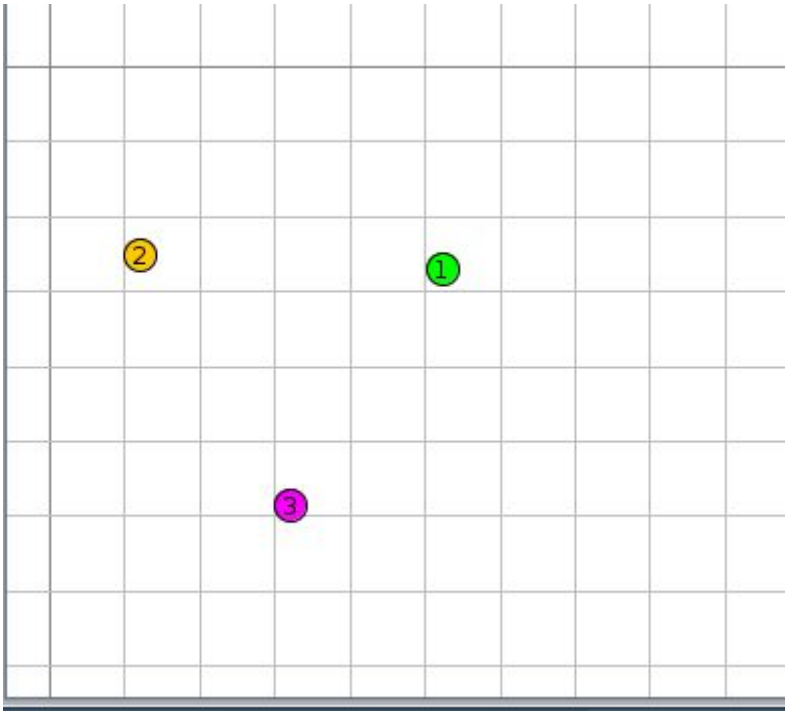


Figura 31. Estado inicial de la red del escenario 1.

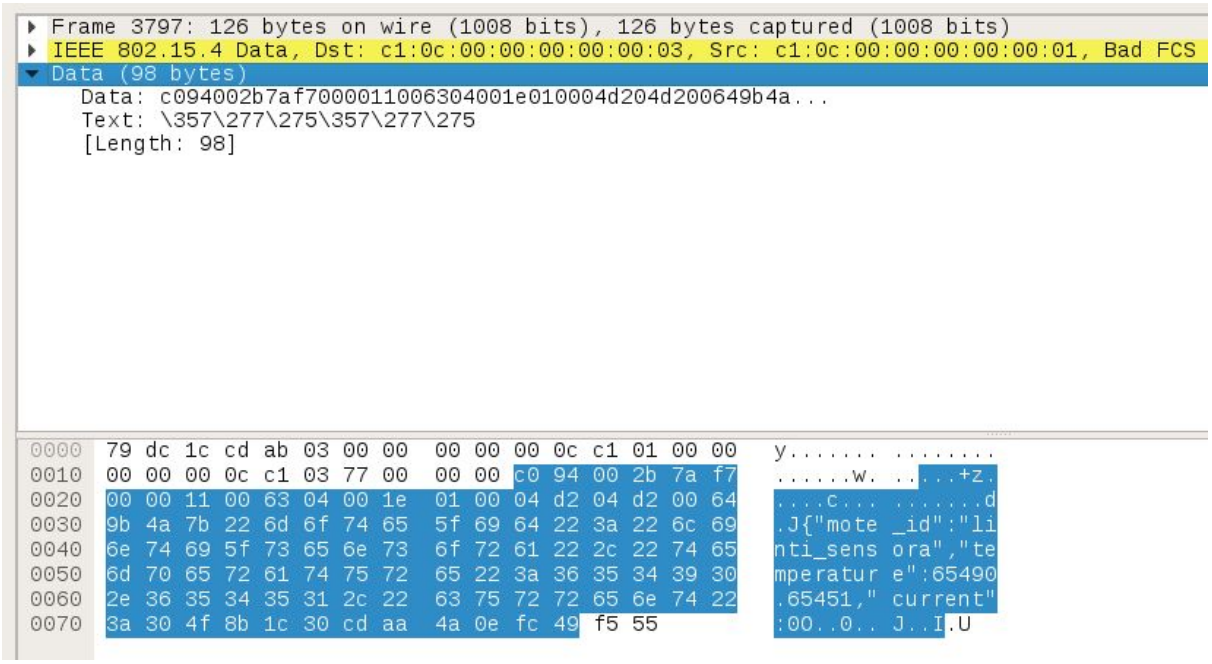
Observando lo que imprime en salida estándar cada mota, se ve que las tres primeras se comunican adecuadamente:

```
03:20.884  ID:2   Enviando mensaje a:fd00::c30c:0:0:3
03:21.075  ID:3   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 91: '{"mote_id":"linti_control","temperature":65490...
03:26.889  ID:1   Enviando mensaje a:fd00::c30c:0:0:3
03:27.075  ID:3   Datos recibidos de fd00::c30c:0:0:1 en el puerto 1234 desde el puerto 1234 con longitud 91: '{"mote_id":"linti_sensora","temperature":65490...
03:36.829  ID:2   Enviando mensaje a:fd00::c30c:0:0:3
03:36.952  ID:3   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_control","temperature":65490...
03:44.202  ID:1   Enviando mensaje a:fd00::c30c:0:0:3
03:44.324  ID:3   Datos recibidos de fd00::c30c:0:0:1 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
03:46.470  ID:2   Enviando mensaje a:fd00::c30c:0:0:3
03:46.578  ID:3   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_control","temperature":65490...
04:02.866  ID:1   Enviando mensaje a:fd00::c30c:0:0:3
04:03.078  ID:3   Datos recibidos de fd00::c30c:0:0:1 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
04:04.900  ID:2   Enviando mensaje a:fd00::c30c:0:0:3
04:05.076  ID:3   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_control","temperature":65490...
04:10.765  ID:1   Enviando mensaje a:fd00::c30c:0:0:3
04:10.950  ID:3   Datos recibidos de fd00::c30c:0:0:1 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
```

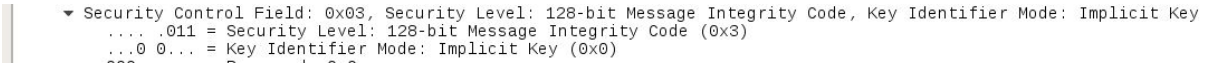
Figura 32. Salida estándar de las tres primeras motas en el escenario 1.

Además, utilizando la herramienta del simulador que permite hacer sniffing sobre todos los mensajes intercambiados por los nodos de la red, hay dos cuestiones que merecen ser resaltadas:

- El payload de los paquetes no se encripta, respetando de esta forma el nivel de seguridad elegido :



- El nivel de seguridad y el modo de identificación de de la clave (en este caso es el modo 0, que indica que la clave está implícita en el emisor y en el receptor) también se presentan acorde a lo esperado:



Ahora se presenta el segundo estado de la red, donde se agrega una mota (la número 4) cuya clave compartida es distinta a la de las motas 1,2 y 3:

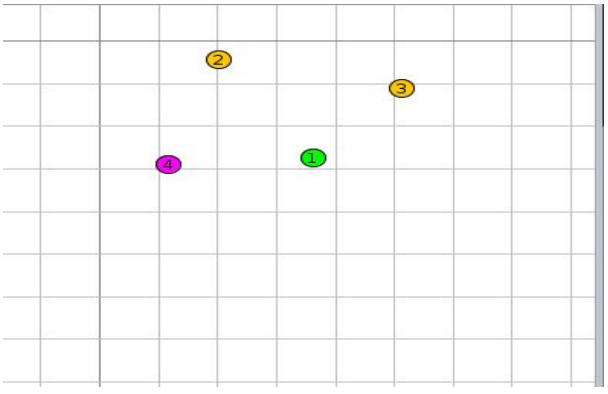


Figura 33. Segundo estado de la red del escenario 1.

```
13    #define SINGLE_CONF_KEY { 0x02 , 0x03 , 0x01 , 0x0F }
```

Figura 34. Clave compartida de la mota cuatro, la cual difiere de la del resto.

Por lo que no puede establecer ninguna comunicación de capa 2 con la red. En la solución adoptada en este trabajo, la mota queda iterando en un loop informando que no pudo encontrar el servicio asociado a la dirección IP de la mota receptora:

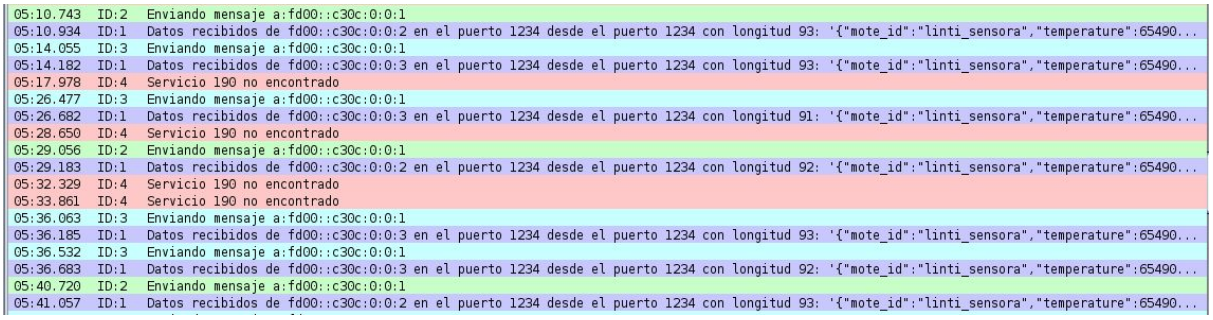


Figura 35. Mota 4 tratando de entablar comunicaciones con la mota receptora.

El escenario que sigue demuestra la implementación del cifrado de capa de enlace que ofrece Adaptivesec.

Escenario 2: prueba del cifrado de los mensajes enviados

Al igual que en el anterior escenario, se despliegan tres nodos en la red, donde uno cumple el rol de receptor de los mensajes enviados por los otros dos. Cabe aclarar que el mensaje que se envía por parte de los emisores continúa siendo el string codificado en JSON que contiene las mediciones tomadas por cada mota.
En primer lugar, las motas sensoras envían las mediciones sin encriptar. Con una simple modificación, se ve cómo las mismas envían sus mensajes impidiendo que un atacante pueda leer el contenido real de los mismos.

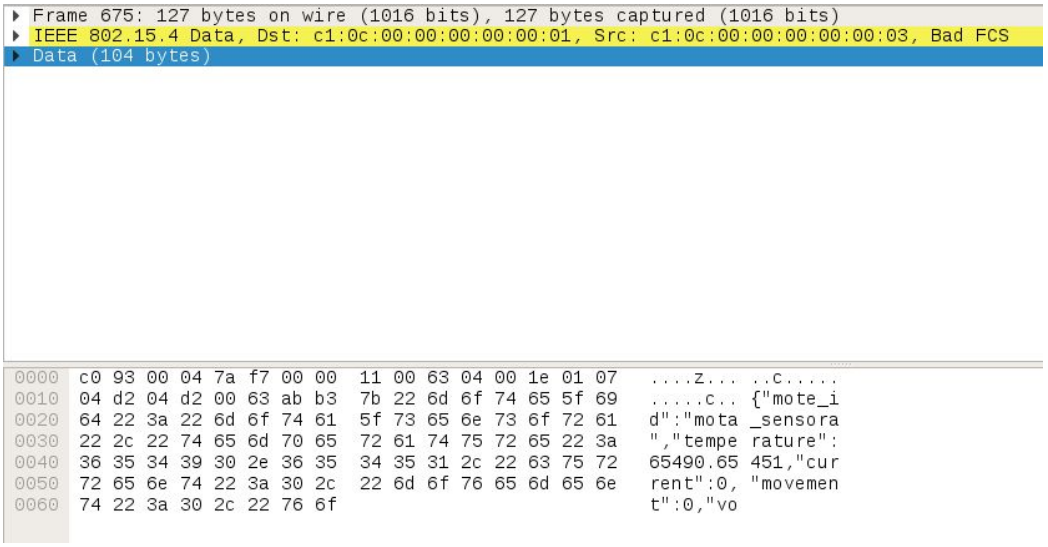


Figura 36. Mensaje sin cifrar desde un nodo emisor (en este caso la mota 3) al receptor.

Para hacer efectivo el uso del cifrado, se elige el nivel 5 de seguridad 802.15.4:

```
1   /* Cifrado adaptivesec */
2   #if WITH_LLSEC==1
3     #undef ADAPTIVESEC_CONF_UNICAST_SEC_LVL
4     #define ADAPTIVESEC_CONF_UNICAST_SEC_LVL 5
5     #undef ADAPTIVESEC_CONF_BROADCAST_SEC_LVL
6     #define ADAPTIVESEC_CONF_BROADCAST_SEC_LVL 5
7
8     #undef LLSEC802154_CONF_USES_AUX_HEADER
9     #define LLSEC802154_CONF_USES_AUX_HEADER 1
10
11    #include "net/llsec/adaptivesec/noncoresec-autoconf.h"
12
13    #define SINGLE_CONF_KEY {'C','C','C','\0'}
14
15    #undef CSPRNG_CONF_SEEDER
16    #define CSPRNG_CONF_SEEDER iq_seeder
17  #endif
18
19  //Falla sin esto
20  #define LPM_CONF_MAX_PM 1
21
22  /* Z1 platform has limited RAM */
23
24  #undef NBR_TABLE_CONF_MAX_NEIGHBORS
25  #define NBR_TABLE_CONF_MAX_NEIGHBORS 7
```

Figura 37. Archivo de configuración de proyecto para el escenario 2.

Por lo tanto, la configuración elegida para este escenario asegura no sólo la autenticidad de las componentes de la red, sino que además la confidencialidad de los mensajes intercambiados:

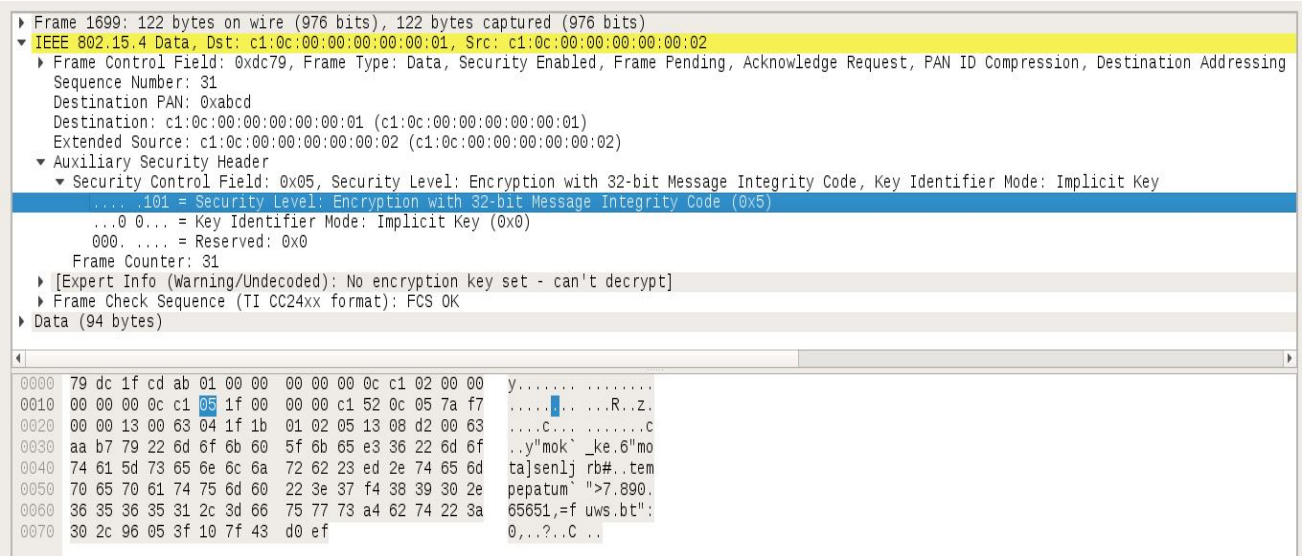


Figura 38. Paquete de datos medidos cifrado.

Se puede observar que, con respecto al escenario anterior, además de haber variado el payload del paquete también varió el campo security level, mostrándose el nivel cinco. En cuanto al campo Key Identifier Mode, nunca varía, ya que las claves para el cifrado son conocidas implícitamente por el emisor y el receptor.
Si se quisiera un MIC de mayor longitud, sólo habría que incrementar el nivel de Adaptivesec en el archivo de configuración del proyecto.
Vale aclarar, que Adaptivesec forma los nonces de la siguiente manera. Los primeros 8 bytes contienen la información de direccionamiento de un emisor *u*. Después, sigue el frame counter $C_{u,v}$ o $C_{u,*}$, dependiendo si *u* envía un frame unicast o broadcast. Por último, sigue el índice $I_{v,u}$ o, en el caso de un frame broadcast, 0xFF (figura 39).

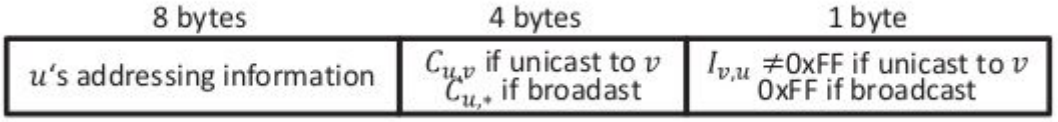


Figura 39. Formato del nonce en Adaptivesec.

Escenario 3: reinicios de las motas y ataques anti replay

Como se explicó en el presente trabajo, la implementación de seguridad de capa de enlace ofrecida por Contiki presenta falencias significativas, una de las cuales engloba a los reinicios de las motas y a los ataques anti replay. Por lo tanto, en este escenario se muestra cómo gestionan la información anti replay las dos implementaciones tratadas en el trabajo, tratando de arribar a una conclusión acerca de cuál es la más adecuada para esta situación.

Se plantea una simulación con dos motas, donde una cumple el rol de sensora y la otra de receptora. A diferencia de los escenarios anteriores, las motas corren la rama principal de la versión oficial de Contiki [41].

Por el lado de la capa de seguridad, se configuran ambas motas en el nivel 3 del estándar 802.15.4. Además, cabe recordar que la implementación de Contiki sólo ofrece el esquema de predistribución de clave compartida, por lo que no cambiará el esquema utilizado en los anteriores escenarios.

```
1  #if WITH_LLSEC==1
2      #define AES_128_CONF aes_128_driver
3      #undef LLSEC802154_CONF_ENABLED
4      #define LLSEC802154_CONF_ENABLED       1
5      #undef NETSTACK_CONF_FRAMER
6      #define NETSTACK_CONF_FRAMER          noncoresec_framer
7      #undef NETSTACK_CONF_LLSEC
8      #define NETSTACK_CONF_LLSEC          noncoresec_driver
9      #undef NONCORESEC_CONF_SEC_LVL
10     #define NONCORESEC_CONF_SEC_LVL       3
11     #define NONCORESEC_CONF_KEY "dM9KGhSvO6"
12
13 #endif
14
15 #undef NBR_TABLE_CONF_MAX_NEIGHBORS
16 #define NBR_TABLE_CONF_MAX_NEIGHBORS 4
```

Figura 40. Configuración de seguridad de capa de enlace para el escenario 3.

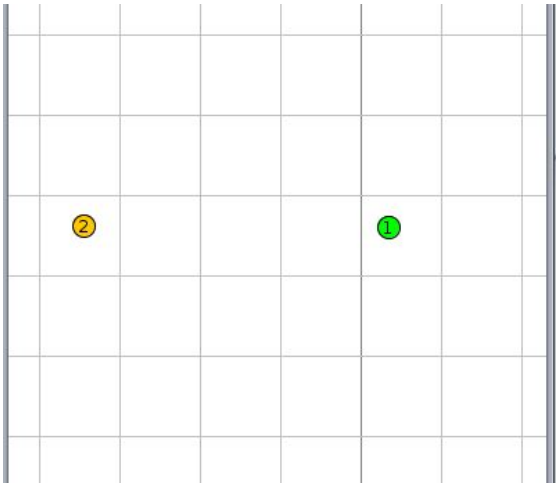


Figura 41. Motas receptora (1) y sensora (2) para el escenario 3.

Se corre la simulación por tres minutos. Luego, con el objetivo de simular el reinicio de la mota número 2, se elimina la misma y se vuelve a agregar en el minuto 5 , analizando cómo la red reacciona ante dicha situación:

```
00:20.000  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
00:20.186  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 91: '{"mote_id":"linti_sensora","temperature":65490...
00:33.352  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
00:33.558  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
00:53.243  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
00:53.433  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
01:04.922  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
01:05.059  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
01:17.281  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
01:17.433  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
01:19.188  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
01:19.310  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
01:34.570  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
01:34.683  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
01:53.195  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
01:53.308  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
01:58.656  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
01:58.808  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
02:09.500  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
02:09.686  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
02:16.266  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
02:16.432  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
02:18.828  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
02:18.936  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
02:37.742  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
02:37.933  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
02:50.977  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
02:51.183  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
02:57.125  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
02:57.311  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
03:02.906  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
03:03.058  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
03:04.664  ID:2  Enviando mensaje a:fd00::c30c:0:0:1
03:04.811  ID:1  Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 92: '{"mote_id":"linti_sensora","temperature":65490...
```

Figura 42. Salida de las motas en lo primeros tres minutos de simulación. Escenario 3.

En el minuto número cinco, se vuelve a agregar la mota 2. La misma, nuevamente envía mediciones a la mota receptora. Sin embargo, dado que el frame counter de la mota 2 es menor que el de la mota 1, los mensajes son rechazados por esta última. Esto se debe a que debido al reinicio de la mota, el frame counter se vió afectado y

no pudo mantener su valor anterior al reinicio. Como consecuencia, la mota receptora aceptará los mensajes de la sensora solamente cuando el frame counter de esta última sea mayor al de la primera.
Una posible solución a este problema puede ser el envío a nivel aplicación de mensajes de reconocimiento por parte del receptor, que sean procesados por la mota sensora para evaluar si continúa enviando mediciones. Sin embargo, esto demandaría un mayor tráfico de mensajes y un overhead adicional por el procesamiento de los mismos en ambos puntos de la comunicación.

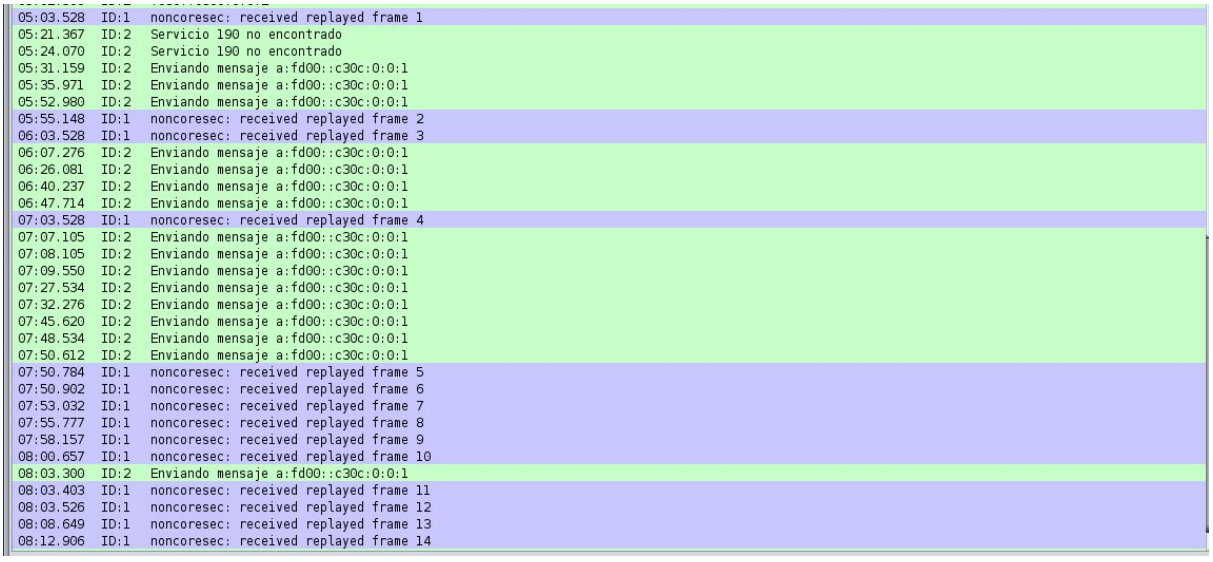


Figura 43. Mota 1 rechazando frames reenviados de la mota 2.

Además, debe tenerse en cuenta que es probable que el frame counter de la mota 2 nunca llegue a superar al de la mota 1. Dado que esta nunca deja de enviar mensajes con las mediciones a la mota receptora , tiene un significativo gasto energético, pudiendo provocar que la mota se apague. Como se dió a entender en este trabajo, los dispositivos IoT deben hacer énfasis en esta cuestión debido a las grandes limitaciones de recursos que presentan, por lo que el enfoque de la implementación de seguridad en capa de enlace de Contiki no es aceptable.

Como se explicó anteriormente, Adaptivesec gestiona los reinicios y la movilidad de los nodos de la red haciendo que cada nodo almacene información de sus vecinos (tabla 2). A continuación se demuestra cómo reacciona ante la situación presentada en esta sección.

```
1   /* Cifrado adaptivesec */
2   #if WITH_LLSEC==1
3     #undef ADAPTIVESEC_CONF_UNICAST_SEC_LVL
4     #define ADAPTIVESEC_CONF_UNICAST_SEC_LVL 3
5     #undef ADAPTIVESEC_CONF_BROADCAST_SEC_LVL
6     #define ADAPTIVESEC_CONF_BROADCAST_SEC_LVL 3
7
8     #undef LLSEC802154_CONF_USES_AUX_HEADER
9     #define LLSEC802154_CONF_USES_AUX_HEADER 1
10
11    #include "net/llsec/adaptivesec/noncoresec-autoconf.h"
12
13    #define SINGLE_CONF_KEY "XWrmtBomqB"
14
15    #undef CSPRNG_CONF_SEEDER
16    #define CSPRNG_CONF_SEEDER iq_seeder
17
18    #define AES_128_CONF aes_128_driver
19
20  #endif
21
22  #undef NBR_TABLE_CONF_MAX_NEIGHBORS
23  #define NBR_TABLE_CONF_MAX_NEIGHBORS 4
```

Figura 44. Configuración de seguridad de capa de enlace (Adaptivesec) para el escenario 3.

Nuevamente se correrá la simulación por tres minutos, con un posterior reinicio de la mota sensora:

```
00:03.151  ID:1   akes-nbr: prolonging
00:03.155  ID:1   akes-nbr: Neighbor C10C000000000002
00:03.160  ID:1   akes-nbr: Group session key: 66e94bd4ef8a2c3b884cfa59ca342b2e
00:03.252  ID:2   akes-nbr: prolonging
00:03.255  ID:2   akes-nbr: Neighbor C10C000000000001
00:03.261  ID:2   akes-nbr: Group session key: 66e94bd4ef8a2c3b884cfa59ca342b2e
00:04.886  ID:2   akes-nbr: prolonging
00:07.779  ID:1   akes-nbr: prolonging
00:08.657  ID:1   akes-nbr: prolonging
00:09.879  ID:2   akes-nbr: prolonging
00:10.111  ID:2   Enviando mensaje a:fd00::c30c:0:0:1
00:10.282  ID:1   akes-nbr: prolonging
00:10.295  ID:1   akes-nbr: prolonging
00:10.310  ID:1   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 91: '{"mote_id":"linti_sensora","temperature":65490...
00:12.634  ID:2   akes-nbr: prolonging
00:14.655  ID:1   akes-nbr: prolonging
00:15.532  ID:1   akes-nbr: prolonging
00:16.778  ID:1   akes-nbr: prolonging
00:24.275  ID:2   Enviando mensaje a:fd00::c30c:0:0:1
00:24.410  ID:1   akes-nbr: prolonging
00:24.424  ID:1   akes-nbr: prolonging
00:24.439  ID:1   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
00:28.638  ID:2   akes-nbr: prolonging
00:31.909  ID:1   akes-nbr: prolonging
00:32.780  ID:1   akes-nbr: prolonging
00:37.626  ID:2   Enviando mensaje a:fd00::c30c:0:0:1
00:37.781  ID:1   akes-nbr: prolonging
00:37.795  ID:1   akes-nbr: prolonging
00:37.810  ID:1   Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 93: '{"mote_id":"linti_sensora","temperature":65490...
00:44.248  ID:2   akes-nbr: prolonging
00:44.769  ID:1   akes-nbr: prolonging
00:57.517  ID:2   Enviando mensaje a:fd00::c30c:0:0:1
00:57.657  ID:1   akes-nbr: prolonging
00:57.671  ID:1   akes-nbr: prolonging
```

Datos recibidos de fd00::c30c:0:0:2 en el puerto 1234 desde el puerto 1234 con longitud 91: '{"mote_id":"linti_sensora","temperature":65490.65451,"current":0,"movement":0,"voltage":0}'

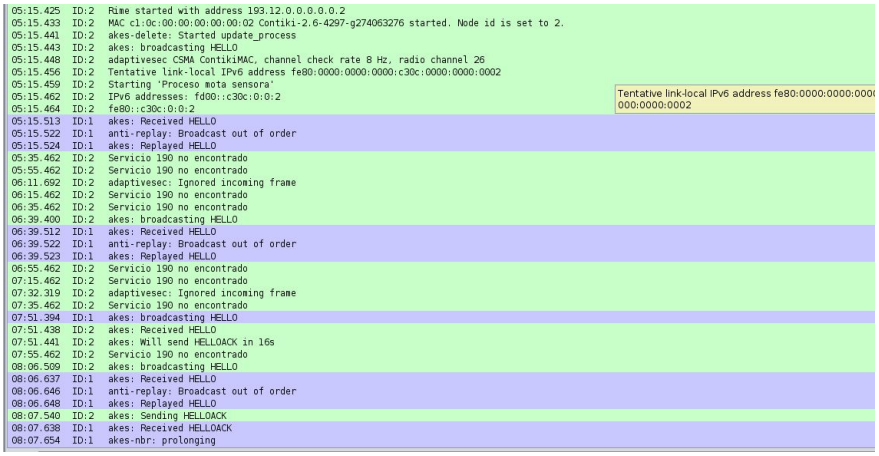
Figura 45. Salida de las motas en lo primeros tres minutos de simulación utilizando Adaptivesec.

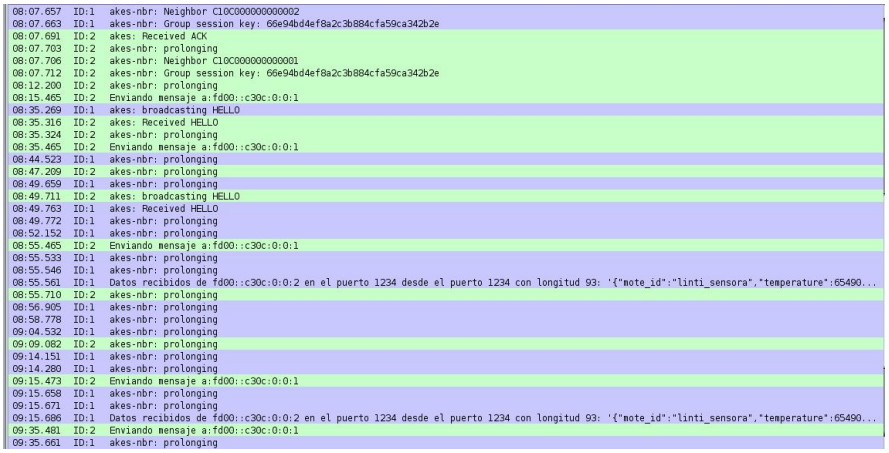
```
04:24.308  ID:1   akes: broadcasting HELLO
05:01.261  ID:1   akes-delete: #permanent = 1
```

Figura 46. Mota receptora eliminando a la mota sensora como vecino permanente.

La mota receptora, al vencer el tiempo de vida de la mota 2 (que no se renueva ya que al no estar más presente en la red, la mota receptora no recibe ningún otro mensaje de la mota 2), envía un mensaje UPDATE a la sensora para chequear si todavía se encuentra en su rango. Al no recibir respuesta alguna, la elimina de su lista de vecinos permanentes, lo cual implica que no es posible que se envíen frames de datos.

Después de agregar nuevamente la mota 2, el comportamiento de ambas es distinto con respecto a la anterior implementación:





Una vez reiniciada, la mota sensora intenta nuevamente entablar la clave de sesión entres pares para cifrar la clave compartida de grupo. A tal fin, envía comandos HELLO entramados en frames broadcast. Sin embargo, la mota 1 los descarta ya que su frame counter de mensajes broadcast es mayor que el de los mensajes que recibe. Esto cambia cuando la mota receptora comienza la negociación para establecer la clave de sesión entre pares y, de esa manera, también volver a agregar a la mota sensora como vecino permanente. De igual manera, la mota sensora también agrega a la receptora como vecino permanente cuando recibe un mensaje ACK. Una vez hecho esto, y que la mota 2 encuentre la IP de la receptora, ambas pueden volver a comunicarse.

Es importante notar que, a diferencia de la primer implementación mostrada en este escenario, ambos puntos de la comunicación son conscientes de la situación ocurrida. La mota sensora, por un lado, tendrá como prioridad reconocerse como vecinos permanentes con la receptora antes de continuar enviándole mediciones. De esa manera, se asegura que sus frames no serán descartados ya que se sincronizan los frame counters asociados a la comunicación entre ambas. Por el lado de la mota receptora, será la encargada de comenzar la negociación para establecer una nueva clave de sesión entre pares.
En consecuencia, se reduce considerablemente la cantidad de mensajes intercambiados, lo que lleva a un significativo ahorro de energía en ambos dispositivos (ya que la mota sensora no envía mediciones hasta que su frame counter esté sincronizado con el de la receptora; y esta última no debe procesar los mensajes para verificar si son reenviados, como sí pasaba con noncoresec). De esa manera, se tiene una reacción más rápida frente al reinicio de una mota, sin por ello perder la protección anti replay.

Implementación

Ya presentados los servicios de seguridad que provee la librería introducida en este trabajo, y planteados posibles escenarios de uso, se debe decidir qué configuración final de Adaptivesec es la que adoptará WENU. Teniendo en cuenta que no se transfiere información crítica desde las motas sensoras hacia la receptora, como por ejemplo claves, y que a nivel aplicación la mota receptora no envía ningún mensaje a las demás, se descarta el uso de cifrado de los paquetes UDP.
Por otro lado, sí se desea que un conjunto predefinido de motas sea el único autorizado para enviar mediciones, por lo que el nivel 3 del estándar 802.15.4 que implementa Adaptivesec parece ser el más apropiado para este requerimiento; ya que sólo intervendrán en la comunicación mota sensora -> mota receptora las que posean la clave compartida en toda la red; lográndose autenticidad e integridad de los mensajes enviados.

```
#if WITH_LLSEC==1
  #undef ADAPTIVESEC_CONF_UNICAST_SEC_LVL
  #define ADAPTIVESEC_CONF_UNICAST_SEC_LVL 3
  #undef ADAPTIVESEC_CONF_BROADCAST_SEC_LVL
  #define ADAPTIVESEC_CONF_BROADCAST_SEC_LVL 3

  #undef LLSEC802154_CONF_USES_AUX_HEADER
  #define LLSEC802154_CONF_USES_AUX_HEADER 1

  #include "net/llsec/adaptivesec/noncoresec-autoconf.h"

  #define SINGLE_CONF_KEY "XWrmtBomqB"

  #undef CSPRNG_CONF_SEEDER
  #define CSPRNG_CONF_SEEDER iq_seeder

  #define AES_128_CONF aes_128_driver

#endif

#undef NBR_TABLE_CONF_MAX_NEIGHBORS
#define NBR_TABLE_CONF_MAX_NEIGHBORS 4
```

Figura 47. Configuración final de Adaptivesec en WENU.

Conclusiones

El objetivo de este trabajo ha sido establecer los requerimientos de seguridad de una aplicación IoT, para luego proponer alternativas que los resuelvan. Se centró el estudio de estos requerimientos en la capa de enlace de la aplicación en cuestión.

Para cumplir dicho objetivo; se presentaron en primer lugar los principales conceptos y protocolos de IoT, con un enfoque descendente por las capas de su arquitectura. Posteriormente a esto, se plantearon los aspectos de seguridad generales de una aplicación IoT, arribando a la conclusión que estos no varían significativamente en cuanto a los de una aplicación de Internet.

Buscando aportar un marco teórico y comprender cómo se implementan todos los servicios para securizar una aplicación, se abordó la sección de seguridad del estándar IEEE 802.15.4. Se mostraron los distintos niveles de seguridad de la misma, detallando los servicios que cada uno implementa

Se presentó en detalle el proyecto WENU y se especificó qué déficits presenta el mismo en materia de seguridad en la capa de enlace.

Para resolver los déficits planteados, se presentan dos enfoques. El primero, implementado en la versión oficial de Contiki, a pesar de implementar correctamente los servicios de autenticación y cifrado presenta tres problemas:

1. Sólo un esquema de predistribución de claves permitido: el esquema de clave compartida en toda la red. Si se consigue la clave en una mota, toda la red está comprometida (ya que todas comparten la misma clave).

2. Si una mota se reinicia, todos sus frame counters también lo hacen. Si esta quiere volver a comunicarse con algún vecino, éste lo va a impedir ya que considerará como reenviados los frames que la mota reiniciada envíe.

3. Lo anterior ocurre también cuando un frame counter alcanza su máximo valor (por ejemplo, 2^{32} si se trata de un contador de 4 bytes). A pesar de ser un valor extremadamente alto, hay que tener en cuenta el tiempo de vida de los dispositivos IoT (meses o incluso años), por lo que esta vulnerabilidad no puede ser pasada por alto.

La segunda implementación, denominada Adaptivesec, resuelve las falencias que presenta el enfoque de Contiki, por lo que se optó por esta opción para resolver los requerimientos de seguridad de WENU.

La aplicación de Adaptivesec en WENU no resultó una cuestión trivial. La imposibilidad de utilizarlo en la mota pensada para ejecutar 6LBR causó una reestructuración de la aplicación, con las siguientes modificaciones:

1. La mota Raspberry pi que ejecuta 6LBR se reemplazó por una mota z1 que recibe las mediciones y las imprime en salida estándar.

2. Al no contar con el broker necesario para el uso de MQTT, se decidió no hacer uso de este protocolo para el envío de las mediciones, por lo que éstas se enviarán directamente sobre el protocolo de capa de transporte UDP.

Luego de conseguir integrar la aplicación WENU con la librería estudiada, se presentaron escenarios de prueba en un ambiente de simulación controlado, de manera de prescindir del hardware de las motas. En cada uno de ellos, se planteó una posible amenaza a la red, que afecte a la integridad o a la confidencialidad de los frames intercambiados. Además, se especificó cómo Adaptivesec mitiga estas amenazas mostrando en cada caso la configuración necesaria para su funcionamiento.

Finalmente, se presentó qué configuración de Adaptivesec será la utilizada en esta primera etapa del proyecto WENU; fundamentando por qué se eligió la misma y qué servicios de seguridad aportará.

Así pues, se han obtenido los requerimientos de seguridad de una aplicación de IOT, analizando cada uno y tomando dimensión de los riesgos que implica para la misma no cumplirlos. Para esto se barajaron dos alternativas, eligiendo una por las razones fundamentadas en este trabajo y demostrando mediante casos de prueba que la misma efectivamente cumple con los requerimientos de seguridad de la red en cuestión.
Si bien no se contempla la presencia de un gateway en la red para obtener salida a Internet, y tampoco se trabajó con un protocolo de capa de aplicación; se propone, por un lado, una alternativa para que la librería utilizada sea capaz de securizar también la mota que cumple funciones de border router. Por otro lado, el uso de un protocolo de capa de aplicación no debiera suponer una dificultad adicional para el

uso de la librería empleada, por lo que es totalmente loable extender la arquitectura de WENU en un futuro.

Aportes

En base al trabajo realizado se diseñó un taller de seguridad de IOT en el marco del curso de doctorado de IOT dictado el pasado mes de septiembre en nuestra facultad, en base a las pruebas realizadas en el marco de la presente tesina. Esto representa un paso inicial para el grupo de investigación y docencia en seguridad en la temática de seguridad en IOT.

Trabajos futuros

Tres propuestas toman protagonismo si se piensa en futuras líneas de investigación. En primer lugar, la integración de un protocolo de capa de aplicación para la gestión de mensajes. MQTT y COAP, presentados en este trabajo, asoman como posibles alternativas dadas las características de WENU. Es posible también investigar sobre mecanismos de seguridad que estos protocolos ofrezcan, de manera de cumplir con los requerimientos planteados en este trabajo no sólo en capa de enlace.
En segundo lugar, implementar la alternativa propuesta en este trabajo para que la aplicación tenga salida a Internet, asegurando a la vez a la mota que cumpla funciones de border router con Adaptivesec. La misma consiste en un pequeño script escrito en python para transferir, mediante puerto serial, las mediciones que recibe la mota receptora hacia el Gateway directamente conectado a ella. De esa manera se respetaría la arquitectura original de WENU.
Finalmente, sería deseable poder implementar el esquema de predistribución de clave de a pares en las motas Zolertia z1. De esa manera, se tendría una solución alternativa para securizar la red, menos eficiente pero a la vez más confiable que la utilizada en este trabajo.

Apéndice

Tabla 1: notaciones

Símbolo	Significado
||	Concatenación
u->v	*u* envía un frame unicast a *v*
*u->**	*u envia un frame broadcast*
〈*X*〉	Contenido autenticado X
{X}	Contenido encriptado X
(X)	Contenido opcional X
[X|Y]	X o Y
AES-128(K,P)	Encriptación AES de un texto plano de 16 bytes P con una clave de 128 bits K.

Tabla 2: información almacenada por un nodo *u por cada vecino* v

Estado	Almacena si *v* es permanente o tentativo y, si es tentativo, si se envió un HELLOACK a *v.*
PAN_v	Identificador PAN de *v.*
ID_v	Dirección extendida, corta o simple de *v.*
E_v	Tiempo de vida de *v.*
$K'_{u,v}$	Clave de sesión entre pares de *u* y *v* - puede liberarse después del saludo de tres vías cuando sólo se usan claves de sesión de grupo.
$K_{v,*}$	Clave de sesión de grupo opcional de *v.*
$I_{u,v}$	Indice de *u en la lista de vecinos de v.*
$C_{v,u}$	Frame counter del último frame unicast aceptado de *v.*
$C_{u,v}$	Frame counter del último frame unicast saliente hacia *v.*
$C_{v,*}$	Frame counter del último frame broadcast aceptado de *v*.
H_v	Flag que almacena si *v* envió un HELLO auténtico y reciente desde la última vez que *u envió un* HELLO en un mensaje broadcast.

Bibliografía

[1] https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=Main.

[2] Antonio Liñán Colina, Alvaro Vives, Marco Zennaro, Antoine Bagula, Ermanno Pietrosemoli. Internet of Things IN 5 DAYS.

[3] Naveen Sastry, David Wagner. Security Considerations for IEEE 802.15.4 Networks. Universidad De California, Berkeley.

[4] Felix Büsching, Andreas Figur, Dominik Schürmann, and Lars Wolf. Utilizing Hardware AES Encryption for WSNs.

[5] James F. Kurose, Keith W. Ross. Redes de Computadoras, un enfoque descendente (quinta edición).

[6] http://www.wsnmagazine.com/aes-implementation-on-micaz-with-Contiki-os/.

[7] Ines Robles, Alvaro Retana. IoT Roadmap, lacnic25 (2/6 de mayo- La Habana, Cuba).

[8] Zach Shelby (Wiley). 6LoWPAN: The Wireless Embedded Internet.

.

[9] Osterlind, F.; Dunkels, A.; Eriksson, J.; Finne, N.; Voigt, T. "Cross-Level Sensor Network Simulation with COOJA".

.

[10] Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. in Communications Surveys & Tutorials, IEEE , vol.17, no.4, pp.2347-2376, Fourthquarter 2015. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications".

[11] IEEE Standard for Low-Rate Wireless Networks. IEEE Computer Society.

[12] Mihir Bellare, Joe Kilian, y Phillip Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. Journal of Computer and System Sciences, 61(3):362–399, December 2000.

[13] V. Rijmen, J. Daemen. The Block Cipher Rijndael.

[14] D. Whiting, R. Housley, y N. Ferguson. Counter with cbc-mac (ccm). RFC 3610, September 2003.

[15] Chi-Yuan Chen, Han-Chieh Chao. A survey of key distribution in wireless sensor networks.

[16] Konrad-Felix Krentz, Christoph Meinel. Handling reboots and mobility in 802.15.4 security.

[17] Konrad-Felix Krentz , Hosnieh Rafiee y Christoph Meinel. 6LoWPAN Security: Adding Compromise Resilience to the 802.15.4 Security Sublayer.

[18] P. Levis, T. Clausen, J. Hui, O. Gnawali y J. Ko. The Trickle Algorithm. RFC 6206, 2011.

[19] Uli Kretzschmar. AES128 – A C Implementation for Encryption and Decryption. Texas instruments, 2009.

[20] https://github.com/Zolertia/Resources/wiki/The-Z1-mote.

[21] http://www.Contiki-os.org/.

[22] Wenliang Du, Jing Deng, Yunghsiang S. Han y Pramod K. Varshney. A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks.

[23] https://es.wikipedia.org/wiki/RFID.

[24] https://es.wikipedia.org/wiki/TinyOS.

[25] René Hummen, Jens Hiller, Hanno Wirtz, Martin Henze, Hossein Shafagh, Klaus Wehrle. Communication and Distributed Systems, RWTH Aachen University, Germany. 6LoWPAN Fragmentation Attacks and Mitigation Mechanisms.

[26] https://mosquitto.org/.

[27] http://mqtt.org/.

[28] http://coap.technology/.

[29] https://www.influxdata.com/.

[30] https://github.com/huginn/huginn.

[31] https://www.w3schools.com/js/js_json_intro.asp.

[32] https://github.com/Contiki-os/Contiki/tree/master/core/net/llsec/noncoresec.

[33] https://github.com/kkrentz/Contiki/tree/master/core/net/llsec/adaptivesec.

[34] William Stallings. Organización y Arquitectura de Computadores, quinta edición.

[35] https://github.com/esar/Contiki-mqtt.

[36] https://github.com/cetic/6lbr/wiki.

[37] https://github.com/laurentderu.

[38] Haowen Chan, Adrian Perrig y Dawn Song. Random Key Predistribution Schemes for Sensor Networks.

[39] https://es.wikipedia.org/wiki/Ley_de_Moore.

[40] Alejandro Cama, Emiro De la Hoz, Dora Cama. Las redes de sensores inalámbricos y el Internet de las cosas, 2012.

[41] https://github.com/Contiki-os/Contiki.

[42] http://Contiki.sourceforge.net/docs/2.6/index.html.