Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Seguridad en redes de sensores

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Iván Frago Álvarez

DIRECTOR: Olga León Abarca

DATA: 03 de Julio del 2017

Títol: Seguridad en redes de sensores

Autor: Iván Frago Álvarez

Director: Olga León Abarca

Data: 3 de Julio del 2017

Resum

Las redes inalámbricas de sensores están en expansión debido a los avances en tecnologías inalámbricas y a sus múltiples aplicaciones en diferentes campos, como por ejemplo monitorización de procesos en entornos industriales, seguimiento de las constantes vitales de un paciente en aplicaciones de e-salud o para monitorizar el tráfico.

Uno de los aspectos fundamentales en el diseño estas redes es la seguridad de las comunicaciones. Debido a que usan un medio de transmisión abierto, es fácilmente accesible por un atacante y vulnerable, resulta imprescindible implementar mecanismos que permitan proteger la información transmitida frente a accesos no autorizados y verificar la identidad las partes involucradas en las comunicaciones. Por otro lado, los nodos sensores suelen ser dispositivos limitados en cuanto a capacidad de procesamiento, almacenamiento, etc. y no soportan gran parte de los mecanismos de seguridad tradicionales. Como consecuencia, la seguridad en redes de sensores constituye un reto.

En este proyecto se ha implementado y evaluado una red de sensores con dos nodos: un nodo sensor para la detección de presencia en un hogar y un nodo *router* que recibe la información recogida por los sensores, y la envía a un servidor externo. El escenario desplegado hace uso de distintos mecanismos de seguridad para proporcionar confidencialidad y autenticación tanto en la red de sensores como en la comunicación entre *router* y servidor.

Title: Seguridad en redes de sensores

Author: Iván Frago Álvarez

Director: Olga León Abarca

Date: July 3, 2017

Overview

Wireless sensor networks are rapidly expanding due to advances in wireless technologies but also due to their multiple applications in different fields, such as monitoring processes in industrial environments, monitoring patient's vital signs in e-health applications or traffic monitoring.

One fundamental aspect in the design of wireless sensor networks is the security of the communications established within the network. As in any wireless network, an open transmission medium is used, which opens door to attackers and a wide variety of threats. For this reason, it is essential to design mechanisms to protect the information against unauthorized access and verify the identity of the parties involved in a communication. On the other hand, nodes are limited devices in terms of processing, storage, etc. and therefore they do not support most traditional security mechanisms.

In this work we have implemented and evaluated a wireless sensor network for presence detection. The network is composed of two nodes: a sensor node which detects movements and sends reports to a coordinator node, which in turn sends this information to an external server. The deployed system makes use of security mechanisms that provide confidentiality and authentication in the sensor network and the communication between router and server.

Indice

Lista de Acrónimos

6LoWPAN: IPv6 over Low power Wireless Personal Area Networks.
AES: Advanced Encryption Standard
CA: Certification Authority
CBC: Cipher Block Chaining
CORS: Cross Origin Resource Sharing.
CTR: Counter.
DoS: Denial of Service.
FFD: Full Function Device
HTTP: HyperText Transfer Protocol.
HTTPS: HyperText Transfer Protocol Secure.
IDE: Integrated Development Environment.
IEEE: Institute of Electrical and Electronics Engineers.
ISM: Instrumentation, Scientific and Medical.
ITU: International Telecommunication Union
IV: Initialization Vector.
LAN: Local Area Network.
LLC: Logical Link Control
LR-WPAN: Low Rate Wireless Personal Area Network.
MAC: Message Authentication Code
MAC: Medium Access Layer
ND: Node Discover.
NPM Node Package Manager.
PAN: Personal Area Network.
PFS: Perfect Forward Secrecy.
RFD: Reduced Function Device.
URL: Uniform Resource Locator.
WiFi: Wireless Fidelity.
WBSN: Wireless Body Sensor Network
WLAN: Wireless Local Area Network.
WPAN: Wireless Personal Area Network.
WSN: Wireless Sensor Network.

1. Introducción

Las redes inalámbricas de sensores (*Wireless Sensor Networks* o WSNs) [1][2] han experimentado un gran crecimiento durante los últimos años debido a los avances en tecnologías inalámbricas y a sus múltiples aplicaciones en diferentes campos, como por ejemplo monitorización de procesos en entornos industriales o seguimiento de las constantes vitales de un paciente en aplicaciones de e-salud.

Uno de los aspectos fundamentales en el diseño de una WSN es la seguridad de las comunicaciones. En WSNs, como ocurre en todo tipo de redes inalámbricas, el medio de transmisión es abierto y fácilmente accesible por un atacante. Por dicho motivo, resulta imprescindible proteger la información transmitida frente a accesos no autorizados y verificar la identidad de las partes de una comunicación. Por otro lado, los nodos sensores que componen las WSNs son dispositivos limitados en cuanto a capacidad de procesamiento, almacenamiento, etc. y no soportan gran parte de los mecanismos de seguridad tradicionales.

En este proyecto se ha desplegado una red de sensores sencilla para la detección de presencia en un hogar y se han implementado mecanismos de seguridad básicos. Los nodos sensores recogen información sobre el movimiento y envían informes a un nodo central o nodo *router*.

Con el objetivo de convertir este trabajo en un proyecto más realista, se ha ampliado el escenario con un servidor externo, que podría representar el servidor de una empresa que ofrece servicios de detección de intrusiones en hogares La Figura 1.1 representa el esquema completo de este servicio. El nodo *router* se comunica, de forma segura, con el servidor, reportando cualquier tipo de actividad sospechosa en el hogar. En el caso que se considere necesario, la empresa podría tomar las medidas pertinentes, como avisar a la policía o alertar al usuario mediante una aplicación móvil.

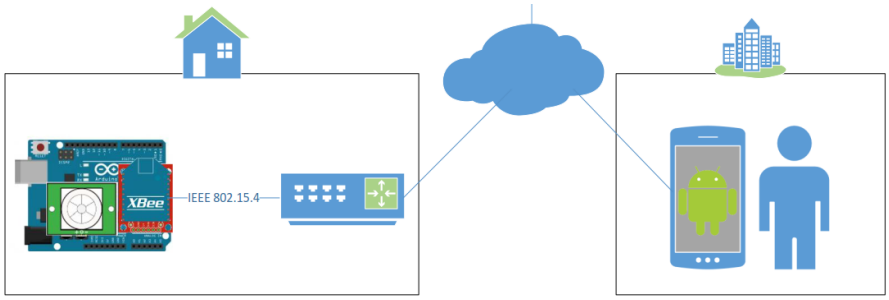

Figura 1.1 Escenario desplegado.

Por lo tanto, el escenario considerado está formado por dos partes: la **red de sensores**, compuesta por un nodo sensor y un *router* (en este proyecto se ha usado un ordenador portátil) que usan el protocolo IEEE 802.15.4 para comunicarse, y la **comunicación *Router*-Servidor**, a través de la red pública (Internet).

La tercera parte, correspondiente a la comunicación entre usuario y servidor a través de una aplicación Android, no se ha implementado en su totalidad se deja como futura línea de trabajo.

El documento se estructura de la siguiente forma. En el capítulo 2 se describen las características principales de las WSNs, las tecnologías existentes para este tipo de redes y los mecanismos de seguridad. En el capítulo 3, se presenta la implementación de la red de sensores realizada en este proyecto, especificando los dispositivos utilizados, su configuración y los mecanismos implementados. El capítulo 4 está dedicado a la implementación de la comunicación entre la red de sensores y el servidor. El capítulo 5 presenta los resultados de las pruebas realizadas en la red de sensores para comprobar el funcionamiento de la misma en función de la distancia y las interferencias. Finalmente, en el capítulo 6 se presentan las conclusiones de este trabajo.

2. Redes de sensores inalámbricas

2.1. Introducción

Una WSN es una red formada por un conjunto de dispositivos de bajo coste y consumo, que usan sensores para controlar diversas condiciones, como pueden ser la temperatura, la presión, el movimiento, etc.

Debido a su carácter inalámbrico, los nodos pueden distribuirse fácilmente en un área determinada para recoger información de su entorno, procesarla y enviarla a un nodo central de coordinación.

Durante los últimos años, este tipo de redes han experimentado un gran crecimiento, debido a sus múltiples áreas de aplicación. A pesar de que inicialmente se usaban con fines militares, actualmente se usan en muchos otros campos como son:

- Eficiencia energética: Monitorizar elementos industriales o eléctricos para hacer un uso más eficiente de la electricidad.
- Entornos de seguridad: Vigilar presencias u objetos sospechosos en zonas poco seguras.
- Control ambiental: Monitorizar el ambiente para poder hacer predicciones meteorológicas.
- Sensores industriales: Monitorizar cadenas de producción para hacerlas más eficientes o detectar algún tipo de fallo.
- Automoción: Recoger y distribuir información sobre tráfico para conseguir una circulación de coches más eficiente y evitar posibles accidentes.
- Medicina: Controlar de forma remota las constantes vitales de pacientes.
- Domótica: Control de electrodomésticos u otros dispositivos en el hogar.

La Figura 2.1 Arquitectura de una red inalámbrica de sensoresmuestra la arquitectura típica de una WSN.

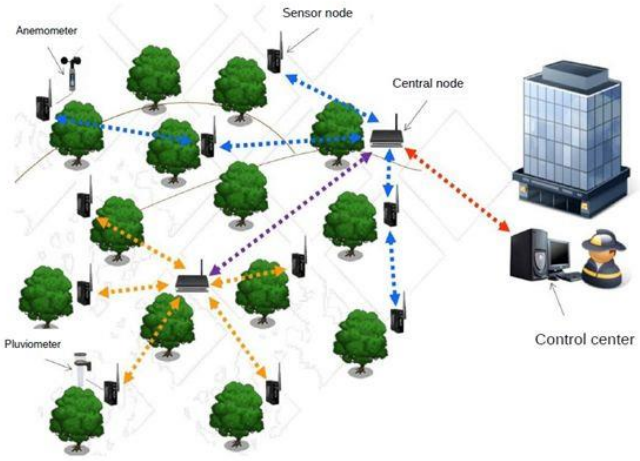

Figura 2.1 Arquitectura de una red inalámbrica de sensores

Las WSNs están formadas por nodos que disponen de un sensor y recogen información sobre el parámetro que se desea monitorizar. Generalmente, los nodos sensores envían dicha información a un nodo coordinador con mayor capacidad, que se encarga de procesarla. Las WSNs pueden funcionar en modo infraestructura, de manera que todos los nodos sensores se comunican únicamente con una entidad central, o en modo ad hoc, de manera que la red es completamente distribuida y los propios nodos sensores participan en el encaminamiento de los mensajes.

Las características principales de las WNSs son:

- Topología dinámica: En este tipo de redes la topología suele cambiar, por lo tanto se requieren protocolos de encaminamiento dinámico para asegurar que todos los nodos de la red pueden comunicarse.
- Variabilidad de canal: Se suele cambiar frecuentemente de canal debido a atenuaciones de la señal o a interferencias.
- Tolerancia a errores: Un nodo en una red de sensores tiene que funcionar a pesar de que el propio sistema tenga errores.
- Comunicaciones multi-salto o *broadcast*: Al disponer (típicamente) de muchos nodos, las redes de sensores tienen protocolos para establecer comunicaciones multi-salto y también permites hacer difusión (por ejemplo para posible cambio de topología) mediante transmisiones *broadcast*.
- Consumo mínimo de energía: Es una de las características más sensibles de este tipo de redes, ya que hay que encontrar un equilibrio entre el bajo consumo y la potencia del procesador del nodo.
- Limitaciones de hardware: para poder conseguir un consumo bajo, hay que utilizar hardware que requiera poca potencia para su funcionamiento. Esto implica que la potencia de cálculo de los nodos sensores es limitada.
- Coste bajo

2.2. Protocolos y tecnologías existentes en las redes inalámbricas de sensores

Las tecnologías inalámbricas más populares en entornos WSNs son:

- IEEE 802.11b [3], para redes WLAN (*Wireless Local Area Network*)
- IEEE 802.15.1 [4] para redes WPAN (*Wireless Personal Area Network*), basado en la tecnología Bluetooth.
- IEEE 802.15.4 / Zigbee [5][6], para redes WPAN de baja velocidad.
- 6LoWPAN [7] (*IPv6 over Low power Wireless Personal Area Networks*).
- IEEE 802.15.6 [8] para redes WBAN (*Wireless Body Area Networks*), redes centradas en la monitorización de las constantes vitales de los seres humanos.

Estos estándares usan las bandas ISM (*Industrial, Scientific and Medical*),

definidas por la ITU (*International Telecommunication Union*) para uso no comercial de radiofrecuencia en aplicaciones industriales, científicas y médicas, como su mismo nombre indica. Cabe mencionar que el uso de estas bandas no requiere de licencia, hecho que ha propiciado la aparición de un gran número de servicios inalámbricos durante los últimos años y la saturación de las mismas. Las bandas de frecuencia son las siguientes:

- 2.4GHz (Universal)
- 902 - 928MHz (EEUU):
- 868 - 870 MHz (Europa)
- 433.05 - 434.79 MHz (EEUU y Europa)
- 314 - 316 MHz (Japón)

En la siguiente tabla se muestra las principales características de los estándares mencionados.

	802.11b	**802.15.1**	**802.15.4 / Zigbee /6LoWPAN**	**802.15.6**
Frecuencias de operación (MHz)	2400	2400	868/915/2400	400/800/900/2400
Velocidad de transmisión	11Mbps/ 6Mbps	Hasta 1 Mbps	Hasta 250 Kbps	Hasta 15.6 Mbps
Cobertura (m)	460	60-240	10-20	< 10 m
Topologías	Estrella, ad hoc	Scatternet (master-slave)	Estrella, P2P y Mesh	Estrella

Tabla 2.1 Comparativa de los estándares

En este proyecto se ha decidido usar el estándar IEEE 802.15.4, puesto que sus características encajan perfectamente con el escenario que se quiere desplegar (detección de presencia): velocidades de transmisión baja, bajo consumo y radio de cobertura medio.

2.2.1 IEEE 802.15.4

El IEEE 502.15.4 es un estándar que define el nivel físico y el control de acceso al medio de las redes PAN, con tasas de transmisión bajas (LR-WPAN, *Low Rate Wireless Personal Area Network*). Se enfatiza el bajo coste de comunicación con nodos cercanos y sin infraestructura o con muy poca, para favorecer aún más el bajo consumo. Este protocolo está pensado para aplicaciones que no requieran protocolos muy pesados ya que afectaría seriamente al consumo de energía. La Tabla 2.2 resume las características más importantes del estándar.

Propiedad	Rango
Rango de TX de datos	868 MHz-> 20Kbps 915 MHz-> 40Kbps 2.4 GHz-> 250Kbps
Alcance	10-20 metros.
Latencia	< 15 milisegundos
Canales	868/915 MHz: 11 canales 2.4 GHz: 16 canales
Bandas de frecuencia	Dos PHY: 868/915 MHz y 2.4 GHz
Direccionamiento	16 o 64 bits
Canal de acceso	CSMA-CA y CSMA-CA ranurado

Tabla 2.2 Características del estándar 802.15.4

El estándar 802.15.4 soporta diferentes topologías, como tipo estrella o la topología *peer-to-peer*. La selección de la topología viene dada en función del escenario: si se quiere dar una mayor cobertura se recomienda *peer-to-peer*.

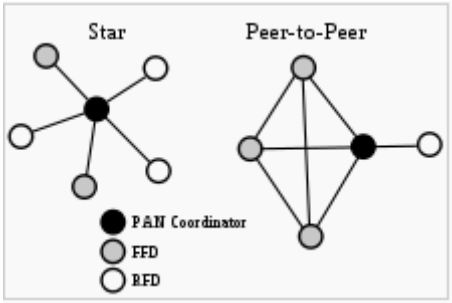

Figura 2.2 Topologías en 802.15.4

Como se puede ver en la Figura 2.2, se definen dos tipos de nodos en la red:
el dispositivo de funcionalidad completa (*full-function device* o FFD), que puede funcionar como coordinador de una red (PAN *coordinator*) o bien como un nodo normal. Puede comunicarse con cualquier dispositivo de la red, encaminar los mensajes y ser responsable de su entorno o bien de la red.
El segundo tipo de nodos es el son los dispositivos de funcionalidad reducida (*reduced-function device* o RFD). Son dispositivos muy sencillos con recursos y necesidades de comunicación muy limitadas, que únicamente pueden comunicarse con un nodo tipo FFD.

Capa de enlace de datos (*Data Link Layer* o DLL)

El estándar divide este nivel en dos subcapas: la del enlace de control de acceso a medios (Medium Access Control o MAC) y la de control de enlaces lógicos (*Logical Link Control*, LLC). La subcapa LLC es común a todos los estándares de la familia 802 mientras que la MAC depende de la capa física. Las funciones de la capa MAC del estándar IEEE 802.15.4 son, entre otras, la gestión de asociación y disociación de nodos, generación de reconocimientos de datos, y gestión de los slots temporales de acceso al medio.

En cuanto al formato de la trama MAC, tiene un diseño para ser flexible y se ajusta a las necesidades de diferentes aplicaciones manteniendo el protocolo simple, tal y como se muestra en la Figura 2.3.

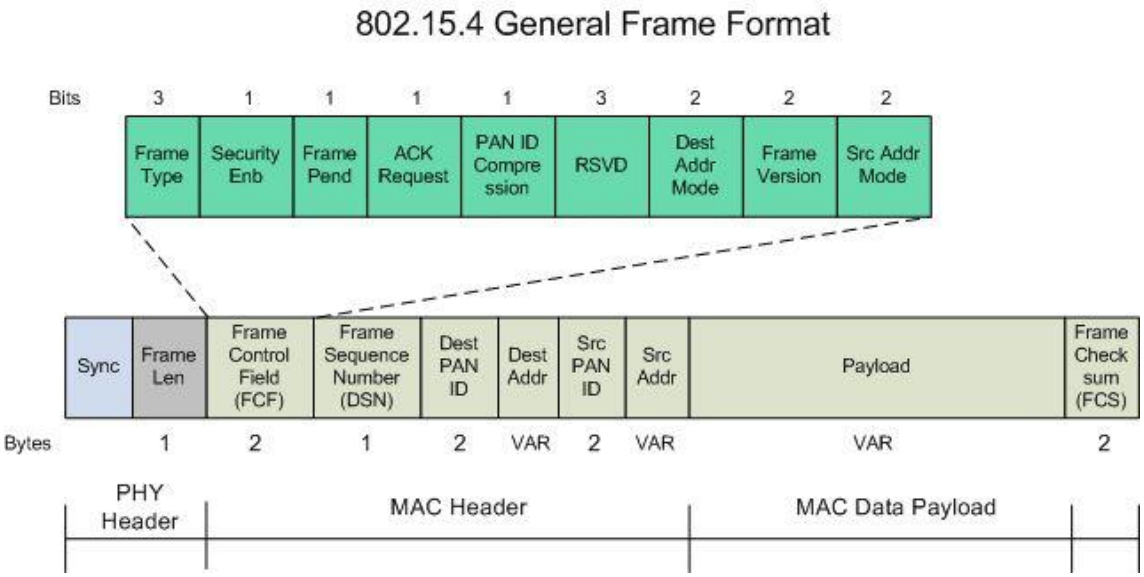

Figura 2.3 Formato trama MAC IEEE 802.15.4

Cabe destacar que el *payload*, que contiene los datos, tiene longitud variable, pero la trama completa no puede exceder los 127 bytes.
El coordinador PAN transmite *superframes* de guía en intervalos variables de 15 ms hasta 245 segundos. Entre cada intervalo el tiempo se definen 16 ranuras durante las cuales los dispositivos pueden transmitir sus *frames*. El coordinador PAN también puede asignar ranuras específicas a dispositivos que requieran un ancho de banda dedicado.

Zigbee se basa en las capas física y de datos del protocolo 802.15.4. A grandes rasgos se podría decir que básicamente añade funcionalidades de la capa de red, permitiendo que la red opere en topología *mesh* (red multi-salto.).

2.3. Seguridad en WSNs

Como ya se ha mencionado en la Introducción, las redes WSNs son especialmente vulnerables debido a su carácter inalámbrico y a los recursos limitados de los nodos sensores que no soportan mecanismos de seguridad muy complejos.

Los servicios básicos de seguridad que debe proporcionar cualquier red son:

- Confidencialidad: asegurar que la información es inaccesible para usuarios no autorizados mediante cifrado de datos.
- Integridad: Proteger los datos contra una manipulación no autorizada (modificación de paquetes o inserción).
- Autenticación: verificar que los datos provienen de una fuente de confianza.

Estos servicios pueden ofrecerse mediante el uso de primitivas criptográficas.

Hay distintos tipos de ataques que pueden ejecutarse contra una red WSN [9]. Algunos de los más comunes son:

- DoS (*Denial of Service*). Consiste en saturar a un nodo sensor enviándole muchos paquetes con el objetivo de que no pueda establecer comunicaciones o bien se le agote la batería.
- *Sniffing.* Capturar paquetes de una conversación ajena. Para evitarlo, es recomendable cifrar de dichas conversaciones.
- *Spoofing*. El atacante puede suplantar la identidad de un nodo de la red (usando su dirección MAC, por ejemplo) para enviar mensajes falsos.
- *Jamming*. Consiste en generar una señal interferente de modo que imposibilita la comunicación entre nodos honestos.
- *Ataques de replay*. Consiste en capturar un paquete generado por un nodo autorizado y reenviarlo posteriormente.

Otros ataques se basan en el protocolo de encaminamiento usado cuando la red funciona en modo ad hoc:

- *Sink-Hole*. Consiste en forzar, falsificando la información de encaminamiento, a que los nodos legítimos envíen la información a través del nodo atacante. Este ataque permite al atacante observar la información enviada por dichos nodos y la vez, ejecutar otros ataques como el reenvío selectivo.
- Reenvío selectivo. Consiste en eliminar algunos de los paquetes que atraviesan el nodo malicioso. Esto es posible siempre y cuando dicho nodo aparezca en la ruta usada por otros nodos sus comunicaciones.

También existen ataques relacionados con el código usado para programar el sensor o con la seguridad física de los dispositivos:

- Atacar al código del sensor. El atacante accede a un nodo sensor para modificar o borrar la información, a menudo aprovechando algún *bug* del código fuente sensor.
- Fuerza física. Este tipo de ataque consiste en manipular de forma física el sensor. En el mercado hay *hardware* preparado para este tipo de ataque, llamado *tamper-proof*, que impide la manipulación del sensor de forma física sin previa autenticación.

2.3.1 Mecanismos de seguridad en el estándar IEEE 802.15.4

El algoritmo de cifrado usado en el IEEE 802.15.4 es el AES (*Advanced Encryption Standard*), un cifrado simétrico en bloques de 128 bits con claves de 128, 192 o 256 bits. Puesto que se trata de un cifrado simétrico, emisor y receptor deben acordar una clave secreta con antelación, y que usarán para cifrar y descifrar, respectivamente. El cifrado de datos proporciona confidencialidad en las comunicaciones.

El mismo algoritmo se usa también para autenticar los mensajes, proporcionando de este modo autenticación e integridad de los datos. Para ello se genera un código MAC (*Message Authentication Code*) que se transmite junto con el mensaje. El receptor debe verificar que dicho código es correcto para autenticar correctamente a la fuente de dicho mensaje y asegurase de que el mensaje ha llegado íntegro. La Figura 2.4 muestra el proceso de generación y verificación del código MAC.

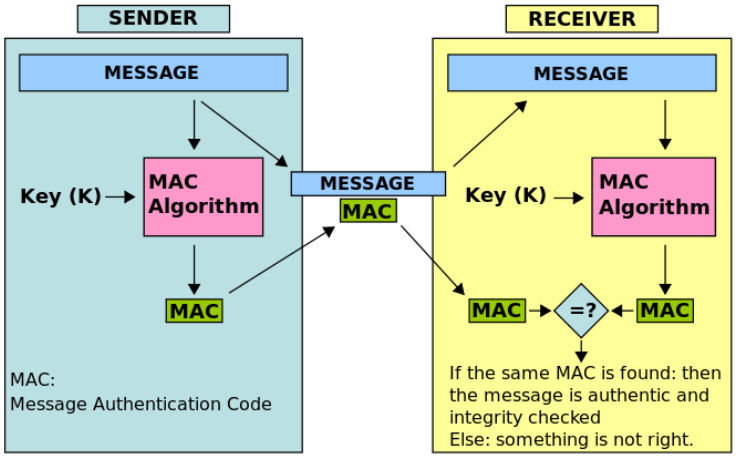

Figura 2.4 Esquema funcional de creación de MAC

Un código MAC generado con AES en modo CBC tiene una longitud de 128 bits, aunque este valor puede truncarse a 32 o 64 bits para aumentar la eficiencia en la transmisión.

Hay tres campos en la trama IEEE 802.15.4 relacionados con la seguridad: *Frame Control*, *Auxiliar Security Control* y *Payload.*

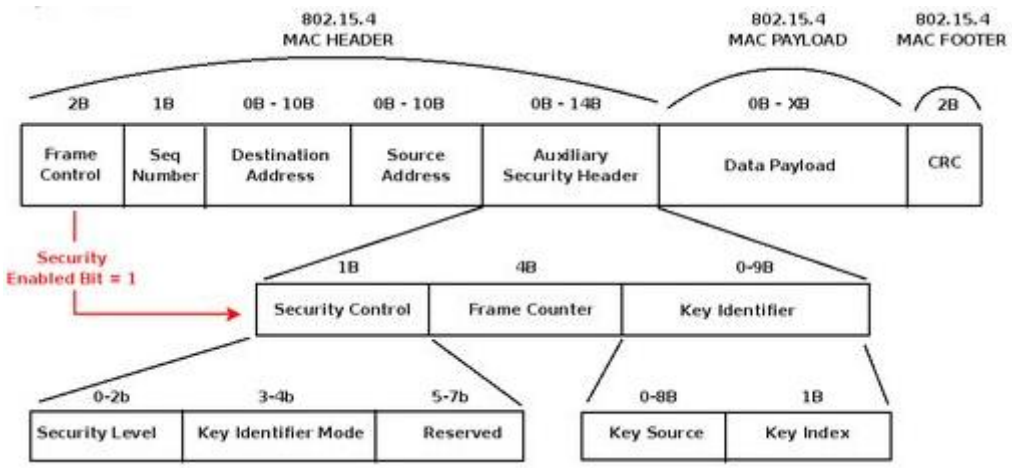

Figura 2.5 Esquema MAC Header

El campo *Auxiliary Security* sólo está activado si el bit *Security Enable* del campo *Control Frame* está a “1”. Esta cabecera especial tiene 3 campos:

- *Security Control* (1 byte): Especifica qué protección se está usando.

- *Frame Counter* (4 bytes): Es un contador con el fin de proteger a ataques de repetición de trama (ataques de *replay*)..
- *Key Identifier* (0-9 Bytes): Específica la información necesaria para saber qué clave usar para el cifrado.

El *Security Control* indica si se va a cifrar y autenticar el paquete y el algoritmo usado, tal y como se muestra en la Tabla 2.3.

Bits	Tipo cifrado	Función
0x00	No security	Data is not encrypted. Data authenticity is not validated.
0x01	AES-CBC-MAC-32	Data is not encrypted. Data authenticity is validated.
0x02	AES-CBC-MAC-64	Data is not encrypted. Data authenticity is validated.
0x03	AES-CBC-MAC-128	Data is not encrypted. Data authenticity is validated.
0x04	AES-CTR	Data is encrypted. Data authenticity is not validated.
0x05	AES-CCM-32	Data is encrypted. Data authenticity is validated.
0x06	AES-CCM-64	Data is encrypted. Data authenticity is validated.
0x07	AES-CCM-128	Data is encrypted. Data authenticity is validated.

Tabla 2.3 Tipos de cifrado en 802.15.4

En la siguiente imagen se puede observar la trama resultante dependiendo de si sólo se cifra el paquete, si sólo se autentica o bien se combinan ambas cosas.

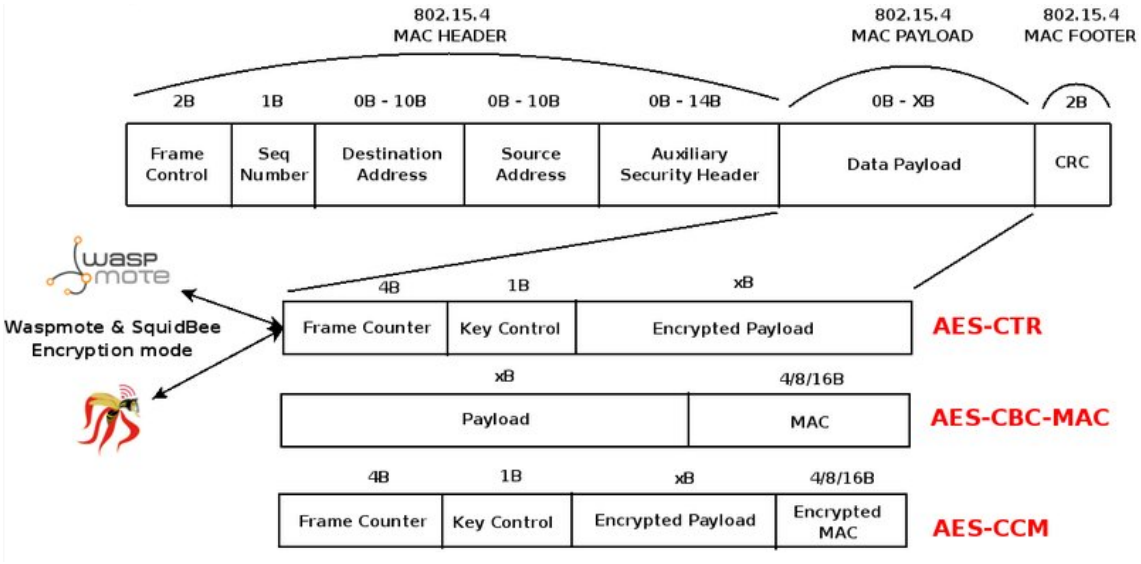

Figura 2.6 Payload de la trama 802.15.4

Hay 3 opciones:

- AES-CTR: Todo los datos son encriptados usando la clave de 128 bits y con el algoritmo AES en modo *counte*r (CTR).

- AES-CBC-MAC: El mensaje está autenticado mediante un código MAC generado con el algoritmo AES en modo CBC (*Cipher Block Chaining*). La longitud del código MAC depende de la seguridad especificada en el campo de Security Policy.

- AES-CCM: Es una combinación de los dos métodos mencionados anteriormente. Cifra los datos como el AES-CTR y también autentica el mensaje con AES en modo CBC.

Las Figuras 2.7 y 2.8 muestran el esquema de cifrado AES en modo CTR y CBC-MAC respectivamente. En ambos casos, se introduce el mensaje bloque a bloque (bloques de 128 bits) dentro del algoritmo. En el caso del modo CTR, en cada iteración se obtiene una porción del mensaje cifrado o criptograma, que es la concatenación del cifrado de cada uno de los bloques. La longitud del criptograma dependerá de la longitud del mensaje. En el caso del modo CBC-MAC, se obtiene un código MAC de 128 bits en la última iteración, independientemente de la longitud del mensaje.

Como puede observarse, además del mensaje, ambos modos usan otro parámetro de entrada: un vector de inicialización (Initialization Vector o IV) en el caso del modo CBC y un *nonce* junto con un contador en el caso del modo CTR. El IV y el *nonce* son valores aleatorios que deben cambiar con cada cifrado y deben ser conocidos por emisor y receptor. El contador es un valor que se inicializa típicamente a cero y se incrementa con cada bloque cifrado.

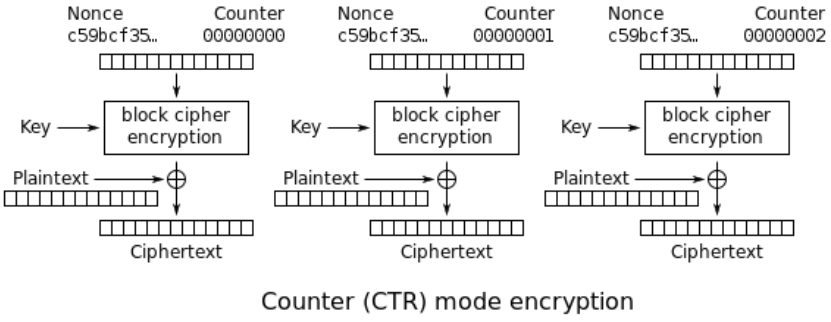

Figura 2.7 Cifrado en modo CTR

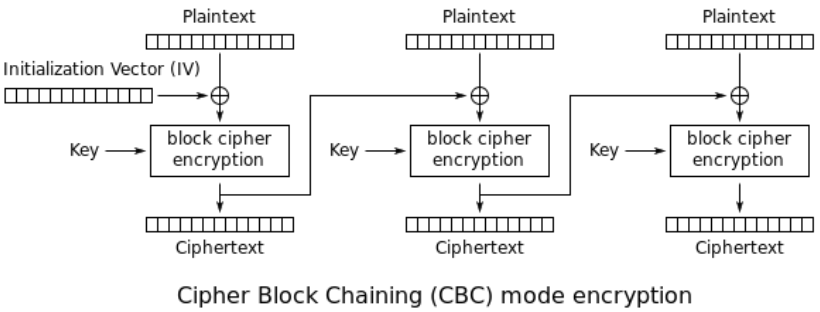

Figura 2.8 Generación de CBC-MAC

3. Implementación de un sistema de detección de presencia para uso domótico

En este capítulo se describe la implementación del sistema de detección de presencia realizada en este proyecto. Tal y como se muestra en la Figura 3.1, el sistema está formado por una red de sensores que envían informes a un nodo-*router* cuando detectan movimiento. El *router* enviará la información recibida a un servidor externo, para que se pueda procesar posteriormente.

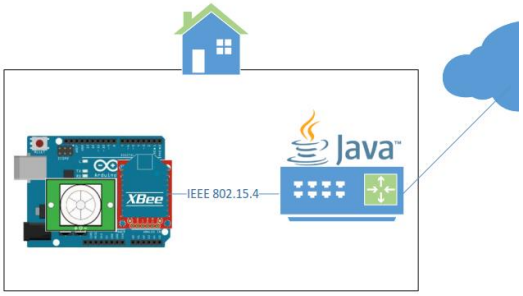

Figura 3.1 Implementación de la red de sensores

3.1. Dispositivos en la red de sensores

Para poder desplegar la red de sensores, se han usado los siguientes dispositivos:

- **Sensor de presencia PIR**: Gracias a este elemento, la red de sensores, será capaz de detectar si hay movimiento en un radio de unos 4m aproximadamente. En el Anexo I se detalla su funcionamiento.

Figura 3.2 Sensor de presencia PIR

- **XBee Series 1:** Los dispositivos Xbee Series 1 de Digi [10] disponen de un interfaz radio 802.15.4 que trabaja en la banda de los 2,4Ghz. Cada nodo de la red dispondrá de un XBee Series 1 para poder comunicarse.

Figura 3.3 Xbee Series 1

- **Placa Arduino UNO:** Arduino UNO es un producto *open-source* que lleva una placa con un procesador ATmega328 incorporado y un *bootloader* que permite conectarla a un PC mediante un cable USB y configurarla. La placa Arduino será el cerebro del nodo sensor: procesará toda la información captada por el sensor PIR, para poder enviarla al nodo-*router* mediante la interfaz XBee Series 1. Todos los nodos sensores de la red tendrán la placa Arduino. En el Anexo II se explica con mayor detalle cómo programar la placa Arduino.

Figura 3.4 Arduino UNO

- **XBee USB Explorer:** Es un módulo Digi XBee. Dispone de un puerto mini USB que permite conectar el XBee Series 1 al PC directamente, de modo que el PC funcione como un nodo más de la red y pueda comunicarse mediante una interfaz 802.15.4.

Figura 3.5 Xbee USB Explorer

- **XBee Shield:** Es un complemento para la placa Arduino que permite conectar perfectamente cualquier módulo XBee a la placa.

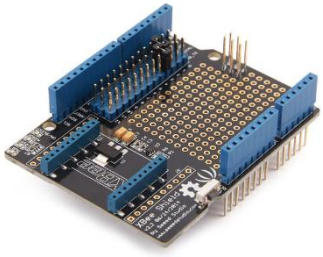

Figura 3.6 Xbee shield

- **Placa Protoboard:** Gracias a la placa Protoboard se van a poder conectar fácilmente todos los elementos que componen los nodos sensores.

Figura 3.7 Placa protoboard

- **PC (*router*):** Se usa un PC que actúa como nodo de la red recolectando la información proporcionada por los nodos sensores y que actúa como nodo frontera con la red exterior, es decir, realiza la función de *router*.

3.1.1 Funcionamiento del XBee Serie 1

XBee es un producto de Digi International [11] para soluciones inalámbricas. El modulo que se ha usado para este proyecto es uno de los más simples y de menor coste de que dispone Digi International, además de ofrecer facilidad de configuración. Debido a sus características y al hecho que implementa el protocolo IEEE 802.15.4, se ha decidido escoger este modelo para implementar la red de sensores. Las características principales del XBee Serie 1 son:

- 3,3 V @ 50 mA
- Velocidad máxima de 250kbps
- Salida de 1 mW (+ 0 dBm)
- Cobertura de 100m (sin obstáculos)
- 6 pines de entrada ADC de 10 bits
- 8 pines IO digitales
- Cifrado AES de 128 bits
- Configuración local o remota (sobre aire)
- Conjunto de comandos AT o API 1 y 2

Cabe destacar que los Xbee series 1 no implementan todas las opciones de seguridad definidas en el estándar. En particular, sólo permiten cifrado de datos con AES de 128 bits en modo CBC, y no ofrece autenticación mediante generación de códigos MAC.

A continuación se describe cómo configurar estos dispositivos.

3.1.1.1 Direcciones de los módulos XBee

Para que los módulos XBee se puedan comunicar dentro de la red tienen dos tipos de direcciones de hardware: la denominada @MAC extensa, de 16 dígitos hexadecimales (64 bits) que es la definida por la IEEE y es única, y otra de 4 dígitos hexadecimales (16 bits) que puede ser modificada por el usuario. La MAC corta permite diferenciar a un sensor dentro de la red de sensores, mientras que la MAC corta identifica de forma unívoca el nodo sensor a nivel global (dentro y fuera de nuestra red de sensores). La dirección MAC corta resulta especialmente útil dentro de la red de sensores, puesto que disminuye el tamaño

de los paquetes transmitidos dentro de la misma.

Figura 3.8 @MAC extensa de un módulo XBee.

La @MAC extensa puede consultarse en la parte inferior del módulo, tal y como puede verse en la Figura 3.8. Está grabada en el firmware de XBee y por lo tanto no se puede modificar. La @MAC extensa está formada por el SH (Serial number High) y el SL (Serial number Low). En el ejemplo de la figura el SH tiene el valor 0013A200 y el SL 40DC073D. El SH es la parte menos significativa y significa que habrá módulos que compartan esa parte de la dirección @MAC extensa, mientras que el SL es único.

3.1.1.2 Configuración de Xbees: XCTU

XCTU es un programa de Digi que permite configurar los Xbee, crear y visualizar los mensajes que se mandan entre sí, y también poder ver la topología de la red.
Para cambiar las características del Xbee, hay que seleccionar el dispositivio y se puede hacer de dos maneras: la primera es añadiendo el Xbee al XCTU o bien escanear los puertos con el botón *Discover Devices*. La única diferencia entre ambos es que esta última opción ofrece más opciones de búsqueda.

Una vez seleccionado el Xbee conectado al ordenador, se cambia la configuración para que pueda establecer conexión con el Xbee conectado a la placa Arduino. Principalmente para que esto pueda suceder, ambos tienen que tener configurado como dirección destino la dirección del otro Xbee y usar el mismo canal para la comunicación y usar el mismo MAC *mode*, es decir, que sepan los dos si habrá paquetes ACK o no. Por último, se debe especificar si la comunicación estará cifrada por AES y la clave que se utilizará en la conversación. Para poder localizar rápidamente a los Xbee (sin tener que usar su dirección MAC), es conveniente asignarles un identificador.

En la Figura 3.9 se puede ver cómo XCTU muestra los parámetros del Xbee y cuales se pueden modificar Tal y como indica la leyenda en la parte superior, los que están marcados en azul son lo que se han cambiado. En esta pantalla además se puede actualizar el *firmware* del Xbee seleccionado..

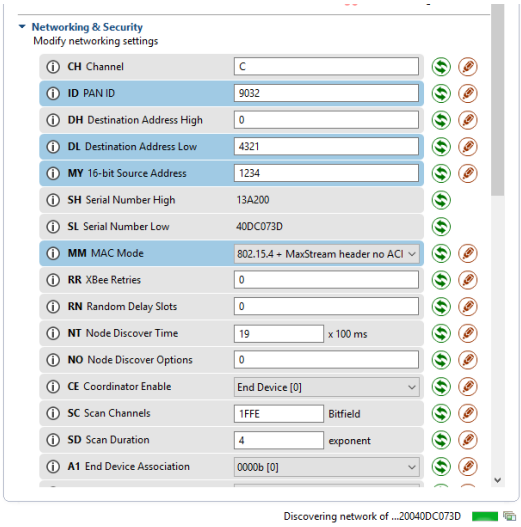

Figura 3.9 Parámetros Xbee

En el Anexo III se proporciona más información sobre el funcionamiento y la configuración de los Xbee.

3.2. Escenario y funcionamiento de la red de sensores

En esta sección se describe el escenario considerado en este proyecto, así como la configuración de los distintos dispositivos. Se especificará qué herramientas se han utilizado y cómo se ha implementado. A nivel de código de la placa Arduino sólo se muestran las funciones más esenciales, el código completo puede consultarse en los Anexos II y IV.

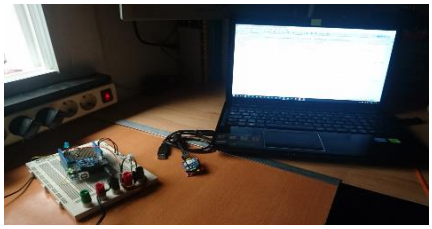

Figura 3.10 Componentes de la red de sensores.

Cuando se enciende la placa Arduino, se calibra el sensor PIR para que se acostumbre a la radiación de su ubicación. A nivel de código, esto implica esperar unos 30 segundos. Cuando el sensor está calibrado, el LED emite una señal lumínica. A continuación, la placa Arduino entra en la función Loop() y ejecuta el código correspondiente.

Durante el funcionamiento de la red, el sensor puede encontrarse en 3 estados diferentes:

- **Estado inicial**: el sensor se acaba de conectar a la red y solicita autenticarse con el *router* (intercambio de mensajes o *handshake*).
- **Estado constante**: el sensor ya está autenticado y envía una notificación

al *router* cuando detecta movimiento, o bien cuando deja de detectarlo.

- **Estado de inactividad**: entra en este estado cuando no detecta movimiento durante unos 3 minutos y por lo tanto durante ese intervalo de tiempo no ha transmitido ningún mensaje. El *router*, al no recibir mensajes del sensor durante este tiempo, envía un mensaje *keep alive*, solicitando al sensor que conteste con otro mensaje para demostrar que no está desconectado o ha dejado de funcionar correctamente.

Con este objetivo, se han definido distintos tipos de mensajes, en función de la situación en la que se encuentra el sensor, tal y como se muestra en la Tabla 3.1.

Situación del sensor	Código del mensaje
Autenticado y detecta movimiento	[*nonce*]/01/MAC
Autenticado y deja de detectar movimiento	[*nonce*]/02/MAC
No autenticado y quiere autenticarse (1ª)	/03/[numero aleatorio]
No autenticado y quiere autenticarse (2º)	/04/E_k[$Challange_{Router}$, ID_{sensor}]
Autenticado y avisa que está operativo	[*nonce*]/05/

Tabla 3.1 Códigos de los mensajes enviados por el sensor

Durante el estado constante, los mensajes intercambiados entre *router* y sensor se envían cifrados con AES y autenticados mediante un MAC, que se genera usando el algoritmo AES en modo CBC. El cifrado de mensajes con AES es una función que ya viene implementada en el XBee, pero ha sido necesario implementar tanto el mecanismo de autenticación mediante el *handshake* como la generación de MACs. En ambos casos, los dispositivos requieren el uso de una clave secreta compartida, que se ha configurado en los dispositivos, tal y como se muestra en la Figura 3.11 Claves compartidas

```
//Valores preconfigurados Sensor-Router
uint8_t key[] = {'48','49','50','51','52','53','54','55','56','57','48','49','50','51','52','53'}; //KEY=0123456789012345
uint8_t KEY_MASTER[] = {'7','6','5','4','2','1','0','8'}; //KEY preconfigurada para los challange del primer hanshake
uint8_t IV_MASTER[] = {'5','6','8','1','9','7','2','8'}; //IVpreconfigurada para el cifrado
```

Figura 3.11 Claves compartidas

En particular, las variables configuradas son:

- Key: la clave secreta usada para crear los MACs.
- IV: el vector de inicialización con el cual se generará el MAC. Este parámetro se escoge en el *router*, no está preconfigurado.
- KEY_MASTER: la clave secreta usada para cifrar el *challenge* del *handshake* principal de la comunicación del escenario.
- IV_MASTER: el vector de inicialización con el cual se cifrará el *challenge* del *handshake*.
- KY: Clave de cifrado AES de los módulos XBee para cifrar los paquetes. Este parámetro se inserta en la propia configuración de los

módulos XBee, no en el código.

Para poder llevar a cabo la comunicación del XBee con la placa Arduino, se ha utilizado la librería XBee.h, y así poder acceder a las funciones del XBee mediante la placa Arduino. También se ha usado la librería SoftwareSerial.h donde están las funciones básicas de Arduino y por último la librería AESLib.h para poder hacer el cifrado en la MAC y el *handshake* inicial entre sensor y *router*.

3.2.1 Mecanismo de autenticación

Uno de los servicios de seguridad más importantes es la autenticación de los dispositivos. Por defecto, los XBee series 1 no ofrecen ningún mecanismo de autenticación, y ello implica que cualquier dispositivo controlado por un usuario malicioso puede conectarse a la red de sensores y enviar información falsa.

En este proyecto se ha implementado un mecanismo sencillo que permite que el *router* y el nodo sensor se autentiquen mutualmente basándose en el conocimiento de una clave secreta. El nodo sensor ejecutará dicho mecanismo cuando se encuentre en el **estado inicial**. La Figura 3.12 muestra el diálogo entre el *router* y el nodo sensor durante el proceso de autenticación.

Cuando el sensor se conecta, envía un mensaje para autenticarse que contiene el *tag* (/03/), junto con un *challenge* de 9 bytes (Challenge@sensor). Un *challenge* no es más que un número aleatorio, que varía con cada sesión o intento de autenticación.

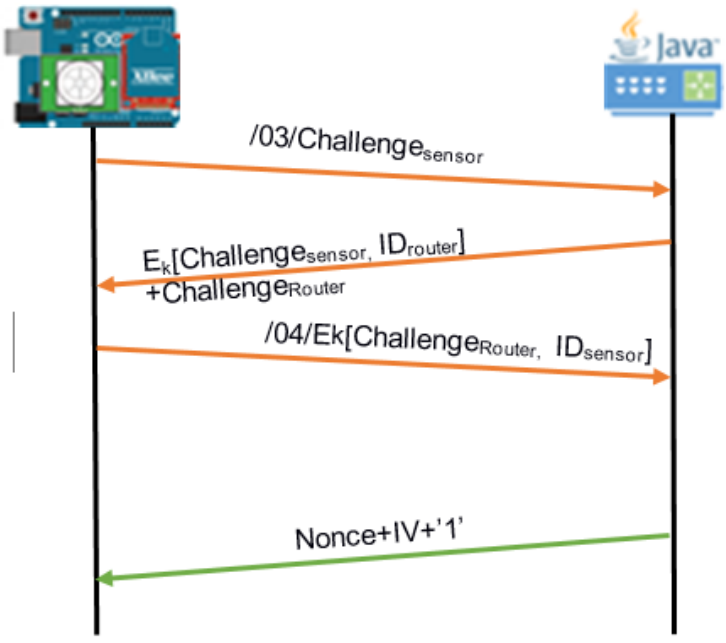

Figura 3.12 Intercambio de mensajes en el estado inicial.

Cuando el *router* recibe el mensaje, cifra el Challenge@sensor usando el algoritmo AES y una clave secreta pre-configurada que comparte con los nodos sensores autorizados en la red. A su vez, éste genera un Challenge@*router* y

envía ambas cosas al sensor.

El sensor descifra el *challenge* cifrado por el *router* y lo compara con el que había enviado. Si coincide el valor de ambos, el *router* queda autenticado (demuestra que conoce el valor de la clave secreta) y el sensor envía un mensaje con el *tag* (/04/) junto con el valor Challenge@*router* cifrado con la misma clave secreta. El *router* realiza la verificación del challenge y de este modo autentica al nodo sensor.
Una vez el *router* recibe el mensaje, con el *tag* "/04/", lo descifra y prueba si es correcto o no. Si es correcto le pregunta al servidor si la dirección MAC está registrada en la base de datos, por lo tanto si es legítimo para el usuario o no. Si todo es correcto el servidor le contesta al sensor con un *nonce* junto con un IV aleatorio y con un 1, indicándole que está.

Cuando el *router* conteste al sensor, le enviará el reto recibido cifrado con la clave compartida que ambos tienen (la KEY_MASTER mencionada anteriormente junto con el IV_MASTER) junto con un reto nuevo para el sensor. En este punto el sensor tiene que hacer dos cifrados; uno para comprobar si el reto cifrado que ha recibido es correcto, si es correcto cifrará el nuevo reto que ha recibido por parte del *router*.

En el caso que falle la autenticación de un sensor, se considera que se trata de un nodo malicioso y el *router* almacena su dirección MAC en una lista negra o *black list*. Cuando el *router* reciba un mensaje que proviene de un nodo que se encuentra en la lista no lo procesará, directamente lo descartará y seguirá escuchando el canal. El nodo malicioso estará en la *black list* durante 10 minutos. Se establece un tiempo límite de estancia en la *black list* por si se ha ejecutado un ataque de suplantación de identidad que ha conseguido insertar en dicha lista a un nodo honesto.
Si falla la autenticación en el lado del sensor (no logra autenticar correctamente al *router*), intentará repetir el proceso de autenticación enviando de nuevo el mensaje con el *tag* "/03/".

3.2.1.1 Generación de challenges

Con el objetivo de generar los *challenges* necesarios para el proceso de autenticación, se ha creado la función que se muestra en la Figura 3.13.

```
void getRandNumber() {
  for (int i = 0; i < sizeof(challange); i++) {
    randomSeed(millis());
    challange[i] = (char)random(48, 57);
    Serial.print(challange[i]);
  }
```

Figura 3.13 Generación de números aleatorios

Dado que el número aleatorio debe ser transmitido por el XBee, es necesario construir un vector de tipo Uint8_t, que es la estructura de datos que se usa para transmitir información entre XBees. La longitud del vector es el equivalente al número de dígitos de dicho número y cada posición del vector contiene el valor

de un dígito. Cada dígito se genera de forma aleatoria usando la función random() y usando como semilla los tics del procesador. Esta función devuelve un número entre 48 y 57, que equivale a crear un valor aleatorio entre 0 y 9. La siguiente tabla muestra la equivalencia entre variables Uint8_t y enteros.

Variable Uint8_t	Variable Integer
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

Tabla 3.2 Equivalencia Uint8_t y enteros

Cada posición del vector equivale a un byte, por lo tanto el número aleatorio tendrá un valor de 9 bytes. Teniendo en cuenta que el vector que se envía equivale a un número de 9 cifras, cada posición solo puede tener 10 valores (entre 0 y 9), se pueden obtener 9^{10} (3.486.784.401) valores distintos.

3.2.2 Detección de movimiento y envío de informes al *router*

En la Figura A-5.3 del Anexo V, se puede ver los procesos de Arduino cuando se envían los mensajes de movimiento. Esta parte es la más sencilla debido a que ambas partes, sensor y *router*, se confían entre sí, solo hace falta envían un pequeño mensaje cada cierto tiempo. También se puede ver la creación medidas contra ataques, como por ejemplo el incremento del *nonce*. El *nonce* se usa para que ningún cifrado (a nivel de capa de enlace, que lo hace el propio XBee), sea idéntico, debido a que solo se enviarán dos mensajes diferentes, “hay movimiento” y “no hay movimiento”. La otra medida de seguridad es la creación de un código MAC, para autenticar el propio mensaje de movimiento, se emplea por si un atacante ha logrado saltarse el cifrado AES a nivel de enlace.

En la Figura 3.14 se puede ver la función principal, donde hará todas las comprobaciones de forma periódica. Como se menciona con anterioridad, lo primero que hace es comprobar si está autenticado o no, si no lo está, enviará el mensaje con el *tag* "/03/”, después al recibir contestación del *router*, el sensor enviará el segundo mensaje en cuanto a la autenticación se refiere, que contiene el *tag* “/04/”.

Una vez que la autenticación sea correcta, Arduino solo ejecutará las siguientes comprobaciones:

- Detectar movimiento.
- Detectar si el movimiento ha cesado.

- Escucha el canal de radiofrecuencia.

```
void loop() {
  if (auth == 0) dataSend(3, nonce);
  else {
    val = digitalRead(PIRpin);
    Serial.print("Val: ");
    Serial.println(val);
    Serial.print("BeforeVal: ");
    Serial.println(valBefore);
    if (val == 0) {
      Serial.println("Listening channel...");
      listenChannel(1000, 1000, 5); // We Send timeACk and timePacket integers to function
    }
    if (val == HIGH) {
      digitalWrite(ledPin, HIGH);
      if (movimiento == 0) {
        movimiento = 1;
        Serial.print("Movimiento Change:  ");
        Serial.print(movimiento);
        Serial.print('\n');
        dataSend(1, nonce ); //MOVE!
        valBefore = val;
      }

    } else {
      digitalWrite(ledPin, LOW);
      if (movimiento == 1) {
        movimiento = 0;
        Serial.print("Movimiento Change:  ");
        Serial.print(movimiento);
        Serial.print('\n');
        dataSend(2, nonce); // Not Move!
        valBefore = val;
      }

    }
  }
}
```

Figura 3.14 Función Loop Arduino

En el primer *else*, si no detecta movimiento consulta la interfaz radio a la espera de mensajes del *router*. En este caso la función ListenChannel(1000, 1000, 5), espera durante 1000 milisegundos a recibir una petición del *router* y otros 1000 milisegundos para leer el mensaje que le envía el *router*. El número 5 es el código del mensaje que se espera, en este caso un paquete *Hello* del *router*.

Cuando revisa la interfaz radio, que tarda unos 2 segundos, pasa a revisar el sensor PIR, a partir del pin de la placa Arduino que tiene conectado. Arduino no puede correr más de un proceso a la vez, así que no hay otro modo de programar las funciones. Esto implica que Arduino tiene que alternar entre la interfaz radio y el sensor PIR, sin permanecer demasiado tiempo con cada uno de ellos para evitar perder datos.

Cuando el sensor envía un mensaje estando autenticado, tiene que enviar dos elementos de seguridad, un *nonce* y un MAC. Un *nonce* es un número aleatorio de un solo uso (number used once) que sirve como contramedida frente ataques de *replay*. Ese valor se incrementa en uno con cada nuevo mensaje

que envía el XBee.

Para poder hacer una operación tan sencilla como incrementar en 1 un número de 9 dígitos (con formato de vector de Uint8_t) se ha tenido que implementar el algoritmo de la Figura A-4.3 del Anexo IV. Aunque aparentemente es una función sencilla, ha resultado complicado programarla. El algoritmo programado convierte el vector de 9 posiciones de tipo Uint8_t en un vector de enteros, incrementa en 1 la última posición (posición 8) y mira si el valor de la misma es un 9. Si es así, comprueba si las posiciones anteriores también tienen el mismo valor. Si es el caso, por ejemplo, las posiciones de la 5 a la 8 tienen un 9, estas se convierten en 0 y la posición 4, la anterior a la posición inferior con un 9, se incrementa en 1. Cuando ya se han realizado estas operaciones, se convierte de nuevo en un vector de uint8_t para poder enviarlo en el siguiente mensaje.

El otro elemento de seguridad es añadir un MAC al mensaje para proporcionar integridad a los mensajes. De este modo, el receptor podrá saber si el mensaje ha sido alterado. En la Figura A-4.4 del Anexo IV se puede ver la función implementada en Arduino.

3.2.2.1 Estado constante

El sensor entra en este estado una vez ya está autenticado. Solo enviará mensajes cuando detecte movimiento y cuando deje de detectarlo. Cuando recibe el último mensaje del estado inicial, guarda el valor del *nonce* y el IV que recibe. Al detectar movimiento, suma 1 al *nonce* y lo envía junto con el mensaje con *tag* "/01/" (movimiento) y el código MAC correspondiente. El MAC es un cifrado del *nonce* junto con el *tag* con AES 128 bits usando el IV que le envía el *router* en el anterior estado. La clave que usa para generar el MAC se configura previamente, por lo tanto no es necesaria transmitirla (exponiéndola así a posibles atacantes).

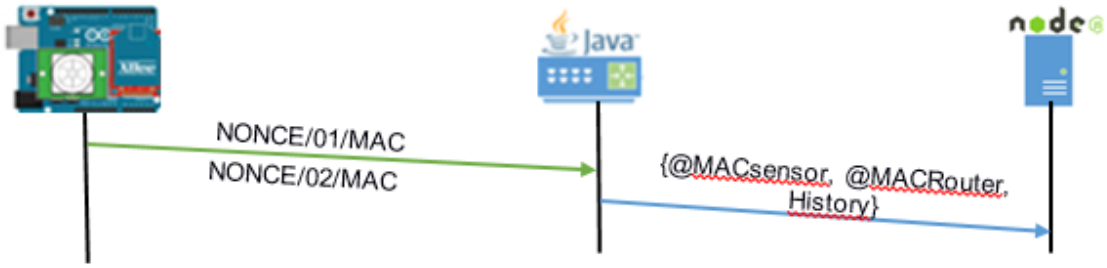

Figura 3.15 Intercambio de mensajes en estado constante

Cuando el *router* recibe el mensaje, comprueba que la dirección del Xbee no esté en la lista negra y que el código MAC sea correcto. Si es así, comprueba el valor del *nonce* (tiene que ser el número que le ha enviado en el anterior estado o en el último mensaje, sumándole 1). Sí no es así, podría tratarse de un ataque de *replay*, así que añade la dirección MAC a la lista negra durante 10 minutos.

Si todas las comprobaciones son correctas, le envía al servidor el siguiente

mensaje de JSON:


```
{
    "mac":"013A20040DC073D",
    "macnet": "013A20040DC0B1A",
    "history":"Move- 18:55:56 2:3:116"
}
```


Figura 3.16 Ejemplo de informe de movimiento

A diferencia del estado inicial, los mensajes no son periódicos, sino que son eventuales.

3.2.3 Estado de inactividad del sensor

Tal y como se ha explicado anteriormente, si el sensor no detecta movimiento durante un cierto tiempo, genera una "alerta" en el *router*, de modo que éste interroga al sensor para comprobar que sigue activo. Para implementar este mecanismo se han definido 2 temporizadores:

- Temporizador de seguridad: Cada 3 minutos, si el sensor no envía ningún mensaje, el *router* le envía un mensaje "hello". Si el sensor contesta, se reinicia el temporizador; si no contesta, el *router* se esperará otros 3 minutos y repetirá la operación.
- Temporizador de vida: Si en 10 minutos el sensor no ha enviado nada y no ha respondido a los paquetes Hello (hay que tener en cuenta que el anterior timing ha saltado 3 veces en este punto) el *router* considera que el sensor "ha muerto "y envía un mensaje el servidor anunciando que se ha desconectado.

Si el sensor contesta al mensaje de Hello, envía el *tag* "/05/, tal y como se muestra en la Figura 3.17.

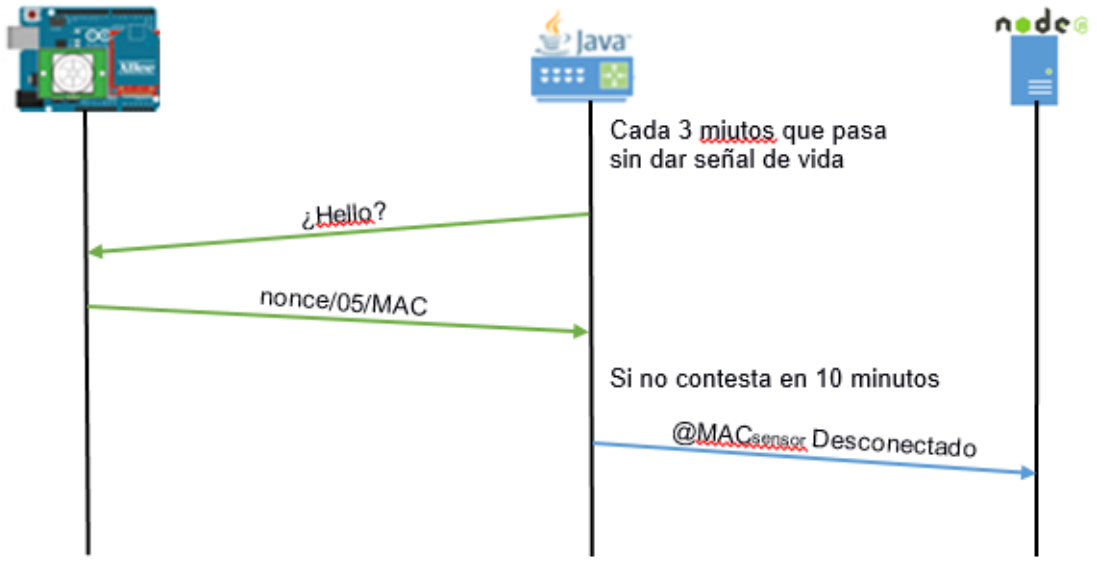

Figura 3.17 Mensajes en el estado de inactividad

Una vez el *router* avisa al servidor que el sensor está desconectado, deja de enviar *Hello packets* al sensor.

3.3. Funciones del *Router*

Como se ha explicado anteriormente, el *router* se sitúa entre la red de sensores y la red externa donde está ubicado el servidor. Por un lado recoge toda la información sobre movimiento que envían los sensores y se encarga de verificar que los nodos sensores están activos, y por otro lado actúa como interfaz entre la red de sensores y el servidor, reenviando la información relevante hacia este último.

Las distintas funcionalidades del *router* se han programado en Java, dado que es un lenguaje orientado a objetos, seguro y sólido ante errores de programación y dispone de numerosas librerías.

La Figura 3.18 muestra los distintos paquetes que componen el código de Java en el *router*.

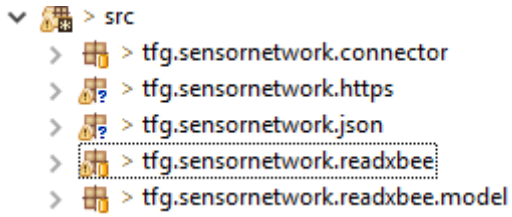

Figura 3.18 Paquetes Java del router.

Los paquetes que se usan para la comunicación con los Xbee y para almacenar los datos recibidos de los mismos son:

- "tfg.sensornetwork.connector": para conectar con la base de datos MySQL y acceder a los datos no temporales.
- "tfg.sensornetwork.readxbee": es el paquete principal donde está el archivo java principal que ejecuta el ordenador.
- "tfg.sensornetwork.readxbee.model": están ubicados los objetos java para poder tratar de una manera más fácil los mensajes JSON y los elementos de la base de datos.

El resto de paquetes se usan en la comunicación con el servidor y se describirán en el siguiente capítulo.

Los datos recibidos por el *router* relativos a los dispositivos XBee que se conectan a la red y a la información sobre movimiento, se almacenan en una base de datos MySQL.

MySQL[12] es un sistema de base de datos relacional, es decir, la forma que tiene para tratar los datos es a partir de tablas que se pueden relacionar entre sí. Se ha escogido MySQL para guardar los datos que pueda manejar el *router*

debido a que las tablas que se tienen que crear son pequeñas. También porque la implementación con el lenguaje Java [13], que se hablará más adelante, es muy sencilla y apenas requiere código para hacer una petición a la Base de Datos.

El *router* maneja 3 tablas con información sobre los sensores:

- Tabla XBee: almacena toda la información relacionada con un XBee.
  - *Address*: dirección MAC de 64 bits.
  - keyMAC: la clave para el *Message Authentication Code*.
  - Iv: el IV para el *Message Authentication Code*.
  - *Challange*.
  - *Connectivity*: si está operativa o no.
  - *Lastnonce*: El *nonce* del último mensaje enviado.
  - *Hellotime*: la hora (en milisegundos) del último mensaje recibido.

- Tabla *BlackList*: almacena las direcciones MAC de los XBees intrusos durante 10 minutos.
  - XBee: dirección MAC de 64 bits del XBee castigado.
  - Tiempo: tiempo que lleva castigado

- Tabla log: registro de los eventos importantes.
  - Fecha: Fecha en formato "dd/m/aaaa hh:mm:ss:ms"
  - XBee: dirección MAC de 64 bits del XBee que ha realizado la acción
  - Operación: indica que acción se ha cometido.

En la siguiente tabla se puede apreciar qué tipos de mensajes tendrá la tabla Log.

Código	Descripción
Connection	Se conecta un XBee
Auth	Se autentica un XBee
Err_*Nonce*	*Nonce* recibido incorrecto
Intrusion_MAC	MAC recibido incorrecto
Err_history	No se ha podido guardar la historia en el servidor
Err_Auth	XBee no registrado en el server
Msg_error	Fallo al crear el mensaje al servidor
Not_Hellopacket	Lleva más de 10 minutos sin enviar un mensaje

Tabla 3.3 Operaciones Log.

Para que el ordenador pueda comunicarse con el XBee en Java, se ha utilizado la librería oficial de Digi. Cuando el *router* se inicia, para que se establezca la comunicación con el XBee del *router*, hay que indicar los parámetros del mismo. También establece una sesión con la base de datos MySQL y crea un *thread* secundario.

El *router* trabaja con dos *threads*: el principal, donde se pone a escuchar el medio para ver si recibe el mensaje de algún XBee y el secundario, que revisa la base de datos para comprobar si los sensores que tiene registrados están activos o no.

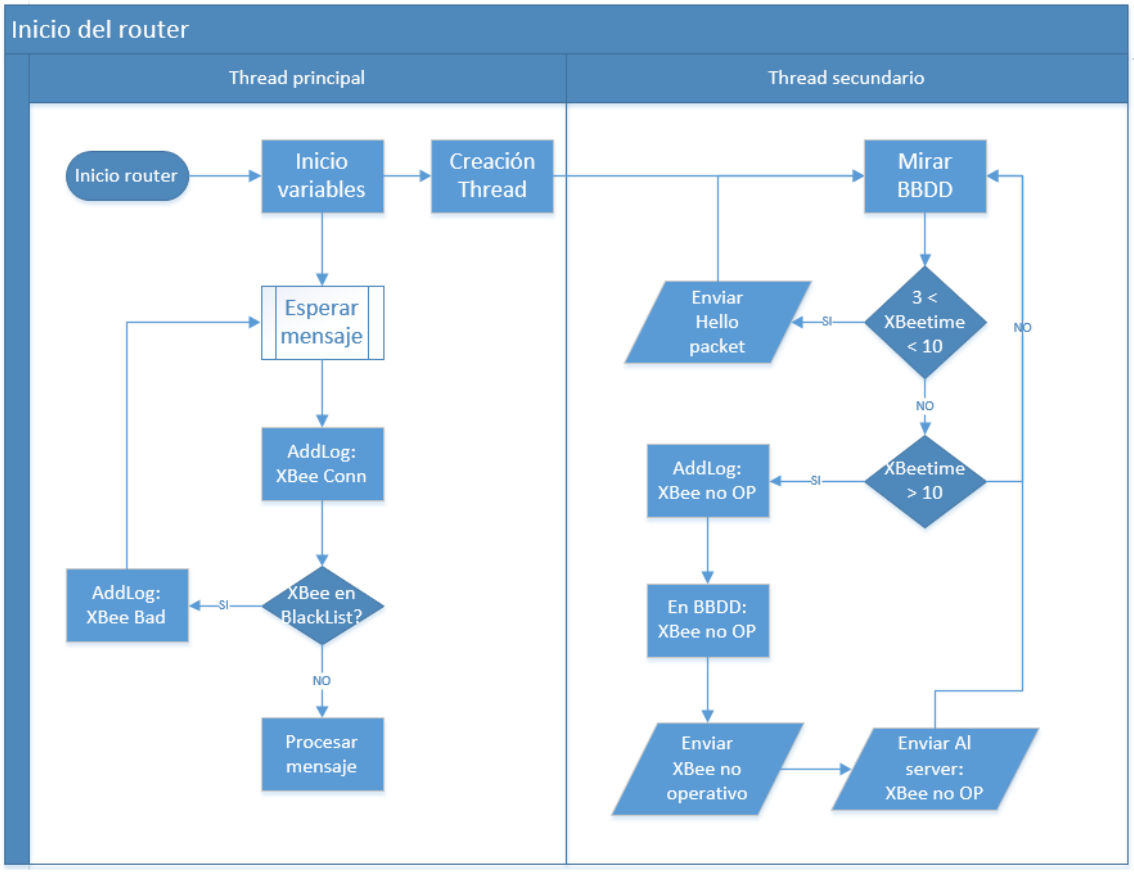

Figura 3.19 *Diagrama de procesos al iniciarse el router*

Como se puede ver en la Figura 3.19, el *thread* secundario hace una llamada a la base de datos y mira la hora en que ha recibido algún mensaje de los XBee que tiene registrados. Si el tiempo que ha pasado es mayor de 3 minutos y menor de 10 el *router* enviará un paquete *Hello* al sensor; si es mayor de 10 minutos enviará un mensaje al servidor avisando de que está desconectado.

En cuanto al *thread* principal, lo primero que hace es buscar en el entorno un XBee con la ID “TX”, la variable la podemos ver en la Figura 3.19, una vez que recibe un mensaje de dicho XBee lo primero que hace es revisar si éste está en su lista negra. Sí no está en la lista negra, por ejemplo cuando se inicia por primera vez, pasa a analizar el mensaje recibido por el XBee.

Cuando el *router* analiza el mensaje que le llega del sensor, lo primero que mira es el *tag*. Si el escenario se está inicializando, el primer *tag* que leerá será el de que se quiere autenticar (/03/). Como se puede apreciar en la Figura A-4.5 del Anexo IV. Cuando el sensor le envía “/03/[Challenge]”, primero de todo comprueba si el sensor está en su base de datos, una vez lo comprueba, procede a cifrar con AES 128bits CBC el *challenge* recibido por el sensor con la clave

preconfigurada que comparten. Una vez cifrado, procede a crear otro *challenge* para que lo resuelva el sensor y lo guarda en la base de datos para luego consultarlo. Por último envía estos dos *challenge*, uno cifrado y el otro en claro, al sensor para probar su identidad. Esto es debido a que si el sensor es un XBee "honesto", tendrá la clave preconfigurada compartida y podrá resolver correctamente el *challenge* nuevo.

El siguiente mensaje a recibir es el que tiene el *tag* "/04/". El *router* espera que el sensor le envíe el *challenge* cifrado con la clave que comparten, por lo tanto, revisa que sea el correcto. Si es positivo envía un mensaje el servidor preguntándole si la dirección MAC del sensor está registrada en su base de datos. En el caso que no estuviese registrado en la base de datos del servidor, el *router* añadiría la dirección MAC a la lista negra, *black list,* tomando el sensor como un sensor malicioso. Por lo contrario, si todo fuese correcto y si estuviese registrado, el *router* generaría un *nonce* y un IV. A continuación el *router* procede a guardar todas las variables creadas en la base de datos, también guarda como autenticado al XBeey como operativo guardando la hora del último mensaje recibido suyo. Por último, añadiendo a la tabla log que se ha autenticado un XBee, el *router* le envía el *nonce*, junto con el IV y la indicación de que está autenticado.

Una vez autenticado, como se ha mencionado antes, el sensor sólo enviará 3 tipos de mensajes, dos en relación a si detecta al movimiento y otro avisando de que está operativo (respuesta a los mensajes *Hello*). En el primer caso, el *router* comprueba si la MAC es correcta, si no lo es, añade el sensor a la *black list* y guarda un registro en la tabla log. Si es correcto, revisa que el *nonce* sea el esperado, si no es correcto ocurre lo mismo que antes. A continuación, el *router* envía al servidor el histórico (si hay movimiento o ha dejado de haberlo) y espera la respuesta tal y como se puede apreciar en la Figura A-4.6 del Anexo IV.

4. Implementación de un servicio remoto de control

En este capítulo se describe las tecnologías y las herramientas usadas para configurar el servidor encargado de gestionar el servicio de control de presencia de forma remota.

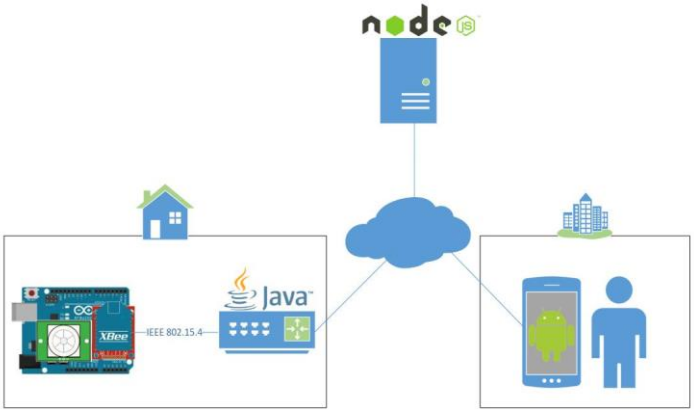

Figura 4.1 Comunicación WSN-Servidor

La comunicación entre *router* y servidor se establece para autenticar sensores en la WSN y para recibir información sobre detección de movimiento. Dicha comunicación está cifrada mediante HTTPS y exige autenticación mutua mediante certificados, es decir, ambos tienen que probar su identidad mediante el intercambio de sus respectivos certificados digitales.

El servidor está preparado para recibir peticiones HTTPS de un usuario a través de su dispositivo móvil. En este caso, la comunicación entre servidor y usuario también está cifrada, pero se usa un método de autenticación basado en *tokens* [18] para que el usuario pruebe que es un usuario autorizado. De este modo, no se exige a un usuario que disponga de un certificado digital.

4.1. Tecnologías y herramientas usadas para el servidor

Para poder establecer una comunicación segura entre *router* y servidor, se usa el protocolo HTTPS (HTTP con TLS). Este protocolo permite establecer una comunicación extremo a extremo con autenticación mutua, es decir, los dos extremo verifican la identidad de la otra parte. El protocolo define un handshake (intercambio de mensajes) que además de autenticación mutua permite a las dos partes negociar una clave de cifrado mediante el algoritmo de *Diffie-Hellman* [14] para los mensajes que intercambiarán posteriormente. (ver Anexo VI).

En cuanto al lenguaje de programación empleado para implementar el servidor se ha escogido Java, dado que Digi International proporciona una librería nativa y muy completa para poder gestionar el módulo XBee.

Para poder programar el servidor, se ha escogido el lenguaje NodeJS [15], un lenguaje orientado a objetos y muy liviano, que no crea una gran carga de procesos. Además, dispone de un intérprete ultra-rápido y es un lenguaje especialmente creado para desarrollar APIs. Por dichos motivos, resulta ideal para implementar el servidor en este proyecto.

Teniendo en cuenta que el servidor está programado en NodeJS, se ha escogido MongoDB como base de datos ya que utiliza el mismo esquema JSON con el que trabaja NodeJS, y por lo tanto no es necesario ningún tipo de *framework* intermedio. Por otro lado, se trata de una base de datos rápida, escalable y fácil de usar.

Para poder programar la aplicación en Java que se ejecutará en el *laptop*, se utiliza el entorno de desarrollo Eclipse Mars IDE (Integrated Development Environment). Este entorno de desarrollo ha sido muy utilizado durante la carrera y es gratuito. En cuanto el entorno de desarrollo para programar NodeJS, se ha utilizado WebStorm IDE, un IDE muy completo de la empresa JetBrains [16]. Es un entorno de pago, pero se ha usado la versión de prueba.

Para poder comprobar el funcionamiento del servidor, se ha usado Postman REST Client, una herramienta muy útil y práctica, que permite enviar peticiones HTTP de una forma personalizada.

Finalmente, para la autenticación mutua entre *router* y servidor, son necesarios sendoscertificados generados por una autoridad de certificación (*Certification Authority* o CA). En este proyecto se ha usado la herramiento OpenSSL para ello.

En el Anexo V, se explica con mayor detalle las tecnologías y lenguajes escogidos.

4.2. Función del *router* en la red externa (comunicación con el servidor)

Tal y como se ha mencionado anteriormente, la comunicación entre *router* yl servidor se hace de forma segura mediante el protocolo HTTPS con autenticación mutua y con cifrado de datos. Tras el *handshake* inicial, el primer mensaje que envía el *router* al servidor es referente al estado inicial de un sensor (cuando éste se quiere autenticar).

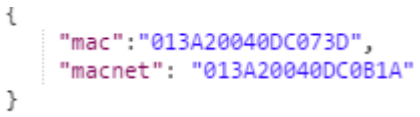

Figura 4.2 Mensaje con información de un sensor

En la Figura 4.2 se puede ver la estructura en JSON del mensaje que el *router* envía al servidor dentro de un mensaje HTTP para averiguar si un dispositivo XBee está registrado. El servidor puede devolver las siguientes respuestas HTTP:

- 200 (access): Sí que está registrado el sensor.
- 404 (no2 found): No está registrado el sensor.
- 500 (err_serv): Error en el servidor.

En el caso que el *router* reciba el 404 o el 500, añade un registro en la tabla de Log de Err_auth. En cambio, si recibe el código 200 procede a a la creación de los parámetros necesarios, explicados en la sección 3.3.

El siguiente caso en el que el *router* establece una conexión con el servidor es durante el estado constante de un nodo sensor, en particular, cada vez que se detecte movimiento.


```
{
    "mac":"013A20040DC073D",
    "macnet": "013A20040DC0B1A",
    "history":"Move- 18:55:56 2:3:116"
}
```


Figura 4.3 Mensaje para añadir una historia.

El *router* enviará un mensaje como el que aparece en la Figura 4.3, mediante una petición HTTP a la URL relativa "/xbeepan/history. Idealmente el *router* tiene que esperar solo una respuesta tipo **200 ok**, pero si el servidor no encuentra en su base de datos el XBee NET o el XBee PAN, devolverá un **404_not found.** La otra posibilidad es que el servidor devuelva un error **500_err server**. Esto ocurrirá si el servidor falla por algún motivo desconocido. Si todo es correcto el *router* procede a actualizar los datos pertinentes en la base de datos.

Otro motivo caso en el que el router envía un mensaje al servidor es cuando el sensor no envía ningún mensaje en un intervalo de 10 minutos. La estructura del mensaje es idéntica al de la Figura 4.3, con la diferencia que la petición se envía a la URL relativa ("/xbeenet/hellotime"). El manejo de errores del servidor ante esta petición es muy similar que en el caso anterior, debido a que tiene en cuenta el hecho de no tener registrado un XBee..

4.3. Función del servidor externo

En este apartado se describe el funcionamiento del servidor montado en Node.js. Los directorios con los diferentes ficheros programados para el servidor se muestran en la Figura 4.4.

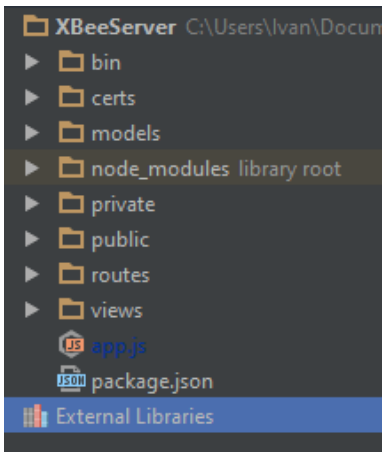

Figura 4.4 Directorios servidor NodeJS.

- El directorio **bin** (véase Figura 4.5) contiene el archivo www.js necesario para que NodeJS arranque el servidor, y es donde se definen los diferentes parámetros del servidor. En este caso, se especifica que el servidor use HTTPS, la ubicación de la clave secreta o la ubicación de los certificados. También se puede apreciar que la variable *requestCert* tiene valor *true*, indicando que es obligatorio que el cliente (el *router* en nuestro caso), proporcione su certificado para autenticarse.
- En el directorio **certs** se encuentran los certificados necesarios para la autenticación con el *router*: el certificado propio del servidor y el de la CA.
- En el directorio **node_modules**, es donde están todos los módulos NPM necesarios para el correcto funcionamiento del servidor. NPM es el gestor por defecto para Node.js.
- Los directorios **private** y **public** contienen las claves privadas y públicas generadas por OpenSSL.
- Por último, en la carpeta **router** y **models** se encuentran los archivos javascprit que permiten ejecutar las distintas operaciones.

```
var app = require('../app');
var debug = require('debug')('XBee:server');
var https = require('https');
var http= require('http');
var fs = require('fs');
var options = {

  // The Server's SSL Key
  key: fs.readFileSync('./certs/server/private/serverXBee.key'),
  // The Server's Cert

  cert: fs.readFileSync('./certs/server/serverXBee.crt'),

  // The CA (us in this case)
  ca: fs.readFileSync('./certs/ca/ca.crt'),
 // Ask for the client's cert
  requestCert: true,
  // Don't automatically reject
  rejectUnauthorized: false
};
```

Figura 4.5 Archivo www.js

- En el archivo **package.json** se especifica el archivo a cargar para inicializar el servidor y las dependencias que se necesitan descargar.
- En el archivo **app.js** (ver Figura 4.6) se configuran las peticiones que puede recibir el servidor mediante filtros CORS (*Cross Origin Resource Sharing)* [17]. En nuestro caso tiene dos tipos de URL (*Uniform Resource Locator*) relativas o rutas (*users y xbees).* La ruta index sólo se usa para realizar pruebas.

```
//Rutas Javascrypt
var routes = require('./routes/index');
var users = require('./routes/users');
var xbees = require('./routes/xbees');
var app = express();

// Mongodb connection
mongoose.connect('mongodb://ivan_root:ivan@ds117199.mlab.com:17199/heroku_73vwwtl6', function(err, res){
  if(err) throw err;
  console.log('Connected to database.js');
});
```

Figura 4.6 Fichero app.js

4.3.1 Gestión de usuarios

El servidor está pensado para que pueda gestionar diferentes usuarios, cada uno con su red de sensores. La Tabla 4.1 muestra las peticiones que puede recibir el servidor y las URL relativa que contiene cada petición.

Petición HTTPS	URL relativa	Atributos JSON	Acción
GET	/users/:username	Nada	Devuelve los datos de un usuario registrado en la base de datos.
POST	/users/	Username y Userpass	Registra un usuario nuevo añadiéndolo a la base de datos.
POST	/users/:username	Userpass y Token	Modifica datos de un usuario
PUT	/users/signin	Username y Userpass	Inicia sesión un usuario al sistema.
DELETE	/users/:username	Username , Userpass y Token	Da de baja a un usuario y a toda su red.

Tabla 4.1 Tabla de URLs relativas de users

Todas estas peticiones están pensadas para que la envíe el cliente desde su terminal móvil, ya que la función principal es gestionar su perfil. Para poder proporcionar seguridad al usuario, se hace uso de credenciales basadas en *tokens* [18], mediante el módulo jwt-simple, que permite generarlos de forma manual de una manera muy sencilla relacionando el *Username* y la hora en la que expira.
El servidor puede devolver las siguientes respuestas:

- **200-*OK***
- **401-*Unauthoritzed*:** El cliente está desautorizado para la petición, o bien por qué ha fallado a la hora de introducir la contraseña o bien por que el *token* es antiguo y requiere que inicie sesión de nuevo.
- **404-*Not found*:** En este caso se muestra cuando se busca un usuario y no está en la base de datos.

- **409-*User exist*:** El usuario tendrá que registrarse con otro *Username*.
- **500-*Error Database*:** Ha ocurrido un problema con la base de datos.

4.3.2 Gestión de la red de sensores

El sistema está diseñado de modo que la información relativa a los Xbees pueda ser actualizada por el *router* (por ejemplo, cuando autentica a un nuevo sensor) y *p*or el usuario (por ejemplo, si el usuario decide añadir a un nuevo sensor en su WSN.
Las URLs relativas que se envían en las peticiones relacionadas con la gestión de los XBee se muestran en la siguiente tabla.

Petición HTTPS	URL relativa	Atributos JSON	Acción
GET	/xbees/:mac	Owner y Token	Devuelve los datos de un xbee registrado en la base de datos.
POST	/xbees/	@MACXBeePAN @MACXBeeNet* Owner y Token	Registra un nuevo XBeePAN de un usuario.
POST	/xbees/:mac	@MACXBeeNet Owner y Token	Registra un nuevo XBeeNET de un usuario, relacionado con el XBeePAN.
POST	**/xbees/xbeepan/history**	**@MACXBeePAN @MACXBeeNet Historia**	**Añade un si hay movimiento o no a la hisotoria de un XeePAN.**
POST	**/xbees/auth/xbeenet**	**@MACXBeePAN@MACXBeeNet**	**Mira si la @MAC del XBeeNet está en la Base de datos.**
POST	**/xbees/xbeenet/hellotime**	**@MACXBeePAN@MACXBeeNet**	**Comunica que el XBeeNet está desconectado.**
DELETE	/xbees/:mac	Owner y Token	Da de baja un XBeePAN y sus XBeeNet's
DELETE	/xbees/:mac/xbeenet	@MACXBeeNet Owner y Token	Da de baja una XBeeNet

Tabla 4.2 Tabla de URLs relativas de Xbees

Las peticiones POST que aparecen en negrita son os mensajes que envía el *router* y que se han mencionado en la sección 4.2. El resto de peticiones son las que envía el usuari, en las cuales es necesario el objeto *Token* como método de autenticación de usuario.
Debido a que no se ha podido implementar la aplicación Android que permite al usuario gestionar su red de sensores, se ha optado por comprobar la respuesta del servidor a dichas peticiones mediante POSTMAN.

5. Análisis del sistema de detección de intrusos: Pruebas y resultados

En este capítulo se presentan los resultados obtenidos en la evaluación del funcionamiento de la WSN. Se han realizado un conjunto de pruebas para determinar el efecto de la distancia, de los obstáculos y de las interferencias en la comunicación de los Xbees. La Tabla 5.1 muestra los parámetros principales configurados para las pruebas. Los parámetros son los mismos que se presentan en el ejemplo de la Figura 3.9.

	XBee Rx	**XBee Tx**
Channel	11	11
PANID	9032	9032
DH	0	0
DL	4321	1234
MY	1234	4321
AES Key	A10234F95E93BC890C09E023B34D459F	A10234F95E93BC890C09E023B34D459F
SH	13A200	13A200
SL	40DC073D	40DC0B1A

Tabla 5.1 Configuración de los Xbees

Los módulos XBee pueden transmitir a diferentes velocidades; 1200, 2400, 4800, 9600, 19200, 38400, 57600 y 115200 bps. Se han realizado pruebas a 9600 bps (valor por defecto) y a 57600 bps (máximo valor real soportado por los Xbees). Configurando ambos módulos a 115200 bps, no se aprecia ningún tipo de comunicación. Leyendo en foros se comprueba que los Xbees no son completamente funcionales con este *bit rate*.

Se han generado mensajes de distintas longitudes (ver código para la generación de mensajes en el Anexo VII): 9 bytes (tamaño de los mensajes usados en el proceso de autenticación del sensor), 34 bytes (tamaño de los paquetes transmitidos para indicar que se ha detectado movimiento) y 103 bytes (longitud máxima que se permite enviar datos desde Arduino mediante la interfaz XBee).

Se ha implementado un pequeño contador con el objetivo de determinar si se producen pérdidas se ha implementado un pequeño contador para poder detectar las pérdidas de paquetes, ya que de forma nativa los Xbees no disponen de esta funcionalidad. Este contador no es más que un número de secuencia que se incluye dentro del paquete a transmitir y se incrementa con cada paquete transmitido.

5.1. Efecto de la de distancia

Uno de los aspectos a tener en cuenta es la situación de los nodos sensores con respecto al nodo *router* que recolecta la información de los sensores. Si la distancia entre ambos es demasiado grandes, es posible que no pueda

establecerse la comunicación entre ambos debido a la atenuación de la señal.

En esta sección se presentan las medidas de *throughput* (bits por segundo recibidos) y tasa de pérdidas en función de la distancia. Para ello se ha considerado que el nodo sensor y el *router* se colocan en diferentes puntos de un piso como el que aparece en la Figura 5.1, con visión directa y con obstáculos entre ambos. Se han considerado 4 casos:

- Distancia de referencia entre sensor y *router*: 20 cm.
- Caso 1: La distancia es de 7 metros.
- Caso 2: La distancia es de 5 metros.
- Caso 3: La distancia es de 10 metros.

En estas primeras pruebas se ha considerado un *bit rate* de 9600 bps. El código programado envía una trama de 34 bytes cada 1200ms, con un contador que se incrementa con cada paquete transmitido. Se ha configurado este tiempo porqué es el tiempo medido para la autenticación del sensor. Teniendo en cuenta lo mencionado anteriormente, en un minuto se tendrían que enviar unos 50 paquetes en condiciones ideales.

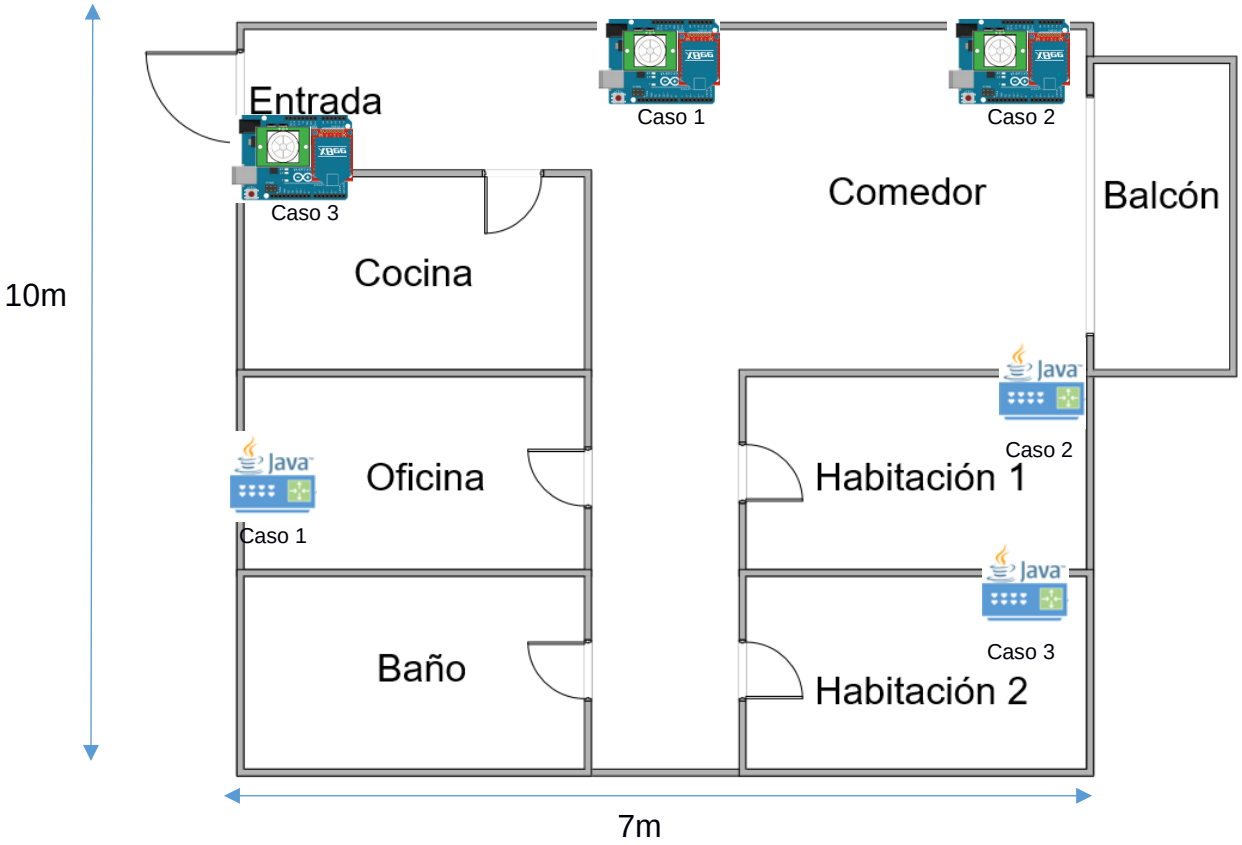

Figura 5.1 Situación del sensor y del router

Para poder visualizar los paquetes y calcular el *throughput*, se ha utilizado el programa nativo de los módulos XBee, XCTU. Se contabiliza el número de paquetes recibidos durante 1 minuto y mediante el contador de paquetes implementado se puede comprobar si se han perdido o no los paquetes.

En la Tabla 5.2 podemos ver el resultado de las pruebas, tanto sin obstáculos entre los XBee como con obstáculos:

Distancia entre sensor y *router*	Núm. de paquetes recibidos sin paredes	*Throughput* sin paredes (bps)	Núm. de paquetes recibidos con paredes	*Throughput* con paredes (bps)
20 cm	49,78	225,69	49,68	225,23
5 m	49,52	224,50	49,00	222,14
7 m	49,33	223,64	48,79	221,17
10 m	47,97	217,33	39,21	177,73

Tabla 5.2 Throughput en función de la distancia y obstáculos

A continuación se analiza con más detalle las pruebas realizadas con obstáculos.

5.1.1 Distancia 5 metros con paredes

En este caso 2 el *router* se encuentra en la habitación 1 y el sensor a la entrada del balcón. Se puede apreciar que hay una única pared entre los XBee. Teniendo en cuenta que está el comedor entre ambos XBee vemos que apenas afecta al *throughput.*

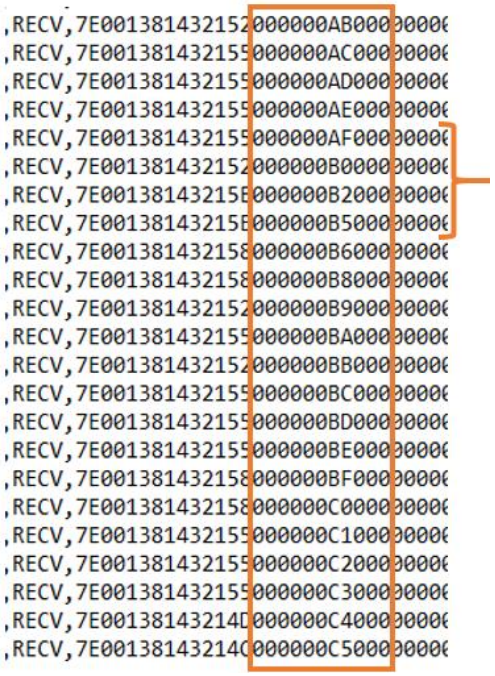

Figura 5.2 Paquetes recibidos con distancia de 5 metros y con paredes

La Figura 5.2 muestra el número de secuencia de los paquetes recibidos. Se puede observar que se pierden 3 paquetes (números de secuencia relativos B1, B3 y B4), ya que los números de secuencia son correlativos. Como se puede apreciar en la Tabla 5.1, el *throughput* no varía mucho, puesto que el número de paquetes perdidos es pequeño en relación al número total de paquetes recibidos.

5.1.2 Distancia 7 metros con paredes

En este caso el *router* se encuentra en la oficina y el sensor entre la entrada y el comedor, de modo que existen 2 paredes entre ambos. De nuevo, el *throughput* prácticamente no se ve afectado. En la Figura 5.3 se puede comprobar que sólo se pierde un paquete por minuto.

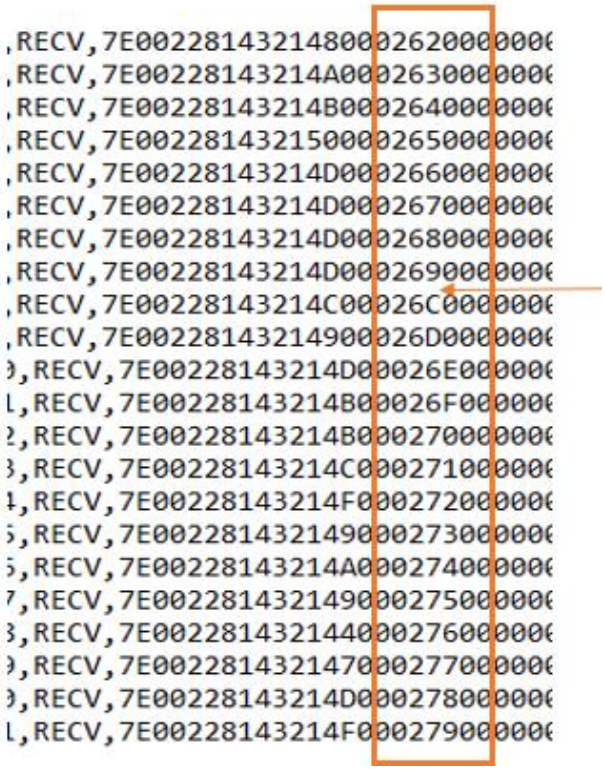

Figura 5.3 Paquetes recibidos con distancia de 7 metros y con paredes

El paquete perdido debería tener número de secuencia relativo 26A.

5.1.3 Distancia 10 metros con paredes

Este es el caso más extremo en el que el *router* se encuentra en la habitación 2 y el sensor justo a la entrada del piso. Entre ambos XBee se encuentra la cocina (y posiblemente el microondas pueda generar interferencias cuando esté en uso) y la oficina (con ordenadores portátiles que pueden generar interferencias por el uso de la conexión WiFi).
Como se puede comprobar en la Tabla 5.2, en este caso se llegan a perder el 30% de los paquetes.

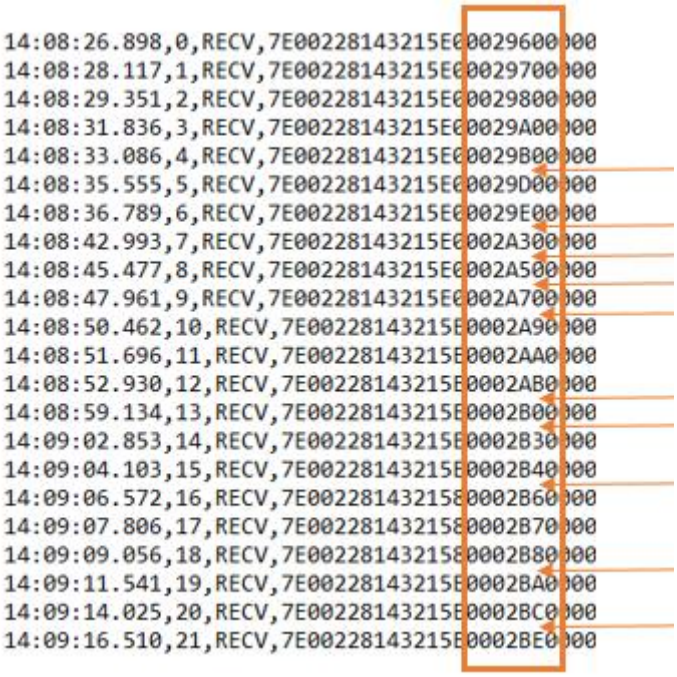

Figura 5.4 Paquetes recibidos con distancia de 10 metros y con paredes

La Figura 5.4 muestra los paquetes perdidos en una de las pruebas con esta distancia. Podemos ver la cantidad de paquetes perdidos que podría haber si los XBee se pusieran en las ubicaciones nombradas anteriormente. Bien es cierto que a nivel de aplicación a penas afectaría, ya que siempre habrá una retransmisión de paquetes. Además los tiempos de los que estamos hablando entre último paquete recibido y primer paquete recibido después de una pérdida, son de pocos segundos, por lo tanto el *router* no pensaría que se ha desconectado.
Aunque este escenario sea el más extremo en cuanto a distancia y obstáculos entre los XBee y aun así no afectase a la comunicación, no se proponer. Esto es debido a que sería mucho más frágil a ataques, como por ejemplo añadir interferencias.

5.2. Efecto del tamaño de paquete y de las interferencias

Uno de los posibles ataques, intencionado o no intencionado, que podría recibir la red de sensores, consiste en generar interferencias en el canal a través del cual se comunican los sensores (*jamming*). Con el objetivo de evaluar el impacto que pueden tener las interferencias en la comunicación de los Xbees, se ha realizado un escaneo de canales para comprobar cuáles son los más usados y los más afectados por las interferencias generadas por un microondas (que opera en la banda de frecuencia de 2.4Ghz, como los Xbee).

Mediante un analizador de canales, en concreto el Fluke networks AirCheck Wifi-tester [19], hemos obtenido el siguiente gráfico:

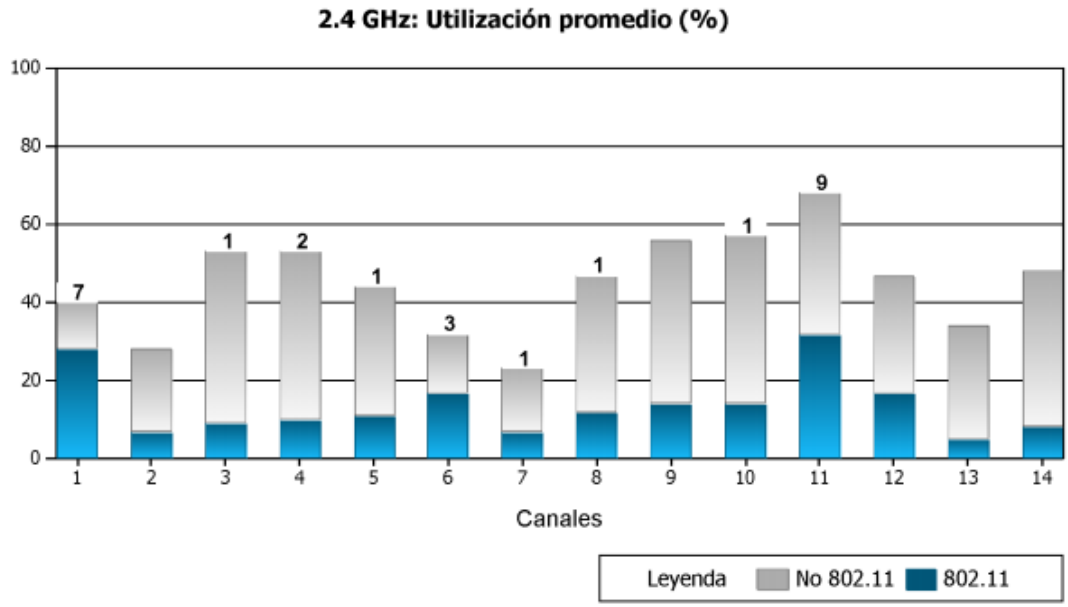

Figura 5.5 Interferencias y uso en la banda de 2.4Ghz.

Las franjas de color azul representan la utilización que se está haciendo en un canal determinado por parte de dispositivos que usan el estándar 802.11, mientras que las franjas de color gris representan el ruido que genera el microondas al calentar un vaso de agua. Los números que vemos en la parte superior de las barras son los puntos de accesos que hay operando en cada canal.
Como se puede observar, los canales más afectados son el 10 y el 11. Por este motivo, se han realizado las pruebas con los dos XBee operando en el canal 11.

La Tabla 5.3 muestra el *throughput* y la tasa de paquetes perdidos con y sin interferencias con el *bit rate* por defecto, 9600bps, y una distancia entre Xbees de 20 cm. En este caso se intenta conseguir el máximo througput, así que se envía un paquete cada 100 ms.

Tamaño paquetes	**Numero de paquetes sin microondas**	***Throughput* sin microondas (bps)**	**Numero de paquetes con microondas**	**Throughput con microondas (bps)**	**% paquetes perdidos**
9 bytes	3.184,92	3.821,90	2.689,22	3.227,07	15,56
34 bytes	1.263,31	5.727,01	918,04	4.161,76	27,33
103 bytes	469,98	6.454,35	279,98	3.845,11	40,43

Tabla 5.3 Throughput con y sin interferencias a 9600 bps

En el caso sin interferencias, el *throughput* aumenta con el tamaño del paquete. Es razonable, ya que se está enviando un paquete (independientemente del tamaño) cada 100ms. Los valores de *throughput* son mayores que en la sección 5.1 puesto que la tasa de envío es mucho más alta.
Sin embargo, se puede observar que con interferencias se reduce considerablemente el *throughput*, especialmente cuando el tamaño del paquete aumenta. La tasa de pérdidas es de un 15% cuando los paquetes tienen un tamaño de 9 bytes. Como se ha comentado, estos son los paquetes que

transmite el sensor para autenticarse. Si el sensor no recibe respuesta intenta autenticarse de nuevo, así que en este caso la pérdida de paquetes no representa un problema grave: sencillamente el sensor tarda un poco más en autenticarse.

Cuando el tamaño de los paquetes es de 34 bytes, es decir, se envían paquetes con información sobre la detección de presencia en la casa, se pierde más de una cuarta parte de los paquetes. Analizando la traza (obtenida mediante XCTU) que contiene el registro de los paquetes recibidos y su número de secuencia se observa que no hay retransmisión de paquetes por parte de los Xbee. En este caso las interferencias sí tienen un efecto nefasto en nuestro sistema, puesto que puede perderse información sensible.

El hecho de que la tasa de pérdidas aumenta con el tamaño del paquete puede deberse a que para una misma tasa de error de bit, la probabilidad de recibir un paquete corrupto aumenta con el tamaño del paquete.

Se ha realizado un conjunto de pruebas análogas a las anteriores pero aumentando el *bit rate* a 57600. Los resultados se muestran en la Tabla 5.4.

Tamaño paquetes	**Numero de paquetes sin microondas**	***Throughput* sin microondas (bps)**	**Numero de paquetes con microondas**	***Throughput* con microondas (bps)**	**% paquetes perdidos**
9 Bytes	9.053,06	10.863,67	6.164,22	7.397,06	31,91
34 Bytes	5.140,94	23.305,59	3.709,40	16.815,92	27,85
103 Bytes	2.577,64	35.399,64	1.405,72	19.305,24	45,46

Tabla 5.4 Throughput con interferencias a 57600 bps

Sin interferencias, el *throughput* aumenta con el tamaño de paquete, tal y como ocurría en el caso con bit rate 9600 bps. Cuando se generan interferencias, la tasa de pérdidas es mucho mayor en este caso.

```
18:02:50.335,5648,RECV,7E000981432128008F00000063
18:02:50.335,5649,RECV,7E0009814321380090000000052
18:02:50.335,5650,RECV,7E000981432128009800000005A
18:02:50.384,5651,RECV,7E0009814321280099000000059
18:02:50.385,5652,RECV,7E000981432138004000000042
18:02:50.385,5653,RECV,7E000981432128004100000051
18:02:50.385,5654,RECV,7E000981432129004200000004F
18:02:50.386,5655,RECV,7E000981432133004300000044
18:02:50.387,5656,RECV,7E000981432134004400000042
18:02:50.388,5657,RECV,7E000981432129004500000004C
18:02:50.496,5658,RECV,7E00098143213A00C900000017
18:02:50.496,5659,RECV,7E00098143212B00D00000001F
18:02:50.496,5660,RECV,7E00098143212A00C800000028
18:02:50.496,5661,RECV,7E00098143212A00C700000029
18:02:50.496,5662,RECV,7E000981432139003F00000022
18:02:50.496,5663,RECV,7E000981432129003E00000033
18:02:50.496,5664,RECV,7E000981432129003D00000034
18:02:50.496,5665,RECV,7E00098143213A00B60000002A
18:02:50.496,5666,RECV,7E00098143212A00B40000003C
18:02:50.496,5667,RECV,7E000981432138003B200000030
```

Figura 5.6 Traza de paquetes recibidos 57600 bps, 9 Bytes con microondas

La Figura 5.6 muestra los paquetes recibidos cuando hay interferencias y se usan paquetes de 9 bytes. En este caso, puede comprobarse que los Xbees sí retransmiten algunos de los paquetes, aunque no se ha conseguido deducir porqué en este caso sí hay retransmisión selectiva (no todos los paquetes son retransmitidos). Una posible solución consiste en forzar las retransmisiones a nivel de aplicación, para asegurarnos de que la información crítica sí llega al receptor.

5.3. Análisis de seguridad

En esta sección se pretende analizar el efecto que puede tener un nodo malicioso que se conecta a la WSN. Se han considerado dos perfiles de atacante:

- El nodo malicioso intenta leer el contenido de los mensajes (*sniffing*) sin disponer la clave AES que comparten los nodos de la red.
- Suplantar la identidad del nodo sensor o del *router*.
- Generar un mensaje falso.

La Figura 5.7 muestra una traza del contenido capturado por un nodo conectado a la WSN.

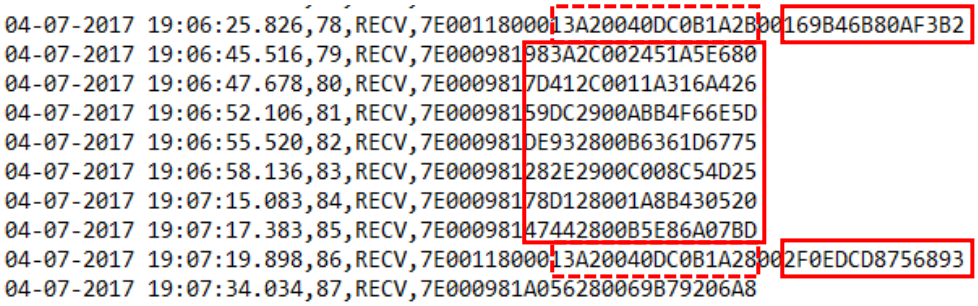

Figura 5.7 Paquetes capturados por el nodo malicioso

Los distintos campos que aparecen en la captura, a excepción de la fecha y hora de recepción del paquete, están codificados en hexadecimal y son los siguientes:

- El primer byte es el delimitador (7E) de trama.
- Los dos siguientes indican la longitud del mensaje.
- El cuarto byte indica el tipo de trama.
- El quinto y sexto byte indican la dirección origen de la trama.

Los recuadros con línea discontinua representan la dirección MAC de 64 bits, mientras que los recuadros con línea continua representan el *payload* cifrado. Por tanto, un nodo malicioso que capture un paquete no podrá leer el contenido del paquete. El único modo que tiene de intentar descifrar el mensaje es mediante un ataque por fuerza bruta, es decir, intentar adivinar la clave que se ha usado para el cifrado. Dado que AES usa claves de 128 bits, el número de posibles claves es 2^{128} y por lo tanto un ataque por fuerza bruta es inviable.

Para suplantar la identidad de un Xbee autorizado, bien sea el sensor o el *router*, el nodo malicioso debe ser capaz de autenticarse correctamente con ell

mecanismo descrito en la sección 3.2.1. Tal y como se ha explicado, para ello es necesario conocer el valor de la clave secreta de 128 bits. De nuevo, la probabilidad de generar un valor de autenticación válido por fuerza bruta es despreciable. Otra posibilidad es ejecutar un ataque de *replay*, es decir, capturar un mensaje enviado por un Xbee legítimo y reenviarlo para poder autenticarse. Gracias a la generación de *challenge* (valor aleatorio que cambia en cada proceso de autenticación), no es posible ejecutar este ataque.

6. Conclusiones y líneas futuras

Una de las posibles aplicaciones de las redes inalámbricas de sensores o WNSs es la detección de presencia en hogares. En este proyecto se ha querido implementar un servicio de detección de intrusos, que podría ofrecer una empresa de seguridad a un conjunto de hogares.

Para ofrecer este servicio, es necesario desplegar una red de sensores en el hogar del cliente. El número de sensores y la localización de los mismos depende de la distribución de la casa o piso. Dichos sensores se encargarán de detectar movimiento e informar a un nodo coordinador, cuya función puede implementarse en un PC o en el *router* que tenga el cliente en su hogar.
El nodo coordinador se comunica con un servidor implementado en Node.js gestionado por la empresa de seguridad, que se encargará de procesar toda la información y generar alarmas de seguridad, si es necesario.

Uno de los objetivos de este proyecto, más allá de diseñar e implementar el servicio de detección de intrusos, era proteger dicho servicio contra posibles ataques. Por este motivo, se han implementado mecanismos de seguridad tanto en la WSN como en la comunicación entre *router* y servidor. Los datos se envían cifrados en ambos casos y se implementan mecanismos de autenticación para verificar la fuente de los datos e impedir que un usuario no autorizado pueda inyectar información falsa, bien suplantando la identidad de un sensor en la WSN, o bien suplantando la identidad del *router* en la comunicación con el servidor. La autenticación dentro de la WSN se consigue mediante el uso de MACS, mientras que en el caso del *router*-servidor se ha optado por usar certificados digitales.

Como resultado del sistema implementado, el servicio es robusto a ataques conocidos como *sniffing*, suplantación de identidad, ataques de *replay*, ataques MITM, etc

Durante el proyecto se han encontrado algunos problemas y limitaciones. La primera limitación ha sido los módulos XBee Series 1. Estos módulos tienen una velocidad de transmisión máxima más baja (57600 bps) de lo que indica el datasheet (255200 bps). Por otro lado, el módulo es muy inestable cuando funciona a su máxima velocidad de transmisión, tal y como se ha podido ver en las pruebas con interferencias.

También hubo problemas con la programación de Arduino, debido a que las librerías de AES para Arduino disponibles online presentan pequeños errores, que se han solucionado revisando el código.

Otro problema encontrado durante el proyecto ha sido la implementación de la autenticación mutua por parte del router. Esto fue debido a como gestiona la propia máquina de Java los certificados digitales. En cuanto a la implementación del servidor, hubo problemas para poder subir el código del servidor a Openshift, hasta que finalmente se optó por usar otra plataforma como Heroku.

El resultado de este trabajo es un producto funcional y con protección frente a

los ataques más típicos. Como posibles líneas futuras, se sugieren los siguientes puntos. En la WSN, se podrían agregar diferentes sensores (de presencia, temperatura, etc) para ampliar las funcionalidades y poder, por ejemplo, indicar la posición en la que se ha detectado movimiento (serían necesarios al menos 3 sensores y aplicar triangulación). Otra posible ampliación consistiría en poner una pequeña cámara para poder visualizar qué ocurre en la casa cuando se detecta movimiento, aunque esta nueva funcionalidad sería más costosa, tanto a nivel de dificultad como económicamente.

En cuanto a la red externa, el primer paso sería desarrollar una aplicación móvil para el cliente tenga acceso a los datos registrados en el servidor relativos a la detección de presencia en su casa, o pudiera recibir alertas. Una vez desarrollada la aplicación básica, se podría personalizar de modo que el usuario pueda, por ejemplo, activar el sistema de detección de forma remota y en el horario deseado.

Este proyecto me ha permitido aplicar gran parte de los conceptos y herramientas que he aprendido durante la carrera y profundizar en ellos, enfocándome principalmente en la parte de redes de comunicación. La principal motivación para desarrollar este escenario fue que combina dos de los temas que, a nivel personal, resultan más interesantes y actualmente se encuentran en auge: los sensores y la seguridad.

7. Referencias

[1] Shio Kumar Singh, M- P- Singh y D. K. Singh. "*Routing Protocols in Wireless Sensor Networks – A Survey*". (Vol.1, No.2, Nov. 2010)

[2] Al-Sakib Khan Pathan, Hyung-Woo Lee y Choong seon Hong. "*Security in Wireless Sensor Neworks: Issues and Challenges"* (Feb. 2006)

[3] 3Com. Technical Paper. IEEE 802.11b Wireless LAN's.

[4] The Institute of Electrical and Electronics Engineers (IEEE). "*802.15.1, Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personas Area Networks (WPANs)*". (2002)

[5] The Institute of Electrical and Electronics Engineers (IEEE). "*802.15.4, Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*". (2003)

[6] Javier M. M. y Daniel R. F. "Informe Técnico: Protocolo ZigBee (IEEE 802.15.4)". (Jun. 2007)

[7] G. Montenegro, N. Kushalnagar, J. Hui, D.Culler. "*Transmission of IPv6 Packets over IEEE 802.15.4 Networks*". (Sept. 2007)
[Disponible Online]:
https://tools.ietf.org/html/rfc4944

[8] IEEE Standards Association. "*802.15.6-2012 – IEE Standard for Local and metropolitan área networks - Part 15.6: Wireless Body Area Networks*". (Feb. 2012)

[9] Rishav Dubey, Vikram Jain, Rohit Singh Thakur y Siddahart Dutt Choubey. "*Attacks in Wirless Sensor Networks*". (May. 2012)

[10] Datasheet Modulo XBee Series 1.[Disponible Online]:
https://www.digi.com/pdf/ds_xbeemultipointmodules.pdf

[11] Página oficial de DigiInterntional. [Disponible Online]:
https://www.digi.com/products

[12] Base de datos MySQL: [Disponible Online]:
https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html

[13] Lenguaje de programación Java. [Disponible Online]:
https://www.thoughtco.com/what-is-java-2034117

[14] RFC 2631. "*Diffie-Hellman Key Agreement Method*". (Jun 1999).

[15] Lenguaje de programación Node JS. [Disponible Online]: https://nodejs.org/en/about/

[16] Plataforma de servicio Heroku. [Disponible Online]: https://devcenter.heroku.com/categories/application-architecture

[17] Mecanismo de restricción Cross Origin Resource Sharing (CORS). [Disponible Online]: http://www.arquitecturajava.com/que-es-cors/

[18] Autenticación mediante Tokens.[Disponible Online]: https://docs.docker.com/registry/spec/auth/token/#requesting-a-token

[19] Medidor Canales Wifi: Fluke networks AirCheck Wifi-tester. [Disponible Online]: https://www.newegg.com/Product/Product.aspx?Item=9SIA91J4343469

[20] Entorno de desarrollo Arduino. [Disponible Online]: https://www.arduino.cc/en/Guide/Environment#

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXOS

TÍTOL DEL TFG: Seguridad en redes de sensores
TITULACIÓ: Grau en Enginyeria Telemàtica
AUTOR: Iván Frago Álvarez
DIRECTOR: Olga León Abarca
DATA: 3 de Julio del 2017

ANEXO I. Funcionamiento del sensor de presencia PIR

Un sensor infrarrojo pasivo (o las siglas en inglés PIR) es un sensor electrónico que calibra la luz infrarroja que irradian los objetos que tiene a su alrededor Así, es capaz de detectar movimiento en radio de visión, que es aproximadamente de unos 4 metros con un ángulo de unos 160º tridimensionalmente.
El sensor PIR mide la radiación calorífica de los objetos, es decir, saca provecho de que todos los objetos con una temperatura por encima de 0º Celsius irradian calor. Por lo tanto, esa radiación se puede detectar aunque no sea visible para el ojo humano, debido a que irradia a unas longitudes de onda infrarrojas. Se denomina sensor pasivo debido a que no genera dichas ondas, solo las detecta.
El componente principal es un sensor piroeléctrico. Dicho sensor dispone de un componente electrónico diseñado para detectar las radiaciones infrarrojas. En realidad cada sensor está dividido en dos campos y hay un circuito eléctrico que compensa ambas mediciones. Si ambos campos reciben la misma radiación la señal eléctrica que generan es nula, pero si un cuerpo pasa por delante, harán que ambos campos emitan una radiación diferente, por lo tanto, esto generará una señal eléctrica.

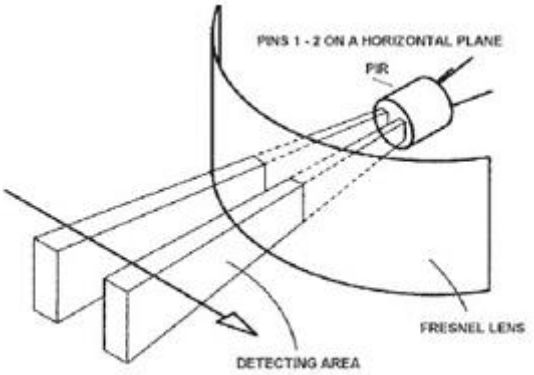

Figura A-1.1 Representación de detección objeto en un sensor PIR

Un elemento importante en el sensor es la óptica, es decir, el encapsulado de plástico que cubre el sensor, tal y como se muestra en la **¡Error! No se encuentra el origen de la referencia.** Básicamente es una cúpula de plástico formada por lentes de *Fresnell*, que divide el espacio en zonas, así enfocando la radiación a cada uno de los campos del componente piroeléctrico.
El sensor PIR es un elemento muy fácil de usar, ya que dispone de 3 patillas para la alimentación y el resto permiten ajustar el sensor, tal y como se muestra en la Fig. ***A-1.2*** *Esquema de patillaje del sensor PIR.*

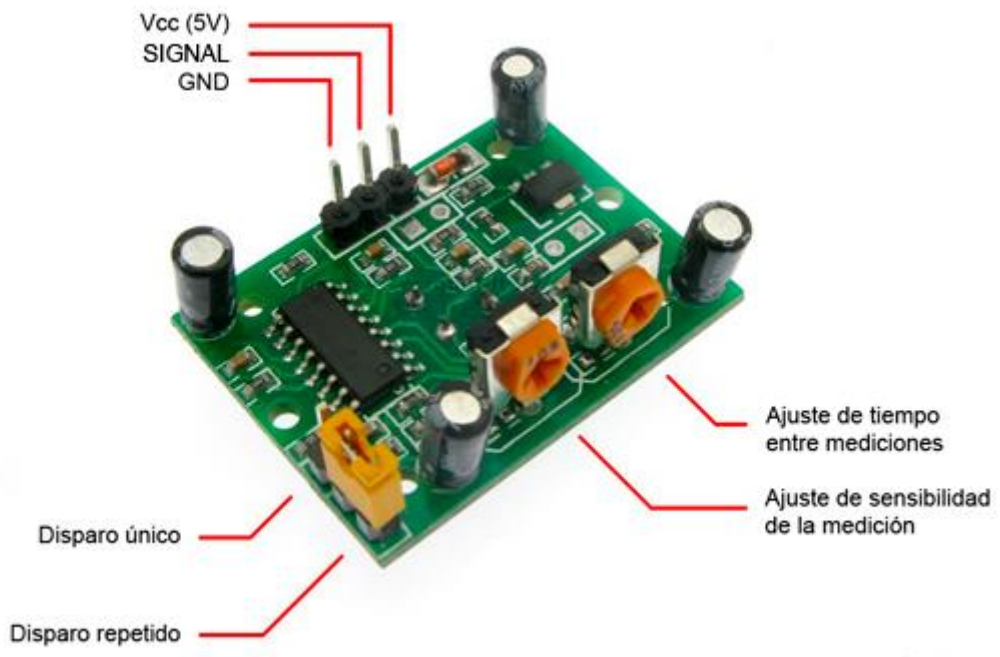

Fig. A-1.2 *Esquema de patillaje del sensor PIR.*

ANEXO II. Programar placa Arduino

Para poder gestionar los datos del sensor PIR y enviarlos al *route*r, es necesario programar la placa Arduino UNO en C++. El código debe de tener una estructura básica: una función setup() y una función loop().

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Fig. A-2.1 Estructura básica C++ para Arduino

En la función de *setup*() se inserta el código que se desea ejecutar cuando se inicia Arduino. A continuación, se ejecuta la función L*oop*() de forma indefinida.
Arduino ofrece un entorno propio de desarrollo integrado en la placa. Gracias a esta herramienta se puede programar el código en C++ y luego cargarlo en la placa Arduino. El IDE de Arduino usa el concepto de *Sketchbook*[20], un lugar estándar para almacenar programas (sketches). Dicho entorno contiene un editor de texto para poder escribir el código y comentarios, un área de mensajes llamada Monitor Serie para ir mostrando valores, una consola de texto para poder visualizar posibles errores a la hora de subir el código, una barra de herramientas con botones para las funciones más comunes y una serie de menús.
Cuando se abre el entorno de desarrollo para Arduino, aparece un editor de texto con la estructura típica de un código en Arduino, con su función *setup*() seguida de la función *loop*().
El IDE de Arduino dispone de un menú con diferentes funciones que permiten, entre otras cosas, compilar el código y cargarlo en la placa. Para realizar esta última tarea hay que tener en cuenta 3 sub-menús: el submentú Puerto, que identifica el puerto serie del ordenador al que está conectado la placa Arduino. El segundo sub-menú, Placa, permite especificar el modelo de placa Arduino con el que se está trabajando. Por último el submenú Monitor Serie, permite visualizar los *Serial.println*() que se han escrito en el código, es el equivalente a un *printf()* en lenguaje C, y empezar a ejecutar dicho código.
Por último, cabe mencionar la consola de texto, a través de la cual Arduino IDE informa del porcentaje utilizado de la memoria de la placa Arduino debido a la ejecución del código. También informa de posibles errores de compilación o simplemente de lo que está sucediendo con el código.

ANEXO III. Funcionamiento XBee Serie 1

a. Modos de funcionamiento

Los módulos XBee tienen diferente modos de comunicarse:

- **Modo Transparente:** Configuración básica, en la que al introducir dirección destino en la configuración propia del módulo, ya empieza a funcionar. Es un modo muy limitado, pero es completamente transparente para el usuario.
- **Modo API 1 y API 2:** Configuración más avanzada que permite enviar y recibir datos a través de "paquetes" mediante estructuras de datos predefinidas.
- **Modo de comandos:** Es utilizado para cambiar parámetros de configuración del XBee, se puede hacer localmente o remotamente, un XBee es configurado a partir de la información que recibe de otro módulo.

El modo transparente es el modo más sencillo de configurar para la comunicación, pero a la vez el más simple y más limitado. Hay funciones casi imposibles de implementar en este modo como transmitir a una dirección *Broadcast*, modificar remotamente la configuración de un módulo XBee o algo tan sencillo como cambiar de dirección destino en cada transmisión.
El modo API es el más complicado a nivel de programación pero provee de una mayor flexibilidad a la hora del envío y recepción de datos.
Como se ha comentado antes, a diferencia del modo transparente, en el modo API, el módulo espera por una secuencia específica de Bytes (7E), como si fuese un *tag*. Gracias a este *tag* el módulo sabe que empieza un *frame* (o paquete). Cada paquete presenta la siguiente estructura:

Fig. A-3.1 Frame en modo API

Donde:

- **Byte 1:** Es el inicio de *frame*, el valor de este primer Byte es siempre 0x7E.
- **Byte 2, 3:** Esta parte indica la longitud del *frame*. Contabiliza los bits entre el byte menos significativo de tamaño hasta el byte de a suma de verificación o *checksum*.
- **Byte 4-N:** En esta parte, como indica la **¡Error! No se encuentra el origen de la referencia.**, contiene la estructura específica a la API, donde se explicará más abajo con más detalle.
- **Byte N+1:** Gracias a este campo se verifica la integridad de los datos. Para calcularlo se suman todos los bytes, excepto los delimitadores y los que dicen la longitud, del resultado solo se coge los 8 bits de la derecha. Por último se le resta a 0XFF los 8 bits del resultado. Por lo tanto para verificarlo, se suma todos los bytes, excepto los delimitadores y las longitudes, y se le resta 0XFF, si no cuadra la operación se descarta el

paquete.

Tanto el contenido como la longitud del paquete varían en función del tipo de datos que se envía. Esto se puede saber en función del comando API. Si se quisiera transmitir un dato a un XBee usando una dirección de 16 bits se tendría que utilizar la siguiente estructura:

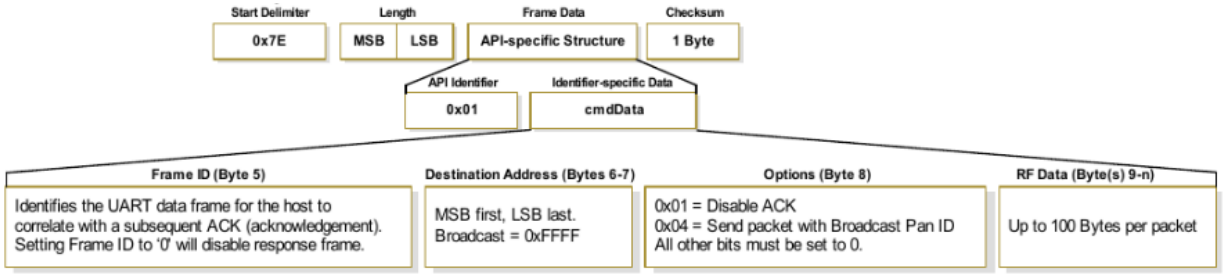

Fig. A-3.2 Estructura de un Frame en modo API

Dicho ejemplo tendría esta estructura:

- **Byte 4:** Identificador de API, gracias a este byte se identifica la acción del *frame*. En este caso el byte 4 es igual a 0x01 que corresponde a "*Transmit Request*: 16-Bit *Address*", una solicitud de transmisión de datos a una dirección de 16 bits.
- **Byte 5:** El valor es un número secuencial del paquete o *frame*. Por cada paquete enviado se recibe un ACK, este número ayuda a verificar que el anterior paquete fue recibido por el destinatario.
- **Byte 6 y 7:** Corresponde a la dirección del XBee a que va dirigido el paquete. Se pone el byte más significativo por delante.
- **Byte 8:** Corresponde a las opciones de envío: 0x01(no ACK) y 0x04(*broadcast*).
- **Byte 9-N:** *Payload* del *frame*. Tiene un máximo de 100 bytes.

Identificador API	Tipo de mensaje
0x8A	Estado del módulo
0x08	Comando
0x09	Aplicar un comando
0x88	Respuesta a un comando
0x00	Requerimiento de transmisión (dirección 64-bits)
0x01	Requerimiento de transmisión (dirección 16-bits)
0x89	Estado de la transmisión
0x80	Paquete recibido (dirección 64-bits)
0x81	Paquete recibido (dirección 16-bits)

Fig. A-3.3 Identificadores API

El modo API 1 y el 2 son prácticamente iguales, con la única diferencia que el modo API 2 (una mejora del 1 por un fallo) tiene secuencias de byte de escape. Con esto nos referimos a que debido que la secuencia 0x7E significa inicio de paquete, si este valor se encuentra dentro del paquete, el módulo XBee descartará lo anterior leído ya que se pensará que 0x7E es el inicio de un paquete nuevo, no un valor del mismo paquete. Por lo tanto el modo API 2 solventa este problema.
El modo comandos es la forma en que se puede configurar el módulo XBee desde el propio puerto Serial. Dichos comandos se realiza a través de los "comandos AT", que estos no son nada más que palabras precedidas de las letras AT, de ahí el nombre. En realidad el software XCTU no es más que una interfaz que nos permite modificar los parámetros del XBee de forma fácil.

b. Descubrimiento de XBees

Para poder ver qué XBees están conectados, mediante XCTU, hay que clicar sobre el icono, tal y como se muestra en la Fig. A-3.4 *Descubrimiento XBee remotos*En ese caso el XBee escanea el medio para ver qué dispositivos tiene alrededor.

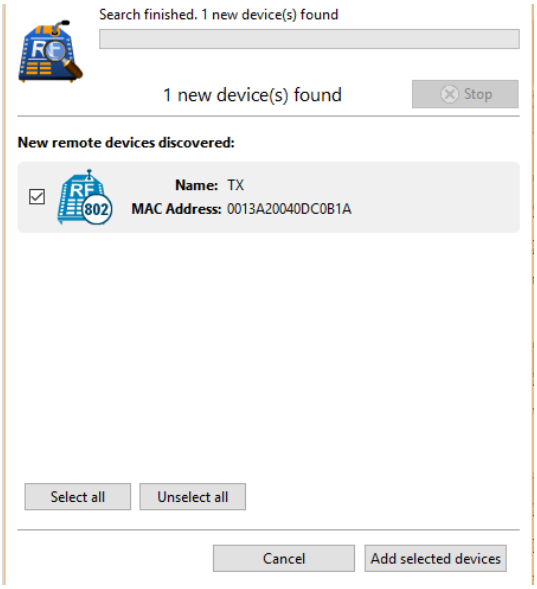

Fig. A-3.4 Descubrimiento XBee remotos

En el ejemplo que se muestra en la Fig. A-3.4 *Descubrimiento XBee remotos* sólo hay un XBee conectado. Una vez seleccionado basta con darle al botón con forma de engranaje para poder ver sus parámetros.

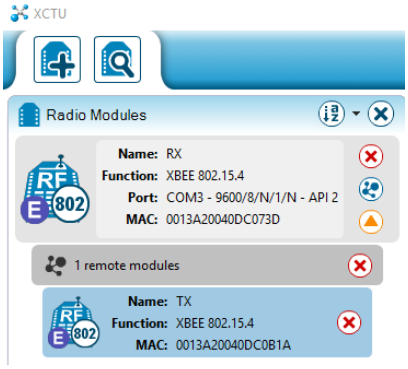

Fig. A-3.5 Configurando XBee remotamente

Nótese que XCTU muestra el identificador y la @MAC de 64 bits del XBee remoto.

c. Visualizar conversaciones de la red

XCTU también muestra los mensajes transmitidos en la red de sensores, Para poder acceder a dicha característica de la aplicación hay que clicar sobre el icono de Consola situado en la esquina superior derecha de la aplicación.

Una vez que entramos en la consola de XCTU, para poder visualizar los mensajes hay clicar el icono con el dibujo de dos conectores. En el ejemplo se puede ver como se envían comandos AT. La sección *Frame details*, nos muestra el mensaje en hexadecimal y explica los diferentes campos del mensaje.

En la consola también se pueden crear *frames* y transmitirlos. La herramienta con la que se crean *frames* en XCTU se llama *Frame Generator*. Dicha herramienta nos permite indicar el protocolo del mensaje, que puede escogerse entre un amplio abanico de protocolos: 802.15.4, *ZigBee*, *Digi-Mesh*, entre otros. También se puede indicar el tipo de *frame* que se quiere generar, como por ejemplo un mensaje *Tx Request* de dirección 64-bit o de 16-bit, o un *Tx* Status o también un AT *Command*. Ademásde poder modificar y generar *frames*, también permite modificar a nivel del *payload*, es decir, modificar la ID del *frame* y las opciones (no ACK, *Broadcast*, con *Traceroute* o simplemente sin opciones).

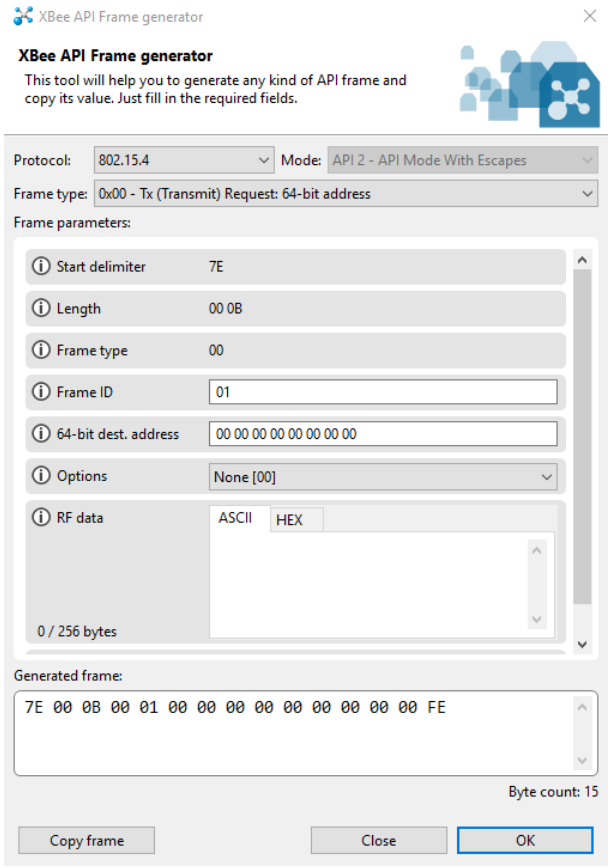

Fig. A-3.6 Frame Generator de XCTU

ANEXO IV. Diagramas y Funciones clave

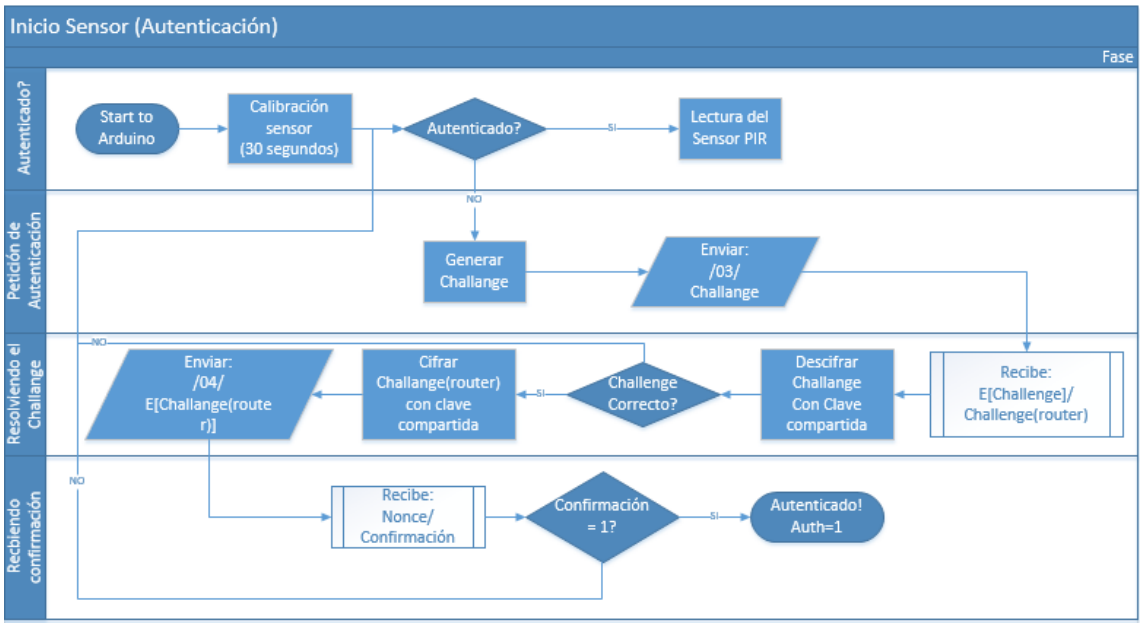

Fig. A-4.1Diagrama de proceso para autenticación

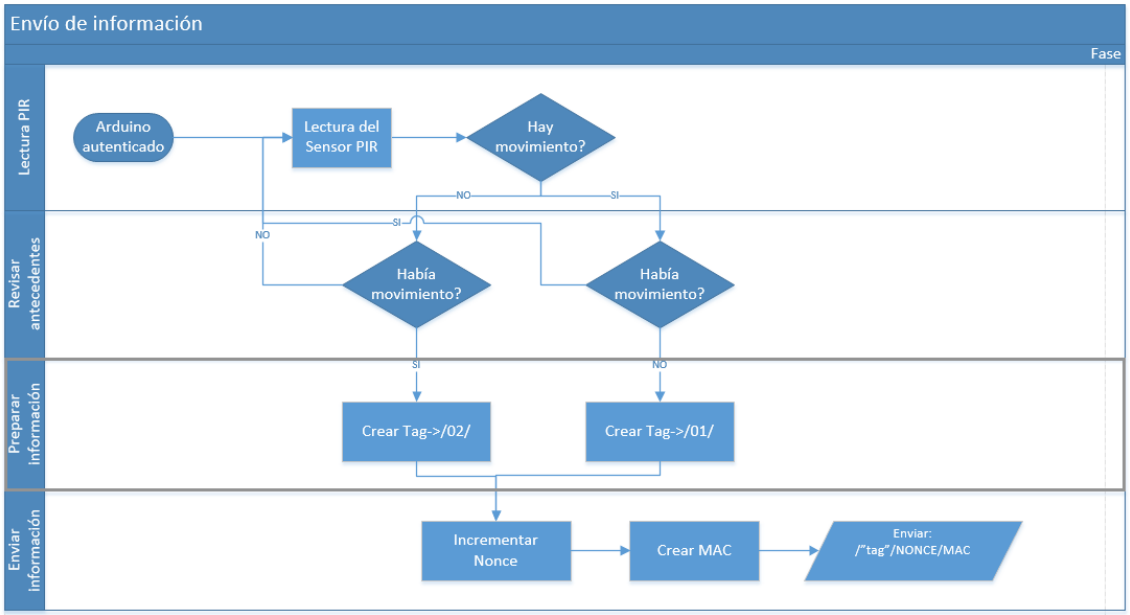

Fig. A-4.2 Diagrama del proceso de detección de movimiento del nodo sensor.

```
void plusNonce () {
  Serial.println("PLUSNONCE()-------------");
  //Convert nonce from uint8_t to integer.
  Serial.println("Size of NonceInt: ");
  Serial.println(sizeof(nonceInt));
  Serial.println("NonceInt: ");
  int i = 0;
  for (i = 0; i < sizeof(nonce); i++) {
    nonceInt[i] = (int) nonce[i];
    Serial.println(nonceInt[i]);
  }
  Serial.println("i: ");
  Serial.println(i);
  //Plus 1 to nonceInt: 48->0, 49->1, 50->2, 51->3, 52->4, 53->5, 54->6, 55->7, 56->8, 57->9.
  for (int j = sizeof(nonce) - 1; j >= 0; j--) { //Usamos J para movernos en el vector
    Serial.println("j:");
    Serial.println(j);
    int cont9 = 0;                          //Usamos cont9 para contar cuantos 9 hay por a partir de nuestra
    //posición para adelante (derecha a izquierda)
    if (j == sizeof(nonce) - 1 && nonceInt[j] == 57) {          // Localizamos si hay '9'
      for (int k = j; k >= 0; k--) {    //Bucle cuenta 9: Contamos las cifras de adelante son '9' también,
        //las almacenamos en la variable cont9
        if (nonceInt[k] == 57)  cont9++; //Por cada 9 (57 en uint8_), aumentamos cont9
        else break;                     //Si ya no encontramos otro 9, se sale del bucle cuenta 9
      }
      if (cont9 != 0) {                 // Si hay alguna cifra consecutiva más con un 9...
        for (int l = 0; l < cont9; l++) { // la convertimos en 0
          nonceInt[j - l] = 48;           // dicho 9 lo pasamos a 0(48 en uint8_t);
        }
      }
      nonceInt[j - cont9] = nonceInt[j - cont9] + 1; //La siguiente cifra despues del ultimo 9, que ahora es 0,
      //le sumamos 1.
      break;                                //Como ya hemos hecho la suma de todos los 9 -> 0
      //y el siguiente +1 salimos del bucle de sumar.
    }
    nonceInt[j] = nonceInt[j] + 1;           //Si no ha encontrado ningún 9 en las unidades, le suma traquilamente.
    break;   // salimos del rollo este
  }
  i = 0;
  //Convert nonceIntfrom integer to uint8_t
  Serial.println("Nonce: ");
  for (i = 0; i < sizeof(nonce); i++) {
    nonce[i] = (uint8_t) nonceInt[i];
    Serial.println(nonce[i]);
  }
}
```

Fig. A-4.3 Función incremento de nonce

```
void sendWithMAC(uint8_t datasend[]) {
  Serial.println("Send data with MAC----------");
  Serial.println("MAC:");
  for (int i = 0; i < sizeof(datasend); i++) {
    MAC[i] = datasend[i];
    Serial.println(MAC[i]);
  }
  aes128_cbc_enc(key, iv, MAC, sizeof(MAC));
  uint8_t datasendWithMAC[] = {'0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
                               '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
                               '0', '0', '0', '0', '0', '0', '0', '0', '0'}; //length=29
  Serial.println("datasendWithMAC:");
  for (int i = 0; i < sizeof(datasendWithMAC); i++) {
    if (i < 13) {
      datasendWithMAC[i] = datasend[i];
      Serial.println(datasendWithMAC[i]);
    } else {
      datasendWithMAC[i] = MAC[i - 13];
      Serial.println(datasendWithMAC[i]);
    }
  }
  tx16 = Tx16Request(0x1234, datasendWithMAC, sizeof(datasendWithMAC));
  xbee.send(tx16);
}
```

Fig. A-4.4 Función inserción código MAC

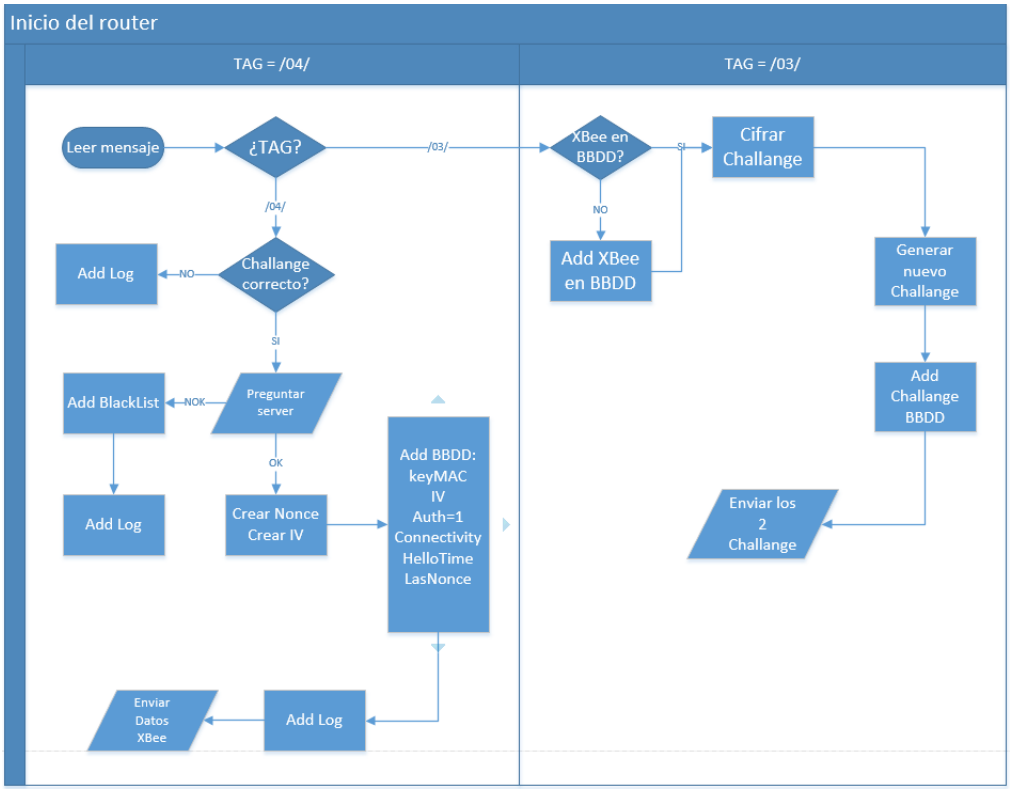

Fig. A-4.5 Proceso de autenticación de un XBee

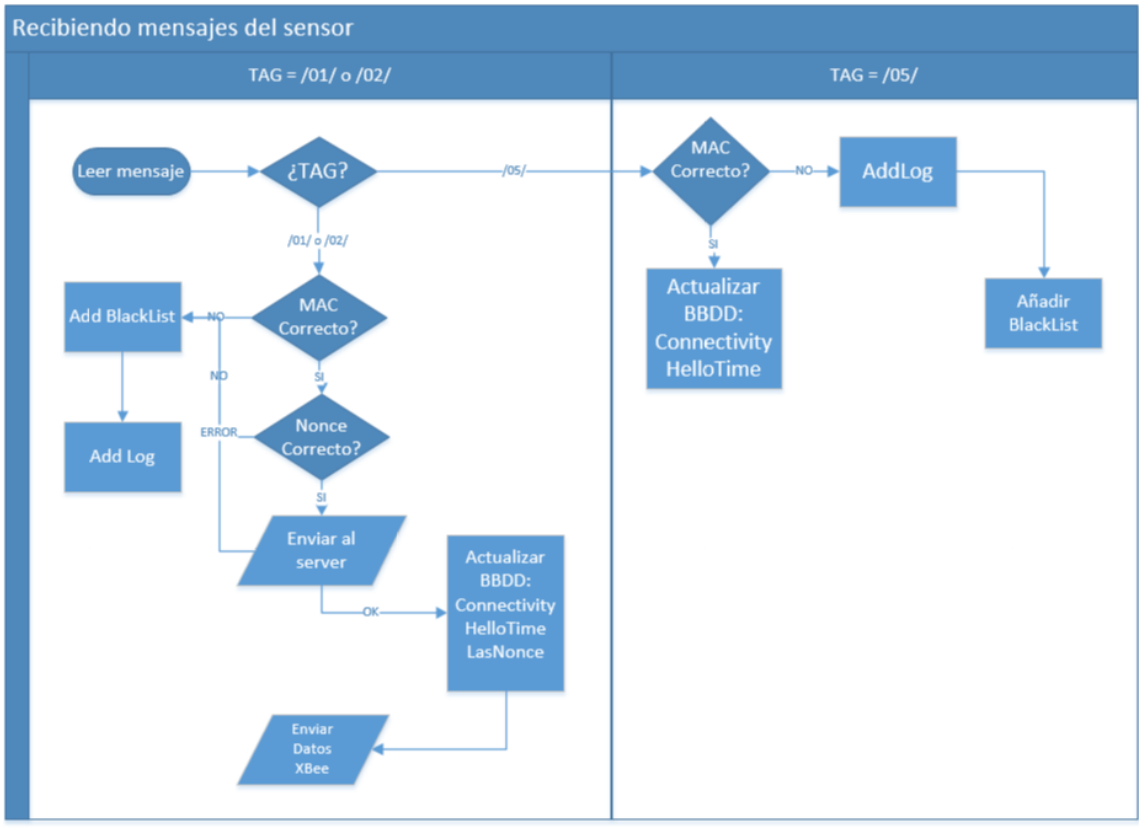

Fig. A-4.6 Proceso envío informe de movimiento

ANEXO V. Tecnologías y herramientas usadas

a. HTTPS y TLS

Para poder comunicar de forma segura el r*oute*r con el servidor y la aplicación Android con el servidor, hace falta un protocolo de seguridad. Éste tiene que cifrar el contenido del mensaje, proporcionar integridad al mensaje y que nos verifique que se está tratando con el receptor que queremos, es decir, evitar la posible suplantación de identidad.
Al tener un servidor con una API REST, que trata peticiones HTTP (GET, PUT, POST, DELTE,…) se tiene que implementar HTTPS (Hypertext Transfer Protocol Secure) en las comunicaciones, para proporcionar seguridad en el escenario.
El protocolo HTTPS no es más ni menos que el HTTP implementando una subcapa de seguridad.

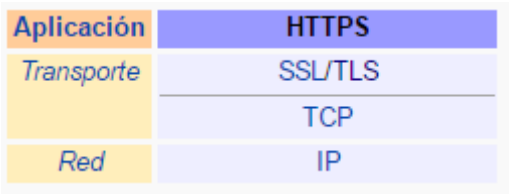

Fig. A-5.0 Ubicación protocolo HTTPS en modelo OSI

HTTPS opera en la capa más alta del modelo OSI, en la capa de aplicación, mientras que la subcapa que le proporciona la seguridad (en la versión más nueva es el protocolo TLS) está por debajo.
El protocolo crea un canal cifrado seguro para que cuando alguien lo intercepte solo vea un flujo cifrado sin sentido alguno. El puerto estándar es el 443.
TLS usa certificados para autenticarse, eso quiere decir que hace uso de criptografía asimétrica para autenticar a la otra parte. También se usan los certificados para intercambiar la clave simétrica de la sesión cifrada. Opcionalmente, aunque muy recomendable, se suele usar un código de autenticación de mensajes (MAC), que consiste en aplicar la función *Hash* al mensaje (proporcionamos integridad) y cifrar la salida de la función con una clave privada (integramos autenticación).
Para poder establecer el canal seguro, cliente y servidor se envían una serie de mensajes. A este intercambio de mensajes se le denomina *handshake* y consiste en lo siguiente.

1. El cliente envía un mensaje “ClientHello” especificando una lista con el conjunto de cifrados que soporta, métodos de compresión y la versión SSL o TLS (en nuestro caso) más alta permitida. También envía un reto (o *nonce* en inglés), un valor aleatorio que varía en cada sesión.
2. El servidor le contesta con un “ServerHello”, en el que elije los parámetros de conexión a partir de las opciones ofertadas en el anterior mensaje.
3. Por último se envía un mensaje “Finished” que es un hash de todos los datos intercambiados y vistos por ambas partes.

Cuando ya se tienen claro los parámetros, cliente y servidor se envían sus respectivos certificados firmados por una Autoridad de Certificación (CA).
Una vez que se intercambian los certificados, empieza la negociación de la clave

secreta (simétrica) para la sesión cifrada. Este intercambio se suele hacer mediante Diffie-Hellman o simplemente cifrando la clave secreta con la clave pública de la otra entidad para que solo él la pueda descifrar con su clave privada. Gracias al *handshake*, se establece los criterios de seguridad en la comunicación de la conversación y se establece un canal seguro.

b. Lenguajes de programación y Base de Datos usados

Para implementar el servidor, se ha optado por el lenguaje de programación de Node.js [15]. Node.js es el lenguaje de programación de JavaScript que se usa en el servidor.
Se ha escogido este lenguaje debido a que Node.js está orientado a eventos y prácticamente está pensado para servidores que sean una API. Una API toma algunos parámetros, los interpreta, arma una respuesta y descarga esa respuesta al usuario..
Por lo tanto las características principales de Node.js se pueden resumir en:

- Orientado a eventos
- Proporciona servidores altamente escalables
- Interprete ultra-rápido
- Lenguaje hecho para API 's
- Lenguaje liviano, no crea una gran carga de procesos en el servidor

Otro punto importante de Node.js, es que trabaja con módulos para poder completar el servidor. Por ejemplo, necesita de módulos para poder arrancar un servidor HTTPS. Adicionalmente, Node.js presenta el Node Package Module, que es una forma integrada de instalar y administrar módulos.
La otra base de datos con la que se trata en este proyecto es MongoDB. Esta base de datos no implementa el concepto de tablas, como por ejemplo hace MySQL, sino el concepto de documento: cada base de datos de MongoDB representa un documento. Es rápida, masivamente escalable y fácil de usar.

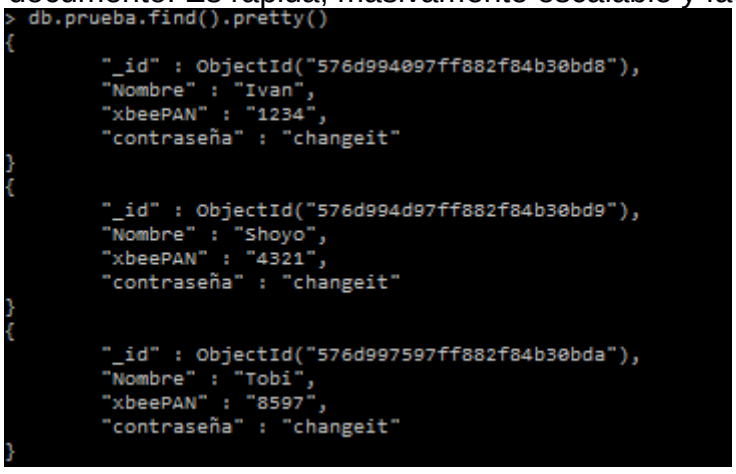

Fig. A-5.2 Estructura documento MongoDB.

Como se puede ver en la Figura A-5.2, el concepto de documento es muy parecido al esquema JSON. Este es uno de los motivos por los cuales se ha escogido esta base de datos para el servidor, ya que en NodeJS trabaja a la perfección con este tipo de esquema.

c. Herramientas usadas en el proyecto

En este apartado se explica con mayor profundidad como son las herramientas usadas en el proyecto, y se da una visión general de su funcionamiento.

7.1.1 Eclipse Mars IDE

La herramienta Eclipse IDE se ha utilizado mucho durante esta carrera, por lo tanto no se explicará tan extensamente como se ha hecho con las anteriores herramientas mencionadas, solo se explicará un poco por encima. En este caso se trabaja con la versión Mars, publicada en Junio del 2015.
Se trata de una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar. Esta plataforma ha sido usada sobre todo para entornos de desarrollo integrado (IDEs), como el IDE de Java y el compilador.
Los lenguajes de programación que se utilizan en la programación con Eclipse son: Java, ANSI C, C++, JSP, sh, perl, PHP y sed, pero mayoritariamente Eclipse se usa para programar en Java.
Eclipse cuenta con un editor de texto con un analizador sintáctico, es decir, que busca errores de código a medida que se escribe. Se puede *debugar* el código cuando se ejecuta el programa, creando puntos denominados BreakPoint. La compilación es en tiempo real. Gracias a *plugins* se puede añadir control de versiones con Subversion y *Github*, e integración con *Hibernate.*

7.1.2 WebStorm IDE

WebStorm es el IDE de pago para JavaScript y NodeJS de la compañía JetBrains. Es un IDE que proporciona asistencia en codificación JavaScript, HTML y CSS. Está equipado para el desarrollo del lado tanto del cliente (JavaScript) como del servidor (NodeJS).
En este proyecto se ha utilizado la versión de prueba para poder realizar la parte del servidor donde será consultado tanto por el router como por el dispositivo móvil.
La interfaz gráfica es muy parecida a Eclipse, es decir, la ubicación de las funciones (como el Debugar, Compilar, *Plugins*, etc.) se asemeja a la de Eclipse. Con esto nos referimos que si se ha trabajo con Eclipse no costará nada adaptarse a WebStorm.

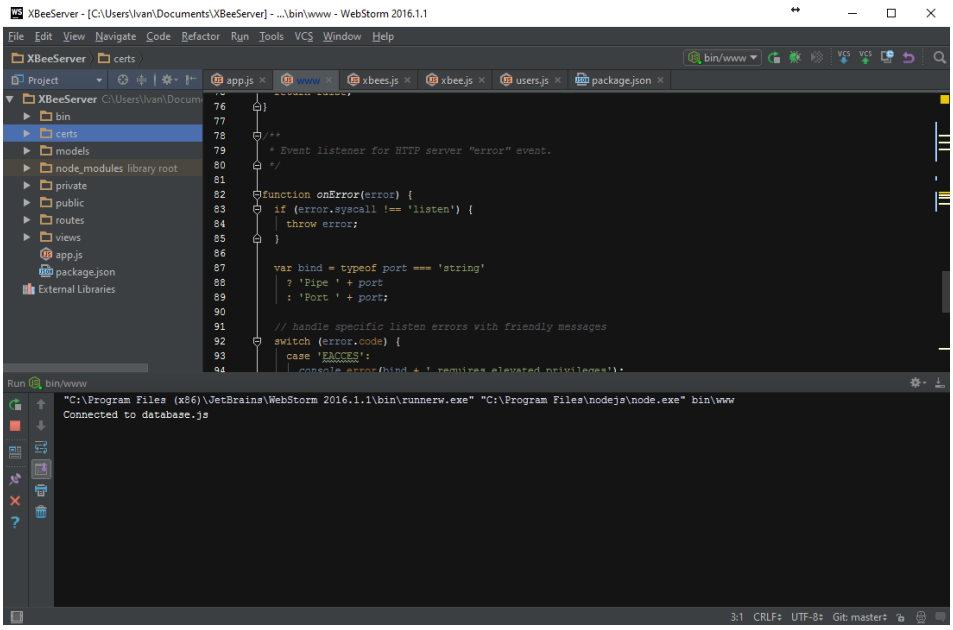

Fig. A-5.3 Interfaz WebStorm IDE.

Como se puede apreciar en la anterior figura, a diferencia del color (que es personalizable), la Interfaz se parece mucho a la de Eclipse.
Cuanto se crea un proyecto con *WebStorm*, da a elegir entre diferentes tipos de proyecto relacionado con las codificaciones JavaScript, HTML y CSS.

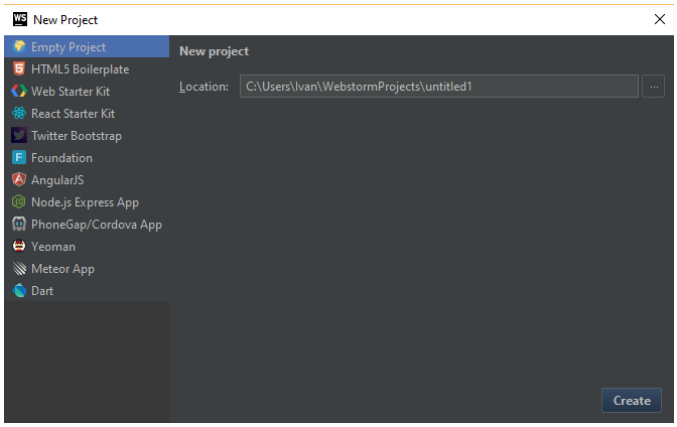

Fig. A-5.4 Tipos de proyecto en WebStorm.

En este caso se usa Node.js Express App. Cuando se elige el tipo de proyecto, *WebStorm* crea todas las carpetas y archivos necesarios para hacer un *Hello World*., que en este caso son las siguientes carpetas:

- Bin: Donde se ubica el archivo www.js (para crear el servidor http o https).
- Node_modules: Donde se ubican todas las dependencias de NodeJS que se usan en el código.
- Routes: Donde se ubican los archivos de rutas. Cuando haya una petición hacia localhost/user, le llevará al archivo user.js (en nuestro caso).
- Public: Donde se ubican los CSS para darle estilo a los HTML o Jade.
- Views: Donde se ubican los HTML o Jade que se mostrarán al usuario

cuando pida una página.

Y los archivos:

- App.js: Es el archivo que se ejecutará en el servidor, es decir, este archivo irá llamando a las funciones de los otros JavaScripts.
- Package.json: Es el archivo donde se indican las dependencias del proyecto y también la ubicación de donde tiene que arrancar el servidor web.

7.1.3 Postman REST Client

Postman es un cliente REST del navegador Chrome, que permite construir y gestionar de forma cómoda las peticiones a servicios REST (*Post, Get, Put, Delete,…*).

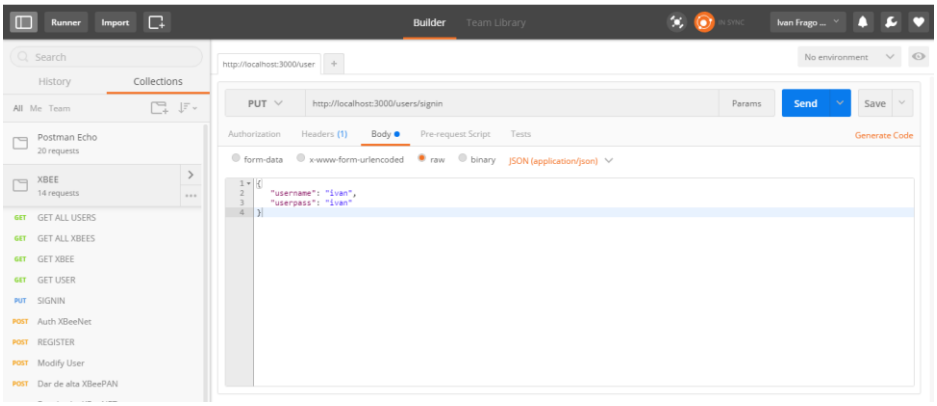

Fig. A-5.5 Interfaz Postman REST Client.

Es una herramienta muy intuitiva y sencilla, pero a su vez es muy completa. Con solo ver la Figura A-5.5 se pueden apreciar todas sus funciones, añadir tipos de autorización (básica/formulario etc), cabeceras (como por ejemplo Content-Type), escribir en formato JSON, Raw,…

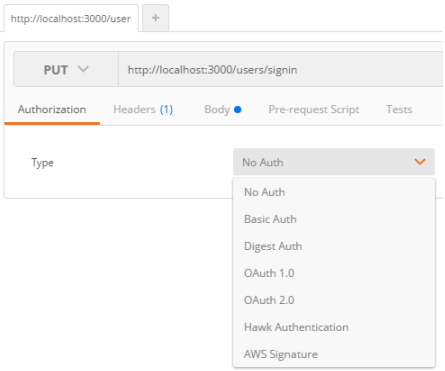

Fig. A-5.6 Tipos de autenticación en Postman

Cuando se pulsa el botón de enviar, *Postman* muestra la respuesta del servidor

de una forma muy bien ordenada y se puede optar por el tipo de formato en que quieres visualizarlo (JSON, XML, HTML o texto plano)

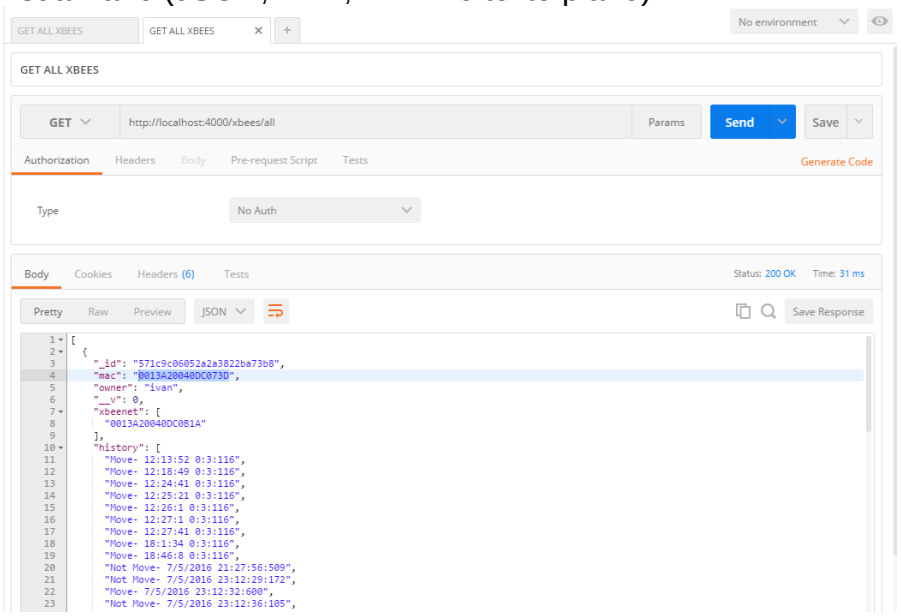

Fig. A-5.7 Visualización de la respuesta del servidor en Postman

Postman mantiene un historial para poder ver las peticiones que se han ido haciendo y así usar una petición anterior y no tener que escribirla toda entera de nuevo. Esto se considera muy útil si se está probando peticiones POST, por ejemplo, con distinta información pero repetidas veces.
Por último, cabe destacar que se pueden crear, importar y exportar colecciones de peticiones. Se puede crear una carpeta donde se contenga todas peticiones guardadas con su respectiva información para luego guardarla y exportarla poder pasarla a otro usuario. También comentar que al inicial sesión, se obtienen las colecciones guardadas con el usuario.

7.1.4 OpenSSL

OpenSSL es un robusto paquete de herramientas de administración y bibliotecas relacionadas con la criptografía, que suministra funciones criptográficas y ayudan a implementar protocolos relacionados con la seguridad. Principalmente se ha usado en este proyecto a la hora de crear una autoridad de certificación, claves privadas y certificados. Las claves privadas se han usado para cifrar por ejemplo el *token* que suministra el servidor. Mientras que los certificados se han usado para la autenticación del *router,* servidor y aplicación Android, es decir, para el protocolo TLS.

7.1.5 Heroku

Heroku[16] es un servicio de *hosting* en la nube, orientado a una plataforma como servicio (PaaS, *Platform as a Service*), que proporciona una plataforma y un entorno que permite alojar tanto un servidor como una base de datos. Normalmente estas plataforma son de pago, pero Heroku tiene un pack gratuito que ofrece un servicio algo limitado, pero suficiente para alojar el servidor en NodeJS y la base de datos. Existen varios tipos de *hosting* que proporcionan un servicio similar, también de forma gratuita.

Para escoger una PaaS se ha tenido en cuenta los siguientes detalles:

- Que permita correr una aplicación en Node JS junto con una base de datos MongoDB.
- Que sea gratuita.
- Que permita comunicación HTTPS.

Hay muchas plataformas PaaS que cumplen con los requisitos anteriormente mencionados. En un principio se había escogido OpenShift, pero como recientemente han actualizado el servicio, mucha documentación para subir las aplicaciones estaba desactualizada y la misma que proporcionaba OpenShift no era del todo clara. Debido a los problemas que estaba dando para poder hacer correr el servidor, se optó a buscar otra plataforma PaaS gratuita.
Heroku en la versión gratuita ofrece un *hosting* con las siguientes características:

- Desplegar aplicación desde Git
- Parchear las aplicaciones automáticamente.
- Aplicaciones unificadas.
- Balanceo de carga.
- Correr dos tipos de procesos diferentes.
- La aplicación alojada duerme 6h al día.
- Dominio customizado.
- 500 MBytes de memoria.

Heroku trabaja con lo que se denomina *dynos,* que es como llama a los procesos. Cada dyno es independiente del otro y también tiene su propio *file system* efímero y cuando el proceso es detenido, este archivo se destruye. Por lo tanto es recomendable usar otro tipo de servicio, como por ejemplo mongoDB. El código corre siempre dentro de un *dyno.*

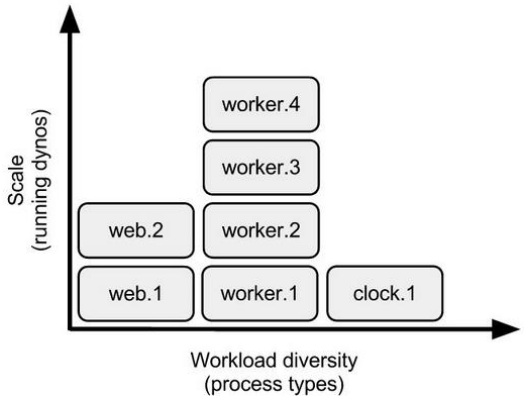

Fig. A-5.8 Representación de Dynos en Heroku

Se pueden distinguir tres tipos de *dynos*:

- *Web dynos:* Solo recibe trafico HTTP de los *router* de Heroku.
- *Worker dynos:* Usado para trabajos en *background*, en cola y trabajos temporizados.
- *One-off dynos:* Procesos temporales, funcionamientos separados.

Cuando la aplicación ya está en funcionamiento y espera a recibir peticiones, las solicitudes las procesa antes el balanceador de carga que las pasa a un conjunto de *routers* de Heroku. Entonces los *routers* determinan la ubicación de

los *Web* dynos de la aplicación y reenvían la solicitud HTTP. El enlace Heroku garantiza escalabilidad, es decir, si la aplicación recibe muchas peticiones, se levantarán otros dynos para poder asumir la carga de trabajo. Heroku es capaz de hacer correr aplicaciones de una gran variedad de lenguajes distintos:

- Ruby
- PHP
- Node js
- Python
- Java
- Clojure
- Scala and Play
- Go

Por cada aplicación es necesario proporcionar un fichero llamado Procfile para indicar como se levanta la aplicación alojada.

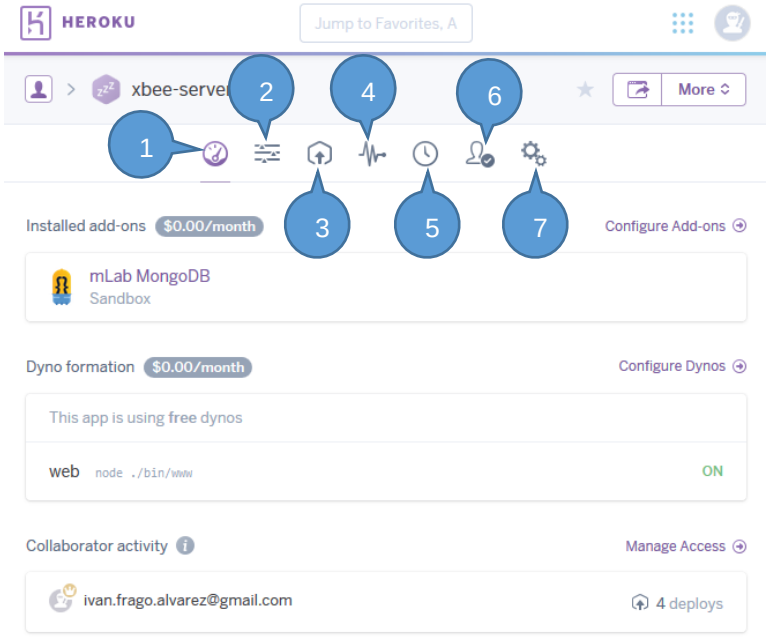

Fig. A-5.9 Dashboard Heroku

Para poder gestionar la aplicación, Heroku proporciona un *dashboard* para:

1. Información genérica del proyecto.
2. Servicios contratados del proyecto.
3. Control de versiones con Github.
4. Métricas de uso.
5. Actividad en el proyecto.
6. Colaboradores.
7. Configuración del proyecto.

También se puede ver un log del dyno clicando a la pestaña “more”, dando las opciones de hacer correr una consola, comprobar las producciones de dynos y por último restaurar todas las dynos.

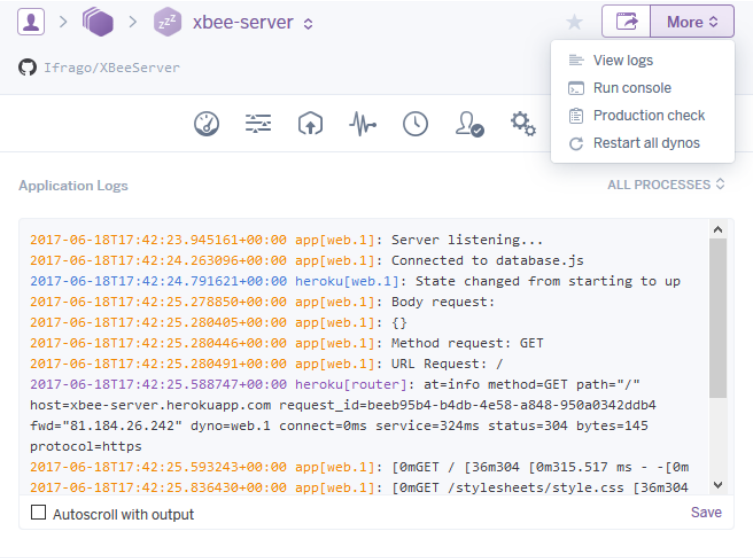

Fig. A-5.10 Log de la aplicación

Como se ha mencionado con anterioridad para poder almacenar datos en Heroku es necesario un servicio independiente. Por lo tanto para hacer correr mongoDB es necesario el uso de *Add-on*, éste *Add-on* necesario es el denominado mLab.
El *Add-on* dispone también de un *dashboard* donde se puede visualizar las tablas de la colección de MongoDB y saber cuanto espacio ocupa cada tabla. También permite gestionar toda la base de datos desde el *dashboard.*

ANEXO VI. Negociación de algoritmos criptográficos e intercambio de certificados entre *router* y servidor.

Cuando el router quiere comunicarse con el servidor, lo hace mediante protocolo HTTPS con autenticación mutua. Gracias a la autenticación mutua incrementamos la seguridad en la parte del proyecto. Cuando se va a establecer una conexión, primero de todo tanto el *router*, que en este caso el cliente, y servidor hacen el *handshake.*

Fig. A-6.1 *Handshake Router-Servidor*

En la figura de arriba se puede ver como *router* y servidor han establecido un *handshake* para poder iniciar una sesión HTTPS. En este escenario, el *router* siempre iniciará la conversación, ya que solo hablaran en tres ocasiones distintas que se explicarán más tarde, pero todas es por qué el *router* quiere dar información al servidor.
Cuando el *router* quiere iniciar una sesión HTTPS con el servidor, le envía un *Client Hello*. Este tipo de mensaje sirve para que el *router* sepa cierta información del servidor, como por ejemplo

- Versión SSL.
- Bytes aleatorios (que se usarán para generar una clave maestra entre ambos)
- Lista de algoritmos de encriptación, también llamados *cipher_suites*.
- Lista de algoritmos de compresión, los denominados *hash.*
- Extensiones, esto es opcional, ya que le pide una lista de extensiones que no son propias del protocolo, pero que se pueden usar para incrementar la seguridad de la sesión.

La respuesta al anterior mensaje es el denominado *Server Hello*, que en el proyecto siempre lo enviará el servidor. Básicamente en el mensaje el servidor le proporciona toda la información que le ha pedido el *router*. También le contesta, mediante otro paquete, tres mensajes más. Estos tres mensajes son el *Certificate, Server Key Exchange*, el *Certificate Request* junto con el *Server Hello Done*.
En el mensaje Certificate, contiene un certificado o una lista de certificados que el *router* tendrá que usar para poder autenticar al servidor y cifrar la información. Mientras que en el mensaje Server *Key Exchange*, le pasa los parámetros de Diffie-Hellman, para que luego puedan generar la *master key*.
En el mensaje *Certificate Request* el servidor está pidiendo al router que se requiere su certificado. Este mensaje es opcional, dependiendo si la sesión es pro autenticación mútuo o unidireccional. Por último el *Hello Done*, con este mensaje el servidor finaliza la parte del *handshake* donde se pasan todos los

parámetros de la sesión HTTPS, es decir, el *router* ya tiene todo lo que se necesita para empezar a cifrar.
Cuando el *router* tiene todo, le envía su propio certificado al servidor y los parámetros de Diffie-Hellman en los mensajes *Certificate* y *Client Key Exchange.*
A partir de aquí lo que hace el *router* es verificar el certificado del servidor y avisarle que a partir de ahora irá todo cifrado mediante el mensaje *Change Cipher Spec*. El servidor le contestará con el mismo mensaje: *Encrypted Handshake Message*.
Como se ha mencionado con anterioridad, el *router* solo interactúa tres veces distinta con el servidor. Una por cada estado (definido en la sección 3.3 del documento principal). Cuando interactúa con el servidor, abre una sesión TLS y mediante el *handshake* establecen las claves para que la comunicación está cifrada, por lo tanto sea segura.

ANEXO VII. Código en Arduino de las pruebas de *throughput* realizadas

```
#include <SoftwareSerial.h>
#include <XBee.h>

// Define SoftSerial TX/RX pins
// Connect Arduino pin 8 to TX of usb-serial device
uint8_t ssRX = 3;
// Connect Arduino pin 9 to RX of usb-serial device
uint8_t ssTX = 2;
// Remember to connect all devices to a common Ground: XBee, Arduino
and USB-Serial device

SoftwareSerial nss(ssRX, ssTX);

XBee xbee = XBee();

XBeeResponse response = XBeeResponse();
uint8_t dataLightCounter[]   = {0,0,0,0}; //4 bytes
uint8_t dataConnect[]        = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0};
uint8_t dataMediumCounter[]  =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t dataHardCounter[]    =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
uint8_t dataLight[]          =
{'48','49','50','51'};//KEY=0123456789012345
uint8_t dataMedium[]         =
{'48','49','50','51','52','53','54','55','56','57','48','49','50','51'
,'52','53','54','55','56','57','48','49','50','51','52','53','54','55'
,'56','57','48','49','50','51','52','53'};//KEY=0123456789012345
uint8_t dataHard[]           =
{'57','48','49','50','51','52','53','54','55','56','57','48','49','50'
,'51','52','48','49','50','51','52','53','54','55','56','57','48','49'
,'50','51','52','53','54','55','56','57','48','49','50','51','52','53'
,'54','55','56','57','48','49','50','51','52','53','49','50','51','52'
,'53','54','55','56','57','48','49','50','51','52','53','54','49','50'
,'51','52','53','49','50','51','52','53','49','50','51','52','53','49'
,'50','51','52','53','49','50','51','52','53','49','50','51','52','53'
,'49','50','51','52','53'};


uint8_t key[] =
{'48','49','50','51','52','53','54','55','56','57','48','49','50','51'
,'52','53'}; //KEY=0123456789012345
uint8_t KEY_MASTER[] = {'7','6','5','4','2','1','0','8'}; //KEY
preconfigurada para los challange del primer hanshake
uint8_t IV_MASTER[] = {'5','6','8','1','9','7','2','8'};
//IVpreconfigurada para el cifrado
// create reusable response objects for responses we expect to handle
Rx64Response rx64 = Rx64Response();
Rx16Response rx16 = Rx16Response();
Tx16Request tx16 = Tx16Request();

int count=0;
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  nss.begin(9600);
  xbee.begin(nss);
  delay(1000);
}

void loop() {

  tx16= Tx16Request(0x1234,dataConnect,sizeof(dataConnect));
  dataConnect[2] = count;
  count++;
  xbee.send(tx16);
  delay(1200);

}
```