Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DISEÑO E IMPLEMENTACIÓN DE UN PROTOCOLO DE COMUNICACIONES ENTRE SENSORES IOT Y UNA PLATAFORMA CLOUD

Alumno: **María Linarejos González Ginés**

Tutor: Prof. D. José Ángel Fernández Prieto
Depto.: Ingeniería de Telecomunicación

Junio, 2018

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Trabajo Fin de Grado

DISEÑO E IMPLEMENTACIÓN DE UN PROTOCOLO DE COMUNICACIONES ENTRE SENSORES IOT Y UNA PLATAFORMA CLOUD

Alumna: **María Linarejos González Ginés**

Tutor: Prof. D. José Ángel Fernández Prieto

Depto.: Ingeniería de Telecomunicación

Junio, 2018

Firma Alumna:

GONZALEZ GINES MARIA LINAREJOS - 26253256K

Firmado digitalmente por GONZALEZ GINES MARIA LINAREJOS - 26253256K
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26253256K, givenName=MARIA LINAREJOS, sn=GONZALEZ GINES, cn=GONZALEZ GINES MARIA LINAREJOS - 26253256K
Fecha: 2018.06.19 14:48:51 +02'00'

María Linarejos González Ginés

Firma Vº Bº Tutor:

FERNANDEZ PRIETO JOSE ANGEL - 26222606F

Firmado digitalmente por FERNANDEZ PRIETO JOSE ANGEL - 26222606F
Nombre de reconocimiento (DN): c=ES, serialNumber=26222606F, sn=FERNANDEZ PRIETO, givenName=JOSE ANGEL, cn=FERNANDEZ PRIETO JOSE ANGEL - 26222606F
Fecha: 2018.06.19 10:12:08 +02'00'

Prof. D. José Ángel Fernández Prieto

Índice:

1. Resumen.

El principal objetivo de este trabajo es diseñar e implementar un protocolo de comunicaciones entre un sensor del Internet de las Cosas (IoT) y una plataforma de Cloud Computing. Esto se ha conseguido desplegando una red de sensores del IoT (Internet de las Cosas, "Internet of Things" en inglés) comunicada mediante el protocolo de enlace LoRa, y ésta a su vez se ha conectado a una red WiFi que envía los datos recogidos a un servidor Cloud mediante el protocolo de aplicación HTTP. Se ha precisado del uso de un Gateway LoRa – WiFi para la interconexión de ambas tecnologías.

El protocolo LoRa es un protocolo privado de nivel de enlace que gracias a las características que tiene su modulación LoRa permite enviar paquetes de datos entre nodos separados por grandes distancias a una velocidad baja (kbytes/segundo), ya que el enlace que los comunica precisa de una gran robustez y una baja sensibilidad frente a interferencias a un coste energético mínimo.

La red de sensores presenta una topología en estrella, en la que los nodos periféricos envían datos a un nodo central (Gateway LoRa – WiFi), y éste los envía a su vez a una plataforma Cloud vía WiFi mediante un protocolo de aplicación HTTP (siendo los protocolos de transporte y de red TCP e IP, respectivamente).

Este tipo de infraestructura permite un despliegue de nodos que corren programas sencillos a un coste energético ínfimo para la recogida de datos, los cuales se envían a un único nodo central (Gateway) que realizará el costoso proceso de utilizar la arquitectura IP, más compleja, para enviar los datos a la plataforma Cloud y de esta forma almacenarlos, reduciendo así los gastos tanto de equipamiento como de complejidad en la red y en los nodos, a costa de precisar de un coste computacional mayor en un único nodo.

Se han realizado pruebas en tres escenarios diferentes: dos exteriores y uno interior.

En los escenarios exteriores, uno de ellos ha sido un entorno urbano, donde la línea de visión se veía obstruida por edificios, y el otro ha sido en campo abierto, donde la línea de visión era directa. Se han llegado a alcanzar 29 km de distancia entre sensores del IoT en campo abierto, mientras que en el entorno urbano esta distancia se ha reducido considerablemente (400 m).

En el escenario interior, las pruebas se han realizado dentro de un mismo edificio, por lo que no se han tenido en cuenta distancias máximas.

2. Introducción

Se habla acerca de la importancia del Internet de las Cosas, pero, ¿qué es el "IoT"?

Como menciona Cisco [1], cada vez estamos más conectados, la red de Internet es más grande, los precios de las conexiones son cada vez inferiores, y gracias a los sensores podemos monitorizar prácticamente de todo, desde el nivel de aceite que tiene el coche, hasta programar la cafetera para que haga un café a una hora en concreto. ¿Y si pudiésemos conectar absolutamente todo lo que nos rodea? Ese es el objetivo del Internet de las Cosas, poder monitorizar todo aquello que nos rodea vía Internet.

Pero, ¿es realmente necesario el uso de Internet?

Está claro que el protocolo más utilizado para las comunicaciones es el protocolo a nivel de red IP, pero ahora mismo presenta un problema: hay demasiados dispositivos para las direcciones IPv4 que hay disponibles. IPv6 tiene un rango mucho mayor, además de que la seguridad también aumenta, aunque no está del todo estandarizado. Mientras esto ocurre, ¿cómo se le pueden asignar las direcciones a los nodos, si no hay suficientes para todos? Y además, el protocolo IP es un protocolo complicado, ¿realmente es necesario tener máquinas que precisen de un coste computacional alto para poder correrlo?

En este caso entran en escena los sensores, máquinas a las que se les incorpora un algoritmo, una secuencia de órdenes que se repiten en el tiempo de forma secuencial, y que tienen un coste computacional bajo. Las órdenes que pueden seguir no son complejas, pero en su favor tienen que su coste energético es mínimo.

Además, hemos de tener en cuenta que si bien el protocolo IP es uno de los más extendidos que hay en el momento, éste presenta un problema: los módulos o sensores que lo implementen han de estar conectados a Internet, ya sea vía Ethernet, WiFi, o por cualquier tecnología que cuente con el envío y recepción de datos móviles. Esto se traduce en un rango de cobertura pequeño, en el caso de las dos primeras opciones, o como ocurre en el tercer caso, que puede haber zonas donde no llegue la cobertura de las antenas destinadas a tal propósito.

¿Qué podemos hacer al respecto?

El protocolo LoRa es un protocolo de nivel de enlace que se basa en una modulación que proporciona robustez a la señal y una mínima sensibilidad al ruido, haciendo que el rango de alcance de la comunicación entre nodos vaya desde cientos de metros en un entorno urbano hasta más de una veintena de kilómetros en un espacio abierto.

La unificación de ambos protocolos en una misma topología hace posible que un despliegue de nodos periféricos que utilicen el protocolo LoRa, envíen los datos recogidos a un nodo central que actúe como Gateway LoRa – WiFi, el cual los envía a una plataforma cloud para almacenarlos y gestionarlos. En el siguiente esquema puede observarse con detalle esta infraestructura de red:

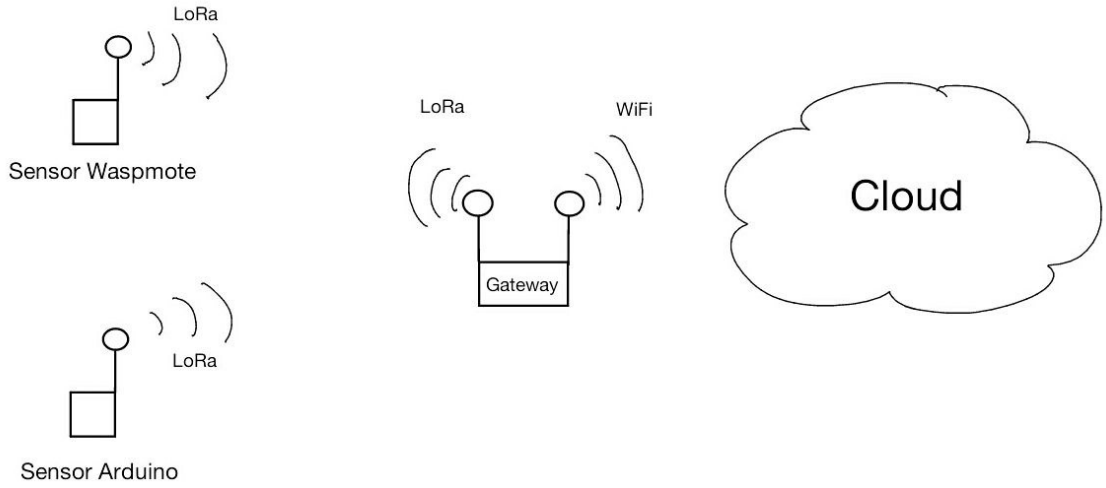

Imagen 2.1. Esquema de red: dos nodos que corren el algoritmo LoRa se comunican con un Gateway LoRa – WiFi, y este a su vez envía los datos recogidos a una plataforma Cloud.

Con esta implementación a larga distancia y con unos sensores en lugar de máquinas de computación (ordenadores, por ejemplo), el coste energético es muy inferior, prolongando la vida de las baterías y minimizando los gastos en ellas, además de que se puede tener incluso en consideración la posibilidad de alimentar los sensores con energía solar, con la adaptación de placas solares para este propósito.

3. Objetivos.

Los objetivos de este Trabajo de Fin de Grado son:

- Establecer un protocolo de comunicaciones entre una red de sensores del Internet de las Cosas (IoT, “Internet of Things”) y una plataforma de Cloud Computing.
- Diseñar una topología de red aunando las tecnologías LoRa y WiFi.
    - Red de sensores IoT con comunicaciones punto a punto (nodos periféricos a nodo central).
    - Red WiFi para envío de datos a la plataforma Cloud.
    - Gateway LoRa – WiFi para hacer posible la comunicación entre ambos protocolos.

4. Materiales y Métodos.

En este apartado, veremos los distintos elementos que se han utilizado para llevar a cabo el trabajo.

4.1. Waspmote.

4.1.1. Hardware.

Waspmote [2][3] está basado en una arquitectura modular, cuya idea es integrar únicamente los módulos necesarios en cada dispositivo, pudiendo ser cambiados y expandidos de acuerdo a las necesidades del usuario.

La placa Waspmote v15 permite conectar a ella dos módulos a sus sockets, siendo estos socket 0 y socket 1. En el socket 0 se conectará el módulo LoRa, mientras que al socket 1 se conectará el módulo WiFi PRO, previo uso de una placa radio de expansión para poder realizar la conexión.

4.1.1.1. Especificaciones.

- Microcontrolador: ATmega1281.
- Frecuencia: 14.7456 MHz.
- SRAM: 8 kB.
- EEPROM: 4 kB.
- FLASH: 128 kB.
- Tarjeta SD: hasta 8 GB.
- Peso: 20 g.
- Dimensiones: 73.5 x 51 x 13 mm.
- Rango de temperatura: [-30 ºC, +70 ºC], aunque se recomienda su uso entre los -20, +60 ºC, ya que solamente soporta temperaturas extremas de forma temporal.

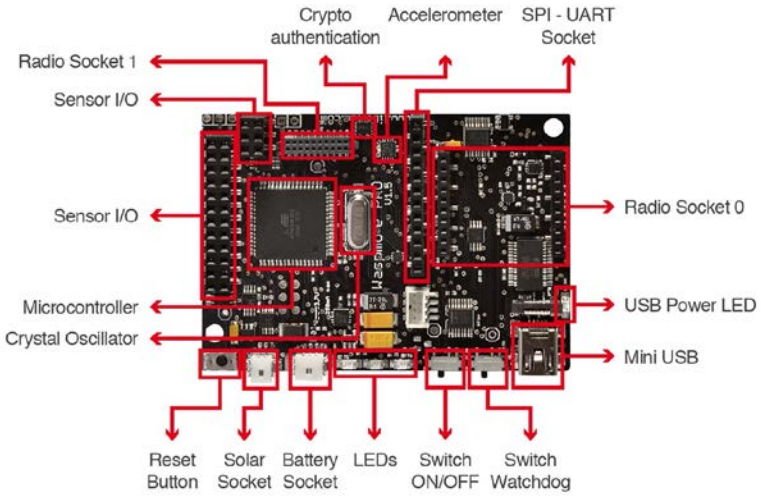

Imagen 4.1.1.1.1. Vista superior de una placa Waspmote. Fuente: Libelium.

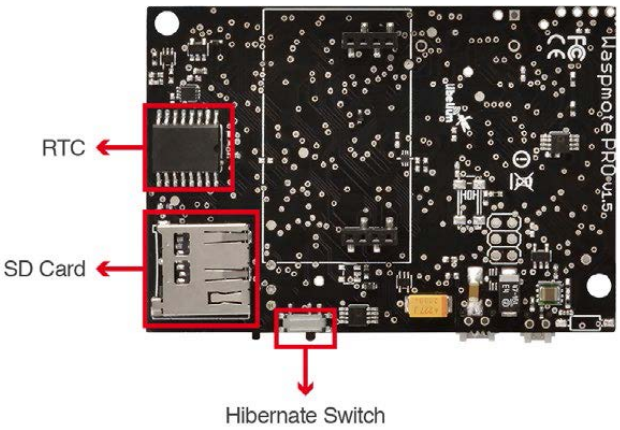

Imagen 4.1.1.1.2. Vista inferior de una placa Waspmote. Fuente: Libelium.

4.1.1.2. Datos eléctricos.

Waspmote precisa de los siguientes valores de alimentación:

- Voltaje mínimo de batería operacional: 3.3 V.
- Voltaje máximo de batería operacional: 4.2 V.
- Voltaje de carga por USB: 5 V.
- Voltaje por carga mediante un panel solar: 6 – 12 V.
- Amperaje de carga mediante USB: 480 mA (máxima corriente de entrada).
- Amperaje de carga por placa solar: 300 mA (máxima corriente de entrada).

Los valores absolutos máximos que soporta son los siguientes:

- Voltaje en cualquier pin: [- 0.5 V, +3.8 V]
- Corriente máxima en cualquier pin digital I/O: 40 mA.
- Voltaje por USB: 7V.
- Voltaje por placa solar: 18 V.

- Voltaje por batería: 4.2 V.

4.1.1.3. *UART*

Waspmote cuenta con dos UARTs (Universal Asynchronous Receiver – Transceiver, Transmisor – Receptor Asíncrono Universal): UART0 y UART1. Además, hay varios puertos que pueden estar conectados a estos UARTs a través de dos multiplexores diferentes, uno para cada UART.

- UART0. Está compartido por el puerto USB y el Socket0. Este Socket será el que utilizaremos para conectar el módulo LoRa. El multiplexor en este UART controla la señal de datos, que por defecto está ligada al Socket0. Cuando el USB necesita enviar información a través del UART0, el multiplexor cambia momentáneamente al puerto USB y luego vuelve de nuevo al Socket0 tras enviar la información.
- UART1 está compartido por 4 puertos: Socket1, socket de GPS, y sockets Auxiliar1 y Auxiliar2. Es posible seleccionar en el mismo programa cuál de los 4 puertos está conectado al UART1 en el microcontrolador.

4.1.1.4. *USB.*

El USB en Waspmote se usa para la comunicación con un ordenador o con dispositivos USB compatibles. Esta comunicación permite cargar el programa en el microcontrolador.

Como se ha dicho anteriormente, se utiliza el UART0 para la comunicación USB en el microcontrolador. El chip FT232RL realiza la conversión al estándar USB.

4.1.1.5. *Arquitectura.*

La arquitectura de Waspmote está basada en el microcontrolador Atmel ATmega1282.

Cuando el sensor Waspmote se enciende y comienza el gestor de arranque (bootloader), hay un tiempo de espera antes de que comience la primera instrucción, el cual es usado para comenzar a cargar las nuevas actualizaciones de los programas compilados. Si se recibe un nuevo programa por el USB en este tiempo, se cargará en la memoria FLASH (128 kB), sustituyendo los programas ya existentes. Por otra parte, si no se recibe ningún programa nuevo, se lanzará el último programa almacenado.

La estructura de los códigos se divide en dos partes básicas: “setup” y “loop”. Ambas partes del código tienen un comportamiento secuencial, ejecutando instrucciones en el orden establecido.

La primera parte del código está compuesta por el fragmento “setup”, el cual se ejecuta una única vez cuando el código es inicializado. En esta parte es recomendable incluir la inicialización de los módulos que se van a utilizar, ya que esta parte del código es importante únicamente en el arranque del Waspmote.

El fragmento correspondiente a “loop” se ejecuta de manera continua, formando un bucle infinito. Debido al comportamiento de esta parte del código, se recomienda el uso de interrupciones para ejecutar las acciones con Waspmote.

Cuando el Waspmote es reseteado o encendido, el código comienza de nuevo con la función “setup”, y a continuación ejecuta la función “loop”.

Por defecto, los valores de las variables declaradas en el código y modificadas durante la ejecución del mismo se perderán cuando se resetee el dispositivo o cuando no tenga batería.

4.1.1.6. Placa radio de expansión.

La placa radio de expansión permite conectar dos módulos de comunicación al mismo tiempo en la plataforma del sensor Waspmote. Esto significa que serán posibles muchas combinaciones de comunicaciones distintas, aunque en este caso la que se usará será LoRa – WiFi.

La API proporciona una función denominada “ON()”, para encender el módulo, y esta función soporta un parámetro que permite seleccionar el socket, siendo posible seleccionar entre el SOCKET0 y el SOCKET1.

Imagen 4.1.1.6.1. Placa radio de expansión Waspmote. Fuente: Libelium.

4.1.2. Software.

Para poder programar el sensor Waspmote y los respectivos módulos que se encuentren colocados en ella, es necesario el uso del IDE de Waspmote.

Las pruebas se han realizado en el sistema operativo Windows, y los pasos para su instalación son los siguientes:

1. Descarga del archivo comprimido que permitirá su instalación, disponible en el siguiente enlace: http://downloads.libelium.com/waspmote-pro-ide-v06.06-windows.zip.
2. Extracción del contenido del archivo comprimido. Se descomprimirá en una carpeta denominada "Waspmote-pro-ide-v06.06", en el caso de esta versión del software.
3. Al entrar en la carpeta anteriormente dicha, aparecerá una lista de elementos. Ejecutar el programa "waspmote.exe", como se indica en la imagen "4.1.2.1".

Imagen 4.1.2.1. Archivo ejecutable para la instalación del IDE de Waspmote.

4. Si Windows Defender está activado, aparecerá una ventana emergente informando de que se ha impedido el inicio de una aplicación desconocida. Hay que marcar la opción "más información" y, a continuación, "ejecutar de todas formas".

Imagen 4.1.2.2. Aviso de Windows Defender.

5. Tras aceptar, se abrirá el IDE de Waspmote.

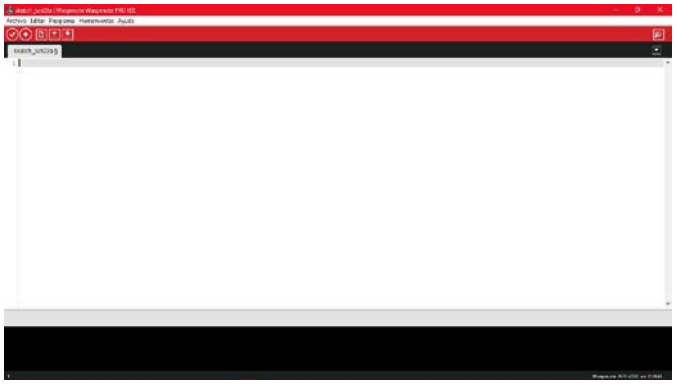

Imagen 4.1.2.3. IDE de Waspmote.

6. A continuación, conectamos la placa Waspmote con los módulos LoRa, WiFi PRO y la placa radio de expansión al ordenador, mediante un cable USB – micro USB.

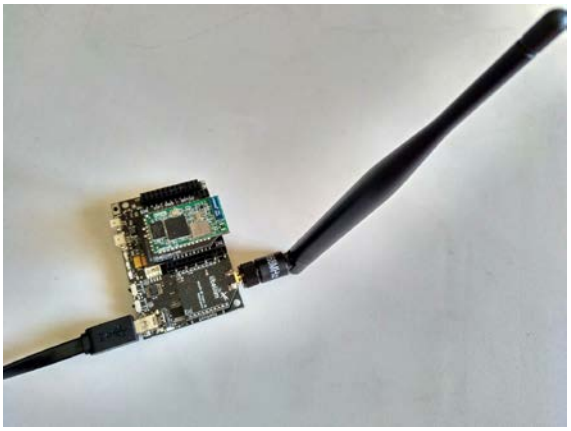

Imagen 4.1.2.4. Sensor Waspmote con módulos LoRa, WiFi PRO y placa radio de expansión conectados.

7. A continuación, se ha de seleccionar el puerto y la placa en el IDE de Waspmote. Para ello, seleccionamos en la barra de herramientas “Herramientas”, y ahí seleccionamos la placa y el puerto.

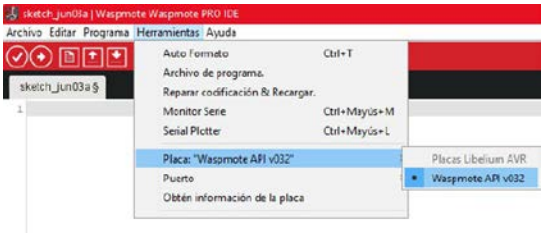

Imagen 4.1.2.5. Selección de placa Waspmote.

Imagen 4.1.2.6. Selección de puerto en la placa Waspmote.

8. Para incluir las librerías necesarias, hay que abrir a pestaña “Programa”, y a continuación “incluir librería”. Para esta situación, es necesario incluir las librerías “SX1272”, para utilizar el módulo LoRa, y “WiFi_PRO”, para poder usar el módulo “WiFi PRO”.

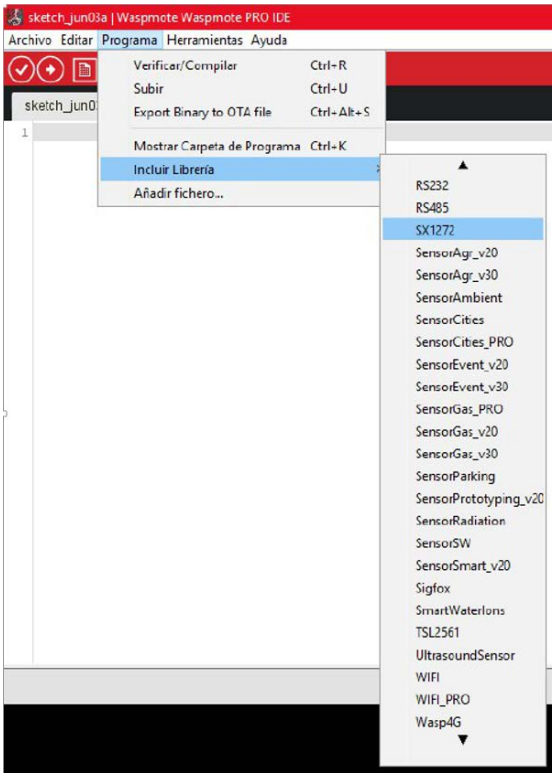

Imagen 4.1.2.7. Inclusión de las librerías necesarias para la usabilidad de los módulos.

9. Una vez incluidas las librerías, se puede comenzar a introducir el código para programar la placa en los bucles previamente comentados (“setup” y “loop”).

4.2. Arduino.

4.2.1. Hardware.

4.2.1.1. Arduino UNO

El sensor Arduino con el que trabajaremos será Arduino UNO [4], el cual está basado en el microcontrolador ATmega328P. Cuenta con 14 pines digitales de entrada

y salida (de los cuales 6 pueden ser usados como salidas PWM), 6 entradas digitales, un cristal de cuarzo de 16 MHz, una conexión USB, un jack de alimentación, una cabecera ICSP y un botón de reseteo.

4.2.1.1.1. Alimentación.

La placa Arduino UNO puede ser alimentada mediante un cable USB o mediante alimentación externa. La fuente de alimentación es seleccionada de manera automática.

La alimentación externa que no se produzca por vía USB puede venir tanto por un adaptador AC/DC como por una batería. El adaptador se puede conectar mediante una clavija de 2.1mm con el centro positivo al jack de alimentación. Los cables de la batería se pueden insertar en los pines GND y Vin del conector de alimentación.

La placa puede funcionar con un suministro externo de 6 a 20 Voltios. Si se le suministran menos de 7 Voltios, el pin de 5 V suministrará menos de 5 voltios y la placa puede volverse inestable. Si se utilizan más de 12 V, el regulador de tensión puede sobrecalentarse y dañar la placa. El rango recomendado es entre 7 y 12 voltios.

4.2.1.1.2. *Memoria.*

El microcontrolador ATmega328 tiene 32 kB de memoria, estando 0.5 kB ocupados por el gestor de arranque. También tiene 2 kB de SRAM y 1 kB de EEPROM.

4.2.1.1.3. Entradas y salidas.

Cada uno de los 14 pines digitales de Arduino UNO puede ser usado como un pin de entrada o de salida, dependiendo de la función que se le esté aplicando, y operan a 5 Voltios. Cada pin puede proveer o recibir 20 mA como condición recomendada de operación, soportando hasta un máximo de 40 mA (si esta cantidad se ve superada puede dañar permanentemente el microcontrolador).

Arduino UNO cuenta además con 6 pines de entrada analógicos, en los que cada uno de ellos provee 10 bits de resolución. Por defecto, miden desde tierra hasta 5 voltios, a pesar de que se pueden cambiar estos valores con las funciones pertinentes.

Otros dos pines que se encuentran en la placa son “AREF”, que es el voltaje de referencia para las entradas analógicas, y “reset”, el cual si se establece a “low”, resetea el microcontrolador.

4.2.1.1.4. *Comunicaciones.*

La placa Arduino UNO tiene facilidad para conectarse con un ordenador, otra placa Arduino o con otros microcontroladores. El microcontrolador ATmega328 provee una comunicación serie UART TTL (5V), la cual está disponible en los pines digitales 0 (recepción) y 1 (transmisión). Un microcontrolador ATmega16U2 en la placa canaliza la comunicación serial a través del USB y aparece como un puerto virtual de comunicación en el software del ordenador. El firmware 16U2 utiliza los drivers estándar USB COM, y no se precisan drivers externos. El IDE de Arduino incluye un monitor serie que permite enviar y recibir datos de texto a la placa.

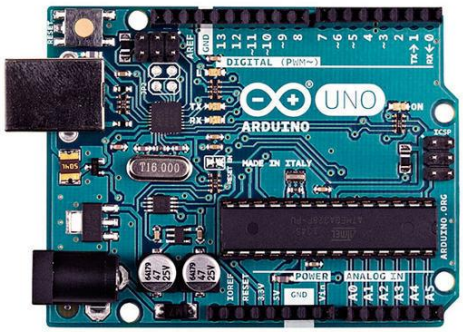

Imagen 4.2.1.1.4.1. Placa Arduino UNO. Fuente: Arduino.

La placa Arduino UNO por sí misma no permite la conexión de ningún módulo de Libelium, por lo que hay que montar sobre ella una placa de radio LoRa para Arduino como adaptador.

4.2.1.2. *Placa Radio Multiprotocolo (Multiprotocol Radio Shield).*

La placa radio multiprotocolo es una placa de interconexión para Arduino, y ha sido diseñada para conectar dos módulos de comunicación a la vez.

Incluye las conexiones bus SPI que permiten el uso de RS-485, Bus CAN, módulos LoRa y LoRaWAN, pero también es compatible con el resto de módulos.

La placa incluye dos sockets, en los cuales se puede conectar cualquier módulo que use UART. Para usar módulos SPI hay dos posibilidades. Si el SPI utiliza niveles de voltaje de 3.3V, se debe conectar en el SOCKET1, y si usa niveles de 5V, en el SOCKET0.

4.2.1.2.1. Características eléctricas.

La placa radio multiprotocolo [5] puede alimentarse a través del PC o mediante un suministro externo de energía. Algunos puertos USB de ordenadores no son capaces de aportar toda la corriente que el módulo necesita para funcionar. Si se tienen problemas, se deberá utilizar una fuente de alimentación externa (12 V – 2 A) en el Arduino.

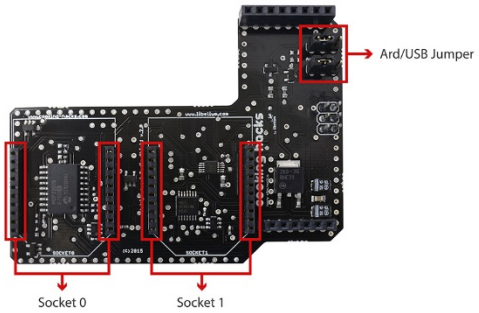

Imagen 4.2.1.2.1.1. Placa de radio LoRa para Arduino. Fuente: Cooking Hacks.

A diferencia de la placa de expansión utilizada en Waspmote, en la placa de radio hemos de colocar el módulo de LoRa en el SOCKET1.

Como el módulo WiFi PRO de Libelium no es compatible con Arduino, no se conectará a esta placa.

4.2.2. Software.

Para poder programar la placa de Arduino y el módulo LoRa que va conectado a ella, es necesario el uso del IDE de Arduino, y será necesaria a su vez la inclusión de una librería externa para el uso del módulo LoRa.

Las pruebas se han realizado en el sistema operativo Windows 10, y los pasos para su instalación son los siguientes:

1. Descarga del archivo ejecutable que permitirá su instalación, disponible en el siguiente enlace: https://www.arduino.cc/download_handler.php
2. Aparecerá una ventana emergente preguntando si queremos permitir que la aplicación haga cambios en el dispositivo, y pulsamos “Sí”.

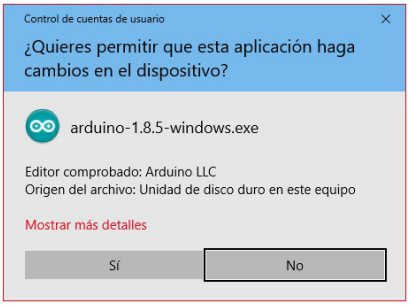

Imagen 4.2.2.1. Pantalla emergente para permitir instalación del IDE de Arduino.

3. Aparecerá otra ventana emergente para aceptar el acuerdo de licencia. Pulsamos sobre “I agree” para proceder a la instalación.

Imagen 4.2.2.2. Aceptación del acuerdo de licencia.

4. Aparecerá una ventana emergente para seleccionar los componentes que queremos instalar. Los seleccionamos todos y pulsamos “Next”.

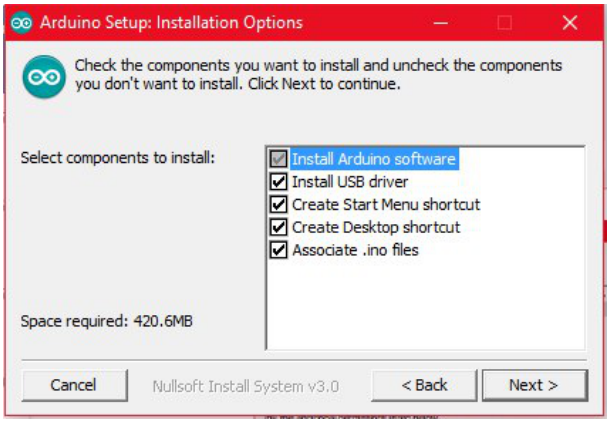

Imagen 4.2.2.3. Opciones de instalación.

5. Se abrirá otra ventana para elegir la carpeta de instalación y proceder a la misma. Pulsamos “install”.

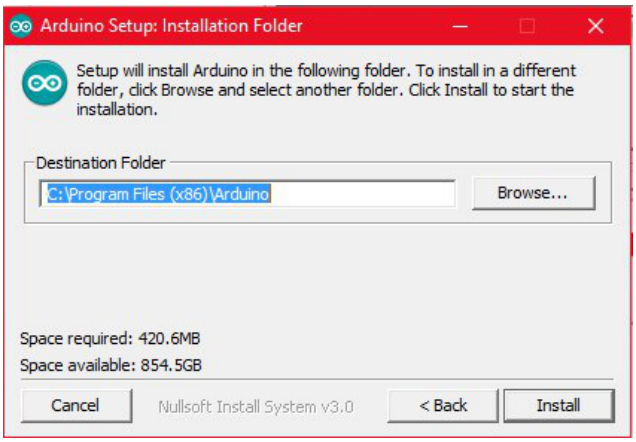

Imagen 4.2.2.4. Selección de la carpeta de instalación.

6. Una vez se haya terminado la instalación, aparecerá una pantalla indicándolo. Pulsamos “close” para terminar con ella.

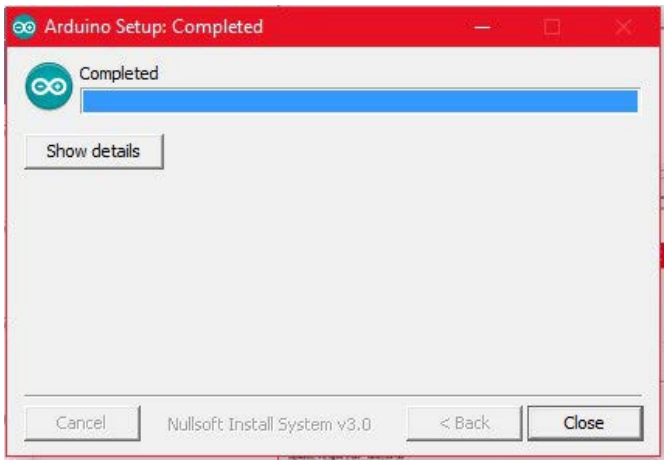

Imagen 4.2.2.5. Instalación del IDE de Arduino completada.

7. Abrimos la aplicación de escritorio “Arduino”.

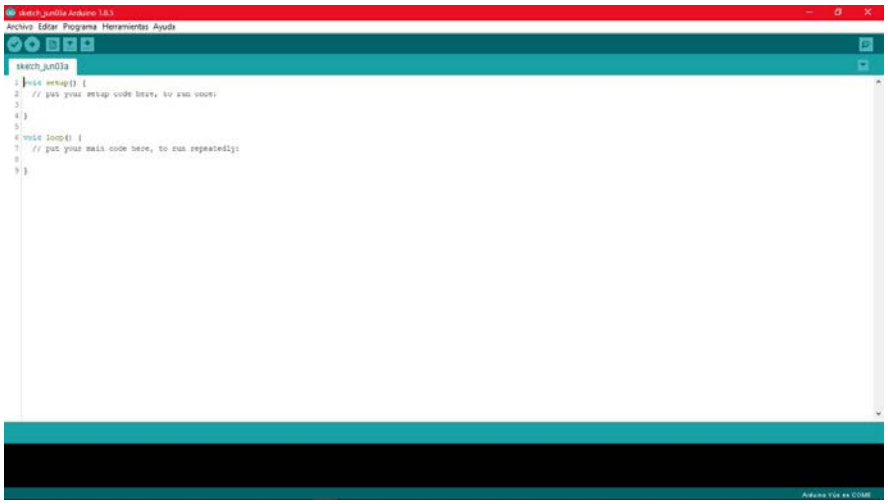

Imagen 4.2.2.6. IDE de Arduino.

8. Conectamos ahora el sensor Arduino con la placa radio de LoRa para Arduino y el módulo SX1272 con la antena, mediante un cable USB 2.0 tipo A macho – tipo B macho.

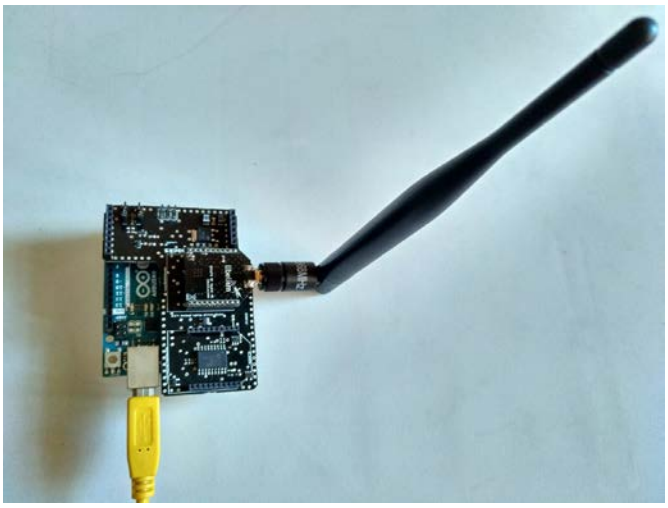

Imagen 4.2.2.7. Sensor Arduino Uno con el módulo LoRa y la placa de radio LoRa para Arduino conectada.

9. Procedemos ahora a seleccionar el puerto y la placa en el IDE de Arduino. Para ello, seleccionamos en la barra de herramientas “Herramientas”, y ahí seleccionamos la placa “Arduino/Genuino Uno” y el puerto.

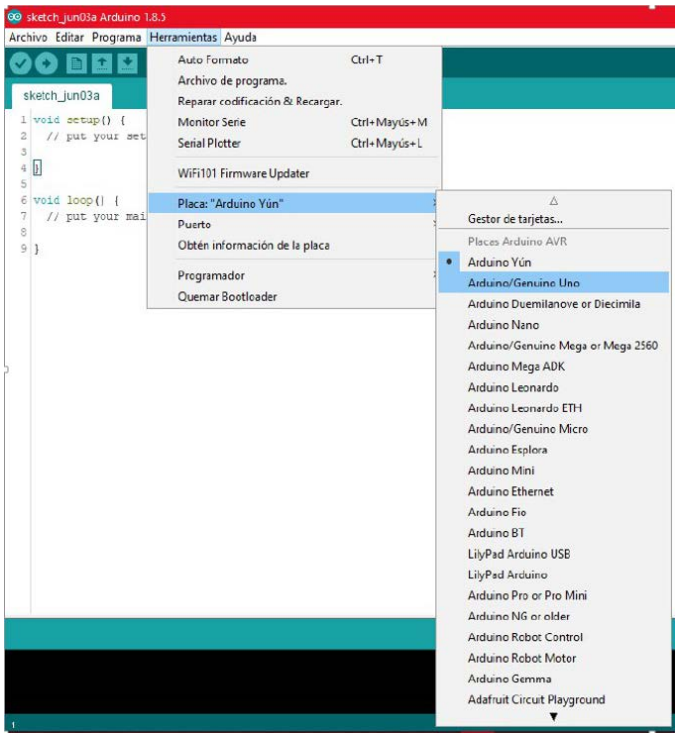

Imagen 4.2.2.8. Selección de placa “Arduino/Genuino Uno”.

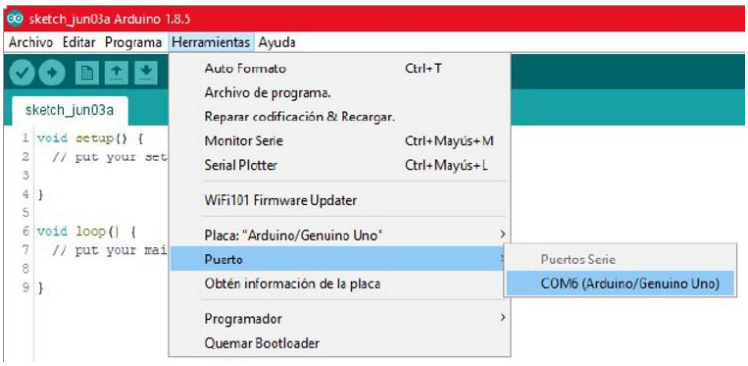

Imagen 4.2.2.9. Selección de puerto en la placa Arduino.

10. A continuación, hemos de descargar las librerías necesarias para el funcionamiento del módulo LoRa. Las descargamos de la siguiente dirección: www.cooking-hacks.com/media/cooking/images/documentation/tutorial_SX1272/SX1272_library_arduino_v1.4.zip
11. Al descargarlas, descomprimimos el archivo, y aparecerán otros dos archivos comprimidos: Arduino-api_v1_4.zip y arduinoLoRa_v1_4.zip. Descomprimimos el archivo "arduinoLoRa_v1_4.zip", ya que hay que cambiar una línea en el archivo "arduinoLoRa.cpp". Lo abrimos y vamos a la línea 29. En lugar de "#include "../SPI/SPI.h", escribimos "#include <SPI.h>". Guardamos, volvemos a comprimir la carpeta que se nos había generado, y nos vamos al IDE de Arduino de nuevo.
12. Procedemos a importar las librerías ahora. Clicamos sobre "Programa", "Incluir Librería" y "Añadir librería .ZIP".

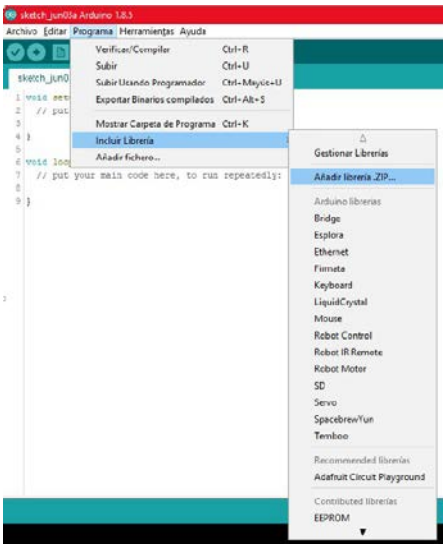

Imagen 4.2.2.10. Inclusión de las librerías necesarias para la usabilidad de los módulos a la lista general de librerías.

13. Incluimos las dos librerías que hemos descargado, repitiendo el proceso para ambas, y teniendo en cuenta que una de ellas la hemos modificado.
14. Ahora, incluimos en el sketch del código ambas, para poder utilizarlas en el programa que vayamos a escribir e introducir en la placa Arduino.

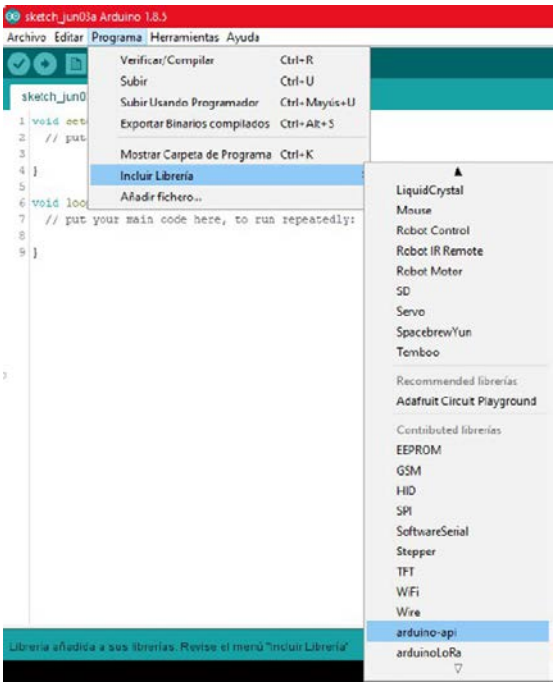

Imagen 4.2.2.11. Inclusión de las librerías necesarias para la usabilidad de los módulos.

15. Una vez incluidas las librerías, se puede comenzar a introducir el código para programar la placa.

4.3. Módulo SX1272.

El transceptor SX1272 [6] presenta el módem LoRa de comunicación de largo alcance, de espectro ensanchado y gran inmunidad frente a interferencias, mientras minimiza el consumo de corriente. El módulo ha de acompañarse por una antena de 868 MHz, pues si no se emplea, éste podría verse dañado por reflexiones de radiofrecuencia.

El uso de la modulación LoRa patentada por Semtech en este módulo hace que éste alcance una sensibilidad superior a -137dBm, usando un cristal oscilador de bajo coste entre otros materiales. Esta gran sensibilidad, unida al amplificador integrado de -20dBm, lo hace óptimo para cualquier comunicación que precise de robustez. LoRa también cuenta con ventajas importantes sobre bloqueo y sensibilidad sobre técnicas

de modulación convencionales, resolviendo el compromiso tradicional entre rango, inmunidad frente a interferencias y consumo de energía.

El módulo SX1272 utiliza los pines SPI para realizar la comunicación, los cuales permiten una mayor velocidad en la comunicación y liberan el UART (Universal Asynchronous Receiver – Transmitter, Transmisor – Receptor Asíncrono Universal) de Waspmote o Arduino para otros propósitos.

4.3.1. Características técnicas.

Los rangos de voltaje oscilan entre los máximos de -0.5 V y 3.9 V, aunque el rango de operación es preferible que se encuentre entre 1.8 V y 3.7 V.

En cuanto a la temperatura, soporta desde -55 ºC hasta 115 ºC, aunque el rango operativo incluye desde los -40 ºC hasta los 85 ºC.

4.3.2. Arquitectura.

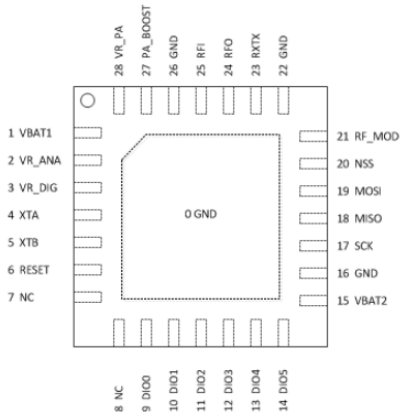

Imagen 4.3.2.1. Diagrama de pines del módulo SX1272. Fuente: Semtech.

La descripción de los pines es la siguiente:

Número	Nombre	Tipo	Descripción
0	GROUND	-	Pista de tierra expuesta.
1	VBAT1	-	Voltaje de suministro
2	VR_ANA	-	Voltaje de suministro regulado para circuitería analógica.
3	VR_DIG	-	Voltaje de suministro regulado para bloques digitales.
4	XTA	I/O	Conexión XTAL o entrada TCXO.
5	XTB	I/O	Conexión XTAL.
6	RESET	I/O	Trigger de reseteo de entrada.
7	NC	-	Puede conectarse a tierra.

8	NC	-	Puede conectarse a tierra.
9	DIO0	I/O	I/O digital, configurado por software.
10	DIO1/DCLK	I/O	I/O digital, configurado por software.
11	DIO2/DATA	I/O	I/O digital, configurado por software.
12	DIO3	I/O	I/O digital, configurado por software.
13	DIO4	I/O	I/O digital, configurado por software.
14	DIO5	I/O	I/O digital, configurado por software.
15	VBAT2	-	Voltaje de suministro.
16	GND	-	Tierra.
17	SCK	I	Reloj de entrada SPI.
18	MISO	O	SPI de salida de datos.
19	MOSI	I	SPI de entrada de datos.
20	NSS	I	Chip SPI de selección de entrada.
21	RF_MOD	O	NC
22	GND	O	Tierra
23	RXTX	O	Conmutador de control Rx/Tx: alto en Tx.
24	RFO	O	Salida de RF.
25	RFI	I	Entrada de RF.
26	GND	O	Tierra.
27	PA_BOOST	O	Salida opcional de alta-potencia PA.
28	VR_PA	O	Suministro regulado para PA.

Tabla 4.3.2.1. Descripción de los pines del módulo SX1272. Fuente: Semtech.

4.3.3. Características clave del producto.

- Módem LoRa.
- Presupuesto de enlace máximo de 157 dB.
- +20dBm a 100mW con una salida de RF constante vs. Suministro de voltaje.
- +14dBm alta eficiencia PA.
- Bit rate programable de hasta 300 kbps.
- Gran sensibilidad: hasta -137 dBm.
- Interfaz a prueba de balas: IIP3 = -12.5 dBm.
- Inmunidad de bloqueo de 89 dB.
- Corriente de recepción de 10 mA, registro de retención de 100 nA.
- Sintetizador completamente integrado con una resolución de 61 Hz.
- Modulaciones FSK, GFSK, MSK, GMSK, LoRa y OOK.
- Sincronizador de bit incorporado para recuperación de reloj.
- Detección de preámbulo.

- RSSI con rango dinámico de 127 dB.
- Sensor automático de radiofrecuencia Y CAD con AFC ultra-rápido.
- Motor de paquetes de hasta 256 bytes con CRC.
- Sensor de temperatura incorporado e indicador de batería baja.

4.4. Módulo WiFi PRO.

El módulo WiFi PRO [7] está gestionado por UART y puede conectarse tanto al SOCKET0 como al SOCKET1 del sensor Waspmote. Las características principales del módulo son:

- Potencia de transmisión:
  - 802.11b: 17 dBm
  - 802.11g: 14 dBm
  - 802.11n: 12 dBm
- Sensibilidad de recepción:
  - 802.11b @11Mbps PER<8%: -87 dBm.
  - 802.11b @1Mbps PER<8%: -94 dBm.
  - 802.11g @54Mbps PER<10%: -73 dBm.
  - 802.11g @6Mbps PER<10%: -86 dBm.
  - 802.11n MCS0 PER<10%: -86 dBm.
  - 802.11n MCS0 PER<10%: -70 dBm.
- Consumo del chipset:
  - Modo TX: 350 mA.
  - Modo RX: 130 mA.
- Protocolos de internet: ARP, ICMP, IP, UDP, TCP, DHCP, DNS, NTP, HTTP, FTP.
- Protocolos de seguridad: SSL3/TLS1, HTTPS, RSA, AES-128/256, 3DES, RC-4, SHA-1, MD-5, WEP, WPA y WPA2.
- Especificaciones inalámbricas:
  - Estándares: IEE 802.11 b/g/n.
  - Frecuencia en Europa: 2412 – 2472 GHz.
  - Canales: 1 a 11.
- Antena integrada.

Figura 4.4.1. Módulo WiFi PRO. Fuente: Libelium.

Cuando un módulo SX1272 transmite datos, los recibe un Gateway LoRa – WiFi, el cual hace posible la transmisión de los datos recogidos por los sensores periféricos a la plataforma cloud, siendo el módulo WiFi PRO el encargado de hacerlo mediante una conexión HTTP con el servidor cloud.

Para poder conectarlo, se utilizará una placa radio de expansión (Expansion Radio Board), ya que en el socket 1 no se puede conectar directamente. No es compatible con Arduino.

Para su correcto funcionamiento, en el código primero hay que encender el módulo, y a continuación configurar la información para acceder al punto de acceso (ESSID del router y contraseña). En caso de querer asignar una dirección IP estática, se podrá hacer, aunque por defecto se asignará unan dirección IP dinámica mediante DHCP.

Para el uso del cliente HTTP, elegiremos el método POST, en el que los datos irán en el cuerpo. El código se encuentra en el anexo 7.2.1.2.

4.5. LoRa.

LoRa [8] se trata de un protocolo privado de nivel de enlace y de espectro ensanchado que tiene unas características únicas que le permiten abarcar grandes distancias a costa de un coste energético mínimo.

Las características que lo definen son las siguientes:

4.5.1. Canales.

El protocolo LoRa trabaja en una banda doble de frecuencia, 868 MHz y 900MHz. En este documento solamente se verán reflejadas las características relacionadas con la banda ISM de 868 MHz, ya que es la utilizada en Europa. Esta banda de frecuencia

cuenta con 8 canales con un ancho de banda de 0.3 MHz por canal. Los canales disponibles son los siguientes:

NÚMERO DEL CANAL	FRECUENCIA CENTRAL
CH_10_868	865.20 MHz
CH_11_868	865.50 MHz
CH_12_868	865.80 MHz
CH_13_868	866.10 MHz
CH_14_868	866.40 MHz
CH_15_868	866.70 MHz
CH_16_868	867 MHz
CH_17_868	868 MHz

Tabla 4.5.1.1. Canales usados por el módulo LoRa en la banda de 868 MHz. Fuente: Libelium.

Debido a las características de propagación de la banda, el campo cercano podría hacer que dos sensores no puedan comunicarse entre sí si están demasiado cerca el uno del otro, por lo que se ha de mantener una distancia mínima de 3-4 metros entre ellos.

4.5.2. Potencia de transmisión.

En cuanto a la sección energética, la potencia de transmisión puede ser ajustada a distintos valores:

Parámetro	Nivel de potencia de salida
'L' (low)	0 dBm
'H' (high)	7 dBm
'M' (Maximum)	14 dBm

Tabla 4.5.2.1. Valores de potencia de transmisión. Fuente: Libelium.

Cuanto mayor sea la potencia de transmisión, mayor será el coste energético que se produzca.

La fuerza de la señal recibida (RSSI, Received Signal Strength) es un parámetro que se puede almacenar en el módulo y que puede mostrarse por pantalla. Se pueden

distinguir dos tipos, de paquete y de canal. Si la RSSI de paquete es superior a la sensibilidad, el paquete enviado se detectará de forma satisfactoria. De no ser así, el paquete se perderá. La RSSI de canal informa del nivel de señal detectado en cada momento, incluso si no hay señal siendo transmitida, informando acerca del nivel de ruido.

Cada escenario precisará de un nivel de potencia diferente, de acuerdo a las necesidades del mismo.

El modo de trabajo ideal es aquel que permite una cobertura máxima con el mínimo nivel de potencia, por lo que aparece un compromiso entre el nivel de potencia y la cobertura. Cada escenario de aplicación precisará de la realización de algunas pruebas para descubrir la mejor combinación de ambos parámetros.

4.5.3. Modulación LoRa.

Cuando se enciende el módulo SX1272, éste está preparado para utilizar el modo LoRa.

En cuanto los estados de operación, son realizados automáticamente por las funciones de la API.

4.5.3.1. Modo LoRa.

Se trata de una modulación avanzada y privada que aumenta significativamente el rango de alcance de la señal transmitida, en comparación con modulaciones clásicas. Este modo de largo alcance provee una comunicación de espectro ensanchado ultra larga, y tiene una gran inmunidad frente a interferencias, a la vez que minimiza el consumo de corriente. Combina espectro ensanchado digital, procesado digital de la señal y corrección de errores en el código. También provee ventajas tanto en bloqueo como en selección sobre otras técnicas de modulación convencionales.

LoRa tiene tres parámetros configurables:

- **Ancho de banda (Bandwidth – BW).** El ancho de banda es la longitud de la extensión de frecuencias en la que se encuentra la mayor potencia de la señal. Solamente se puede escoger entre 3 opciones: 125 kHz, 200 Hz o 500 Hz. Si se requiere una transmisión rápida, la opción de 500 MHz es mejor, pero si se necesita un mayor alcance, el módulo tendrá que configurarse con el valor de 125 kHz. Cuanto menor es el ancho de banda, más tiempo en el aire (Time on Air) permanece la transmisión, pero a su vez la sensibilidad aumenta, por lo que

la comunicación tiene un mayor presupuesto de enlace. Un mayor tiempo en el aire implica un aumento en el consumo de batería.

- **Ratio de codificación (Coding Rate – CR).** Puede tomar uno de estos cuatro valores: 4/5, 4/6, 4/7 y 4/8. Esto implica que habrá 4 bits útiles codificados en 5, 6, 7 u 8 bits de transmisión, respectivamente. Cuanto menor es el ratio de codificación (el más pequeño es 4/8), mayor es el tiempo en el aire de la transmisión. Esto hace más rápida la tarea de recepción, pues cada símbolo está más separado en el tiempo, por lo que el receptor puede demodular los paquetes con una menor potencia de recepción, pero a su vez se tarda más tiempo en transmitir el paquete, lo que se traduce en un aumento en el consumo energético.

- **Factor de expansión (Spreading Factor – SF).** El factor de expansión es el número de chips por símbolo utilizados en el tratamiento de datos antes de la transmisión de la señal. Su valor es un número entero entre 6 y 12 (ambos inclusive). Este parámetro es relevante en la técnica de espectro ensanchado, pues cuanto mayor es su valor, mayor capacidad tiene el receptor para separar el ruido de la señal. Cuanto mayor es el valor tomado, mayor tiempo se tarda en enviar un paquete, pero a su vez mayor es el rango alcanzado porque la sensibilidad del receptor es mayor.

Gracias a los diferentes valores del factor de expansión y del ancho de banda que puede tomar la señal con LoRa, esto hace que dos señales que tengan ambos parámetros diferentes o al menos uno de ellos, serán ortogonales, por lo que podrán transmitirse y recibirse simultáneamente en el mismo canal sin sufrir interferencias.

La combinación de estos valores define el modo de transmisión, el cual puede estar predefinido con unos valores estándar, o bien se pueden definir estos parámetros de forma manual.

En la API hay 10 modos definidos, incluyendo el modo de larga distancia y el de “alta velocidad”, aunque es posible añadir nuevos modos.

Los modos disponibles son los siguientes:

Modo	BW	CR	SF	Sensibilidad (dB)	Comentarios
1	125	4/5	12	-134	Máximo alcance, data rate lento, mayor impacto en la batería.
2	250	4/5	12	-131	-
3	125	4/5	10	-129	-
4	500	4/5	12	-128	-
5	250	4/5	10	-126	-
6	500	4/5	11	-125.5	-
7	250	4/5	9	-123	-
8	500	4/5	9	-120	-
9	500	4/5	8	-117	-
10	500	4/5	7	-114	Mínimo alcance, data rate rápido, mínimo impacto en la batería.

Tabla 4.5.3.1.1. Modos de configuración de LoRa. Fuente: Libelium.

4.5.4. Parámetros de paquete.

Los paquetes tienen una estructura con diversos campos:

- dst. Dirección del nodo de destino. Este parámetro está indicado como un input en la función usada por el usuario.
- src. Dirección del nodo de origen. Es rellenado por la aplicación con la dirección del módulo (previamente fijado por el usuario).
- packnum. Número del paquete. Es rellenado por la aplicación. Es un campo de un byte, con un rango de 0 a 255. Cuando se alcanza este último valor, se reinicia el contador de nuevo desde 0. Si el paquete está intentando ser retransmitido, el número de paquete no incrementa.
- length. Tamaño del paquete. Es rellenado por la aplicación, e indica el tamaño total del paquete.

- data [MAX_PAYLOAD]. Se usa para almacenar los datos que se van a enviar a otros nodos dentro del paquete. El tamaño máximo está definido por MAX_PAYLOAD.
- retry. Contador de reintentos. Es rellenado por la aplicación. Cuando se usa la propiedad “retry”, este valor incrementa desde 0 hasta el valor máximo de reintentos establecidos, siendo el máximo de estos 5.

dst	src	packnum	length	data	retry
1 Byte	1 Byte	1 Byte	1 Byte	Nº de bytes variable	1 Byte

Tabla 4.5.4.1. Estructura del payload del paquete. Fuente: Libelium.

4.5.5. Conexiones.

El módulo LoRa soporta transmisiones Unicast y Broadcast.

Las comunicaciones unicast son utilizadas para enviar un paquete a un nodo específico, y siempre se ejecutan mecanismos de control de tiempo (time-out) para prevenir una espera perpetua de un paquete. Soporta servicios adicionales para una comunicación más robusta, como son la confirmación de ACK o los reintentos, los cuales solamente están disponibles cuando la confirmación de ACK está activada.

Las comunicaciones broadcast se utilizan para enviar un paquete a todos los nodos de una red. Cualquier módulo dentro del rango aceptará el paquete que contiene la dirección de broadcast, que será la 0. En este modo no hay posibilidad de confirmaciones de ACK ni reintentos.

Por lo tanto, la dirección 0 no será asignable a ningún nodo, pues está reservada para la transmisión de tipo broadcast. Además, la dirección de nodo 1 se reservará en su caso para el nodo central de la topología en estrella, por lo que las direcciones asignables al resto de nodos abarcarán el rango de 2 a 255.

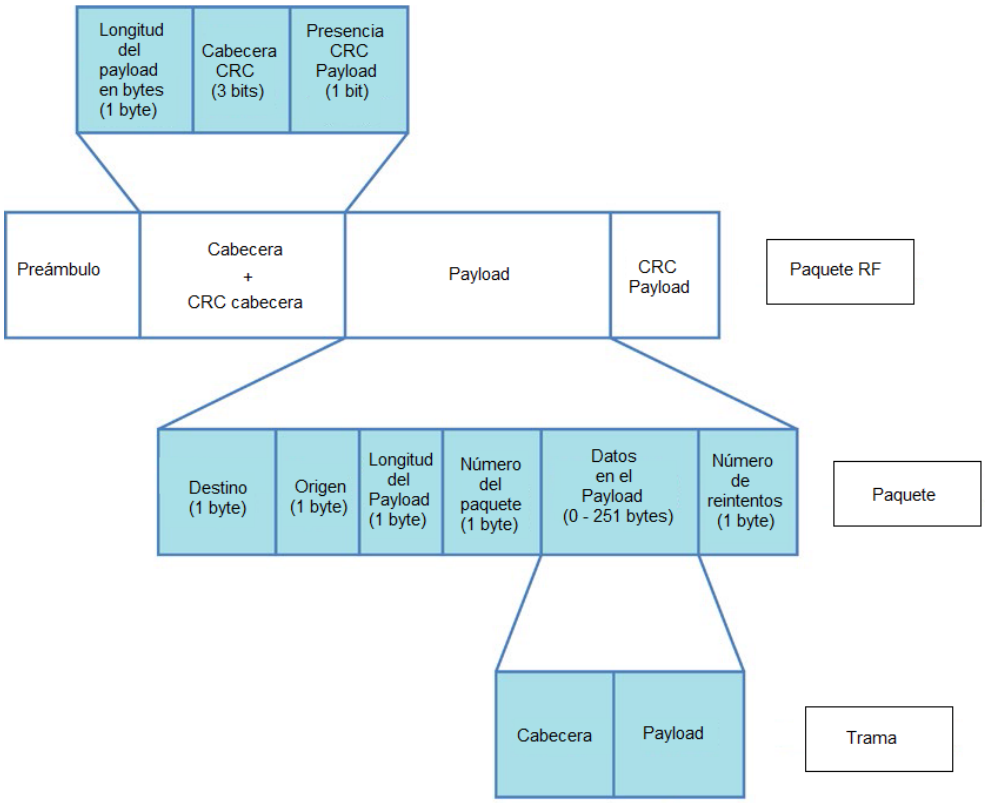

Imagen 4.5.5.1. Formato de paquete LoRa. Fuente: Libelium.

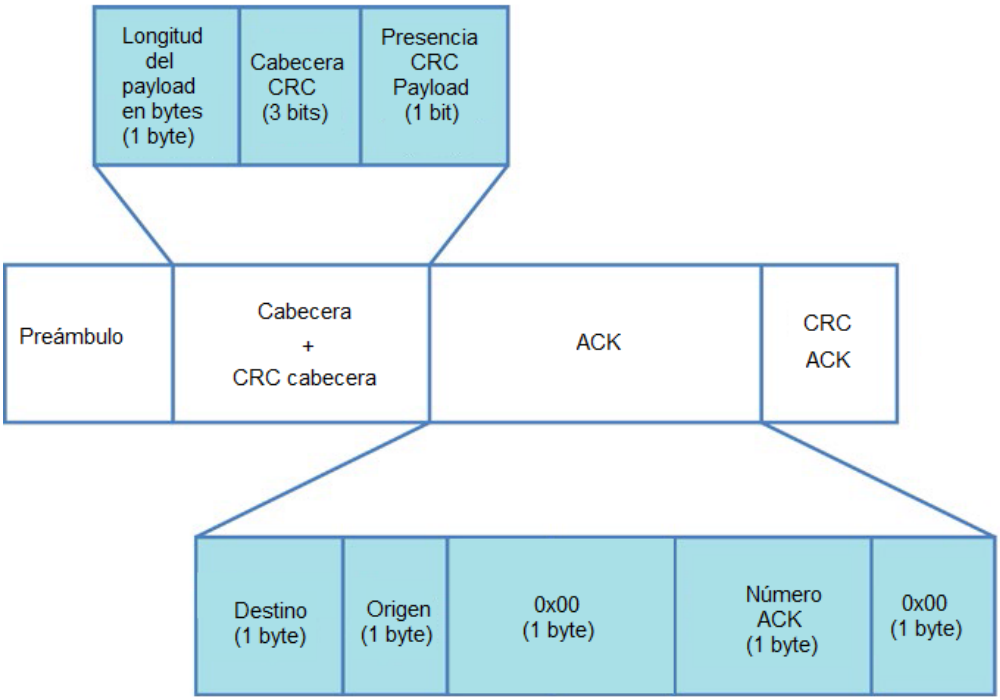

Imagen 4.5.5.2. Formato de paquete ACK LoRa. Fuente: Libelium.

4.5.6. Creación de una red.

Para crear una red, únicamente es necesario fijar dos parámetros en LoRa: canal y modo. En caso de que la red ya esté creada, el nuevo nodo ha de configurar su canal y su modo para que sean los mismos que los de la red a la que quiere pertenecer.

4.6. Protocolo HTTP.

El Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol, HTTP) [9], [10] es un protocolo de nivel de aplicación de protocolo de Internet que utiliza TCP como protocolo de transporte y que generalmente utiliza el puerto número 80. Es uno de los protocolos de comunicación más importantes en los sistemas multimedia cliente – servidor.

A través de HTTP, un cliente puede transferir datos a un servidor o consultar recursos especiales desde el servidor. Dichos recursos pueden incluir archivos tales como datos multimedia o programas que residen en el servidor y que se ejecutan en éste cuando son llamados por el cliente, y devuelven los resultados calculados al cliente.

En nuestro caso, únicamente se procederá al envío de datos tipo texto, ya que debido a las características del protocolo anterior (LoRa), no es posible enviar archivos con un tamaño elevado.

Los recursos se distribuyen en varias máquinas en la red informática, y para poder direccionar uno de ellos que se encuentre en una máquina remota en la red, necesitamos un identificador único: URL (Localizador de Recurso Uniforme, Uniform Resource Locator). Una URL es una dirección a través de la cual un recurso (por ejemplo, una página HTML) puede identificarse de manera única en internet.

4.6.1. Estructura de una URL.

Una URL tiene generalmente la siguiente estructura:

protocolo://[usuario]:[contraseña@]dominio[:puerto][/path][?petición][#ancla]

Los elementos entre corchetes son opcionales.

- Protocolo: nombre del protocolo de red, generalmente HTTP, aunque también ftp, https y otros. En nuestro caso, utilizaremos HTTP.
- Usuario:contraseña: Nombre de usuario y contraseña para autenticarse en el servidor. En nuestro caso, no se utilizará.
- Dominio: nombre del dominio del servidor o dirección IP. En este caso, será: “pruebas.libelium.com”.

- Puerto: Número de puerto del servidor. Si no se especifica ninguno, se utilizará el puerto estándar para el protocolo dado. Por ejemplo, para HTTP, será el puerto 80, que además será el mismo que usaremos en nuestro código.
- Path: directorio o archivo en el servidor. En nuestro caso, utilizaremos “getpost_frame_parser.php”.
- Petición. Una petición tipo String, que se puede utilizar para enviar información adicional al servidor. Si la petición fuese tipo GET, se utilizaría, pero al ser tipo POST, la petición irá en el cuerpo del mensaje y no en la cabecera.
- Ancla: un punto de entrada (fragmento) dentro de una página HTML. En nuestro caso, no procede su uso.

4.6.2. Proceso de comunicación entre el cliente y el servidor.

Los pasos básicos para la comunicación entre cliente y servidor son los siguientes:

1. El cliente establece una conexión con el servidor a través de TCP.
2. El cliente envía una solicitud HTTP y solicita el recurso especificado en la URL del servidor.
3. El servidor procesa la solicitud. Por ejemplo, procede a la generación dinámica de código HTML.
4. El servidor envía su respuesta HTTP al cliente.
5. El cliente procesa la respuesta. Por ejemplo, puede renderizar el código HTML.
6. El cliente o el servidor cierra la conexión.

Una solicitud HTTP o respuesta HTTP consiste en una información de cabecera seguida de unan línea en blanco, seguida a su vez de una carga útil opcional.

HTTP es un protocolo sin estado, lo que significa que el servidor no tiene información sobre solicitudes anteriores de un cliente, es decir, las solicitudes individuales al servidor incluso desde el mismo cliente se consideran independientes entre sí. Esto significa a su vez que la conexión entre el cliente y el servidor no tiene que mantenerse después de una transferencia de datos exitosa.

Hay distintos tipos de métodos HTTP:

- GET: Solicita al servidor que envíe un objeto. Es el método más comúnmente utilizado. Los datos irán en la cabecera, formando parte de la URL como peticiones “query”.
- POST: Transferencia de datos del usuario al servidor. Los datos de usuario se anexan al encabezado. Los datos irán en el cuerpo del mensaje, lo que querrá

decir que habrá carga útil en este caso. Es el método que utilizaremos en nuestro código.

- HEAD: Solamente devuelve el encabezado de respuesta a un objeto; el objeto en sí no se transfiere.
- PUT: pone recursos en el servidor.
- DELETE: elimina un objeto del servidor.

Los métodos PUT y DELETE no son procesados por la mayoría de los servidores por razones de seguridad.

El servidor contestará con un código numérico, indicando si el proceso de solicitud se ha realizado correctamente o no. Las distintas franjas de valores que puede tener son las siguientes:

- 1xx: mensaje informativo.
- 2xx: solicitud exitosa.
- 3xx: la solicitud ha sido redirigida.
- 4xx: solicitud incorrecta.
- 5xx: error de servidor.

5. Resultados y Discusión.

Como resultado de este trabajo se ha obtenido una infraestructura de comunicaciones con dos redes diferentes interconectadas entre sí mediante un Gateway LoRa – WiFi.

Dicha infraestructura se ha probado en tres escenarios diferentes, y los resultados obtenidos han sido los siguientes:

5.1. Metodología experimental.

Para hacer las pruebas pertinentes, se han tenido en cuenta 3 escenarios:

- Escenario urbano.
- Escenario en campo abierto.
- Escenario en el interior de un edificio.

En el escenario urbano y en el escenario en campo abierto se ha tomado la misma configuración tanto en Waspmote como en Arduino, buscando siempre el mayor alcance de la señal. La configuración se puede ver en el Anexo 7.1.

En el escenario interior, se ha buscado una mayor velocidad de transmisión en el envío de tramas, ya que lo que se buscaba era realizar la subida de datos al servidor mediante el protocolo HTTP, y los sensores estaban a tan solo unos metros de distancia. La configuración se puede ver en el Anexo 7.2.

En estos escenarios no se ha podido calcular el porcentaje de paquetes enviados y recibidos en un punto fijo, ya que el dispositivo receptor se encontraba dentro de un coche en movimiento sin posibilidad de parar/estacionar.

El RSSI solamente se ha podido calcular en el escenario urbano.

5.1.1. Escenario urbano.

Para estas pruebas, se ha considerado una topología punto a punto en la que el transmisor se encontraba en un punto fijo a la altura de un tercer piso, mientras el receptor se encontraba en un punto móvil en el interior de un coche. La fuente de energía del transmisor era una batería portátil, mientras que el receptor estaba conectado a un ordenador portátil para poder ser monitorizado.

Los resultados de alcance son los siguientes:

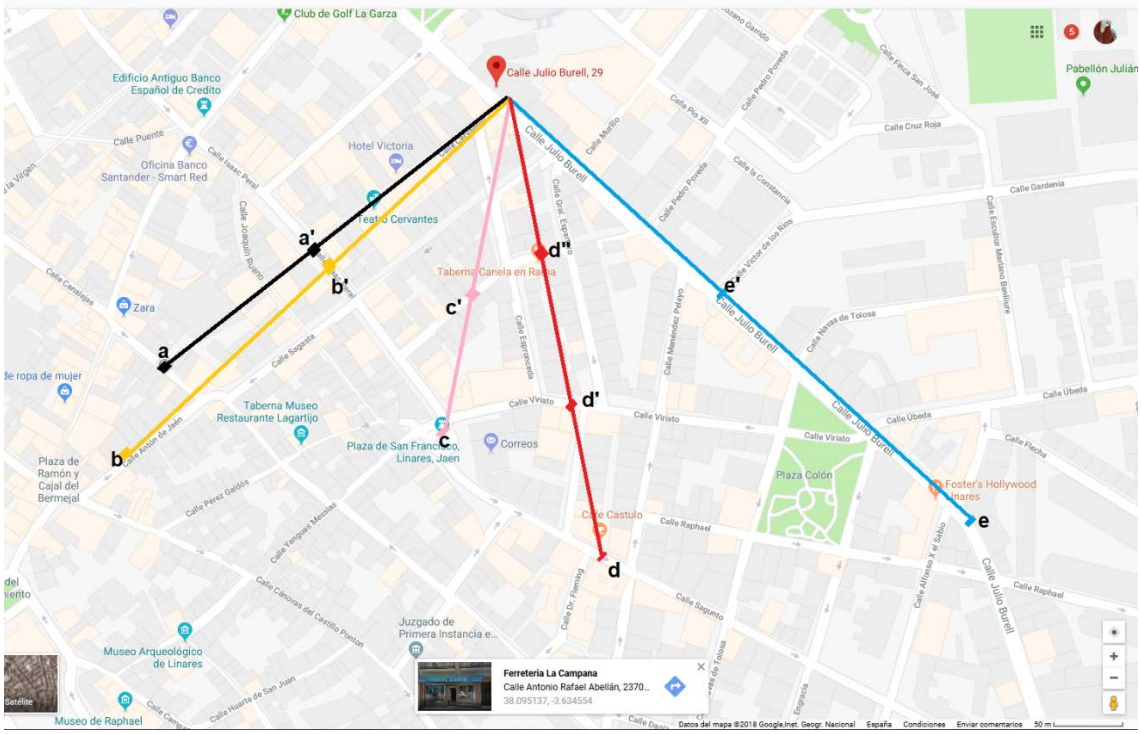

Imagen 5.1.1.1. Alcance en entorno urbano.

Letra	Alcance (m)	Nº edificios atravesados.	RSSI de paquete	RSSI de canal
a	326.11	6	-132	-114
a'	184.58	0	-82	-113
b	394.67	24	-131	-110
b'	182.89	9	-101	-109
c	256.64	14	-125	-109
c'	149	10	-102	-106
d	352.5	20	-120	-110
d'	233.76	13	-96	-106
d''	119.57	7	-98	-109
e	468.16	0	-98	-114
e'	215.18	0	-79	-114

Tabla 5.1.1.1. Medidas del alcance de la señal transmitida por el módulo LoRa en un entorno urbano.

5.1.2. Escenario en campo abierto.

Para realizar este tipo de pruebas, se han tenido en cuenta dos escenarios, uno entre Linares y Bailén (en la provincia de Jaén), y otro entre el embalse del río Fresnedas y las poblaciones de Calzada de Calatrava y Aldea del Rey, en la provincia de Ciudad Real (se especificarán las coordenadas geográficas exactas más adelante).

5.1.2.1. Linares – Bailén.

En este escenario, el transmisor se encontraba en un punto fijo en el polígono industrial de Linares (coordenadas geográficas 38º05'05.2"N 3º39'21.5"W), y el receptor se situaba en un punto fijo en la rotonda de entrada a Bailén (coordenadas geográficas 38º05'59.3"N 3º45'22.7W).

Imagen 5.1.2.1.1. Distancia entre los dos puntos donde se produce comunicación entre Linares y Bailén mediante el módulo LoRa.

La fuente de energía del transmisor era una batería portátil, mientras que el receptor estaba conectado a un ordenador portátil para poder ser monitorizado.

En esta ocasión, la línea de visión era directa al no encontrarse obstruida por ningún obstáculo. Se alcanzó una distancia de 9.03 km.

5.1.2.2. Embalse del río Fresnedas – Aldea del Rey.

En esta ocasión, el transmisor se encontraba en la planta del embalse del río Fresnedas, en Ciudad Real (coordenadas geográficas 38º45'23.1"N 3º49'55.6"W).

Imagen 5.1.2.2.1. Alcance de la señal transmitida por LoRa entre el embalse del río Fresnedas y Aldea del Rey, Ciudad Real.

La fuente de energía del transmisor era una batería portátil, mientras que el receptor estaba conectado a un ordenador portátil para poder ser monitorizado.

El alcance de esta comunicación alcanzó los 29.23 km de distancia en línea de visión directa.

Imagen 5.1.2.2.2. Vistas desde la planta del embalse del río Fresnedas, en dirección a Calzada de Calatrava.

5.1.3. Escenario interior.

En esta topología, se ha establecido una conexión entre dos nodos, un transmisor (Arduino y/o Waspmote) y un receptor (Waspmote), y el receptor a su vez contaba con un módulo WiFi PRO, el cual mediante el protocolo HTTP realiza una conexión con un servidor de pruebas de Libelium.

Tanto transmisor como receptor en esta ocasión estaban conectados a un ordenador (cada uno a uno diferente, ya que los IDE tanto de Arduino como de Waspmote no permiten monitorizar más de una placa a la vez), en dos habitaciones distintas separadas por una pared.

En esta ocasión, se escogió el modo LoRa de transmisión 10, ya que no hacía falta alcanzar un máximo alcance (eran pruebas de comunicación).

El código empleado para este escenario se puede encontrar en el anexo 7.2.

Una de las redes que se ha desplegado ha sido una interconectada mediante el protocolo de enlace LoRa. Esta red cuenta con sensores del Internet de las Cosas, siendo estos tanto Arduino como Waspmote, los cuales llevan acopladas unas placas de adaptación y/o expansión, a las que se les ha conectado módulos SX1272, que incluyen en su interior un módem LoRa. Estos sensores se han dispuesto mediante una topología en estrella, siendo el nodo central el Gateway Lora – WiFi.

La otra red es una red de comunicaciones basada en la tecnología WiFi, la cual permite una comunicación con el servidor de nuestra Gateway LoRa – WiFi, permitiendo a su vez el transporte de los datos recibidos de la red de sensores con una plataforma cloud, estableciéndose la comunicación mediante un protocolo HTTP, enviando los datos con un método POST.

Las transmisiones se han realizado enviando una cadena de caracteres con respuesta de ACK habilitada y un número de reintentos fijado a 3, y han sido las mismas tanto en Arduino como en Waspmote, obteniéndose con ambos los mismos resultados.

En un entorno urbano se ha alcanzado una distancia de comunicación entre nodos de hasta 468 metros, y en un entorno en campo abierto, más de 29 km.

Para finalizar, aunque no menos importantes, las pruebas en el entorno interior sirvieron para confirmar que el módulo WiFi PRO es capaz de establecer una conexión con el servidor de pruebas de Libelium mediante el protocolo HTTP.

6. Conclusiones.

En este Trabajo de Fin de Grado se ha diseñado un protocolo de comunicaciones entre una red de sensores del Internet de las Cosas, y una plataforma de Cloud Computing.

Para ello, se ha diseñado una topología de red que ha aunado las tecnologías LoRa y WiFi, estableciendo una red de sensores comunicada mediante el protocolo de enlace LoRa, y una red WiFi para el envío de datos a una plataforma de Cloud Computing. Con el fin de aunar ambas tecnologías, se ha diseñado un Gateway LoRa – WiFi. Los sensores periféricos transmiten datos a un nodo central situado en el Gateway mediante el protocolo LoRa, y éste envía vía WiFi los datos recabados a una plataforma de Cloud Computing mediante el protocolo HTTP.

Se han realizado pruebas en tres escenarios diferentes: escenario urbano, escenario en campo abierto, y escenario interior.

En el escenario urbano se ha conseguido realizar una comunicación de hasta 468 metros, debido a la presencia de elementos arquitectónicos que impiden la línea directa de visión entre sensores. Las pruebas realizadas muestran que aunque si bien el poder de penetración y la distancia alcanzada en un entorno urbano son generosos, no se cumple tanto como el fabricante promete, presumiblemente por la mitigación de la señal al atravesar los edificios.

En cuanto a las pruebas en campo abierto con línea de visión directa, se han sobrepasado con creces los parámetros dados por el fabricante, llegando a alcanzar más de 29 km de distancia entre dos nodos, estando además uno de ellos en movimiento sin posibilidad de parada.

En el escenario interior se ha comprobado el correcto funcionamiento del Gateway LoRa – WiFi, realizando satisfactoriamente conexiones a un servidor Cloud, enviando los datos recibidos desde los sensores. La el envío de datos a la plataforma de Cloud Computing se ha realizado mediante el protocolo HTTP, en una petición tipo POST.

7. Anexos.

En los siguientes anexos se incluye el código utilizado para la configuración de las placas de Waspmote y Arduino para cada uno de los distintos escenarios.

7.1. Escenario urbano y en campo abierto.

7.1.1. Waspmote.

7.1.1.1. Código del transmisor.

```
// Libreria para tx con el modulo sx1272
#include <WaspSX1272.h>

// Declaracion de la direccion del nodo de destino.
uint8_t rx_address = 2;

// Variable de estado. Si e == 0, no hay error.
int8_t e;

//Codigo de establecimiento. Se ejecuta una vez.
void setup()
{
  // Inicializacion del puerto USB.
  USB.ON();
  USB.println(F("Modo: Modulo TX LoRa con ACKs y reintentos"));
  USB.println(F("----------------------------------------"));
  USB.println(F("Configurando:"));
  USB.println(F("----------------------------------------"));

  // Inicializacion del modulo SX1272
  sx1272.ON();

  ////Opciones para configurar ////

  // Seleccionar el canal de frecuencia.
  e = sx1272.setChannel(CH_11_868);
  USB.print(F("Estableciendo canal CH_11_868.\t Estado: "));
  USB.println(e);

  //Seleccion de cabecera explicita.
```

```
  e = sx1272.setHeaderON();
  USB.print(F("Estableciendo cabecera (Header) ON.\t\t state "));
  USB.println(e);

  // Seleccion de modo. Se ha escogido el 1 porque tiene un mayor alcance.
  e = sx1272.setMode(1);
  USB.print(F("Estableciendo modo '1'.\t\t Estado: "));
  USB.println(e);

  // Seleccion de CRC
  e = sx1272.setCRC_ON();
  USB.print(F("Estableciendo CRC ON.\t\t\t Estado: "));
  USB.println(e);

  // Seleccion de potencia de salida maxima para obtener un mayor alcance.
  e = sx1272.setPower('M');
  USB.print(F("Estableciendo potencia a 'M'.\t\t Estado: "));
  USB.println(e);

  // Seleccion de direccion de nodo, establecida a 12.
  e = sx1272.setNodeAddress(12);
  USB.print(F("Estableciendo dirección de nodo a '12'.\t Estado: "));
  USB.println(e);

  // Seleccion de numero maximo de reintentos, establcido a 3.
  e = sx1272.setRetries(3);
  USB.print(F("Estableciendo numero de reintentos a '3'.\t Estado: "));
  USB.println(e);
  USB.println();

  delay(1000);

  USB.println(F("----------------------------------------"));
  USB.println(F("Enviando...:"));
  USB.println(F("----------------------------------------"));
}

//Codigo para repetir en bucle.
void loop()
{
```

```
  // Envio de paquete con reintentos, espera de ACK de respuesta.
  e = sx1272.sendPacketTimeoutACKRetries( rx_address, "Mensaje enviado.");

  // Comprobacion del estado de envio.
  if ( e == 0 )
  {
    USB.println(F("Paquete enviado correctamente."));
  }
  else
  {
    USB.print(F("Error enviando el paquete."));
    USB.print(F("Estado: "));
    USB.println(e, DEC);
  }
  delay(2500);
}
```

7.1.1.2. *Código del receptor.*

```
// Libreria para tx con el modulo sx1272
#include <WaspSX1272.h>

// Variable de estado. Si e == 0, no hay error.
int8_t e;

//Codigo de establecimiento. Se ejecuta una vez.
void setup()
{
  // Init USB port
  USB.ON();
  USB.println(F("Modo: módulo RX de LoRa con ACKs y reintentos."));
  USB.println(F("----------------------------------------"));
  USB.println(F("Configurando:"));
  USB.println(F("----------------------------------------"));

  // //Codigo de establecimiento. Se ejecuta una vez.
  sx1272.ON();

  //// Opciones para configurar ////

  // Seleccionar el canal de frecuencia.
  e = sx1272.setChannel(CH_11_868);
  USB.print(F("Estableciendo canal CH_11_868.\t Estado: "));
  USB.println(e);

   //Seleccion de cabecera explicita.
  e = sx1272.setHeaderON();
  USB.print(F("Estableciendo cabecera (Header) ON.\t\t state "));
  USB.println(e);

  // Seleccion de modo. Se ha escogido el 1 porque tiene un mayor alcance.
  e = sx1272.setMode(1);
  USB.print(F("Estableciendo modo '1'.\t\t Estado: "));
  USB.println(e);

  // Seleccion de CRC
  e = sx1272.setCRC_ON();
  USB.print(F("Estableciendo CRC ON.\t\t\t Estado: "));
```

```
  USB.println(e);

  // Seleccion de potencia de salida maxima para obtener un mayor alcance.
  e = sx1272.setPower('M');
  USB.print(F("Estableciendo potencia a 'M'.\t\t Estado: "));
  USB.println(e);

  // Seleccion de direccion de nodo, establecida a 2.
  e = sx1272.setNodeAddress(2);
  USB.print(F("Estableciendo dirección de nodo a '2'.\t Estado: "));
  USB.println(e);
  USB.println();

  delay(1000);

  USB.println(F("----------------------------------------"));
  USB.println(F("Recibiendo:"));
  USB.println(F("----------------------------------------"));
}

void loop()
{
  // Recibiendo paquete y enviando ACK de respuesta
  e = sx1272.receivePacketTimeoutACK(10000);

  // Comprobando estado de recepcion
  if( e == 0 )
  {
    USB.println(F("\nMostrando paquete recibido: "));
    // Mostrar paquete recibido.
    sx1272.showReceivedPacket();
  }
  else
  {
    USB.print(F("\nRecepcion de paquete TIMEOUT, estado: "));
    USB.println(e, DEC);
  }
}
```

7.1.2. Arduino.

7.1.2.1. Código del transmisor.

```
#include <Wire.h>

// Librerias Cooking API
#include <arduinoUtils.h>

// Librerias SX1272 y SPI
#include "arduinoLoRa.h"
#include <SPI.h>

int e;
char message1 [] = "Enviando paquete desde Arduino hasta Waspmote.";

void setup()
{
  // Abriendo comunicaciones serie y esperando al puerto para que se abra.
  //Ratio de baudios establecido a 115200 para que sea compatible con Waspmote.
  Serial.begin(115200);

  // Mensaje de encendido.
  Serial.println(F("Modulo SX1272 y Arduino: enviando paquetes con ACK y reintentos."));

  //Estableciendo la configuración inicial
    Serial.println(F("----------------------------------------"));
    Serial.println(F("Estableciendo la configuración inicial:"));
    Serial.println(F("----------------------------------------"));

  // Encendiendo el modulo
  e = sx1272.ON();
  Serial.print(F("Encendiendo(Setting power ON). \t\t Estado: "));
  Serial.println(e, DEC);

  // Configurando el modo. Se ha escogido el 1 porque tiene un mayor alcance.
  e |= sx1272.setMode(1);
  Serial.print(F("Configurando el modo 10. \t\t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de cabecera explicita
```

```
  e |= sx1272.setHeaderON();
  Serial.print(F("Estableciendo Header ON. \t\t Estado: "));
  Serial.println(e, DEC);

  // Seleccionar el canal de frecuencia.
  e |= sx1272.setChannel(CH_10_868);
  Serial.print(F("Estableciendo canal CH_10_868.\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de CRC
  e |= sx1272.setCRC_ON();
  Serial.print(F("Configurando CRC ON. \t\t\t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de potencia de salida.
  e |= sx1272.setPower('M');
  Serial.print(F("Configurando nivel de potencia 'M'. \t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de direccion de nodo, establecida a 3.
  e |= sx1272.setNodeAddress(3);
  Serial.print(F("Estableciendo dirección de nodo a 3. \t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de numero de reintentos, establecida a 3.
  e |= sx1272.setRetries(3);
  Serial.print(F("Numero de reintentos establecido a 3. \t Estado: "));
  Serial.println(e,DEC);

  // Muestra de mensaje de exito
  if (e == 0){
    Serial.println(F("----------------------------------------"));
    Serial.println(F("SX1272 configurado satisfactoriamente"));
    Serial.println(F("----------------------------------------"));
  }else{
    Serial.println(F("SX1272 inicialización fallida."));
  }
}

void loop(){
```

```
  // Enviar "message1" e imprimir el resultado
  e = sx1272.sendPacketTimeoutACKRetries(8, message1);
  if(e==0){
    Serial.println(message1);
    Serial.print(F("Estado: "));
    Serial.println(e,DEC);
  }else{
    Serial.print(F("Error enviando el paquete. Estado: "));
    Serial.println(e,DEC);
  }
  delay(2500);
}
```

7.1.2.2. *Código del receptor.*

```
#include <Wire.h>

//Librerias Cooking API
#include <arduinoUtils.h>

// Librerias SX1272 y SPI
#include "arduinoLoRa.h"
#include <SPI.h>

int e;
char my_packet[100];

void setup()
{
  // Abriendo comunicaciones serie y esperando a que se abra el puerto.
  // La tasa de baudios se ha fijado a 115200 para que sea compatible con Waspmote.
  Serial.begin(115200);

  // Imprimiendo mensaje de encendido
  Serial.println(F("Modulo SX1272 y Waspmote: recibiendo paquetes con ACK y reintentos"));
  Serial.println(F("----------------------------------------"));
  Serial.println(F("Estableciendo la configuracion inicial:"));
  Serial.println(F("----------------------------------------"));

  // Encendido del módulo.
  e = sx1272.ON();
  Serial.print(F("Encendiendo el modulo. \t\t\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Configurando el modo de transmision a 1 (el de mayor alcance)
  e |= sx1272.setMode(1);
  Serial.print(F("Estableciendo el modo 1.\t\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Estableciendo cabecera explicita
  e |= sx1272.setHeaderON();
  Serial.print(F("Estableciendo cabecera explicita (Header ON).\t\t Estado: "));
  Serial.println(e, DEC);
```

```
  // Estableciendo canal de frecuencia
  e |= sx1272.setChannel(CH_10_868);
  Serial.print(F("Estableciendo canal CH_10_868.\t\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Estableciendo CRC
  e |= sx1272.setCRC_ON();
  Serial.print(F("Configurando CRC ON.\t\t\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Configurando potencia maxima de salida de la señal
  e |= sx1272.setPower('M');
  Serial.print(F("Configurando nivel de potencia 'M'.\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Configurando direccion de nodo
  e |= sx1272.setNodeAddress(8);
  Serial.print(F("Estableciendo direccion de nodo a '8'.\t\t\t Estado: "));
  Serial.println(e, DEC);

 // Mostrar mensaje de exito
  if (e == 0){
    Serial.println(F("SX1272 configurado correctamente"));
  }else{
    Serial.println(F("SX1272 inicialización fallida"));
  }

  Serial.println(F("---------------------------------------"));
  Serial.println(F("Recibiendo:"));
  Serial.println(F("---------------------------------------"));
}

void loop(void)
{
  // Recepcion de mensaje
  e = sx1272.receivePacketTimeoutACK(10000);
  if ( e == 0 )
  {
    Serial.print(F("Recibido paquete con ACK y reintentos. Estado: "));
    Serial.println(e, DEC);
```

```
    for (unsigned int i = 0; i < sx1272.packet_received.length; i++)
    {
      my_packet[i] = (char)sx1272.packet_received.data[i];
    }
    Serial.print(F("Mensaje: "));
    Serial.println(my_packet);
  }
  else {
    Serial.print(F("Recibiendo paquete con ACK y reintentos, estado:  "));
    Serial.println(e, DEC);
  }
}
```

7.2. Escenario interior.

7.2.1. Waspmote.

7.2.1.1. Código del transmisor.

```
// Libreria modulo SX1272
#include <WaspSX1272.h>

// Variables de estado
int8_t e;
char message1 [] = "varA=A&varB=B";

void setup()
{
  // Inicializar puerto USB.
  USB.ON();
  USB.println(F("Modulo SX1272 Semtech: enviando paquetes con ACK y reintentos"));

  USB.println(F("----------------------------------------"));
  USB.println(F("Estableciendo la configuracion inicial:"));
  USB.println(F("----------------------------------------"));

  // Inicializando modulo SX1272.
  sx1272.ON();
  USB.print(F("Encendiendo. Modulo: ON). \t\t Estado: "));
  USB.println(e, DEC);

  //// Opciones de configuracion ////

  // Estableciendo canal de frecuencia.
  e = sx1272.setChannel(CH_11_868);
  USB.print(F("Estableciendo canal CH_11_868.\t\t Estado: "));
  USB.println(e);

  // Seleccion de cabecera explicita.
  e = sx1272.setHeaderON();
  USB.print(F("Estableciendo Header ON. \t\t Estado: "));
  USB.println(e);

  // Configuracion de modo 10.
  e = sx1272.setMode(10);
```

```
  USB.print(F("Configurando el modo 10. \t\t Estado: "));
  USB.println(e);

  // Configuracion CRC ON
  e = sx1272.setCRC_ON();
  USB.print(F("Configurando CRC ON. \t\t\t Estado: "));
  USB.println(e);

  // Seleccion de potencia de salida de la señal
  e = sx1272.setPower('H');
  USB.print(F("Configurando nivel de potencia 'H'. \t Estado: "));
  USB.println(e);

  // Seleccion de direccion de nodo
  e = sx1272.setNodeAddress(3);
  USB.print(F("Estableciendo direccion de nodo a 3. \t Estado: "));
  USB.println(e);

  //Seleccion de numero maximo de reintentos
  e = sx1272.setRetries(3);
  USB.print(F("Numero de reintentos establecido a 3. \t Estado: "));
  USB.println(e);
  USB.println();

  if (e == 0){
    USB.println(F("----------------------------------------"));
    USB.println(F("SX1272 configurado satisfactoriamente"));
    USB.println(F("----------------------------------------"));
  }else{
    USB.println(F("SX1272 inicialización fallida."));
}
  USB.println(F("----------------------------------------"));
  USB.println(F("Enviando:"));
  USB.println(F("----------------------------------------"));
}

void loop()
{
  // Enviando paquete con reintentos, espera de ACK de respuesta.
  e = sx1272.sendPacketTimeoutACKRetries( 8, message1);
```

```
  //  Comprobando estado de envio
  if(e==0){
    USB.println(message1);
    USB.print(F("Estado: "));
    USB.println(e,DEC);
  }else{
    USB.print(F("Error enviando el paquete. Estado: "));
    USB.println(e,DEC);
  }
  delay(2500);
}
```

7.2.1.2. *Código del receptor.*

```
// La libreria del modulo SX1272
#include <WaspSX1272.h>

// Variables de estado
int8_t e;
char my_packet[100];

// Libreria del modulo WiFi PRO
#include <WaspWIFI_PRO.h>

// Socket modulo WiFi
////////////////////////////////////////
uint8_t socket = SOCKET1;
////////////////////////////////////////

// Ajustes del Punto de Acceso
////////////////////////////////////////
char ESSID[] = "MiFibra-5328";
char PASSW[] = "Rm2KwvzV";
////////////////////////////////////////

// Ajustes del servidor; en este caso, la peticion sera tipo POST.
////////////////////////////////////////
char type[] = "http";
char host[] = "pruebas.libelium.com";
char port[] = "80";
char url[]  = "getpost_frame_parser.php?";
////////////////////////////////////////

uint8_t error;
uint8_t status;
unsigned long previous;

void setup()
{
  // Inicializacion del puerto USB
  USB.ON();
  USB.println(F("Modulo SX1272 y Waspmote: recibiendo paquetes con ACK y reintentos."));
```

```
USB.println(F("---------------------------------------"));
USB.println(F("Estableciendo la configuracion inicial:"));
USB.println(F("---------------------------------------"));

// Inicializacion del modulo SX1272
sx1272.ON();

//// Opciones de configuracion: ////

// Seleccion del canal de frecuencia
e = sx1272.setChannel(CH_11_868);
USB.print(F("Estableciendo canal CH_11_868.\t\t\t\t Estado: "));
USB.println(e);

// Seleccion de cabecera explicita
e = sx1272.setHeaderON();
USB.print(F("Estableciendo cabecera explicita (Header ON).\t\t Estado: "));
USB.println(e);

// Seleccion de modo.
e = sx1272.setMode(10);
USB.print(F("Estableciendo el modo 10.\t\t\t\t Estado: "));
USB.println(e);

// Seleccion de CRC
e = sx1272.setCRC_ON();
USB.print(F("Configurando CRC ON.\t\t\t\t Estado: "));
USB.println(e);

// Seleccion de potencia de salida
e = sx1272.setPower('H');
USB.print(F("Configurando nivel de potencia 'H'.\t\t\t Estado: "));
USB.println(e);

// Seleccion de direccion de nodo
e = sx1272.setNodeAddress(8);
USB.print(F("Estableciendo direccion de nodo a '8'.\t\t\t Estado: "));
USB.println(e);
USB.println();
```

```
  USB.println(F("----------------------------------------"));
  USB.println(F("Recibiendo:"));
  USB.println(F("----------------------------------------"));
}

void loop()
{
  // Toma el tiempo actual
  previous = millis();

  // Encendido del modulo WiFi PRO
  error = WIFI_PRO.ON(socket);

  if (error == 0)
  {
    USB.println(F("Modulo WiFi encendido"));
  }
  else
  {
    USB.println(F("El modulo WiFino se ha inicializado correctamente"));
  }

  // Establecimiento de la URL
  error = WIFI_PRO.setURL( type, host, port, url );

  // Comprobacion de la respuesta
  if (error == 0)
  {
    USB.println(F("Establecimiento de la URL correcto"));
  }
  else
  {
    USB.println(F("Error llamando a la funcion 'setURL'."));
    WIFI_PRO.printErrorCode();
  }

   // Union al punto de acceso
   // Comprobando conectividad
  status =  WIFI_PRO.isConnected();
```

```
// Comprueba si el modulo esta conectado
if (status == true)
{
  USB.print(F("El modulo WiFi esta conectado al punto de acceso."));
  USB.print(F(" Tiempo(ms):"));
  USB.println(millis()-previous);
  WIFI_PRO.getIP();
  USB.println(WIFI_PRO._ip);

// Recibiendo el paquete LoRa y enviando ACK de respuesta
e = sx1272.receivePacketTimeoutACK(10000);

// Comprobar estado de recepcion
if( e == 0 )
{
  USB.println(F("\nMostrando paquete recibido: "));

  // Recepcion de paquete
  USB.print(F("Recibido paquete con ACK y reintentos. Estado: "));
  USB.println(e, DEC);

  for (unsigned int i = 0; i < sx1272.packet_received.length; i++)
  {
    my_packet[i] = (char)sx1272.packet_received.data[i];
  }
  USB.print(F("Mensaje: "));
  USB.println(my_packet);
  sx1272.showReceivedPacket();
  sx1272.showFramefromPacket();
}
else
{
  USB.print(F("\nPaquete recibido: TIMEOUT, estado: "));
  USB.println(e, DEC);
}

  // http request
  error = WIFI_PRO.post(my_packet);
```

```
    // Comprueba respuesta
    if (error == 0)
    {
      USB.print(F("HTTP POST OK. "));
      USB.println(millis()-previous);
      USB.print(F("\nRespuesta del servidor:"));
      USB.println(WIFI_PRO._buffer, WIFI_PRO._length);
    }
    else
    {
      USB.println(F("Error al llamar a la funcion 'post'"));
      WIFI_PRO.printErrorCode();
    }
  }
  else
  {
    USB.print(F("ERROR conexion WiFi"));
    USB.print(F(" Tiempo(ms):"));
    USB.println(millis()-previous);
  }
}
```

7.2.2. Arduino.

7.2.2.1. Código del transmisor.

```
#include <Wire.h>

// Librerias Cooking API
#include <arduinoUtils.h>

// Librerias SX1272 y SPI
#include "arduinoLoRa.h"
#include <SPI.h>

int e;
char message1 [] = "varA=A&varB=B";

void setup()
{
  // Abriendo comunicaciones serie y esperando al puerto para que se abra.
  // Ratio de baudios establecido a 115200 para que sea compatible con Waspmote.
  Serial.begin(115200);

  // Mostrar mensaje de encendido:
  Serial.println(F("Modulo SX1272 y Arduino: enviando paquetes con ACK y reintentos."));

  //Estableciendo la configuración inicial
    Serial.println(F("----------------------------------------"));
    Serial.println(F("Estableciendo la configuración inicial:"));
    Serial.println(F("----------------------------------------"));

  // Inicializando modulo SX1272.
  e = sx1272.ON();
  Serial.print(F("Encendiendo(Setting power ON). \t\t Estado: "));
  Serial.println(e, DEC);

  // Configuracion de modo 10.
  e |= sx1272.setMode(10);
  Serial.print(F("Configurando el modo 10. \t\t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de cabecera explicita.
```

```
  e |= sx1272.setHeaderON();
  Serial.print(F("Estableciendo Header ON. \t\t Estado: "));
  Serial.println(e, DEC);

  // Estableciendo canal de frecuencia.
  e |= sx1272.setChannel(CH_11_868);
  Serial.print(F("Estableciendo canal 11.\t\t\t Estado: "));
  Serial.println(e, DEC);

  // Configuracion CRC ON
  e |= sx1272.setCRC_ON();
  Serial.print(F("Configurando CRC ON. \t\t\t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de potencia de salida de la señal
  e |= sx1272.setPower('H');
  Serial.print(F("Configurando nivel de potencia 'H'. \t Estado: "));
  Serial.println(e, DEC);

  // Seleccion de direccion de nodo
  e |= sx1272.setNodeAddress(3);
  Serial.print(F("Estableciendo dirección de nodo a 3. \t Estado: "));
  Serial.println(e, DEC);

  //Seleccion de numero maximo de reintentos
  e |= sx1272.setRetries(3);
  Serial.print(F("Numero de reintentos establecido a 3. \t Estado: "));
  Serial.println(e,DEC);

  // Mostrando mensaje de exito
  if (e == 0){
    Serial.println(F("----------------------------------------"));
    Serial.println(F("SX1272 configurado satisfactoriamente"));
    Serial.println(F("----------------------------------------"));
  }else{
    Serial.println(F("SX1272 inicialización fallida."));
}
}

void loop(){
```

```
  // Enviando paquete con reintentos, espera de ACK de respuesta.
  e = sx1272.sendPacketTimeoutACKRetries(8, message1);
  if(e==0){
    Serial.println(message1);
    Serial.print(F("Estado: "));
    Serial.println(e,DEC);
  }else{
    Serial.print(F("Error enviando el paquete. Estado: "));
    Serial.println(e,DEC);
  }
  delay(2500);
}
```

7.2.2.2. *Código del receptor.*

No hay código del receptor, ya que en el escenario en el interior del edificio se ha utilizado el módulo WiFi PRO para la conexión con el servidor de pruebas de Libelium, el cual no es compatible con Arduino.

8. Referencias bibliográficas.

[1] Evans, D. The Internet of Things. How the Next Evolution of the Internet Is Changing Everything (2011). Recuperado de: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] Waspmote. Technical Guide. Recuperado de: http://www.libelium.com/development/waspmote/documentation/waspmote-datasheet/

[3] Waspmote. Datasheet. Recuperado de: http://www.libelium.com/development/waspmote/documentation/waspmote-datasheet/

[4] Arduino Uno Rev 3. Recuperado de: https://store.arduino.cc/arduino-uno-rev3

[5] Multiprotocol Radio Shield v2.0 for Arduino. https://www.cooking-hacks.com/documentation/tutorials/multiprotocol-shield-connect-two-xbee-connector-arduino-raspberry-pi-galileo/

[6] SX1272/73 Datasheet. Semtech. Recuperado de: https://www.semtech.com/uploads/documents/sx1272.pdf

[7] WiFi-PRO Module. Networking Guide. Recuperado de: http://www.libelium.com/development/waspmote/documentation/wifi-networking-guide/

[8] Waspmote – LoRa – 868MHz_915MHz – SX1272. Networking Guide. Recuperado de: http://www.libelium.com/development/waspmote/documentation/waspmote-lora-868mhz-915mhz-sx1272-networking-guide/

[9] Chantelau, K.; Brothuhn, R. (2009). Multimediale Client-Server Systeme. Springer Verlag.

[10] Fielding, R.; Irvine, UC.; Gettys, J.; Mogul, J.C.; Frystyk, H. et Al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. Network Working Group; 1999.