Universidad Nacional de la Patagonia San Juan Bosco

FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA

Informe Final de
Proyecto Final de Ingeniería Electrónica

"Prototipo de Red de Sensores inalámbricos empleando LoRaWAN."

Alumnos: Farías, Brian Ariel. Muñoz, Marlene Mariela.
Directores: Dr. De Marziani. Dr. Berns.

Agradecimientos de Brian

Les agradezco infinitamente este logro a mis padres: María y Diego. Gracias por estar presente en cada aspecto de mi vida (no solo en lo académico) y por alentarme a elegir esta carrera de la cual sigo maravillándome. Nada de lo que logré en estos 6 años hubiese sido posible sin ustedes, los amo y gracias. Este logro es suyo.

También, quiero destacar la enorme ayuda de la familia Garrido, la cual fue el primer empujón para poder iniciar la universidad. Ustedes igual fueron una parte fundamental para que yo pudiera llegar hasta acá. Su enorme bondad es siempre admirable y por todo esto, muchas gracias.

Gracias a mi compañera de vida, Ayelén, que me apoyó desde el primer día y se convirtió en mi principal sostén emocional.

Finalmente, agradezco a todos los amigos y familiares que formaron parte de este camino. En especial a Niko, Luci, Juan y Marlene, quienes fueron un pilar esencial en mi carrera y en mi vida. Gracias.

Agradecimientos de Marlene

A mi abuela y hermano, papá y mamá que me cuidan desde el cielo.

A mi tía, la mujer más valiente y la persona que más admiro, que me dió todo para que tuviera una infancia y una vida feliz.

A Darío, Ximena, Juan y Brian, que han sabido ser hogar cuando me encontraba lejos de casa.

A todos los profesores que estuvieron y están involucrados en mi formación, por su dedicación y predisposición.

A Hugo y Gustavo, que me enseñaron que sí, profesionalmente se alcanzan objetivos, se mejoran rendimientos, pero siempre siendo fiel a lo que uno es y a sus valores.

Resumen

Este proyecto tiene como objetivo el análisis y despliegue de una red LoRaWAN para el monitoreo de variables ambientales en entornos donde el acceso al internet es limitado.

La implementación de la red se llevó a cabo utilizando como hardware principal al Raspberry Pi 4B y a la placa de desarrollo LoRa32u4 II v1.3. La red cuenta con 3 nodos encargados de enviar la información recopilada de los sensores a un gateway LoRa, el cual luego redirige a un servidor. Los elementos de la red se hospedaron en un mismo hardware considerando la simplicidad de la misma. Se realizó una investigación de los protocolos involucrados en el sistema a realizar para su posterior explicación en el presente informe. Además, se realizaron análisis sobre los resultados obtenidos referentes al alcance de la comunicación, al consumo eficiente de energía y autonomía de los nodos, y a las posibles aplicaciones del sistema presentado.

Palabras clave: Redes LPWAN, LoRaWAN, ChirpStack, IoT

Abstract

This project aims to analyze and deploy a LoRaWAN network for monitoring environmental variables in areas with limited internet access. The network implementation was built upon a Raspberry Pi 4B and the LoRa32u4 II v1.3 development board as the main hardware. The network comprises three nodes responsible for transmitting sensor data to a LoRa gateway, which then forwards it to a server. To simplify the system, all network components were hosted on the same hardware. A detailed investigation of the protocols involved in the system was conducted and explained in this report. Additionally, analyses were performed on the obtained results, focusing on communication range, energy-efficient operation and node autonomy, as well as potential applications of the proposed system.

Keywords: LPWAN networks, LoRaWAN, ChirpStack, IoT

Índice General

Índice de Figuras

Índice de Tablas

Glosario

ABP	Activation By Personalization
ACK	Acknowledge
ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
API	Application Programming Interface
BW	Bandwidth
CR	Coding Rate
CRC	Cyclic Redundancy Check
CSS	Chirp Spread Spectrum
DNS	Domain Name System
DR	Data Rate
EIRP	Equivalent Isotropically Radiated Power
EUI	Extended Unique Identifier
FECR	Forward Error Correction Rate
FSK	Frequency Shift Keying
GPIO	General Purpose Input Output
gRPC	Google Remote Procedure Call
IC	Integrated Circuit
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific, Medical
LR-FHSS	Long-Range Frequency Hopping Spread Spectrum
MAC	Medium Access Control
MIC	Message Integrity Code
MQTT	Message Queuing Telemetry Transport
OS	Operating System
OTA	Over The Air
PIR	Passive Infrared Sensor
QoS	Quality of Service
RF	Radio Frequency
RFU	Reserved for Future Use
RPi	Raspberry Pi
RSSI	Received Signal Strength Indicator
SF	Spreading Factor
SNR	Signal Noise Ratio
SPI	Serial Peripheral Interface
SQL	Structured Query Language
TCP	Transmission Control Protocol
TDOA	Time Difference of Arrival
ToA	Time on Air
TTN	The Things Network
UDP	User Datagram Protocol
USB	Universal Serial Bus
WDT	Watchdog Timer

1. Introducción

1.1 Descripción del Problema

La expansión tecnológica en el área industrial y la reciente tendencia IoT (Internet of Things) incentivó el desarrollo de soluciones de baja potencia y largo alcance para la transmisión de datos tales como, LoRaWAN, ZigBee, SigFox, Low Power LTE, etc. Estas soluciones buscan implementar de forma masiva dispositivos en zonas remotas disminuyendo la cantidad de intervenciones en los mismos, basándose en protocolos ligeros pero seguros que, junto a estrategias de ahorro de energía, permiten el despliegue de redes de amplio alcance y bajo mantenimiento.

Este proyecto se enfoca en el protocolo de comunicación LoRaWAN y su modulación LoRa y cómo estos logran transmitir información de sensores en áreas remotas. Además, se implementa un prototipo de red de sensores, presentando resultados y analizando la factibilidad del protocolo como solución en un entorno simulado.

1.2 LoRaWAN y Tecnologías Afines

LoRa (abreviación de Long Range) es una técnica de modulación de espectro ensanchado derivada de la tecnología CSS (chirp spread spectrum) propiedad de Semtech [1]. Mientras que LoRaWAN es un estándar de comunicación IoT administrado y mantenido por LoRa Alliance, una asociación sin fines de lucro [2].

Actualmente, el protocolo LoRaWAN es utilizado ampliamente para la interconexión de sensores y actuadores, especialmente en zonas donde el acceso a internet es restringido. Esto se debe a las características de la tecnología LoRa, de las que se destacan principalmente: largo alcance, bajo consumo y baja tasa de transmisión de datos [3].

LoRa utiliza la técnica de modulación de espectro ensanchado, lo que permite una comunicación de mayor alcance y con alta inmunidad a interferencias. Su frecuencia de operación se encuentra entre los 400 y 900 MHz. En Argentina, la banda permitida para estas conexiones es la banda ISM de 915 MHz y con un máximo de 36 dBm de potencia de transmisión [4]. Entre las ventajas y desventajas de la red, podemos destacar las siguientes:

Ventajas:

- Largo alcance: 10 km a 20 km.
- Alta inmunidad a interferencias.
- Encriptación de la información AES 128 bits.
- Conexión múltiple: se pueden conectar varios dispositivos en una sola red, ideal para aplicaciones que requieren la recopilación de datos de distintos dispositivos.
- Bajo costo.
- Bajo consumo.

Desventajas:

- Baja tasa de transferencia de datos.
- Bajo payload debido a la limitación del tiempo en el aire (ToA).

Existen distintos protocolos orientados a las redes IoT (como por ejemplo ZigBee, Bluetooth Low Energy, etc.), cada uno consta de un conjunto de ventajas y desventajas que justifican su uso. La elección de LoRaWAN para este proyecto se basa en el ahorro energético, mayor alcance, simplicidad de conexión y bajo costo de un sistema basado en LoRaWAN [5].

1.3 Objetivo General del Proyecto

El objetivo general del proyecto es la implementación de una red de comunicación de larga distancia basada en el protocolo LoRaWAN, para la transmisión y recepción de información de monitoreo de variables ambientales en zonas de difícil acceso.

1.3.1 Objetivos Específicos

Se definen los siguientes objetivos específicos:

1. Investigar y comprender los principios y fundamentos de la tecnología LoRaWAN y su aplicación en la transmisión de información de monitoreo de variables ambientales en zonas remotas.
2. Diseñar la arquitectura de la red, considerando los requerimientos de cobertura, capacidad y eficiencia energética para la transmisión de datos de monitoreo remoto.
3. Seleccionar y adquirir los dispositivos y componentes necesarios para la implementación de la red, incluyendo los nodos de sensores, gateways y otros elementos de infraestructura.
4. Desarrollar y adaptar el software y los protocolos necesarios para la comunicación entre los nodos de sensores y los gateways LoRaWAN, asegurando la integridad y confiabilidad de los datos transmitidos.
5. Implementar la red en el entorno específico de zonas remotas, considerando aspectos como la topología de la red, la ubicación y configuración del gateways, y la cobertura requerida para el monitoreo de las variables ambientales.
6. Realizar pruebas y validación de la red LoRaWAN implementada, evaluando su desempeño en términos de alcance, capacidad de transmisión, consumo de energía y confiabilidad de la comunicación.
7. Integrar los nodos de sensores de monitoreo de variables ambientales en la red, asegurando su correcto funcionamiento y la transmisión de los datos de forma eficiente.
8. Establecer un sistema de gestión y análisis de los datos recolectados por la red, que permita visualizar y extraer información relevante sobre las variables ambientales monitoreadas en zonas remotas.

1.4 Esquema General del Sistema a Desplegar

La red contará con tres nodos finales, encargados de transmitir la información sensada del exterior, un gateway, que será el otro extremo de la comunicación LoRa, un servidor de red y un servidor de aplicación, cuya función es servir como red de retorno y de almacenamiento de datos.

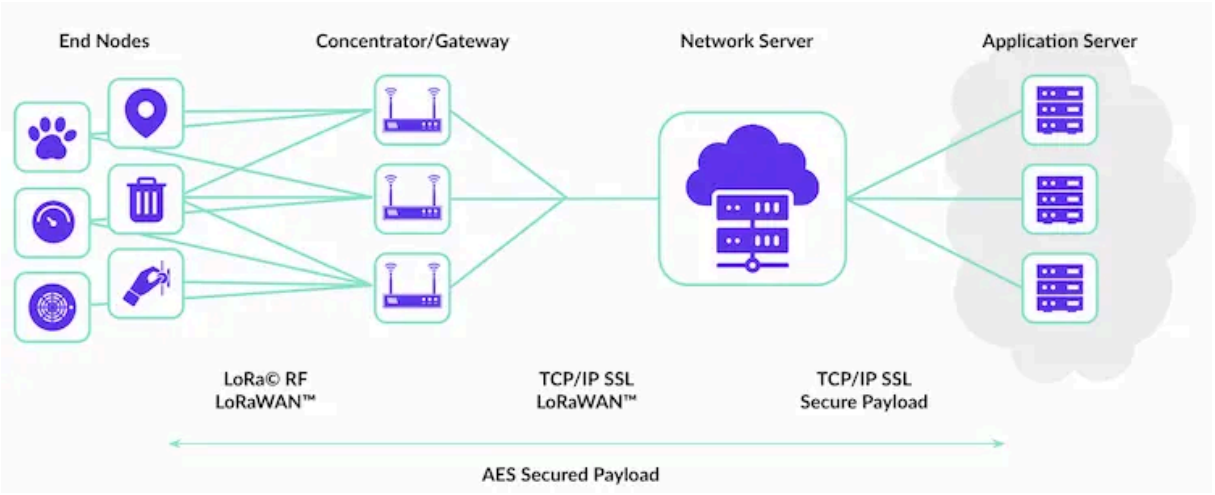


Figura 1: Diagrama genérico de una red LoRaWAN.

2. Fundamento Teórico

2.1 Modulación LoRa

Como se mencionó anteriormente, LoRa es una tecnología propiedad de Semtech. Esta técnica de modulación utiliza señales chirp como símbolo en la transmisión de datos. Una señal chirp es una señal que varía su frecuencia (de forma lineal en este caso), como se muestra en la siguiente figura.

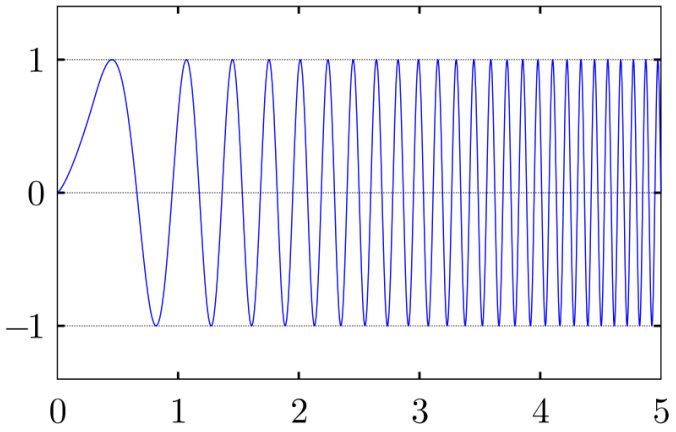


Figura 2: Señal chirp en un gráfico amplitud vs tiempo.

Semtech definió los siguientes conceptos para caracterizar una señal up-chirp (chirp con frecuencia ascendente) o down-chirp (chirp con frecuencia descendente) [6]:

- Factor de Dispersión (SF): es una variable que representa la cantidad de bits puros que transmite un símbolo. Cada símbolo puede representar 2^{SF} valores.
- Ancho de banda de modulación (BW): es el ancho de banda de la señal chirp.
- Chip: Un chip es la unidad de tiempo fundamental dentro de la señal chirp. Existen 2^{SF} chips a lo largo de una señal chirp.

Con estos valores se puede definir la tasa de transmisión de bits R_b.

$$R_b = SF * \frac{BW}{2^{SF}}$$

Es de interés observar que la tasa de baudios, $R_S = \frac{BW}{2^{SF}}$, es proporcional al ancho de banda de la señal chirp y que la misma se divide en dos por cada bit que se agrega al símbolo. La tasa de chips se mantiene constante por definición, ya que $R_C = R_S * 2^{SF} = \frac{BW}{2^{SF}} * 2^{SF} = BW$, es decir, la cantidad de chips en una señal chirp puede variar pero la duración de los chips se mantiene constante (para un BW de transmisión dado) [6]. Cuando se selecciona un factor de dispersión mayor, el RSSI del enlace se ve mejorado debido al aumento de energía por símbolo. Sin embargo, el costo de esto es una menor tasa de datos y mayor consumo de energía [7].

Finalmente, el símbolo LoRa se codifica usando la señal chirp caracterizada con las propiedades mencionadas, más un desplazamiento de frecuencia inicial que indica el dato en sí. En [7], se describe el símbolo LoRa representado análiticamente como

$$s(t) = exp(j2\pi \int_0^t [(\beta x + \gamma_n)_{modBW} - \frac{BW}{2}]dx)$$

Donde β es la tasa de variación de la frecuencia y γ_n es el desplazamiento en frecuencia de la señal chirp cíclica. A partir del dato a transmitir ($m_n \in [0, 1, ..., 2^{SF} - 1]$), se obtiene el desplazamiento inicial en frecuencia. Se expresa lo mencionado en la siguiente igualdad: $\gamma_n = m_n * \frac{R_C}{\beta} = m_n * BW * R_S/BW = m_n * R_S$. Esto indica que los saltos de frecuencia ocurren en múltiplos de la tasa de símbolo.

A continuación, se sintetizan los conceptos de generación de un símbolo LoRa en un gráfico de dos señales chirps codificadas con diferentes datos [7].

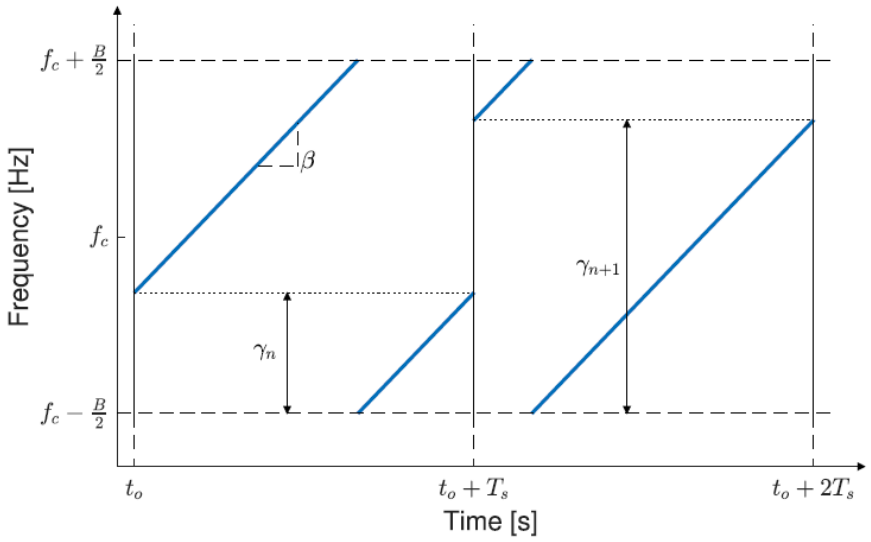


Figura 3: Ejemplo de dos símbolos up-chirps LoRa modulados.

2.2 Protocolo LoRaWAN

Por motivos de simplicidad y debido a la limitada variedad de software libre compatible con la versión 1.1, se profundizará en la versión 1.0.3 del protocolo y luego se mencionarán los cambios significativos en la actual versión 1.1.

LoRa Alliance separa la definición del protocolo en tres documentos, el primero se refiere a la capa física y los parámetros de la modulación según la región en la que se implementa, el segundo se refiere a la capa de enlace de datos, y el último se refiere a la interfaz de backend, es decir, la comunicación entre los servidores de red, los servidores de aplicación y los servidores de unión [8]. Se debe destacar que la versión 1.0.3 no tiene en cuenta el documento de la interfaz de backend y que este último hace referencia a muchos cambios introducidos por la versión 1.1.

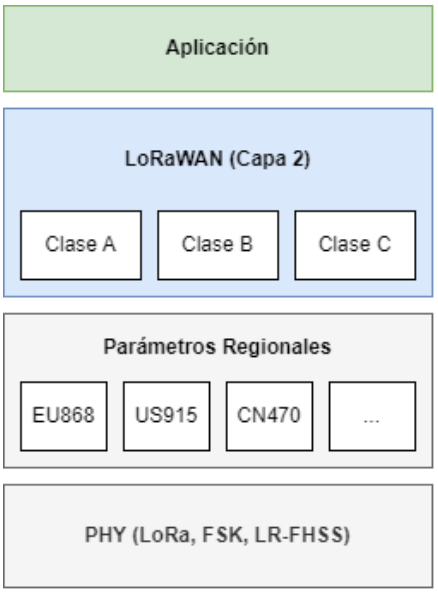


Figura 4: Pila del protocolo LoRaWAN.

2.2.1 Capa Física

LoRa Alliance establece una tabla de referencia para la implementación de los parámetros regionales adecuados según el país [9]. En el caso de Argentina, el plan recomendado es el AU915-928, el cual será detallado a continuación.

La banda del plan AU915-928 se divide en los siguientes canales [9]:

- Subida – 64 canales, numerados del 0 al 63, utilizando LoRa 125 kHz BW, variando desde DR0 hasta DR5, utilizando tasa de codificación 4/5, iniciando en 915,2 MHz e incrementando linealmente en 200 kHz hasta 927,8 MHz.
- Subida – 8 canales, numerados del 64 al 71 utilizando LoRa 500 kHz BW con DR6 o LR-FHSS 1,523 MHz BW con DR7, iniciando en 915,9 MHz e incrementando linealmente en 1,6 MHz hasta 927,1 MHz.
- Bajada – 8 canales, numerados del 0 al 7, utilizando LoRa 500 kHz BW variando desde DR8 hasta DR13, iniciando en 923,3 MHz e incrementando linealmente en 600 kHz hasta 927,5 MHz.

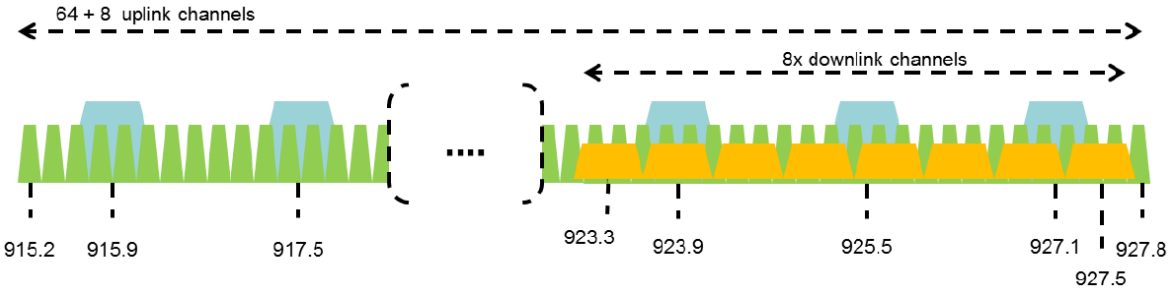


Figura 5: Canales de frecuencia del plan AU915-928.

Se entiende por subida (uplink) al proceso de enviar un paquete desde un dispositivo final al servidor y, por bajada (downlink) al proceso de enviar un paquete desde el servidor a un dispositivo final.

LR-FHSS (Long Range Frequency Hopping Spread Spectrum) es otro método de modulación soportado por el protocolo LoRaWAN (además de LoRa), el mismo se encuentra fuera del alcance de este informe y por lo tanto, a pesar de ser mencionado, no se profundizará sobre él.

Los valores de DR (data rate) mencionados anteriormente hacen referencia a las configuraciones de SF (factor de dispersión) y de BW (ancho de banda). A continuación se presentan los valores de DR y sus respectivas configuraciones de modulación.

Data Rate	Configuración	Tasa indicativa de bits físicos [bits/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF8 / 500 kHz	12500
7	LR-FHSS CR1/3: 1.523 MHz BW	162
8	LoRa: SF12 / 500 kHz	980
9	LoRa: SF11 / 500 kHz	1760
10	LoRa: SF10 / 500 kHz	3900
11	LoRa: SF9 / 500 kHz	7000
12	LoRa: SF8 / 500 kHz	12500
13	LoRa: SF7 / 500 kHz	21900
14	RFU (Reservado para Uso Futuro)	

Tabla 1: Tasa de datos (DR) del plan AU915-928.

En [9] además se recomienda un EIRP (potencia isotrópica radiada equivalente) de +30 dBm, siendo el máximo posible en Argentina, +36 dBm, para esta técnica de modulación [4].

La estructura física de un paquete LoRa consiste en un preámbulo, una palabra de sincronización, una cabecera con un CRC de cabecera (verificación por redundancia cíclica), el payload del paquete con un CRC de payload opcional y el CRC propio del paquete (presente únicamente en paquetes de uplink). La cabecera física (PHDR) indica la longitud del payload en bytes, la presencia de un CRC de payload y la tasa de corrección de errores hacia adelante (FECR).

Tamaño	8 símbolos	4,25 símbolos	8 símbolos		L bytes (definido en PHDR)	2 bytes
Estructura del paquete	Preámbulo	Palabra de sincronización	PHDR	PHDR_CRC	PHYPayload	CRC (solo uplink)

Tabla 2: Estructura física de un paquete LoRa.

El tamaño máximo de MACPayload, descrito en la siguiente sección, es específico de la región en la que se emplea y depende del UplinkDwellTime. El valor UplinkDwellTime es un flag que indica limitaciones en el tiempo en el aire (ToA) máximo debido a regulaciones locales. Cuando este valor es 1, indica que el tiempo máximo de transmisión activa por uplink no puede superar los 400ms. Sin embargo cuando este valor es 0, se entiende que no hay limitaciones por regulaciones locales. Para la región AU915, los tamaños máximos de MACPayload dependiendo del DR y UplinkDwellTime se observan en la siguiente tabla.

Data Rate	**UplinkDwellTime=0**		**UplinkDwellTime=1**	
	M	**N**	**M**	**N**
0	59	51	N/A	N/A
1	59	51	N/A	N/A
2	59	51	19	11
3	123	115	61	53
4	250	242	133	125
5	250	242	250	242
6	250	242	250	242
7	58	50	58	50
8	61	53	61	53
9	137	129	137	129
10	250	242	250	242
11	250	242	250	242
12	250	242	250	242
13	250	242	250	242
14:15	No definido		No definido	

Tabla 3: Tamaño máximo de payload.

El valor M indica el número de bytes máximos del MACPayload mientras que el valor N indica el número de bytes máximos del payload de aplicación en el caso de no poseer campo de FOpts.

2.2.2 Capa de Enlace de Datos

En el siguiente apartado se describirán los procedimientos más relevantes descritos en [10] referente a la capa de enlace de datos.

Los dispositivos finales se pueden describir según su comportamiento frente a las comunicaciones con el servidor. El protocolo define tres clases que los dispositivos finales pueden adoptar:

- Dispositivos finales bi-direccionales (Clase A): Los dispositivos finales clase A permiten la comunicación en ambos sentidos, en donde a cada uplink le siguen dos ventanas de recepción. Cualquier downlink que se deseara enviar fuera de estos tiempos, deberá esperar a la siguiente ventana de recepción. Esta clase posee la ventaja de ser la de menor consumo de energía debido a los pocos estados de actividad que presenta. Es útil para sistemas en donde se requieran downlinks únicamente luego de un uplink.
- Dispositivos finales bi-direccionales con ventanas de recepción programadas (Clase B): Los dispositivos finales clase B poseen las mismas características que uno clase A, con la adición de más ventanas de recepción en tiempos programados. Para poder abrir sus ventanas de recepción en los tiempos establecidos entre ambos extremos, los gateways deben enviar balizas sincronizadas de forma periódica.
- Dispositivos finales bi-direccionales con ventanas de recepción completas (Clase C): Los dispositivos finales clase C mantienen sus ventanas de recepción abiertas casi todo el tiempo, cerrándolas únicamente durante la transmisión (uplink).

El tiempo entre el final de una transmisión y el inicio de la primera ventana de recepción se denomina RECEIVE_DELAY1 y RECEIVE_DELAY2 para el caso de la segunda ventana de recepción.

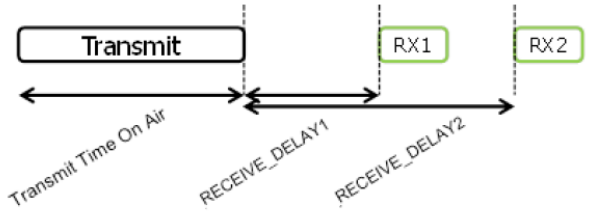


Figura 6: Tiempos de las ventanas de recepción de un dispositivo final.

En [9], se establecen valores por defecto recomendados aplicables a todas las regiones para RECEIVE_DELAY1, RECEIVE_DELAY2, JOIN_ACCEPT_DELAY1 y JOIN_ACCEPT_DELAY2. JOIN_ACCEPT_DELAY es un parámetro equivalente a RECEIVE_DELAY pero es utilizado únicamente durante la activación. Luego, el servidor puede solicitar modificaciones en los valores de retraso de apertura de ventanas utilizando comandos MAC descritos en un apartado posterior. En los parámetros regionales de AU915 también se especifica el DR de los paquetes de downlink en las ventanas de RX1 y RX2. El DR de downlink de la ventana RX1 es una función del DR de uplink y de un valor llamado RX1DROffset, la siguiente tabla define estos valores. Mientras que el DR de downlink de la ventana RX2 es fijo y se define como DR8.

Data Rate de upstream	Data Rate de downstream					
RX1DROffset	**0**	**1**	**2**	**3**	**4**	**5**
DR0	DR8	DR8	DR8	DR8	DR8	DR8
DR1	DR9	DR8	DR8	DR8	DR8	DR8
DR2	DR10	DR9	DR8	DR8	DR8	DR8
DR3	DR11	DR10	DR9	DR8	DR8	DR8
DR4	DR12	DR11	DR10	DR9	DR8	DR8
DR5	DR13	DR12	DR11	DR10	DR9	DR8
DR6	DR13	DR13	DR12	DR11	DR10	DR9
DR7	DR9	DR8	DR8	DR8	DR8	DR8

Tabla 4: Mapeo del DR del downlink de RX1.

La subtrama PHYPayload, introducida en la sección de la capa física, se desencapsula según la siguiente figura.

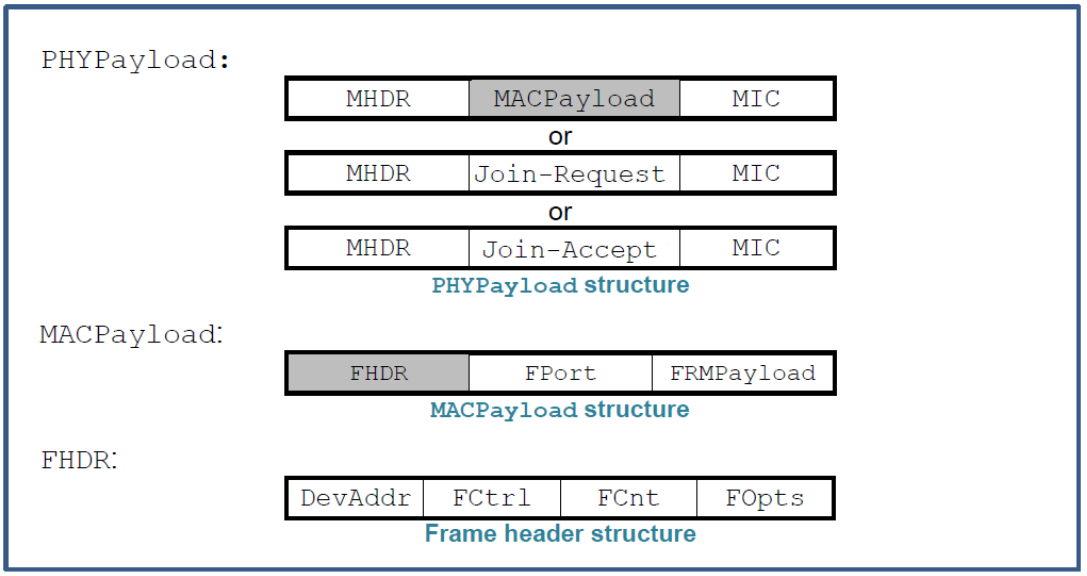


Figura 7: Elementos de una trama LoRaWAN.

En el MHDR (MAC Header) se describe el tipo de trama y, en el caso de ser una trama de Join-Request o Join-Accept, el formato del MACPayload se modifica. El MIC (Message Integrity Code) se calcula con los valores del payload y su función es verificar la integridad del mensaje.

Bits	[7:5]	[4:2]	[1:0]
MHDR	FType	RFU	Major

Tabla 5: Formato del MHDR

FType	Descripción
000	Join-Request
001	Join-Accept
010	uplink no confirmado
011	downlink no confirmado
100	uplink confirmado
101	downlink confirmado
110	RFU
111	Propietario

Tabla 6: Tipos de trama MAC.

Las tramas con FType=000 y FType=001 son tramas utilizadas en la activación OTA (Over The Air) y serán explicadas en detalle en un apartado posterior.

Las tramas confirmadas, ya sea de uplink o de downlink, deben ser reconocidas por el receptor respondiendo con el flag de ACK=1. Cuando el servidor es el que recibe una trama confirmada, debe enviar una trama de reconocimiento (ACK=1) en una de las ventanas de recepción abiertas por el dispositivo que envió el uplink confirmado. Cuando es el dispositivo final el que recibe una trama confirmada, puede elegir entre enviar una trama de reconocimiento sin payload o esperar al próximo uplink que tenga programado y agregarle el flag de ACK. Esto da libertad al dispositivo final para implementar la estrategia más conveniente que reduzca al mínimo el consumo de energía.

Las tramas propietarios pueden ser usadas para implementar formatos no estándar utilizados únicamente entre dispositivos que conocen estas extensiones de formato.

El campo Major se usa para indicar la versión del protocolo utilizada, en la versión 1.0.4 se encuentra definido solamente el valor 00 = LoRaWAN R1. El resto de combinaciones de bits (01,10 y 11) están reservadas para usos futuros.

El campo opcional FPort (Port Field) es un valor comprendido entre 0 y 255. Una trama con FPort=0 indica que la trama contiene únicamente comandos MAC. Alternativamente, los valores de FPort desde 1 hasta 223 son específicas según la aplicación, es decir, permiten el agrupamiento de distintos dispositivos finales en la capa de aplicación. Esto es análogo a la función de los puertos en la capa de transporte de la pila TCP/IP. Los valores 224 ... 255 están reservados para uso y ubicación de LoRa Alliance.

El valor DevAddr, compuesto por 4 bytes, indica la dirección del dispositivo final del cual proviene la trama (en el caso del uplink), o al cual va dirigida la trama (en el caso del downlink). En el campo FCtrl se encuentran los flags de ACK, ADR (Adaptative Data Rate), Clase B (solo en el uplink), FPending (indica tramas en cola de downlink, solo en downlink), FOptsLen (indica la longitud del campo FOpts en bytes). El campo opcional FOpts se utiliza para adjuntar comandos MAC en una trama de datos, por lo tanto, si FPort=0, el campo FOpts debe estar ausente y FOptsLen debe ser igual a 0.

Por cada dispositivo final, existen dos contadores de tramas que el dispositivo debe mantener actualizados, FCntUp se incrementa por cada uplink y FCntDown se incrementa por cada downlink. Estos contadores son de 4 bytes, mientras que FCnt debe contener los 2 bytes menos significativos del contador correspondiente.

ADR (Adaptive Data Rate) es un mecanismo que permite que los dispositivos finales utilicen la mayor tasa de datos con la menor potencia de transmisión siempre que sea posible. Esta negociación requiere que el dispositivo final y el servidor tengan sus flags de ADR en 1 para que, luego, el servidor pueda enviar comandos MAC con los parámetros de optimización de tasa de datos. Si el dispositivo final tiene su bit ADR en 0, aún puede aceptar algunos de los parámetros recomendados por el servidor, mientras que si lo tiene en 1, debe aceptar o rechazar el conjunto completo de parámetros que le son recomendados.

Si el campo FRMPayload está presente, este debe ser encriptado utilizando el algoritmo de encriptación AES-128. Si FPort=0, la clave de encriptación utilizada es la NwkSKey (Network Session Key), mientras que si FPort=1…255, se utiliza la clave AppSKey (Application Session Key). El MIC se calcula luego de la encriptación del payload. Cómo se obtienen las claves NwkSKey y AppSKey se explica en el apartado de activación de los dispositivos finales.

Una vez finalizado el proceso de activación, los dispositivos finales deben almacenar la siguiente información: la dirección del dispositivo (DevAddr), un identificador de aplicación (AppEUI), una clave de sesión de red (NwkSKey) y una clave de sesión de aplicación (AppSKey).

Bits	[31 … 25]	24 … 0
DevAddr	NwkID	NwkAddr

Tabla 7: Campos del DevAddr.

El campo NwkID se usa como identificador del servidor de red entre distintos servidores en el modo roaming y se deriva de los siete bits menos significativos del NetID. NwkAddr identifica al dispositivo final dentro de la red y son asignados arbitrariamente por el servidor de red.

Existen dos métodos de activación de dispositivos finales, estos son: Activación OTA (Over The Air) y Activación por Personalización.

En la activación OTA, el dispositivo final debe contar con la siguiente información antes de iniciar el proceso de activación: un identificador de dispositivo final único global (DevEUI), el identificador de aplicación (AppEUI) y una clave AES-128 (AppKey). Los EUI son identificadores únicos en el espacio de direcciones EUI64 (64-bit Extended Unique Identifier) de la IEEE [11].

Desde el punto de vista del dispositivo final, el proceso de activación consta de dos comandos MAC, el Join-Request por parte del dispositivo final y el Join-Accept por parte del servidor. El Join-Request es enviado sin encriptación.

Size (bytes)	8	8	2
Join-Request Payload	AppEUI	DevEUI	DevNonce

Tabla 8: Formato del payload de un Join-Request.

El DevNonce es un valor aleatorio generado por el dispositivo final. El servidor de red debe seguir el rastro de los DevNonce utilizados y descartar los frames que contengan algún DevNonce utilizado anteriormente.

Size (bytes)	3	3	4	1	1	16 (opcional)
Join-Request Payload	AppNonce	NetID	DevAddr	DLSettings	RXDelay	CFList

Tabla 9: Formato del payload de un Join-Accept.

AppNonce es un valor único por cada Join-Accept que provee el servidor de red. Este valor es utilizado por el dispositivo final para derivar las claves de sesión NwkSKey y AppSKey. DLSettings contiene configuraciones referentes a los downlinks y RXDelay contiene los valores RX_DELAY1 y RXDELAY2 mencionados anteriormente. CFList es una lista opcional de parámetros de red. El Join-Accept se encripta con la clave de aplicación (AppKey).

En el caso de activación por personalización (ABP), las claves de sesión y el DevAddr son almacenados en el dispositivo final antes de ser alimentado. Este método ahorra el proceso de unión a la red, teniendo toda la información para interactuar con la red desde el momento en el que el dispositivo final es encendido. Como contraparte, información sensible es almacenada directamente en la memoria del dispositivo final. Es por esto que la norma recomienda el uso de claves de sesión únicas por cada dispositivo final, con el fin de evitar comprometer la seguridad de toda la red en el caso de filtrado de claves.

La habilitación de la clase B en un dispositivo final consta de los siguientes pasos:

- La capa de aplicación del dispositivo final solicita a la capa de enlace de datos habilitar la clase B. La capa de enlace inicia el proceso de búsqueda de balizas (enviadas periódicamente por los gateways).
- Una vez el dispositivo final encuentra una baliza, puede habilitar la clase B.
- Una vez habilitada la clase B en el dispositivo final, el flag de clase B contenido en el campo FCtrl debe mantenerse en 1.
- La capa MAC debe programar de forma autónoma ventanas de recepción para las siguientes balizas y los siguientes ping slots (ventanas de recepción clase B).
- Si no se reciben balizas durante un tiempo establecido, el dispositivo final debe cambiar a 0 su flag de clase B.

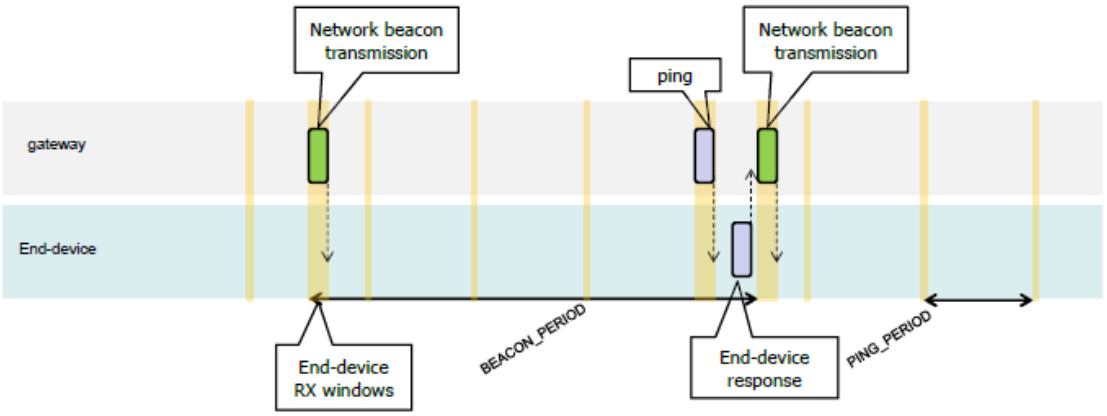


Figura 8: Ejemplo de recepción de balizas, ventanas de recepción y ping slots.

En este ejemplo se pueden observar las balizas en verde enviadas de manera periódica, mientras que los ping slots pueden ser usados o no por el servidor para enviar downlinks. Se puede observar también que, antes de la segunda baliza, el servidor utilizó un ping slot para enviar un ping (downlink durante un ping slot).

En el caso de que una o varias balizas no hayan sido recibidas de forma satisfactoria por causas externas, el dispositivo final debe mantener su secuencia de ping slots pero aumentando el ancho de estas ventanas. Esto contrarresta el error de clock drifting (desvío del reloj) debido a la falta de balizas. Una vez se recibe una baliza antes del tiempo máximo permitido, el ancho de la ventana debe volver a su duración predeterminada.

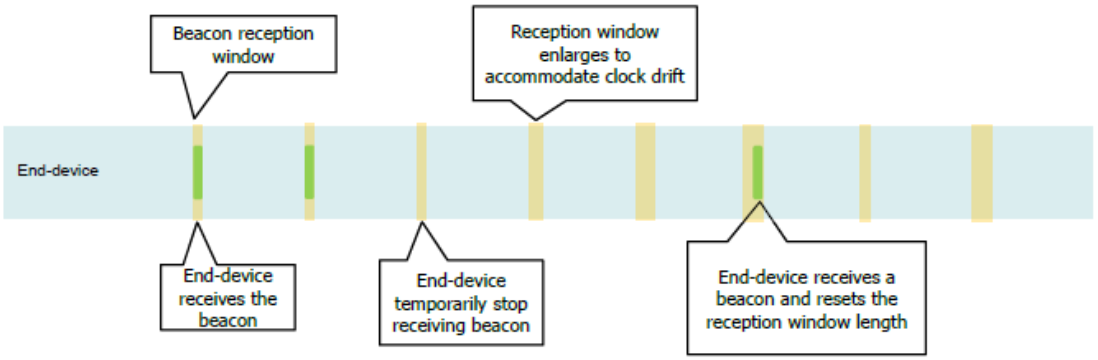


Figura 9: Operación temporal sin balizas.

Un dispositivo clase C tiene las mismas ventanas de recepción que un dispositivo clase A, con el agregado de ventanas de recepción C (RXC), las cuales se encuentran abiertas siempre que el dispositivo no esté transmitiendo o tenga una ventana de recepción tipo A abierta (RX1 y RX2).

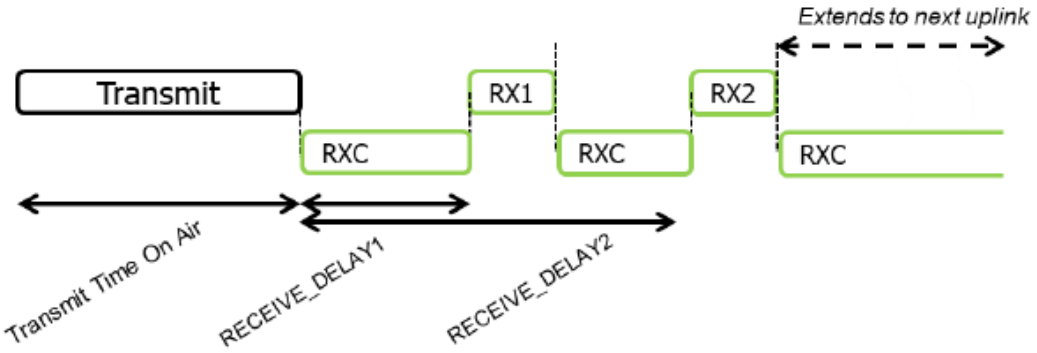


Figura 10: Tiempos de las ventanas de recepción de un dispositivo final clase C.

Si un dispositivo final recibe un paquete en una de sus ventanas de recepción tipo A mientras se encuentra demodulando un paquete que llegó en la ventana RXC, este debe descartar el paquete proveniente de la ventana RXC y atender al paquete proveniente de RX1 o RX2. El servidor no puede enviar comandos MAC (por FOpts o por FPort=0) a un dispositivo final mediante su ventana RXC. Si un dispositivo final recibe un paquete con comandos MAC en RXC, debe descartarlo.

2.2.3 Interfaz Backend

En [12], se describe cómo interactúan el servidor de red (NS), el servidor de aplicación (AS), el servidor de unión (JS) y el dispositivo final (ED), la función que cumple cada uno y los pasos que ejecuta cada elemento de la red en el proceso de activación del dispositivo final. El modelo de referencia de red utilizado se ilustra en la figura 10. El proceso de activación para el caso de dispositivos finales en modo roaming se encuentra fuera del alcance de este informe y no será detallado en la presente sección.

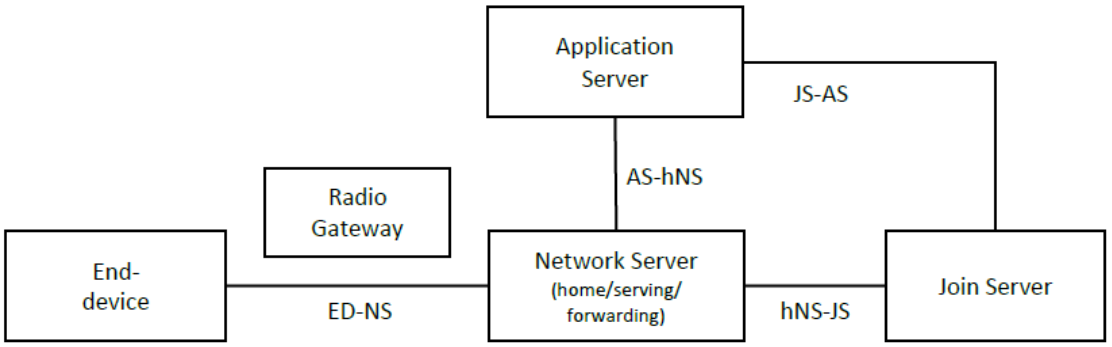


Figura 11: Modelo de referencia de una red LoRaWAN. Dispositivo final en la red objetivo (home network).

Dispositivo final: Los dispositivos finales pueden ser actuadores o sensores. Estos se conectan de forma inalámbrica a una red LoRaWAN mediante los gateways. La capa de aplicación de estos dispositivos se conecta a un servidor de aplicación específico en la nube.

Gateway: Los gateways reenvían todos los paquetes radio recibidos hacía una red de retorno IP. Los gateways operan enteramente en la capa física. Su rol es decodificar los paquetes recibidos por aire y reenviarlos sin procesar al servidor de red. De forma análoga,

para los downlinks, los gateways ejecutan solicitudes provenientes del servidor de red sin realizar ninguna interpretación del payload.

Servidor de red: Las funciones de un servidor de red son:

- Verificación de dirección de dispositivo final.
- Autenticación de trama y verificación de contadores de trama.
- Reconocimientos (ACK).
- Adaptación de la tasa de datos (ADR).
- Responder a todas las solicitudes de la capa MAC del dispositivo final.
- Reenviar uplinks con payload de aplicación al servidor de aplicación apropiado.
- Gestionar cola de downlinks provenientes de los servidores de aplicación.
- Reenviar mensajes de Join-Request y Join-Accept entre los dispositivos finales y los servidores de unión.

Servidor de unión: El servidor de unión gestiona el proceso de activación OTA. Este cuenta con la información necesaria para procesar las tramas de Join-Request y generar las tramas de Join-Accept. También deriva las claves de sesión de red y de aplicación y comunica a los servidores sus respectivas claves de sesión. Para cumplir con tal propósito, el servidor de unión debe contar con la siguiente información referente a cada dispositivo final:

- DevEUI.
- AppKey.
- Identificador del servidor de red objetivo (home).
- Identificador del servidor de aplicación.
- Versión de LoRaWAN del dispositivo final.

Servidor de aplicación: El servidor de aplicación maneja todos los payloads de la capa de aplicación de los dispositivos finales asociados y provee servicios de capa de aplicación al usuario final. Además, genera el payload de downlink específicos de aplicación.

El proceso de activación OTA desde el lado de los servidores se puede resumir en los siguientes ocho pasos:

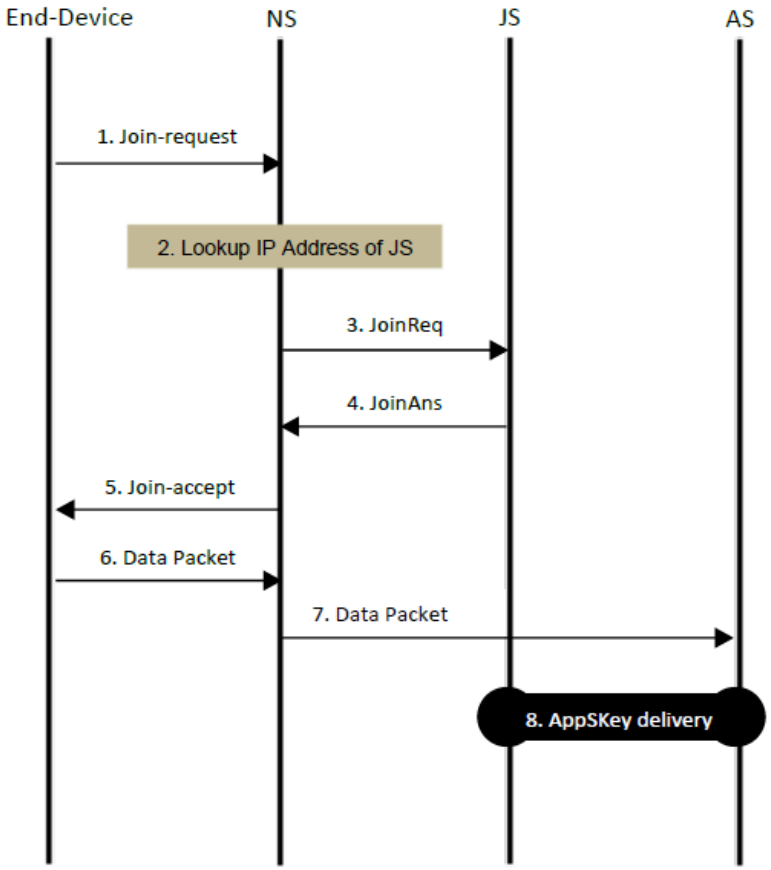


Figura 12: Flujo de mensajes en el proceso de activación OTA.

Paso 1: El dispositivo final envía un frame de Join-Request.

Paso 2: El servidor de red utiliza un servidor DNS para obtener la dirección IP del servidor de unión mediante el JoinEUI (AppEUI) provisto por el dispositivo final.

Paso 3: El servidor de red envía el Join-Request añadiendo también Versión MAC, DevEUI, DevAddr, DLSettings, RxDelay, y opcionalmente CFList.

Paso 4: El servidor de unión debe responder con un paquete Join-Ans que, en el caso de ser exitosa la solicitud de unión, debe contener Result=Success, el payload del Join-Accept, la clave de sesión de red, y la clave de aplicación encriptada. En el caso de ser fallido el intento de unión a la red, el payload del Join-Ans contendrá Result=UnknownDevEUI, Result=MICFailed, Result=FrameReplayed o Result=JoinReqFailed, en los casos donde el servidor de unión no reconozca al dispositivo final, donde no se pueda comprobar la integridad del mensaje, donde DevNonce se utilice más de una vez o donde exista algún otro error respectivamente.

Paso 5: El servidor de red reenvía el payload recibido en un frame de Join-Accept. El dispositivo final deriva las claves de sesión a partir del contenido del Join-Accept.

Paso 6: Cuando el servidor de red recibe un uplink proveniente del dispositivo final cuyo destino es el servidor de aplicación, el servidor de red puede agregar al paquete la clave de sesión de aplicación encriptada (AppSKey encriptada).

Paso 7: Cuando el servidor de aplicación recibe la clave de sesión de aplicación encriptada, debe desencriptarla con una clave secreta compartida entre el servidor de unión y el servidor de aplicación.

Paso 8: En caso de no recibir la clave de sesión de aplicación encriptada junto al primer uplink del dispositivo final, el servidor de unión debe compartir dicha clave en otro canal de comunicación seguro.

2.2.4 Diferencias con LoRaWAN 1.1

Como primer punto importante a destacar es la incorporación de un servidor de unión en la negociación de conexión de dispositivos y la generación de claves. Este punto fue anticipado en la introducción de la sección 3.2 y fue descrito en el apartado de la interfaz de backend. El agregado de un tercero en la generación de claves evita que sea el servidor de red quien genere las claves de sesión de aplicación. En esta nueva versión también se dividió la clave de sesión de red en tres claves de sesión denominadas Forwarding Network Session Integrity Key (FNwkSIntKey), Serving Network Session Integrity Key (SNwkSIntKey) y Network Session Encryption Key (NwkSEncKey) [13]. Esta adición de nuevas claves de sesión aumenta la seguridad y la robustez del sistema incluso cuando el dispositivo final se encuentra en roaming.

En la activación OTA, además de la clave de aplicación (AppKey) utilizada para derivar la clave de sesión de aplicación, ahora el dispositivo debe almacenar una clave compartida con el servidor de unión llamada clave de red (NwkKey) de la cual se derivan las tres claves de sesión de red anteriormente mencionadas.

En esta versión se añadió un contador de tramas de downlink siendo el total de dos contadores de tramas de downlinks, uno para los downlinks que se originan en el servidor de red (comandos MAC) y otro para los downlinks originados en el servidor de aplicación.

DevNonce y JoinNonce (anteriormente llamado AppNonce) ya no son valores aleatorios. Estos valores se inicializan en 0 por cada dispositivo final y se incrementan en 1 por cada Join-Request. Estos valores deben almacenarse en memoria persistente para el caso de los dispositivos finales alimentados por ciclos.

2.3 Topología del sistema

La arquitectura de la red LoRaWAN está basada en una topología estrella. La misma está compuesta por nodos finales, que son los encargados de enviar la información recopilada de los sensores, que son quienes interactuarán con el mundo físico. Estos se conectan con el gateway, que demodula esta información y la envía a través de otro protocolo, por ejemplo, 3G, 4G, WiFi.

Podemos diferenciar dos áreas principales dentro del protocolo. La primera, en donde se establece la comunicación entre nodos finales y gateways, que se caracteriza por contar con LoRa o FSK (Frecuency Shift Keying) en su capa física. Mientras que en la segunda, la comunicación entre gateways y el servidor de red se realiza mediante la pila TCP/IP.

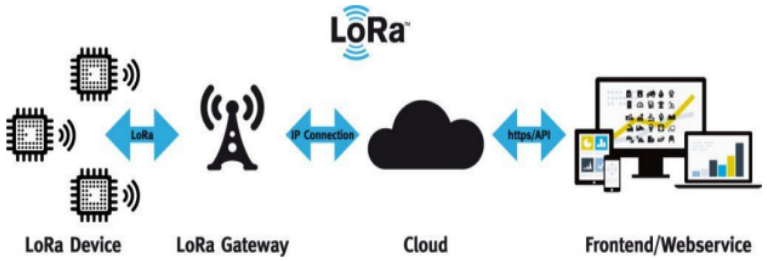


Figura 13: Arquitectura LoRa.

2.3.1 Servidor LoRaWAN

2.3.1.1 Soluciones Comerciales

Existen numerosas implementaciones comerciales de servidores de red y de aplicación LoRaWAN, de los que se destacan: Chirpstack, The Things Network, Loriot, entre otros.

ChirpStack, antiguamente llamado LoRaServer, se compone de una pila de servidores de red LoRaWAN de código abierto que incluye interfaz web para la administración de dispositivos y APIs para realizar integraciones. Soporta las clases A ,B y C, reconfiguración de canal, parámetros regionales y todas las versiones de LoRaWAN [14]. La arquitectura de ChirpStack es similar a la arquitectura TTN, está basada en gateways y servidores que interconectan los nodos LoRaWAN con aplicaciones externas aunque, en ChirpStack hay varios servicios que pueden o no estar en un mismo servidor, como se puede observar en la siguiente figura.

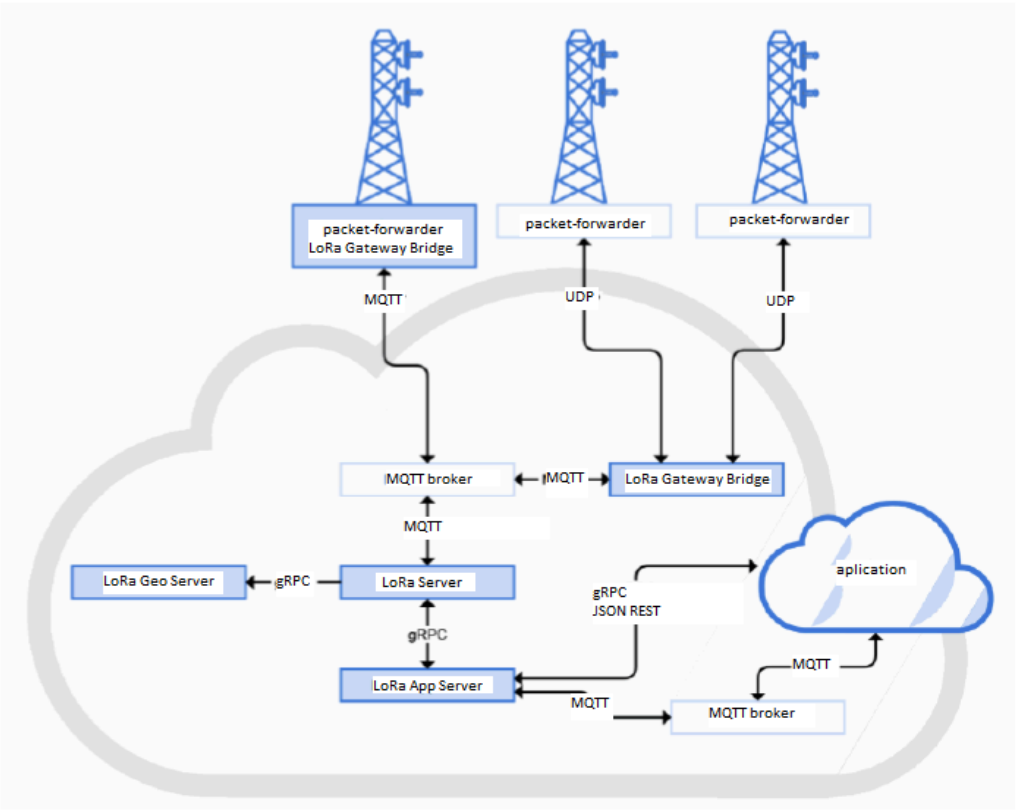


Figura 14: Estructura del servidor de red de ChirpStack.

Los elementos de esta estructura son:

LoRa Gateway Bridge: Es un servicio que convierte los paquetes provenientes de los packet forwarders como Semtech Packet Forwarder (UDP) o LoRa Basics Station (TCP) en paquetes MQTT dirigidos al MQTT Broker.

LoRa Server: Es el servidor de red LoRaWAN propiamente dicho, y es la parte principal del proyecto ChirpStack. La responsabilidad del componente servidor de red es la deduplicación y el manejo de las tramas de enlace ascendente recibidas por las puertas de enlace o gateways, el tratamiento de la capa MAC LoRaWAN y la programación de transmisiones de datos de enlace descendente.

LoRa App Server: Es un servidor de aplicaciones LoRaWAN. Es responsable de la parte del inventario de dispositivos de una infraestructura LoRaWAN, del control de las solicitudes de unión y el tratamiento y cifrado de las cargas útiles de las aplicaciones. Ofrece una interfaz web donde se pueden administrar usuarios, organizaciones, aplicaciones y dispositivos. Para la integración con servicios externos, ofrece una API REST y gRPC.

LoRa Geo Server: Es un servidor de geolocalización. Se puede usar para resolver la ubicación de dispositivos basados en metadatos TDoA (diferencia de tiempo de llegada) proporcionados por los gateways LoRaWAN.

The Things Network (TTN) ofrece un servicio de pago dirigido a empresas para el despliegue y gestión de una red LoRaWAN, sin embargo, cuenta con una versión Community Edition enfocada en redes pequeñas y sin necesidad de suscripción. La principal característica de TTN es su servicio en la nube que permite gestionar los servidores de red y de aplicación sin necesidad de hardware específico, dejándole al usuario final únicamente la implementación de los gateways y los dispositivos finales. Adicionalmente, TTN ofrece la posibilidad de unirse a redes públicas con infraestructura compartida (gateways) o desplegar una red privada [15]. Esto hace de TTN una solución fácil de implementar sin requerir mucho conocimiento técnico.

Loriot es otra solución orientada a empresas con alternativas destinadas a la comunidad. La principal característica de Loriot es su sistema de gestión de red híbrida, compuesta por el protocolo LoRaWAN y el protocolo mioty, otro protocolo LPWAN basado en software. Con estos dos protocolos conviviendo en el sistema, Loriot se vuelve una solución para dispositivos IoT masivos [16].

2.3.1.2 Protocolos de mensajería

MQTT es un protocolo de mensajería cliente-servidor de publicación/suscripción [17]. Múltiples clientes se conectan a un bróker y se suscriben a tópicos en los que están interesados, también se conectan al bróker para publicar mensajes en esos mismos tópicos. Es un protocolo abierto y simple, diseñado para ser fácil de usar. Esto lo hace perfecto para usarse en las tecnologías que incumben al IoT, donde el ancho de banda es bajo y los

protocolos no son complejos. Funciona sobre TCP/IP o sobre cualquier otro que provea conexiones bidireccionales que no tengan pérdidas y los paquetes lleguen ordenados. Algunas de sus características más importantes son:

- El uso de un patrón de mensajería de publicación/suscripción que provee una distribución one-to-many y desvinculación de las aplicaciones.
- El transporte de los mensajes no depende del contenido del payload.
- Tiene tres tipos de QoS dependiendo de los requisitos demandados:

1. "Como mucho una vez", es decir, que el mensaje no llegará por duplicado, pero la pérdida de mensajes puede ocurrir si se pierde el mensaje enviado. Esto es útil para dispositivos que actualizan periódicamente su estado y no importa perder una de sus actualizaciones.

2. "Por lo menos una vez", es decir, que los mensajes están garantizados que lleguen, pero pueden llegar duplicados.

3. "Exactamente una vez", es decir, que los mensajes están garantizados que lleguen sin haber posibilidad de duplicados. Esto es útil para aplicaciones que no puedan permitirse pérdida ni redundancia de paquetes.

- No sobrecarga la red de tráfico.
- Notifica cuando ocurre una desconexión no prevista.

2.3.1.3 Base de Datos

PostgreSQL es una base de datos relacional de código abierto que usa y extiende el lenguaje SQL combinado con multitud de características que hace que el almacenamiento seguro de los datos más complejos sea posible [18]. En esta base de datos es posible generar numerosos tipos de datos, construir funciones y programar en diferentes lenguajes sin necesidad de volver a compilar la propia base de datos.

2.3.2 Gateway

2.3.2.1 Raspberry Pi + RAK2287

Es posible la implementación de un gateway de menores prestaciones para redes pequeñas utilizando un Raspberry Pi, encargado de almacenar y ejecutar el software del concentrador, y el RAK2287, siendo este el hardware del concentrador. La comunicación entre estos puede ser mediante SPI o USB.

Raspberry Pi es el nombre comercial de una serie de computadoras de bajo costo. Estas computadoras se clasifican en tres series principales: Flagship, Zero y Compute Module (CM) [19]. Esta sección describe al Raspberry Pi 4 modelo B de la serie Flagship. El Raspberry Pi 4 modelo B cuenta con un procesador ARM-Cortex A72 1,5GHz de cuatro núcleos, interfaces Bluetooth, WiFi, Gigabit Ethernet, 2 puertos USB 2.0 y 2 puertos USB 3.0. Además posee 28 pines GPIO (General Purpose Input/Output) con soporte para las siguientes interfaces: UART, SPI e I2C [20]. Soporta sistemas operativos basados en Linux. Se alimenta con una fuente de tensión continua capaz de entregar 5V/3A.

El RAK2287 es un módulo concentrador LPWAN con un factor de forma mini PCIe. Está compuesto por el chip de procesamiento LoRa en banda base SX1302 y dos chips de frente de radiofrecuencia SX1250 que se encargan de la modulación y demodulación de los paquetes LoRa [21]. Para su correcto funcionamiento en un Raspberry Pi, es necesaria la placa interfaz RAK2287 Pi HAT, que adapta el conector mPCIe al factor de forma del RPi.

Figura 15: Raspberry Pi 4B + RAK2287.

2.3.2.2 Cisco Wireless Gateway for LoRaWAN

Cisco ofrece una solución integral para entornos industriales con soporte para todas las versiones de LoRaWAN y para las siguientes regiones: EU 863 - 870 MHz, India 865 - 867 MHz, U.S. 902 - 928 MHz, Australia 915 - 928 MHz, y AS 923 MHz, 902.2-923.4 MHz Tailandia [22]. Posee soporte para la geolocalización de dispositivos finales y opera con el protocolo IPSec.

Figura 16: Gateway LoRa Cisco.

Es un producto resistente diseñado específicamente para implementaciones exteriores duras, y también adecuado para aplicaciones en interiores. Se adhiere al diseño de referencia de hardware de puerta de enlace Semtech de próxima generación (versión 2), ofrece hasta 16 canales de enlace ascendente y las capacidades de geolocalización a través de las técnicas de Diferencia de Tiempo de Llegada (TDOA) e Indicación de Fuerza de Señal Recibida (RSSI).

2.3.2.3 Multitech Programable Gateway

El gateway Conduit se caracteriza por ser rápido para implementar y fácil de administrar, lo que permite realizar aplicaciones IoT de manera efectiva. Además, cuenta con una amplia variedad de tarjetas de expansión MultiConnect mCard que proporcionan conectividad local y wireless para activos en el campo y se conectan directamente a la parte posterior del gateway LoRa [23]. Posee terminales con protección AEP (advanced endpoint protección), que proporcionan mayor seguridad contra ciberamenazas.

El software de inteligencia de bordo mPower Edge Intelligence incluye programabilidad, flexibilidad de red, seguridad y gestión mejoradas para soluciones de IoT industrial escalable.

Figura 17: Gateway Multitech.

2.3.3 Nodo Final

2.3.3.1 LoRa32U4 II v1.3

LoRa32U4 II es una placa de desarrollo, imitación del Adafruit Feather 32U4 LoRa, diseñada por BSFrance y distribuida por DIYMall. Cuenta con un microcontrolador Atmel ATmega32u4 AVR y un módulo de radio LoRa HPD13A v1.1 de HPDTek alimentados a través de un regulador LDO. Posee soporte para baterías Li-Ion/LiPo, carga de batería y monitoreo de la misma. También, incorpora un conector IPEX/U.FL para la antena del módulo LoRa [24].

Figura 18: Placa de desarrollo LoRa32u4 II.

Este módulo se basa en el chip de radio SX1276 de Semtech, el cual puede proveer una potencia máxima de transmisión de +20dBm y posee un rango de trabajo de 137 MHz - 1020 MHz [25].

2.3.3.2 Regulador MT3608

Los convertidores DC/DC son circuitos capaces de transformar un nivel de voltaje a otro de mayor o menor nivel. Existen dos tipos de convertidores o reguladores DC-DC: lineales y conmutados (switching). Poseen únicamente componentes que no presentan pérdidas, es decir, que no absorben energía. Los componentes son básicamente de 2 tipos: conmutadores y almacenadores. Los conmutadores son interruptores de paso de corriente, que idealmente no presentan pérdidas por conmutación, normalmente son transistores mosfet. Los componentes almacenadores son los inductores y capacitores que almacenan la energía temporalmente para luego devolverla al circuito. Podemos clasificar a conmutadores DC-DC por su voltaje de salida en: reductores (Step-Down o Buck), elevadores (Step-Up o Boost) y reductores-elevadores (Step-Up-Down o Buck-Boost) [26].

El convertidor DC-DC MT3608 es un regulador de tipo conmutador elevador (Step-Up o Boost) con una alta eficiencia de conversión, excelente regulación de línea y bajo

voltaje de rizado. El módulo reduce al mínimo el uso de componentes externos para simplificar el diseño de fuentes de alimentación y permite obtener un voltaje regulado a partir de una fuente con un voltaje inferior [27].

Figura 19: Regulador boost MT3608.

Este módulo posee un potenciómetro con el que se puede ajustar fácilmente la salida al voltaje necesario, pero la corriente de salida será menor en comparación con la corriente de entrada. Incluye además bloqueo por bajo voltaje, limitación de corriente y protección contra la sobrecarga térmica para evitar posibles daños.

2.3.3.3 Sensores

DHT11

El IC DHT11 es un sensor de temperatura y humedad relativa que utiliza un sensor capacitivo de humedad y un termistor para medir el aire circundante, y muestra los datos mediante una señal digital en el pin de datos.

Tiene una resolución de 8 bits para representar valores decimales, posee una respuesta rápida, la interfaz de 1 hilo le proporciona una alta fiabilidad haciéndolo de una conexión rápida y fácil, además transmite a una distancia de 20 metros. Es bastante simple de usar, pero requiere sincronización precisa para tomar datos [28].

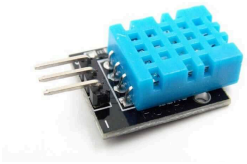


Figura 20: DHT11 - Sensor de temperatura y humedad relativa.

Rango de medición humedad	20% a 90% RH
Rango de medición temperatura	0ºC a +50°C
Resolución humedad	+/- 5.0% RH
Resolución temperatura	+/- 2.0ºC
Alimentación	3.3V a 5V DC
Corriente máxima	2.5 mA
Frecuencia máxima de muestreo	1 Hz

Tabla 10: Características técnicas de DHT11.

Cuando el módulo LoRa envía una señal de arranque al sensor, este pasa del modo bajo consumo a modo de funcionamiento, esperando a que el módulo complete la señal de arranque. Una vez completada, el DHT11 envía una señal de respuesta de datos de 40 bits que incluye la información de humedad relativa y temperatura. El usuario puede elegir leer algunos datos. Sin la señal de inicio del módulo, el sensor no dará la señal de respuesta. Una vez recogidos los datos, el sensor volverá a su modo de bajo consumo de energía hasta que reciba nuevamente la señal de inicio [28].

HC-SR501

El módulo HC-SR501 es un sensor de movimiento de bajo costo que utiliza el PIR LHI778 y el controlador BISS0001 para medir variaciones de emisiones infrarrojas de forma diferencial. Se alimenta con una tensión de 5 V y la corriente durante la inactividad es menor a 50 µA. Cuando el módulo detecta una variación, lleva al pin de datos a high (3,3 V). Este pulso se mantiene en alto por un tiempo configurable mediante potenciómetros, pudiendo ser este desde 3 segundos hasta 5 minutos. Una vez el sensor vuelve a su estado inactivo, lleva al pin de datos a low (0 V). Además, al pasar al estado inactivo, el sensor ignora cualquier disparo durante 3 segundos, sin importar el modo de configuración.

Este módulo además cuenta con los modos “un disparo” y “redisparable”. Cuando se encuentra en el modo “un disparo”, los disparos ocurridos en tiempo de actividad del sensor son ignorados, por lo tanto, el pin de datos se mantiene en high durante el tiempo preestablecido. En el modo redisparable, cada disparo suma “x” segundos al tiempo en high del módulo, siendo x el tiempo preestablecido por un disparo. También es posible modificar la sensibilidad del detector mediante un potenciómetro, dándole un alcance variable de entre 3 y 7 metros [29].

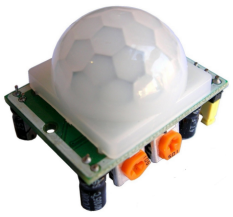


Figura 21: HC-SR501 - Sensor de movimiento.

HC-SR04

El módulo HC-SR04 es un sensor ultrasónico de bajo costo que utiliza una serie de pulsos ultrasónicos para determinar la distancia entre el sensor y los objetos. Se alimenta con una tensión de 5 V y consume una corriente de 15 mA durante la operación. En estado de inactividad su corriente es menor a 2 mA. Posee dos pines para la operación de medición. El pin “trigger” dispara los pulsos ultrasónicos cuando se pone en high durante al menos 10 µs. Luego de haberse enviado los 8 pulsos ultrasónicos de 40 kHz, el pin “echo” se pone en high hasta que detecta una recepción de pulso ultrasónico, en dado caso, el pin “echo” se pone en low. La distancia se puede obtener teniendo en cuenta que la duración en high del pin “echo” es exactamente el tiempo de vuelo de la onda sonora desde el sensor hasta el objeto y del

objeto hasta el sensor. Por lo tanto, se puede obtener la distancia con la siguiente fórmula: $distancia = \frac{duraciónPulso\ (en\ \mu s)}{2} * 0,01716\ \frac{cm}{\mu s}$, donde la constante es la velocidad del sonido en el aire expresado en cm/µs [30].

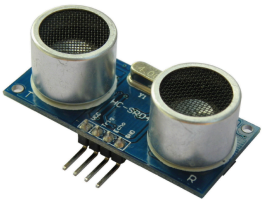


Figura 22: HC-SR04 - Sensor ultrasónico.

3. Definición del Sistema

En la presente sección se definirán el hardware y software a implementar como elementos de red LoRaWAN, descritos en la sección 2.3. Para llevar a cabo la implementación de la red, se valoran las distintas soluciones disponibles en el mercado, dado que existen opciones para el desarrollo electrónico, tanto como si se quisiera un armado de cada parte del sistema o un módulo con todas sus partes integradas. Lo que le da al proyecto flexibilidad y la posibilidad de ser ajustado en función a las necesidades propias de cada implementación.

3.1 Marco Inicial

Uno de los primeros puntos a establecer es la simplificación del servidor de red, servidor de unión, servidor de aplicación y el software del gateway en un único hardware. Si bien es posible almacenar cada software en servidores dedicados, el uso de un único hardware para todo el software de red reduce significativamente los costos y se justifica con la consideración de que la red será de baja escala. El software para la implementación de los servidores será ChirpStack debido a su compatibilidad con numerosas marcas de hardware de gateways, soporte para LoRaWAN 1.1 y por ser una solución open-source muy popular.

En el otro extremo de la red, se define una cantidad equilibrada de 3 nodos considerando que, por ser un prototipo, la red no debe ser de grandes dimensiones pero tampoco lo suficientemente pequeña como para no observar los problemas asociados a una red con acceso compartido al medio de transmisión.

3.2 Selección de Componentes

Los gateways industriales mencionados en la Sección 2.3.2 se encuentran en el mercado alrededor de los 2 mil y 3 mil usd, dependiendo el proveedor, el país y la disponibilidad, también las características y modelo del gateway. Es por esto por lo que, realizando una evaluación adecuada de costos y operatividad, se define la construcción e implementación del gateway LoRa utilizando la combinación de los componentes definidos en 2.3.2.1. Resta definir el hardware de los dispositivos finales, el hardware del gateway y el hardware sobre el cual se ejecutará ChirpStack.

Chirpstack ofrece una distribución de Linux basada en OpenWRT, llamada ChirpStack Gateway OS, para Raspberry Pi, con el objetivo de simplificar la instalación y configuración del conjunto de software anteriormente mencionado [31]. Como todo el software a implementar se ejecuta en Linux, se selecciona al Raspberry Pi 4 modelo B como hardware más adecuado para la implementación de los servidores y parte del gateway.

Dentro de las opciones de hardware para el diseño del gateway encontradas, el RAK2245 se adecúa bastante bien en cuanto a precio y, además, cuenta con una interfaz para conectarse a los pines GPIO del Raspberry Pi. Investigando un poco más en las soluciones de RAKWireless, observamos una solución similar al RAK2245, el RAK2287. Este módulo concentrador cuenta con el chip procesador de paquetes de nueva generación (SX1302) a

comparación de su predecesora (SX1301). Por un precio prácticamente idéntico, se escoge al RAK2287 como preferido.

El dispositivo final consta de tres partes: un microcontrolador sobre el cual se implementa el protocolo LoRaWAN desde su capa de enlace de datos hasta la capa de aplicación, un módulo LoRa encargado de actuar como capa física, y los sensores encargados de tomar datos externos al sistema. Inicialmente se consideró la posibilidad de adquirir el microcontrolador y el módulo LoRa de forma separada con el fin de dar flexibilidad al dispositivo final. Pero luego, y teniendo como mayor ventaja al costo, se decidió utilizar una placa de desarrollo que contenga ambos elementos integrados y previamente conectados, en particular, se seleccionó la placa de desarrollo LoRa32u4 II v1.3.

Para que la red sea implementable en áreas remotas los módulos en los nodos finales son alimentados por baterías, dentro de la amplia variedad disponible se utiliza la batería 600 mah 3.7v LG recargable, es una batería del tipo LiPo (polímero de Litio), este tipo de baterías proporciona una fuente de alimentación muy eficiente, son muy ligeras y soportan altos picos de corriente. Incorporando un conector tipo Molex/JST para conectarlo directamente al módulo LoRa, donde puede ser cargado a través de un cable micro USB [32].

En lo que respecta a la recolección de variables externas, se seleccionaron los sensores descritos en la sección 2.3.3.3. Este tipo de sensores de bajo costo se encuentran ampliamente distribuidos en paquetes Arduino o en kits de placas de desarrollo. La facilidad de adquisición y la precisión de los datos obtenidos fueron los principales motivos para su elección.

Para que estos sensores sean compatibles con la placa de desarrollo, se requiere un regulador step-up que pueda convertir la tensión variable de la batería (3 V a 4,2 V) a 5 V para alimentar a los sensores que no puedan ser alimentados con 3,3 V. El convertidor MT3608 cumple de forma efectiva esta función manteniendo buena regulación ante las variaciones de tensión de la batería y una alta eficiencia en el consumo de energía. Con los elementos de red definidos, se representa finalmente la arquitectura de red a implementar.

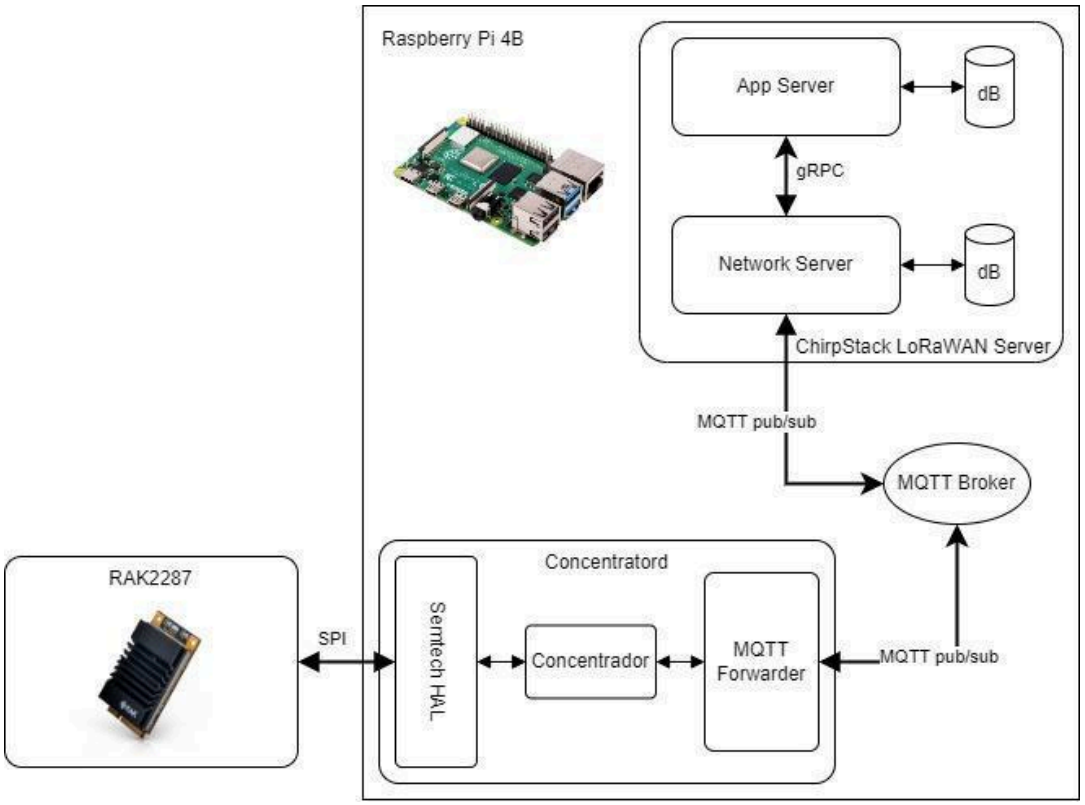


Figura 23: Arquitectura de red.

Cantidad	Componente	Precio unitario ($ ARS)	Envío ($ ARS)	Precio final ($ ARS)
1	RAK Wireless RAK2287 LoRa Módulo Concentrador	27.000	25.000	52.000
1	Raspberry Pi 4 Model B Quad Core 64-bit Wifi Bt 4gb	97.092	0	97.092
3	Placa de desarrollo Lora32u4 II v1.3	34.400	0	103.200
	Costo parcial (Abril 2024)($ ARS)			252.292
3	Fuente Convertidor Boost Elevador Dc Dc Step Up 2A - MT3608	2.919	0	8.757
3	Bateria Lipo 3.7V- 600mah Conector Jst	3.789	0	11.367
	Costo parcial (Octubre 2024) ($ ARS)			20.124
	Total ($ ARS)			272.416

Tabla 11: Resumen de costos del hardware seleccionado.

4. Implementación y Pruebas Experimentales

4.1 Instalación y Configuración

4.1.1 Gateway y Servidor LoRaWAN

Como primer paso en la implementación, se procedió con la instalación del sistema operativo ChirpStack Gateway OS Full en el Raspberry Pi. Luego, se montó el shield del hardware del módulo concentrador y se realizaron las configuraciones necesarias descritas en [33].

Particularmente, en la configuración del concentradord se seleccionó como chipset al SX1302 y como shield al RAK2287 sin GPS ni interfaz USB. El plan de frecuencias seleccionado es el AU915-928. Los problemas asociados a la implementación del gateway se describen en la sección 4.3.

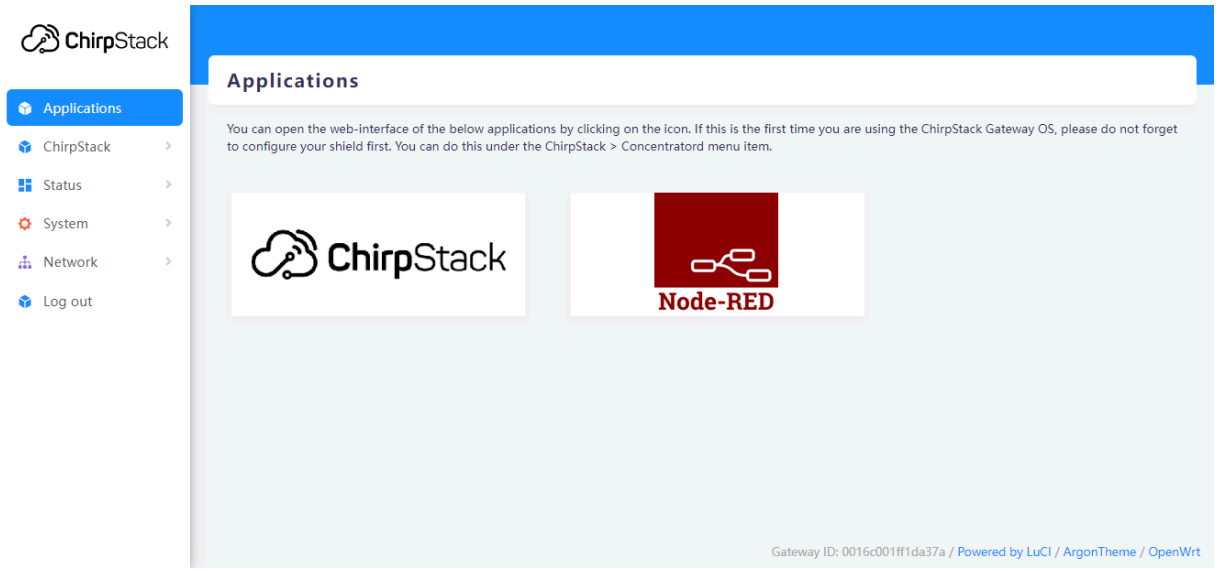


Figura 24: Interfaz web de los servicios del OS.

Antes de proceder a la configuración del servidor, se instaló y configuró la VPN ZeroTier One con el objetivo de facilitar la configuración, pruebas y monitoreo del gateway de forma remota.

Luego, en la configuración del servidor LoRaWAN, se agregó el gateway con sus parámetros regionales correspondientes y el gateway ID, se generaron los perfiles de dispositivo final que utilizarán estos. Las configuraciones correspondientes al perfil del dispositivo final se centran en la clase del dispositivo, la versión LoRaWAN que soporta, los parámetros regionales, el método de activación y el codec utilizado.

Para agregar un gateway, además de contar con la configuración realizada, se debe conocer el gateway ID que, en el caso de los concentradores basados en el chip SX1302, es un valor intrínseco del hardware. Este identificador se puede visualizar en la página de inicio de la interfaz web que sirve el OS en el RPi. Una vez agregado el gateway al servidor, se deben recibir mensajes de “gateway status” cada 30 segundos.

4.1.2 Dispositivos Finales

Antes de poder agregar dispositivos finales a la red, es necesario crear una aplicación sobre la cual los dispositivos finales se registren. Una aplicación es una colección de dispositivos con el mismo propósito o del mismo tipo. Una vez creada una aplicación, dentro de la misma se pueden registrar los dispositivos a partir del perfil de dispositivo, DevEUI y AppKey. Se debe destacar que Chirpstack soporta la activación OTA mediante servidores de unión externos. Sin embargo, se utilizó el mismo software como servidor de red y servidor de aplicación por lo que el valor de AppEUI no es relevante.

Para programar el dispositivo final, el IDE de Arduino debe contar con la librería "mcci-catena arduino-lmic", que es un fork de la librería "arduino lmic" de IBM. Las siglas LMIC provienen de LoRa MAC In C [34]. Esta librería es la encargada de la gestión de la capa MAC del dispositivo final y de la comunicación con el módulo de RF LoRa. Es el software open-source más avanzado en términos de mantenimiento de código y actualización de versiones.

Como primera prueba, se toma el código plantilla ofrecido por la librería para verificar una activación OTA con el servidor y la transmisión y recepción exitosa de paquetes LoRa. Dicho programa envía un paquete con el String "Hello World!" cada 60 segundos luego de realizar exitosamente la activación OTA [35]. Los resultados fueron satisfactorios. Se observó la comunicación de Join-Request/Join-Accept, uplinks de aplicación (el String "Hello World!"), y uplinks y downlinks de comandos MAC (específicamente los DeviceStatusRequest y DeviceStatusAnswer).

Figura 25: Registro de eventos del dispositivo final.

Para implementar correctamente una secuencia periódica de transmisión de datos, debemos conocer el funcionamiento interno de la librería y cómo ésta recomienda la programación de la capa de aplicación. La librería LMIC ofrece una arquitectura basada en la gestión de eventos, enviando todos los eventos del protocolo a una función de retorno *onEvent()*. Todo el código de aplicación corre sobre los denominados *jobs*, los cuales son ejecutados por la función *os_runloop_once()*, que es una función planificadora en tiempo de ejecución [36]. Se recomienda que la ejecución de los *jobs* sea de corta duración para asegurar el sincronismo requerido por el protocolo.

El código utilizado para programar los dispositivos finales se encuentra en el Anexo I, II y III. Para el caso de la transmisión de valores de temperatura ambiente y humedad relativa, se define cierto tiempo constante entre transmisiones y los valores se envían de la misma forma en la que los entrega el sensor. Estos valores son decodificados en el servidor de aplicación con el objetivo de reducir al mínimo la carga de los dispositivos finales. El dispositivo encargado de detectar movimientos enviará un paquete cada vez que se detecte un movimiento. El dispositivo de medición ultrasónica trabajará de manera similar al dispositivo de medición de humedad y temperatura, enviando paquetes separados por un intervalo constante.

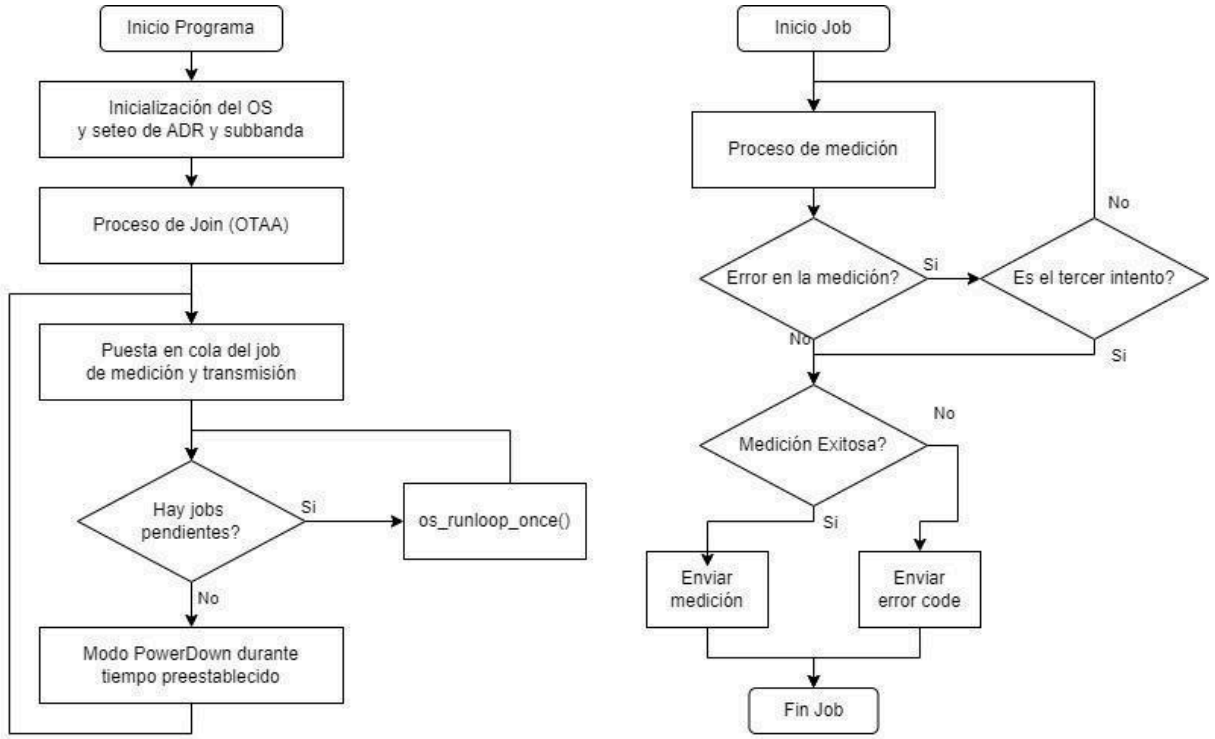


Figura 26: Diagrama de flujo de alto nivel del programa.

El diagrama de flujo representa de forma simplificada la ejecución del programa de los dispositivos finales cuya medición se realiza a intervalos fijos, como el de medición de temperatura y humedad y el de medición ultrasónica. Para obtener los datos del DHT11, se utilizó una versión modificada de la librería en [37]. La implementación de los modos de bajo consumo se basan en la librería LowPower de Rocket Scream Electronics [38]. El código de estas librerías y las respectivas modificaciones se encuentran en el Anexo IV.

4.2 Resultados y Análisis

4.2.1 Resultados de la comunicación

Los resultados de la comunicación fueron exitosos en los tres dispositivos finales, recibiendo los datos decodificados de forma satisfactoria. Esto se puede observar en el panel de presentación provisto por Chirpstack.

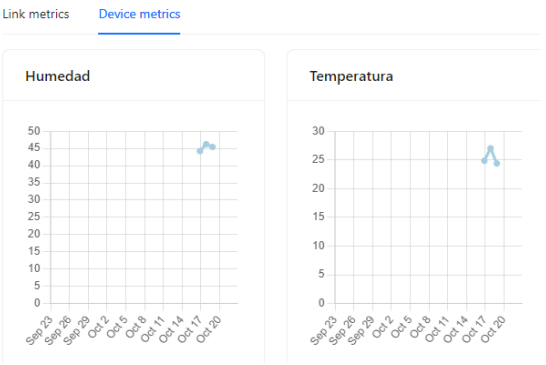


Figura 27: Panel de presentación con los datos decodificados.

Es posible configurar los perfiles de dispositivo con un codec encargado de tomar el payload de aplicación y presentarlos en formato JSON para su posterior almacenamiento en un base de datos o directamente transmitirlo a las integraciones soportadas por el servidor. Análogamente, el codec prepara los bytes de downlink y les da el formato deseado.

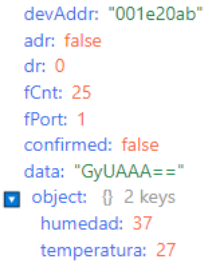


Figura 28: Valores de los campos asociados a un uplink.

4.2.2 Análisis de Alcance

El primer análisis de alcance se realizó con el ADR desactivado y con el DR fijo, siendo este el DR0. Las pruebas realizadas se hicieron en inmediaciones de la Universidad con una distancia máxima registrada de 700 m. Los paquetes de uplink se enviaron de forma periódica a medida que nos íbamos alejando del gateway con el objetivo de observar la disminución en el RSSI y el SNR. A continuación se muestra de forma representativa la topología del suelo y las alturas de las antenas (no tiene en cuenta los obstáculos tales como edificios o vegetación) de la distancia máxima alcanzada.

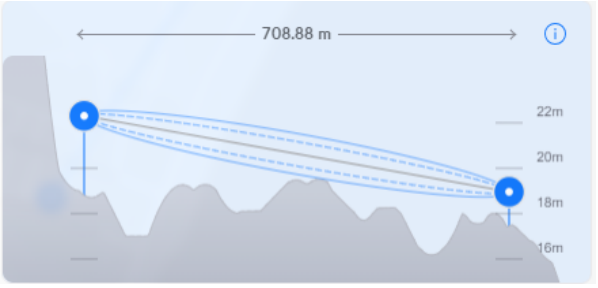


Figura 29: Perfil topológico del suelo en la prueba.

Tiempo	Nombre de Aplicación	DeviceEUI	DR	Frame Count	Field Port	Humedad	Temperatura	SNR	RSSI	Canal
19/10/2024 15:05:16	Temperatura	e52f39790751a22e	0	1	0			6,75	-28	3
19/10/2024 15:42:11	Temperatura	e52f39790751a22e	0	0	1	44	23,031	7,75	-44	3
19/10/2024 15:42:14	Temperatura	e52f39790751a22e	0	1	0			7	-52	6
19/10/2024 15:43:31	Temperatura	e52f39790751a22e	0	2	1	44	23,031	7,75	-46	5
19/10/2024 15:44:48	Temperatura	e52f39790751a22e	0	3	1	43	24,02	7,5	-34	1
19/10/2024 15:46:06	Temperatura	e52f39790751a22e	0	4	1	42	24,02	8,25	-45	4
19/10/2024 15:47:23	Temperatura	e52f39790751a22e	0	5	1	41	24,02	5,75	-69	7
19/10/2024 15:48:40	Temperatura	e52f39790751a22e	0	6	1	39	24,004	3,5	-86	2
19/10/2024 15:49:58	Temperatura	e52f39790751a22e	0	7	1	39	22,008	3,75	-79	1
19/10/2024 15:51:15	Temperatura	e52f39790751a22e	0	8	1	39	21	-7,25	-101	5
19/10/2024 15:52:32	Temperatura	e52f39790751a22e	0	9	1	39	21	-10,5	-104	7
19/10/2024 15:53:49	Temperatura	e52f39790751a22e	0	10	1	39	21,031	-1	-93	4
19/10/2024 15:55:06	Temperatura	e52f39790751a22e	0	11	1	39	21	-1	-92	3
19/10/2024 15:57:42	Temperatura	e52f39790751a22e	0	13	1	43	20,023	-16,5	-110	6
19/10/2024 16:01:34	Temperatura	e52f39790751a22e	0	16	1	39	22,023	-12,25	-104	4
19/10/2024 16:02:51	Temperatura	e52f39790751a22e	0	17	1	38	23,031	-4,25	-96	2
19/10/2024 16:04:08	Temperatura	e52f39790751a22e	0	18	1	38	24,031	-6,5	-101	7
19/10/2024 16:05:25	Temperatura	e52f39790751a22e	0	19	1	36	25,031	-14,75	-108	5
19/10/2024 16:06:43	Temperatura	e52f39790751a22e	0	20	1	35	26,008	-9,5	-100	2
19/10/2024 16:08:00	Temperatura	e52f39790751a22e	0	21	1	37	27,023	-11,25	-104	3

Figura 30: Datos obtenidos en la prueba.

Los metadatos recopilados, en especial el RSSI y el SNR, describen la pérdida de potencia recibida a medida que el nodo se aleja del gateway. Se debe destacar que hubo instantes donde la conexión se perdió por completo. Esto explica por qué el contador de frames no aumenta de forma lineal sino que existen frames que se pierden, como el frame 12, 14 y 15. El RSSI más bajo registrado fue -108 dBm.

4.2.3 Análisis de Ciclo de Batería

Desde el lado de los dispositivos finales, se busca aprovechar al máximo los ciclos de vida de la batería. Para lograr esto, se implementan estrategias de ahorro de batería tales como el establecimiento de intervalos de espera considerables entre transmisiones y programación del modo “power down” del microcontrolador en los tiempos de no transmisión. El análisis siguiente es una estimación en el consumo de corriente del sistema teniendo en cuenta el tiempo de transmisión, las dos ventanas de recepción, el tiempo de sensado, la corriente de fuga de los elementos cuando se encuentran inactivos y la eficiencia de los reguladores de tensión.

Para obtener el tiempo estimado de transmisión, suponemos un uplink sin el campo FOpts, con un payload de aplicación de 4 bytes (los bytes obtenidos del sensor DHT11), una tasa de datos igual a DR2 (SF10/125 kHz) y consecuentemente una tasa de datos de downlink igual a DR8 (SF12/500 kHz) en ambas ventanas, suponiendo RX1DROffset = 0. También se toman los valores por defecto establecidos en los parámetros regionales de AU915.

Con estos datos se puede calcular la tasa de símbolos de subida y de bajada. Del apartado 3.1 se tiene que la tasa de símbolo es $R_S = \frac{BW}{2^{SF}}$, entonces:

$$R_{S/up} = 125\ kHz/2^{10} = 122,07\ símbolos/s$$

$$R_{S/down} = 500\ kHz/2^{12} = 122,07\ símbolos/s$$

Un paquete de datos entonces tendrá: preámbulo (8 S) + sync word (4,25 S) + PHDR y PHDR_CRC (8 S) + PHYPayload (L bytes) + CRC (2 bytes). PHYPayload está compuesto

por: MHDR (1 byte) + FHDR (7 bytes) + FPort (1 byte) + FRMPayload (4 bytes) + MIC (4 bytes). Por lo tanto $L + CRC = 17\,bytes + 2\,bytes = 152\,bits * \frac{1\,símbolo}{10\,bits} = 16\,S$. El número total de símbolos a enviar por paquete es 36,25 símbolos y, a una tasa de $R_{S/up}$, el tiempo de transmisión es aproximadamente 300 ms. El tiempo de las ventanas de recepción son tales que pueden detectar un preámbulo. La librería utilizada mantiene las ventanas de recepción durante el lapso de 6 símbolos [34], equivalente a 50 ms aproximadamente. Por defecto, los valores RECEIVE_DELAY1 y RECEIVE_DELAY2 son 1 y 2 segundos respectivamente. Esto se debe tener en cuenta ya que, a pesar de que el módulo LoRa se encuentre en modo Idle, el microcontrolador se encuentra activo gestionando la sincronización de las ventanas de recepción inminentes.

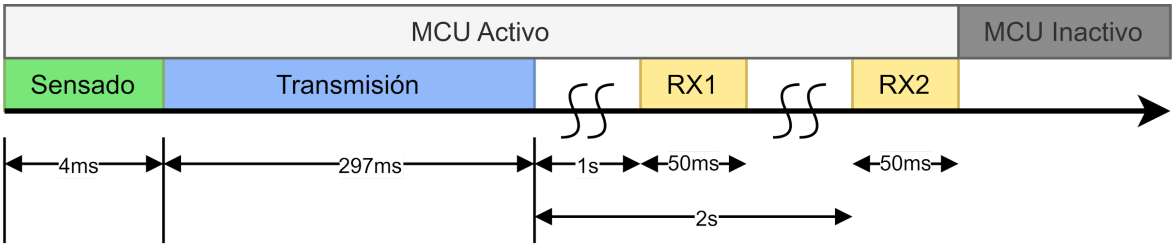


Figura 31: Secuencia de trabajo del dispositivo final.

Elemento	Corriente durante actividad (mA)	Corriente durante inactividad (μA)	Tiempo de actividad (ms)
Microcontrolador	5,5	5,5	2351
Módulo LoRa	120 (TX) / 11,5 (RX)	1,5 (Idle Mode)	297 (TX) / 50 (RX)
Sensor (DHT11)	2,5	150	4

Tabla 12: Consumo máximo asegurado según estado.

Estos valores nos dan una corriente promedio de 21,30 mA durante el ciclo de actividad del dispositivo final, y una corriente de 157 μA durante el ciclo de inactividad. La siguiente tabla muestra los valores de autonomía del dispositivo final estimados dados los ciclos de trabajo y la capacidad de la batería.

Ciclo de trabajo	**Autonomía del dispositivo final (días)**		
Capacidad de batería	**150 mAh**	**600 mAh**	**1500 mAh**
60 s	6	25	63
1800 s	34	135	339
3600 s	36	146	366

Tabla 13: Autonomía estimada según ciclo de trabajo y capacidad de batería.

Se debe tener en cuenta que estos valores pueden aumentar de forma lineal con el aumento de la capacidad de la batería y que el aumento del ciclo de trabajo no produce un aumento lineal de la autonomía del dispositivo ya que, aunque se reduce el valor de corriente

promedio por ciclo, ésta no puede ser inferior a la corriente de inactividad. En estas estimaciones no se tuvieron en cuenta los paquetes de Join ni los comandos MAC que podrían intercambiarse entre los dispositivos y el servidor debido al poco impacto que tienen en el alto volumen de paquetes de datos durante un ciclo de vida de batería. Y como última observación, los dispositivos habilitan el algoritmo ADR que reduce el consumo de potencia siempre que el enlace lo permita, por lo tanto estos valores de autonomía mejoran en la práctica.

4.2.4 Integraciones

En la presente sección se mostrarán ejemplos de aplicación y cómo estas se integran a la red. ChirpStack admite múltiples integraciones mediante APIs. Se seleccionó como integraciones a implementar a la base de datos PostgreSQL y Node-RED.

4.2.4.1 PostgreSQL

Se configuró la base de datos PostgreSQL para el almacenamiento de los eventos del servidor. Luego se utilizó el mismo como fuente de datos en una planilla de Excel. Se debe destacar que la base de datos tiene el potencial de servir como fuente de información de cualquier aplicación de mayor complejidad como Power Bi o Tableau, según la necesidad.

Tiempo	Nombre de Aplicación	DeviceEUI	DR	Field Port	Humedad	Temperatura	SNR	RSSI	Canal
14:02:44	Temperatura	e52f39790751a22e	2	1	45	22,023	9,5	-31	7
14:04:00	Temperatura	e52f39790751a22e	2	1	45	22,023	12,75	-84	4
14:05:16	Temperatura	e52f39790751a22e	2	1	45	22,023	9,25	-74	2
14:06:32	Temperatura	e52f39790751a22e	2	1	45	22,008	13,25	-78	3
14:07:49	Temperatura	e52f39790751a22e	2	1	46	21,031	13,25	-34	6
14:09:05	Temperatura	e52f39790751a22e	2	1	46	21,031	13,25	-76	1
14:10:21	Temperatura	e52f39790751a22e	2	1	46	22,008	10,5	-33	7
14:11:37	Temperatura	e52f39790751a22e	2	1	46	22,008	13,5	-79	
14:12:54	Temperatura	e52f39790751a22e	2	1	46	22,008	13,25	-33	5
14:14:10	Temperatura	e52f39790751a22e	2	1	46	22,008	12,5	-79	4
14:15:26	Temperatura	e52f39790751a22e	2	1	46	22,008	10,25	-85	2
14:16:42	Temperatura	e52f39790751a22e	2	1	46	22,008	13,25	-78	3
14:17:59	Temperatura	e52f39790751a22e	2	1	46	22,008	12,75	-78	4
14:19:15	Temperatura	e52f39790751a22e	2	1	46	22,008	12,75	-31	6
14:20:32	Temperatura	e52f39790751a22e	2	1	46	22,023	10,25	-76	2
14:21:49	Temperatura	e52f39790751a22e	2	1	46	22,023	13,75	-77	1
14:32:00	Temperatura	e52f39790751a22e	2	1	47	22,023	13,5	-78	
14:33:16	Temperatura	e52f39790751a22e	2	1	47	22,023	13	-76	3
14:34:32	Temperatura	e52f39790751a22e	2	1	47	23	13,75	-33	6
14:35:50	Temperatura	e52f39790751a22e	2	1	47	23	13	-79	4
14:37:06	Temperatura	e52f39790751a22e	2	1	47	23	13	-34	5

Figura 32: Tabla obtenida a partir de BD de Chirpstack.

4.2.4.2 Node-RED

Node-RED es una herramienta de programación que enlaza hardware, APIs y servicios en línea mediante diagramas de flujo basados en nodos. Posee un editor sobre una interfaz web que facilita el cableado de los nodos y la gestión de flujos [39].

Se programó un flujo de Node-RED que se encarga de analizar los datos obtenidos de los sensores y envía un correo electrónico cuando se supera un umbral establecido previamente. En el caso del sensor de movimiento, el correo es enviado en cada detección. La siguiente figura muestra los nodos que componen al sistema y el flujo que recorre un evento de uplink.

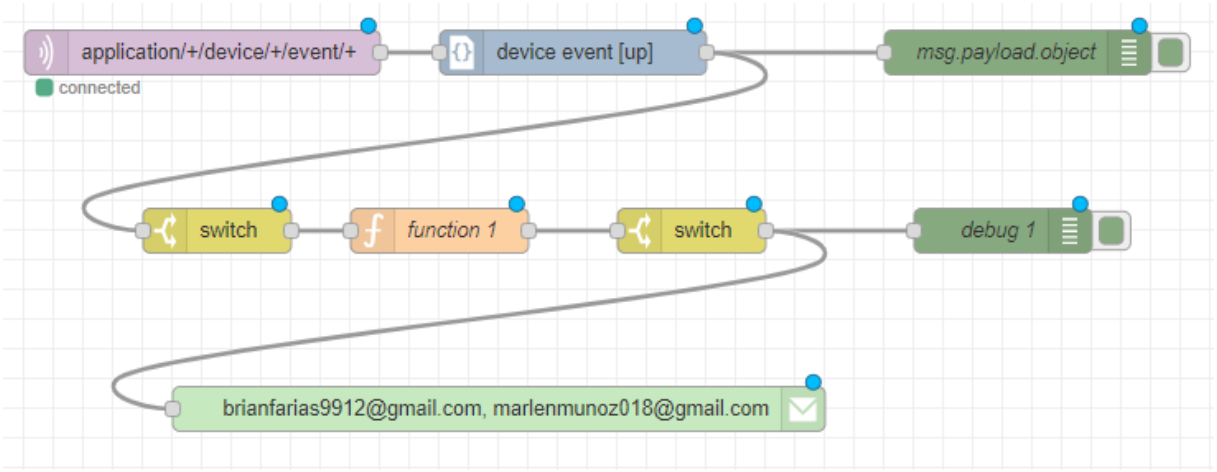


Figura 33: Flujo de la automatización de correos.

La primer fila de nodos se encarga de la obtención de los eventos utilizando un nodo mqtt-in, que se suscribe a los eventos cuyo tópico es “application/+/device/+/event/+”, y un nodo ChirpStack que filtra los eventos permitiendo que los eventos de uplink continúen en el flujo. La segunda fila tiene dos filtros switch que permiten el paso a los eventos de aplicación (fPort = 1) y a los eventos que tengan el flag de enviar en 1 (send = 1) respectivamente. La función denominada “function 1” se encarga de preparar el asunto y el cuerpo del correo según el evento entrante y crea el flag “send” comparando los valores medidos con sus respectivos umbrales. El código de la función se encuentra en el Anexo V.

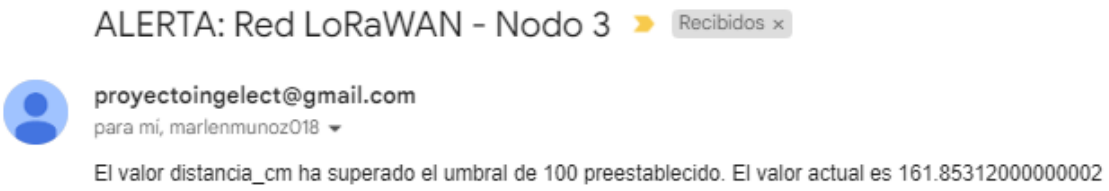


Figura 34: Formato y cuerpo del correo automático.

4.3 Problemas Encontrados

4.3.1 Reinicio Continuo del Software del Concentrador

El primer obstáculo enfrentado fue la nula detección del concentrador RAK2287 por parte del software Concentratord. El programa lograba identificar el hardware del gateway y devolvía el gateway ID (indicio de comunicación exitosa entre el RPi y el RAK2287). Pero luego de la configuración del software del concentrador, se reiniciaba dicho proceso

(Concentratord) y este ya no volvía a detectar el hardware, arrojando “could not find gateway ID” (no se pudo encontrar el gateway ID).

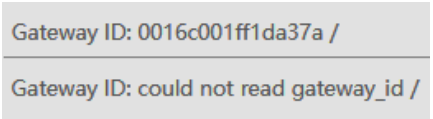


Figura 35: Indicador de Gateway ID antes y después del primer reinicio.

La primera medida tomada fue el restablecimiento del sistema operativo y reconfiguración del mismo. Esta acción derivó en resultados similares: instantes en los cuales la comunicación entre hardware y software del concentrador era exitosa y luego, pérdida de conexión.

Una siguiente medida más drástica fue utilizar el sistema operativo por defecto del RPi, Raspbian, e instalar el software del concentrador creado y recomendado por el fabricante [40]. Siguiendo los pasos indicados, el resultado fue similar y, evaluando los registros de eventos ocurridos durante la pérdida de conexión en ambos sistemas operativos, se pudo observar que el suceso tenía lugar cuando el software del concentrador le indicaba al hardware que se reinicie para aplicar los cambios nuevos en la configuración.

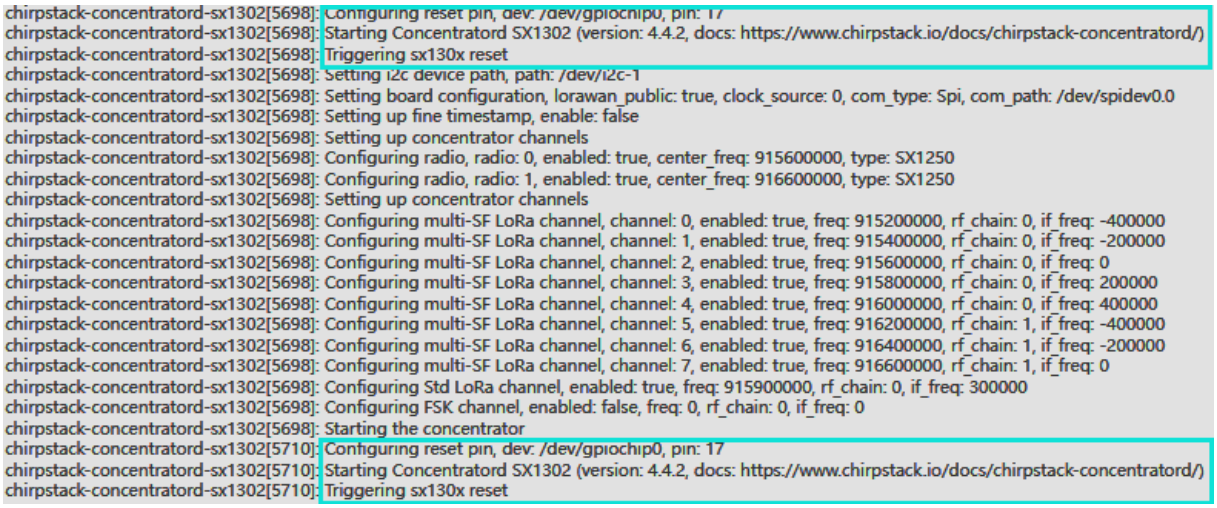


Figura 36: Reinicio constante del software observado en el registro de eventos.

Luego de una ardua investigación, se encontró que el conjunto RAK2287 + RAK2287 Pi HAT provienen de una versión comercial de un gateway utilizado para minar criptomonedas Helium. Este tipo de hardware difiere únicamente en la ubicación del pin de RESET de la placa Pi HAT, siendo el pin 25 para este tipo de concentrador y el pin 17 para la última versión de concentradores RAK2287 vendidos actualmente [41]. Cambiando la configuración del software provisto por RAK, se redefinió el pin de RESET de 17 a 25. Este cambio solucionó los problemas durante el reseteo del gateway manteniendo una comunicación constante entre el hardware y el software del concentrador.

Sin embargo, el software del concentrador provisto por ChirpStack, concentratord, no posee archivos de configuración modificables. Es decir, el programa integra todos estos valores de pin de reset de distintos fabricantes dentro de su código fuente [42]. Es por esto por lo que se decidió crear los archivos binarios compilados con la modificación pertinente. Para

esto, se utilizó WSL (Windows Subsystem for Linux) que permite instalar una distribución de Linux en Windows. Una vez instalada la distribución de Linux, se clonó el repositorio del concentratord y, modificando la línea 576 del archivo ubicado en "chirpstack-concentratord-sx1302/src/config/vendor/rak/rak2287.rs", se compiló el código fuente siguiendo los pasos en [43].

Una vez obtenido el programa compilado, se sobreescribió el archivo binario del concentratord incluido al instalar el OS de ChirpStack. Con esto se logró implementar el gateway sin resignar los objetivos fijados previamente.

4.3.2 Adaptación de las librerías existentes

Este problema es habitual en cualquier proyecto. A pesar de esto, se enfatiza en el mismo debido a las limitaciones en el hardware de los dispositivos finales al implementar código que requiere una precisión crítica en el reloj del CPU. El código utilizado para leer datos en el DHT11 no presenta errores al ser implementado en un ATMega328P con un clock externo. Sin embargo, su implementación en el ATMega32u4 utilizando el oscilador interno reduce la probabilidad de éxito de la medición. Es por esto por lo que se modifica la librería con el objetivo de minimizar la probabilidad de error. Se optimizó el código y se eliminaron las funcionalidades no utilizadas con el fin de ahorrar almacenamiento.

El código final cuenta con código con bucles infinitos potenciales en caso de desincronismo. Por lo tanto, se habilitó el WDT (Watchdog Timer) para el monitoreo y gestión de cuelgue del CPU en caso de ocurrir. El agregado del WDT requiere la definición de la subrutina de interrupción. En consecuencia, esto entra en conflicto con la librería de LowPower, ya que ésta igualmente define una subrutina de interrupción para el WDT. La solución finalmente fue modificar la librería LowPower eliminando la definición de la ISR, desplazando la misma al código principal de la aplicación.

5. Conclusiones

Durante el desarrollo integral del Proyecto de Ingeniería Electrónica, se integraron muchos de los conceptos vistos en la carrera, destacando fundamentalmente a: protocolos de comunicación LPWAN, sistemas embebidos, implementación de redes y servidores, programación de microcontroladores.

Los objetivos establecidos inicialmente se cumplieron de forma satisfactoria, resolviéndose los problemas encontrados durante el desarrollo de los mismos. Se observó la comunicación exitosa entre nodos y servidor de aplicación, incluyendo decodificación de datos, la presentación de la información en una aplicación de visualización como Excel y la automatización de tareas basada en eventos programables mediante Node-RED.

Principalmente, se destaca la factibilidad de implementación de una red LPWAN de bajo costo utilizando software open-source fundamentando su escalabilidad y la posibilidad de integración ante diferentes requerimientos y entornos.

Referencias

[1] “LoRa PHY | Semtech”. Semtech. [En línea]. Disponible: https://www.semtech.com/lora/what-is-lora

[2] “About LoRaWAN® - LoRa Alliance®”. LoRa Alliance®. [En línea]. Disponible: https://lora-alliance.org/about-lorawan/

[3] D.-H. Kim, E.-K. Lee y J. Kim, “Experiencing LoRa Network Establishment on a Smart Energy Campus Testbed”, Sustainability, vol. 11, n.º 7, p. 1917, marzo de 2019. [En línea]. Disponible: https://doi.org/10.3390/su11071917

[4] *Resolución 4653 ENACOM/19*, RES 4653/19, Ente Nacional de Comunicaciones, Ciudad de Buenos Aires, 2019. [En línea]. Disponible: https://www.enacom.gob.ar/multimedia/normativas/2019/res4653.pdf

[5] “LoRa contra Zigbee: La mejor tecnología para la conectividad IoT”. Tecnología IoT LoRaWAN para sus proyectos. [En línea]. Disponible: https://www.mokolora.com/es/lora-vs-zigbee-which-is-better/#:~:text=LoRa%20es%20superi

[6] “LoRa™ Modulation Basics”, 2 de mayo de 2015, AN1200.22. [En línea]. Disponible: https://semtech.my.salesforce.com/sfc/p/E0000000JelG/a/2R0000001OJk/yDEcfAkD9qEz6oG3PJryoHKas3UMsMDa3TFqz1UQOkM

[7] B. Al Homssi, K. Dakic, S. Maselli, H. Wolf, S. Kandeepan y A. Al-Hourani, “IoT Network Design Using Open-Source LoRa Coverage Emulator”, *IEEE Access*, vol. 9, pp. 53636–53646, 2021. [En línea]. Disponible: https://doi.org/10.1109/access.2021.3070976

[8] “LoRa Alliance Technical Committee Publications”, lora-alliance-technical-committee-publications.pdf. [En línea]. Disponible: https://resources.lora-alliance.org/technical-specifications/lora-alliance-technical-committee-publications

[9] *LoRaWAN® Regional Parameters RP002-1.0.4*, RP002-1.0.4, LoRa Alliance, Fremont, 2022. [En línea]. Disponible: https://resources.lora-alliance.org/technical-specifications/rp002-1-0-4-regional-parameters

[10] *LoRaWAN® 1.0.3 Specification*, TS001-1.0.3, LoRa Alliance, Beaverton, 2018. [En línea]. Disponible: https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-0-3

[11] Guidelines for Use of EUI, OUI, and CID. [En línea]. Disponible: https://standards.ieee.org/wp-content/uploads/import/documents/tutorials/eui.pdf

[12] *LoRaWAN Backend Interfaces Specification*, TS002-1.1.0, LoRa Alliance, Fremont, 2020. [En línea]. Disponible: https://resources.lora-alliance.org/technical-specifications/ts002-1-1-0-lorawan-backend-interfaces

[13] *LoRaWAN® 1.1 Specification*, TS001-1.1, LoRa Alliance, Beaverton, 2017. [En línea]. Disponible: https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1

[14] “ChirpStack open-source LoRaWAN Network Server”. ChirpStack open-source LoRaWAN Network Server. [En línea]. Disponible: https://www.chirpstack.io

[15] “Basic Concepts”. The LoRaWAN Network server for scale | The Things Industries. [En línea]. Disponible: https://www.thethingsindustries.com/docs/the-things-stack/concepts/

[16] “LORIOT - The LoRaWAN® Network Server and miotyÂ® service center Provider”. LORIOT - The LoRaWAN® Network Server and miotyÂ® service center Provider. [En línea]. Disponible: https://www.loriot.io/about.html

[17] “Estándar MQTT 5.0. Oasis Standard”. marzo de 2019. [En línea]. Disponible: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[18] “PostgreSQL: About”. PostgreSQL: The world's most advanced open source database. [En línea]. Disponible: https://www.postgresql.org/about

[19] “Raspberry Pi Documentation”. Raspberry Pi. [En línea]. Disponible: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[20] “Raspberry Pi 4 Model B Datasheet”, marzo de 2024. [En línea]. Disponible: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[21] “RAK2287 WisLink LPWAN Concentrator Datasheet”. RAK Documentation Center. [En línea]. Disponible: https://docs.rakwireless.com/Product-Categories/WisLink/RAK2287/Datasheet/

[22] “Cisco Wireless Gateway for LoRaWAN Data Sheet”. Cisco. [En línea]. Disponible: https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/datasheet-c78-737307.html#Productfeatures

[23] “MultiTech Conduit® Cellular Gateways”. MultiTech. [En línea]. Disponible: https://multitech.com/all-products/cellular/cellular-gateways/multitech-conduit/

[24] “Big LoRa32u4 boards topic”. The Things Network. [En línea]. Disponible: https://www.thethingsnetwork.org/forum/t/big-lora32u4-boards-topic/15273

[25] “SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver”, DS_SX1276-7-8-9_W_APP_V7, mayo de 2020. [En línea]. Disponible: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE

[26] Daniel W. Hart. “Electrónica de Potencia”-Editorial Universitaria. Cap 6 Convertidores CC-CC. pág 267-277

[27] Olimex. “DatasheetMT3608”. [En línea]. Disponible: https://www.olimex.com/Products/Breadboarding/BB-PWR-3608/resources/MT3608.pdf

[28] “DHT11 Technical Datasheet Translated Version”. mouser.com. [En línea]. Disponible: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[29] “HC-SR501 PIR MOTION DETECTOR”. Electronic Components and Accessories | MPJA.COM. [En línea]. Disponible: https://www.mpja.com/download/31227sc.pdf

[30] “HC-SR04”. ALLDATASHEET.ES. [En línea]. Disponible: https://www.alldatasheet.es/datasheet-pdf/download/1132203/ETC2/HC-SR04.html

[31] “Introduction - ChirpStack open-source LoRaWAN® Network Server documentation”. ChirpStack open-source LoRaWAN Network Server. [En línea]. Disponible: https://www.chirpstack.io/docs/chirpstack-gateway-os/index.html

[32] Datasheet Batería Litio 600mA 3.7V. [En línea]. Disponible: https://tienda.bricogeek.com/download/BAT-0012/Datasheet-105050-2P-lipo-battery.pdf

[33] “Getting started - ChirpStack open-source LoRaWAN® Network Server documentation”. ChirpStack open-source LoRaWAN Network Server. [En línea]. Disponible: https://www.chirpstack.io/docs/chirpstack-gateway-os/getting-started.html

[34] “GitHub - mcci-catena/arduino-lmic: LoraWAN-MAC-in-C library, adapted to run under the Arduino environment”. GitHub. [En línea]. Disponible: https://github.com/mcci-catena/arduino-lmic

[35] “arduino-lmic/examples/ttn-otaa-feather-us915/ttn-otaa-feather-us915.ino at master · mcci-catena/arduino-lmic”. GitHub. [En línea]. Disponible: https://github.com/mcci-catena/arduino-lmic/blob/master/examples/ttn-otaa-feather-us915/ttn-otaa-feather-us915.ino

[36] “LMIC-v4.1.0”. [En línea]. Disponible: https://github.com/mcci-catena/arduino-lmic/blob/master/doc/LMIC-v4.1.0.pdf

[37] “GitHub - dhrubasaha08/DHT11: DHT11 Arduino Library: A simple and efficient library for reading temperature and humidity data from the DHT11 sensor without dependencies.” GitHub. [En línea]. Disponible: https://github.com/dhrubasaha08/DHT11

[38] “GitHub - rocketscream/Low-Power: Low Power Library for Arduino”. GitHub. [En línea]. Disponible: https://github.com/rocketscream/Low-Power

[39] “Node-RED”. Node-RED. [En línea]. Disponible: https://nodered.org

[40] “GitHub - RAKWireless/rak_common_for_gateway”. GitHub.. [En línea]. Disponible: https://github.com/RAKWireless/rak_common_for_gateway

[41] “ERROR: Failed to set SX1250_0 in STANDBY_RC mode”. RAKwireless Forum. [En línea]. Disponible: https://forum.rakwireless.com/t/error-failed-to-set-sx1250-0-in-standby-rc-mode/6340/11

[42] “Where exactly to set Reset pin”. ChirpStack Community Forum. [En línea]. Disponible: https://forum.chirpstack.io/t/where-exactly-to-set-reset-pin/8898

[43] “GitHub - chirpstack/chirpstack-concentratord: Concentrator HAL daemon for LoRa gateways.” GitHub. [En línea]. Disponible: https://github.com/chirpstack/chirpstack-concentratord/tree/master

Anexo I: Código del Nodo de Temperatura y Humedad

```
#include <lmic.h>
#include <hal/hal.h>
#include <DHT11Mod.h>
#include <avr/wdt.h>
#include <LowPowerMod.h>

#define LMIC_CLOCK_ERROR_PERCENTAGE 3
#define TX_INTERVAL_8S 8 //Número de veces que el micro entra en modo PowerDown durante 8 segundos. Tiempo en dicho modo = TX_INTERVAL_2S * 8.
//#define APP_DEBUG 1

static const u1_t PROGMEM APPEUI[8]={ 0xcf, 0xfb, 0xd9, 0x60, 0x42, 0x57, 0xfb, 0x55 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

static const u1_t PROGMEM DEVEUI[8]={ 0x2e, 0xa2, 0x51, 0x07, 0x79, 0x39, 0x2f, 0xe5 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

static const u1_t PROGMEM APPKEY[16] = { 0x47, 0x2b, 0x66, 0xcc, 0x3b, 0x0b, 0x5d, 0x67, 0x17, 0x4d, 0xd9, 0x28, 0xc1, 0x8f, 0x64, 0x50 };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

static osjob_t sendjob;
static DHT11 dht11(6);
static uint8_t datos[4];
static uint8_t resultado;

volatile bool txing = false;

// Mapeo de pines
const lmic_pinmap lmic_pins = {
  .nss = 8,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 4,
  .dio = {7, 5, LMIC_UNUSED_PIN},
};
```

```
// Función llamada cuando ocurre un evento en la capa MAC.
void onEvent (ev_t ev) {
  #ifdef APP_DEBUG
  Serial.print(os_getTime());
  Serial.print(": ");
  #endif

  switch(ev) {
    case EV_JOINED:
      #ifdef APP_DEBUG
      Serial.println("Joined.");
      #endif

      // Deshabilita la verificación del enlace.
      LMIC_setLinkCheckMode(0);
      LMIC_setDrTxpow(0,20); // Se selecciona Data Rate 0 y potencia de
transmisión +20dBm

      // Prepara la primera transmisión del sensor.
      os_setTimedCallback(&sendjob, os_getTime() + ms2osticks(2), do_send);
      break;
    case EV_JOIN_FAILED:
      #ifdef APP_DEBUG
      Serial.println(F("EV_JOIN_FAILED"));
      #endif
      break;
    case EV_REJOIN_FAILED:
      #ifdef APP_DEBUG
      Serial.println(F("EV_REJOIN_FAILED"));
      #endif
      break;
    case EV_TXSTART:
      txing = true;
      break;
    case EV_TXCOMPLETE:
      #ifdef APP_DEBUG
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
```

```
      if (LMIC.txrxFlags & TXRX_ACK)
      Serial.println(F("Received ack"));
      if (LMIC.dataLen) {
        Serial.print(F("Received "));
        Serial.print(LMIC.dataLen);
        Serial.println(F(" bytes of payload"));
      }
      #endif

      txing = false;
      break;
    case EV_JOIN_TXCOMPLETE:
      #ifdef APP_DEBUG
      Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
      #endif
      break;
    default:
      break;
    }
}

// Función que lee valores del sensor y prepara la transmisión.
void do_send(osjob_t* j){
  // Lee valores de temperatura y humedad y se almacenan sin procesar.
  resultado = 1;
  for (uint8_t i = 0; i < 3; i++) {
    #ifdef APP_DEBUG
    Serial.println("Reading");
    #endif
    //Inicializa el WDT
    wdt_enable(WDTO_60MS);
    WDTCSR |= (1 << WDIE);

    resultado = dht11.readTemperatureHumidity(datos); // Inicia la
medición.

    wdt_disable(); // Se deshabilita el WDT
```

```
    if (resultado == 0 || i==2) break;


    unsigned long tiempo_inicio = millis();
    while (millis() - tiempo_inicio <= 1000) {}
  }


  // Prepara subida en el próximo tiempo posible.
  // Si luego de 3 intentos de medir TyHR no es posible obtener datos, se
envía el error code.
  resultado == 0 ? LMIC_setTxData2(1, datos, sizeof(datos), 0) :
LMIC_setTxData2(1, &resultado, 1, 0);


  #ifdef APP_DEBUG
  Serial.print("Packet queued. ");
  Serial.println(resultado);
  #endif
}


void setup() {
  #ifdef APP_DEBUG
  Serial.begin(9600);
  #endif
  // LMIC init
  os_init();
  // Resetea el estado de la capa MAC.
  LMIC_reset();


  LMIC_setClockError(LMIC_CLOCK_ERROR_PERCENTAGE * (MAX_CLOCK_ERROR /
100.0));  //Define el máximo error de clock (aumenta la duración de las
ventanas de recepción).
  LMIC_setAdrMode(0);
//Habilita el ADR.
  LMIC_selectSubBand(0);
//Selecciona la subbanda 0.
  LMIC_startJoining();
//Inicia el proceso de Activación OTA.
}


void loop() {
  os_runloop_once();
```

```
  if(!os_queryTimeCriticalJobs(sec2osticks(8*TX_INTERVAL_8S + 3)) &&
!txing){
    sleepSRT();
  }
}

void sleepSRT(){
  #ifdef APP_DEBUG
  Serial.println("Entrando en modo PowerDown");
  Serial.flush();
  Serial.end();
  delay(1000);

  USBCON |= (1 << FRZCLK);  //Detiene el clock de la interfaz USB
  PLLCSR &= ~(1 << PLLE);   //Deshabilit el PLL
  USBCON &= ~(1 << USBE);   //Finaliza la conexión USB
  #endif

  for (uint8_t i = 0; i < TX_INTERVAL_8S; i++) {
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  }

  #ifdef APP_DEBUG
  USBDevice.attach();  //Vuelve a habilitar la conexión USB
  delay(2000);
  Serial.begin(9600);
  Serial.println("Saliendo del modo PowerDown");
  Serial.flush();
  #endif

  os_setTimedCallback(&sendjob, os_getTime() + ms2osticks(2), do_send);
}



//Subrutina de interrupción del WDT
ISR(WDT_vect){
  if(dht11.isRunning()){
```

```
    dht11.setAbort();
  }
  else{
    wdt_disable();
  }
}
```

Anexo II: Código del Nodo de Detección de Movimiento

```
#include <lmic.h>
#include <hal/hal.h>
#include <LowPower.h>

#define LMIC_CLOCK_ERROR_PERCENTAGE 3
#define JOB_CHECK_TIME 360 // Intervalo de segundos sobre los cuales se verifican jobs pendientes.
#define PIN_DETECCION 10
//#define APP_DEBUG 1

static const u1_t PROGMEM APPEUI[8]={ 0x68, 0xec, 0xa9, 0x15, 0x7b, 0x9b, 0xa7, 0xc4 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

static const u1_t PROGMEM DEVEUI[8]={ 0x83, 0x7a, 0x15, 0xc6, 0xd6, 0x7e, 0x0e, 0x73 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

static const u1_t PROGMEM APPKEY[16] = { 0x5e, 0xa2, 0x98, 0xee, 0x6c, 0xbc, 0x04, 0x5e, 0xd5, 0xef, 0x8c, 0xbb, 0xfe, 0x77, 0x0f, 0xbf };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

static osjob_t sendjob;
uint8_t msj = 1;

volatile bool txing = false;
volatile bool sleep = false;

// Mapeo de pines
const lmic_pinmap lmic_pins = {
  .nss = 8,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 4,
  .dio = {7, 5, LMIC_UNUSED_PIN},
};

// Función llamada cuando ocurre un evento en la capa MAC.
void onEvent (ev_t ev) {
```

```
  #ifdef APP_DEBUG
  Serial.print(os_getTime());
  Serial.print(": ");
  #endif

  switch(ev) {
    case EV_JOINED:
      #ifdef APP_DEBUG
      Serial.println("Joined.");
      #endif

      // Deshabilita la verificación del enlace.
      LMIC_setLinkCheckMode(0);
      LMIC_setDrTxpow(0,20);    // Se selecciona Data Rate 0 y potencia de
transmisión +20dBm
      txing = false;

      break;
    case EV_JOIN_FAILED:
      #ifdef APP_DEBUG
      Serial.println(F("EV_JOIN_FAILED"));
      #endif
      break;
    case EV_REJOIN_FAILED:
      #ifdef APP_DEBUG
      Serial.println(F("EV_REJOIN_FAILED"));
      #endif
      break;
    case EV_TXSTART:
      txing = true;
      break;
    case EV_TXCOMPLETE:
      #ifdef APP_DEBUG
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));

      if (LMIC.txrxFlags & TXRX_ACK)
      Serial.println(F("Received ack"));
      if (LMIC.dataLen) {
```

```
        Serial.print(F("Received "));
        Serial.print(LMIC.dataLen);
        Serial.println(F(" bytes of payload"));
      }
      #endif

      txing = false;
      break;
    case EV_JOIN_TXCOMPLETE:
      #ifdef APP_DEBUG
      Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
      #endif
      break;
    default:
      break;
    }
}

// Función que lee valores del sensor y prepara la transmisión.
void do_send(osjob_t* j){
  // Prepara subida en el próximo tiempo posible.
  LMIC_setTxData2(1, &msj, 1, 0); // Se envía un 1 indicando detección de
movimiento.

  #ifdef APP_DEBUG
  Serial.print("Packet queued. ");
  #endif
}

void setup() {
  #ifdef APP_DEBUG
  Serial.begin(9600);
  #endif
  pinMode(PIN_DETECCION, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
```

```
  PCMSK0 |= (1 << PCINT6);
// Habilita las interrupciones por cambio de estado para el pin 9
  // LMIC init
  os_init();
  // Resetea el estado de la capa MAC.
  LMIC_reset();


  LMIC_setClockError(LMIC_CLOCK_ERROR_PERCENTAGE * (MAX_CLOCK_ERROR /
100.0));  // Define el máximo error de clock (aumenta la duración de las
ventanas de recepción).
  LMIC_setAdrMode(0);
// Habilita el ADR.
  LMIC_selectSubBand(0);
// Selecciona la subbanda 0.
  LMIC_startJoining();
// Inicia el proceso de Activación OTA.



}


void loop() {
  os_runloop_once();


  if(os_queryTimeCriticalJobs(sec2osticks(JOB_CHECK_TIME)) == 0 && !txing){
    sleepSRT();
  }
}


void sleepSRT(){


  #ifdef APP_DEBUG
  Serial.println("Entrando en modo PowerDown");
  Serial.flush();
  Serial.end();
  delay(1000);


  USBCON |= (1 << FRZCLK);  // Detiene el clock de la interfaz USB
  PLLCSR &= ~(1 << PLLE);   // Deshabilit el PLL
  USBCON &= ~(1 << USBE);   // Finaliza la conexión USB
```

```
  #endif

  PCICR |= (1 << PCIE0); // Se habilitan las interrupciones por cambio de
estado
  sleep = true;
  while(sleep){
    LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);
  }
  PCICR &= ~(1 << PCIE0); // Se deshabilitan las interrupciones por cambio
de estado

  #ifdef APP_DEBUG
  USBDevice.attach();  // Vuelve a habilitar la conexión USB
  delay(2000);
  Serial.begin(9600);
  Serial.println("Saliendo del modo PowerDown");
  Serial.flush();
  #endif

  os_setTimedCallback(&sendjob, os_getTime() + ms2osticks(2), do_send); //
Puesta en cola del job del sensor
}

// Subrutina de interrupción del pin de detección
ISR(PCINT0_vect){
  delayMicroseconds(20);
  if(digitalRead(PIN_DETECCION) == HIGH){
    sleep = false;
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else{
    digitalWrite(LED_BUILTIN, LOW);
  }
}
```

Anexo III: Código del Nodo de Distancia

```
#include <lmic.h>
#include <hal/hal.h>
#include <LowPower.h>

#define LMIC_CLOCK_ERROR_PERCENTAGE 3
#define TX_INTERVAL_8S 8 // Número de veces que el micro entra en modo PowerDown durante 8 segundos. Tiempo en dicho modo = TX_INTERVAL_2S * 8.
#define TRIGGER 6
#define ECHO 9
#define TIMEOUT_PULSE 100000 // Número máximo de microsegundos en los que la función espera el cambio de nivel lógico.
//define APP_DEBUG 1

static const u1_t PROGMEM APPEUI[8]={ 0xab, 0xdb, 0xdc, 0x55, 0xbe, 0xae, 0x0b, 0xdb };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

static const u1_t PROGMEM DEVEUI[8]={ 0xe1, 0x78, 0x02, 0xdd, 0x00, 0x27, 0x0e, 0x68 };
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}

static const u1_t PROGMEM APPKEY[16] = { 0x6c, 0x50, 0x86, 0x23, 0x7d, 0xc8, 0x66, 0x88, 0x9c, 0x23, 0x1d, 0x88, 0xdb, 0x77, 0x6f, 0x50 };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

static osjob_t sendjob;
static unsigned long tiempo_echo;

volatile bool txing = false;

// Mapeo de pines
const lmic_pinmap lmic_pins = {
  .nss = 8,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 4,
  .dio = {7, 5, LMIC_UNUSED_PIN},
};
```

```
// Función llamada cuando ocurre un evento en la capa MAC.
void onEvent (ev_t ev) {
  #ifdef APP_DEBUG
  Serial.print(os_getTime());
  Serial.print(": ");
  #endif

  switch(ev) {
    case EV_JOINED:
      #ifdef APP_DEBUG
      Serial.println("Joined.");
      #endif

      //Deshabilita la verificación del enlace.
      LMIC_setLinkCheckMode(0);
      /*
      LMIC_setAdrMode(0);       //Habilita el ADR.
      LMIC_setDrTxpow(0,20);    //Se selecciona Data Rate 0 y potencia de
transmisión +20dBm
      */
      LMIC_setAdrMode(1);

      //Prepara la primer transmisión del sensor.
      os_setTimedCallback(&sendjob, os_getTime() + ms2osticks(2), do_send);
      break;
    case EV_JOIN_FAILED:
      #ifdef APP_DEBUG
      Serial.println(F("EV_JOIN_FAILED"));
      #endif
      break;
    case EV_REJOIN_FAILED:
      #ifdef APP_DEBUG
      Serial.println(F("EV_REJOIN_FAILED"));
      #endif
      break;
    case EV_TXSTART:
      txing = true;
      break;
```

```
    case EV_TXCOMPLETE:
      #ifdef APP_DEBUG
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));

      if (LMIC.txrxFlags & TXRX_ACK)
      Serial.println(F("Received ack"));
      if (LMIC.dataLen) {
        Serial.print(F("Received "));
        Serial.print(LMIC.dataLen);
        Serial.println(F(" bytes of payload"));
      }
      #endif

      txing = false;
      break;
    case EV_JOIN_TXCOMPLETE:
      #ifdef APP_DEBUG
      Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
      #endif
      break;
    default:
      break;
    }
}

//Función que lee valores del sensor y prepara la transmisión.
void do_send(osjob_t* j){
  #ifdef APP_DEBUG
  Serial.println("Reading");
  #endif

  tiempo_echo = readDistance(); //Inicia la medición.

  // Prepara subida en el proximo tiempo posible.
  LMIC_setTxData2(1, (xref2u1_t)&tiempo_echo, sizeof(tiempo_echo), 0);

  #ifdef APP_DEBUG
  Serial.print("Packet queued. ");
```

```
  #endif
}

void setup() {
  pinMode(TRIGGER, OUTPUT);
  digitalWrite(TRIGGER, LOW);
  pinMode(ECHO, INPUT);

  #ifdef APP_DEBUG
  Serial.begin(9600);
  #endif
  // LMIC init
  os_init();
  // Resetea el estado de la capa MAC.
  LMIC_reset();

  LMIC_setClockError(LMIC_CLOCK_ERROR_PERCENTAGE * (MAX_CLOCK_ERROR /
100.0));  //Define el máximo error de clock (aumenta la duración de las
ventanas de recepción).
  LMIC_selectSubBand(0);
//Selecciona la subbanda 0.
  LMIC_startJoining();
//Inicia el proceso de Activación OTA.
}

void loop() {
  os_runloop_once();

  if(!os_queryTimeCriticalJobs(sec2osticks(8*TX_INTERVAL_8S + 3)) &&
!txing){
    sleepSRT();
  }
}

void sleepSRT(){
  #ifdef APP_DEBUG
  Serial.println("Entrando en modo PowerDown");
  Serial.flush();
  Serial.end();
```

```
  delay(1000);

  USBCON |= (1 << FRZCLK);  //Detiene el clock de la interfaz USB
  PLLCSR &= ~(1 << PLLE);   //Deshabilit el PLL
  USBCON &= ~(1 << USBE);   //Finaliza la conexión USB
  #endif

  for (uint8_t i = 0; i < TX_INTERVAL_8S; i++) {
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  }

  #ifdef APP_DEBUG
  USBDevice.attach();  //Vuelve a habilitar la conexión USB
  delay(2000);
  Serial.begin(9600);
  Serial.println("Saliendo del modo PowerDown");
  Serial.flush();
  #endif

  os_setTimedCallback(&sendjob, os_getTime() + ms2osticks(2), do_send);
}

unsigned long readDistance(){
  digitalWrite(TRIGGER, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER,LOW);

  return pulseIn(ECHO, HIGH, TIMEOUT_PULSE);
}
```

Anexo IV: Código de Librería DHT11 Modificada

```
/**
 * DHT11.cpp
 * Library for reading temperature and humidity from the DHT11 sensor.
 *
 * Author: Dhruba Saha
 * Version: 2.1.0
 * License: MIT
 */

#include "DHT11Mod.h"
#define ERROR_CHECKSUM 254
#define ERROR_TIMEOUT1 253
#define ERROR_TIMEOUT2 252
#define TIMEOUT_DURATION 2000
/**
 * Constructor for the DHT11 class.
 * Initializes the pin to be used for communication and sets it to output
mode.
 *
 * @param pin: Digital pin number on the Arduino board to which the DHT11
sensor is connected.
 */
DHT11::DHT11(uint8_t pin) : _pin(pin), _abort(false), _running(false){
  pinMode(_pin, OUTPUT);
  digitalWrite(_pin, HIGH);
}

/**
 * Reads raw data from the DHT11 sensor.
 * This method handles the direct communication with the DHT11 sensor and
retrieves the raw data.
 * It's used internally by the readTemperature, readHumidity, and
readTemperatureHumidity methods.
 *
 * @param data: An array of bytes where the raw sensor data will be stored.
 *              The array must be at least 5 bytes long, as the DHT11
sensor returns 5 bytes of data.
```

```
 * @return: Returns 0 if the data is read successfully and the checksum
matches.
 *          Returns DHT11::ERROR_TIMEOUT if the sensor does not respond or
communication times out.
 *          Returns DHT11::ERROR_CHECKSUM if the data is read but the
checksum does not match.
 */
uint8_t DHT11::readRawData(byte data[5]){
  startSignal();
  unsigned long timeout_start = micros();

  while (digitalRead(_pin) == HIGH){
    if (micros() - timeout_start > TIMEOUT_DURATION){
      return ERROR_TIMEOUT1;
    }
  }

      while(digitalRead(_pin) == LOW){}

      for (uint8_t i = 0; i < 5; i++){
            data[i] = readByte();
      }
      if (data[4] == ((data[0] + data[1] + data[2] + data[3]) & 0xFF)){
            return 0; // Success
      }
      else{
            return ERROR_CHECKSUM;
      }
  return ERROR_TIMEOUT2;
}

/**
 * Reads a byte of data from the DHT11 sensor during the communication
process.
 *
 * @return: A byte of data read from the sensor.
 */
byte DHT11::readByte(){
  byte value = 0;
```

```
      while (digitalRead(_pin) == HIGH){
            if(_abort)
                  return 0;
      }
  for (int8_t i = 7; i >= 0; i=i-1){
    while (digitalRead(_pin) == LOW){}
            delayMicroseconds(26);
    if (digitalRead(_pin) == HIGH){
      value |= (1 << i);
                  if(i !=0){
                        while (digitalRead(_pin) == HIGH){
                              if(_abort)
                                    return 0;
                        }
                  }
    }
  }
  return value;
}

/**
 * Sends a start signal to the DHT11 sensor to initiate a data read.
 * This involves setting the data pin low for a specific duration, then
high,
 * and finally setting it to input mode to read the data.
 */
void DHT11::startSignal(){
  pinMode(_pin, OUTPUT);
  digitalWrite(_pin, LOW);
  delay(20);
  pinMode(_pin, INPUT);
}

/**
 * Reads and returns the temperature and humidity from the DHT11 sensor.
 * Utilizes the readRawData method to retrieve raw data from the sensor and
then extracts
 * both temperature and humidity from the data array.
```

```
 *
 * @param temperature: Reference to a variable where the temperature value
will be stored.
 * @param humidity: Reference to a variable where the humidity value will
be stored.
 * @param temperatureDEC: Reference to a variable where the decimal part of
temperature value will be stored.
 * @param humidityDEC: Reference to a variable where the decimal part of
humidity value will be stored.
 * @return: An integer representing the status of the read operation.
 *          Returns 0 if the reading is successful, DHT11::ERROR_TIMEOUT if
a timeout occurs,
 *          or DHT11::ERROR_CHECKSUM if a checksum error occurs.
 */
uint8_t DHT11::readTemperatureHumidity(uint8_t values[4]){
      _running = true;
  byte data[5];
  uint8_t error = readRawData(data);
      _abort = false;
      _running = false;
  if (error != 0){
    return error;
  }
  values[0] = data[2];
      values[1] = data[0];
      values[2] = data[3];
      values[3] = data[1];
  return 0; // Indicate success
}

void DHT11::setAbort(){
      _abort = true;
}

bool DHT11::isRunning(){
      return _running;
}
```

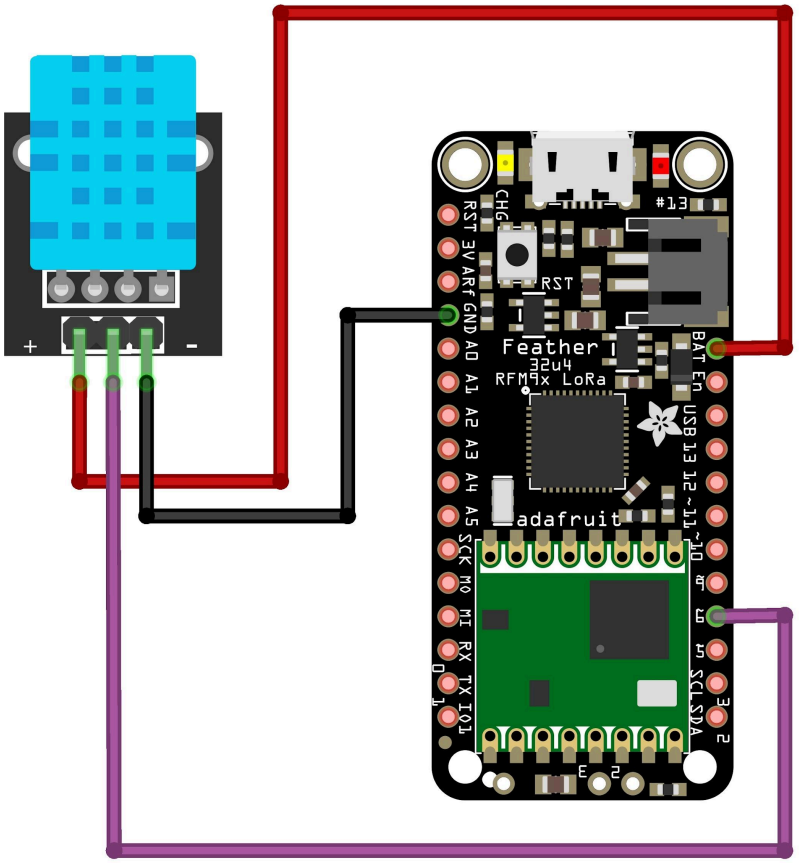
Anexo V: Código de la función de Node-RED

```
// Código ejecutado durante la iniciación del nodo

// Seteo de los umbrales
context.set("temperaturaUmbral", 40); // En grados Celsius
context.set("humedadUmbral", 80); // En % de humedad relativa
context.set("distancia_cmUmbral", 100); // En centímetros

// Código ejecutado por cada mensaje recibido

let nombreNodo = msg.payload.deviceInfo.deviceName;

// Se compara el identificador del perfil del dispositivo con
// el identificador del perfil del nodo de movimiento
if (msg.payload.deviceInfo.deviceProfileId ==
"d4326f08-7af7-48e2-9dbc-affb175eec75") {
    msg = {
        "topic": "ALERTA: Red LoRaWAN - " + nombreNodo,
        "payload": "Se ha disparado el detector de movimiento.",
        "send": true
    };
}
else {
    let data = msg.payload.object;
    let send = false;
    let keys = Object.keys(data);
    let key;
    let valorActual;
    let umbral;

    // Se recorre la lista de propiedades generadas por el codec
    // y se compara con los umbrales.
    // Cabe destacar que una vez detectado un valor que supere el umbral
    // el programa deja de leer propiedades. Esto no detectaría más de una
    // propiedad que supera su umbral (como en el caso del nodo de
temperatura
    // y humedad).
```

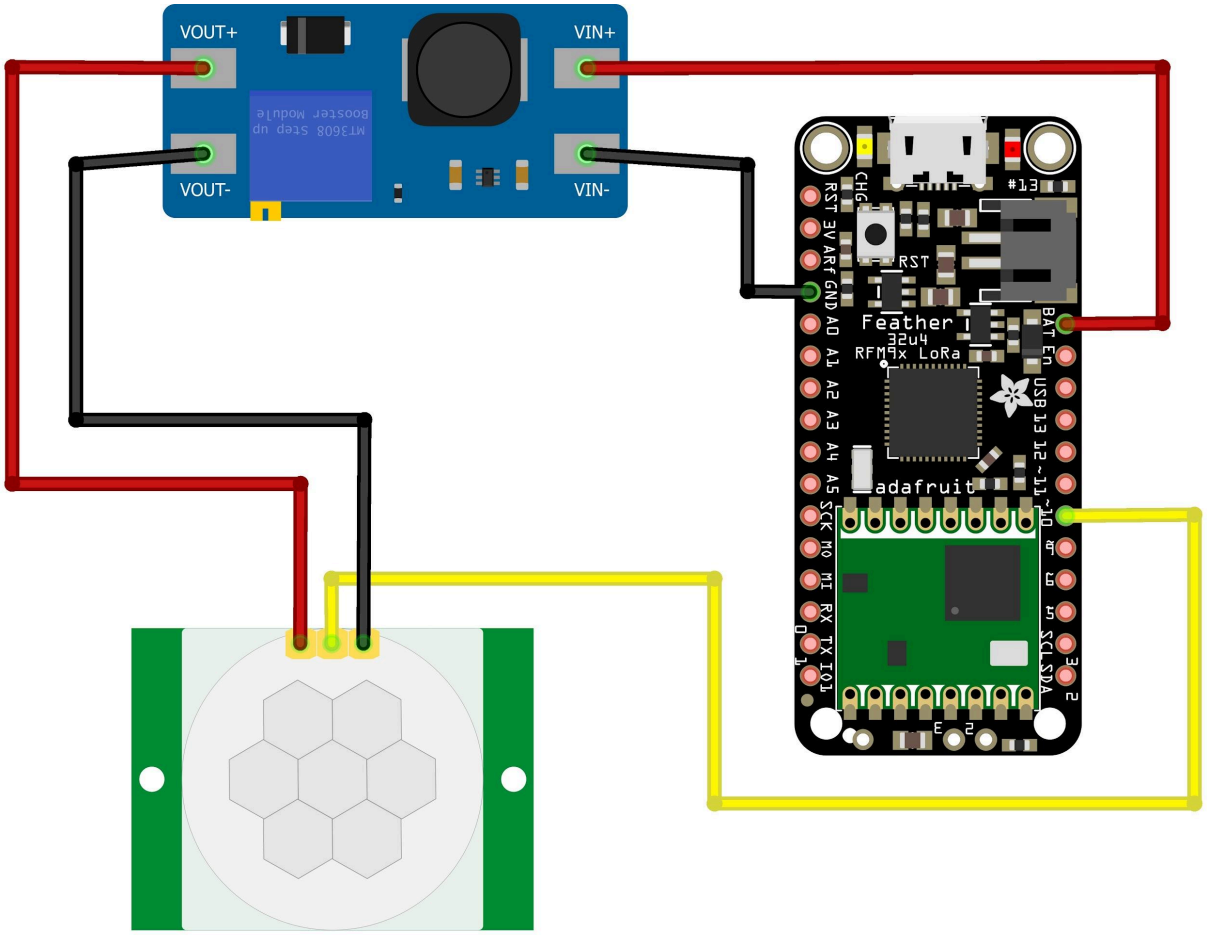
```
    for(let i = 0; i < keys.length; i++){
        key = keys[i];
        umbral = context.get(key + "Umbral");
        valorActual = data[key];
        if(valorActual >= umbral){
            send = true;
            break;
        }
    }

    msg = {
        "topic": "ALERTA: Red LoRaWAN - " + nombreNodo,
        "payload": "El valor " + key + " ha superado el umbral de " +
umbral + " preestablecido. El valor actual es " + valorActual,
        "send": send
    }
}

return msg;
```

Anexo VI: Diagramas Esquemáticos

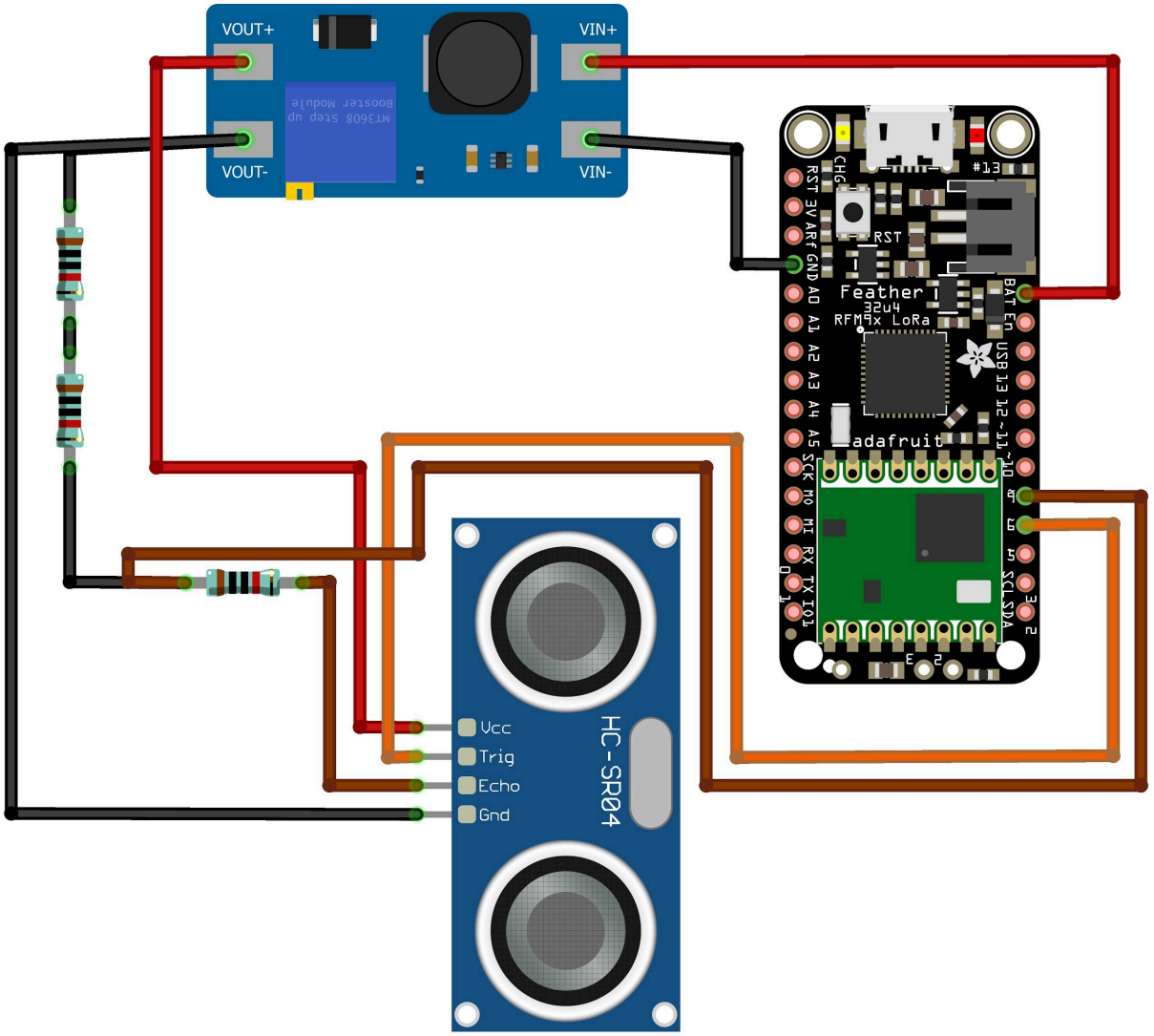
Nodo de Temperatura y Humedad



Nodo de Detección de Movimiento

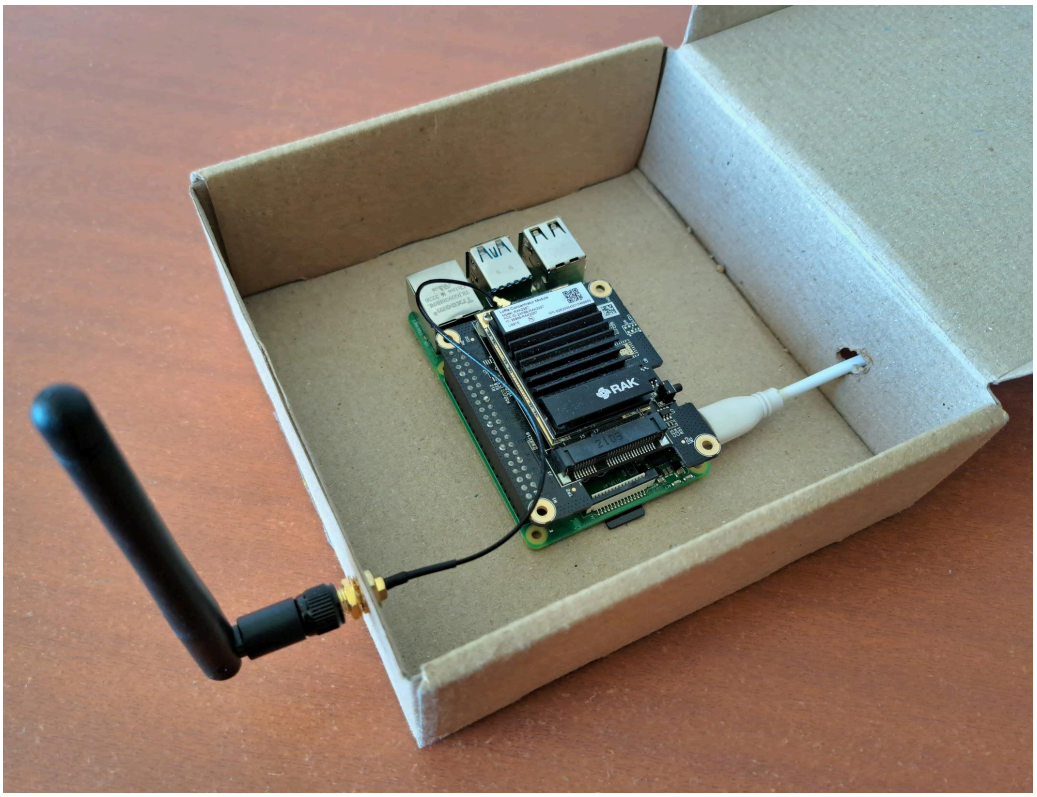


Nodo de Distancia

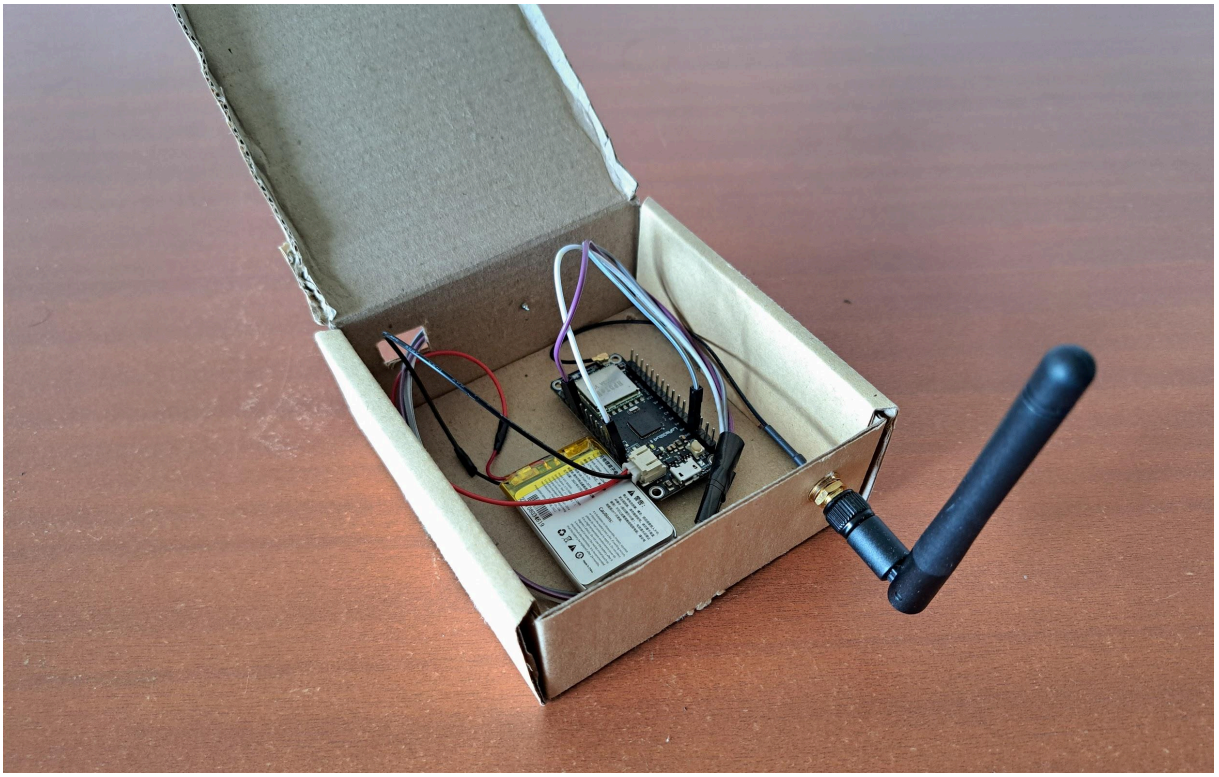


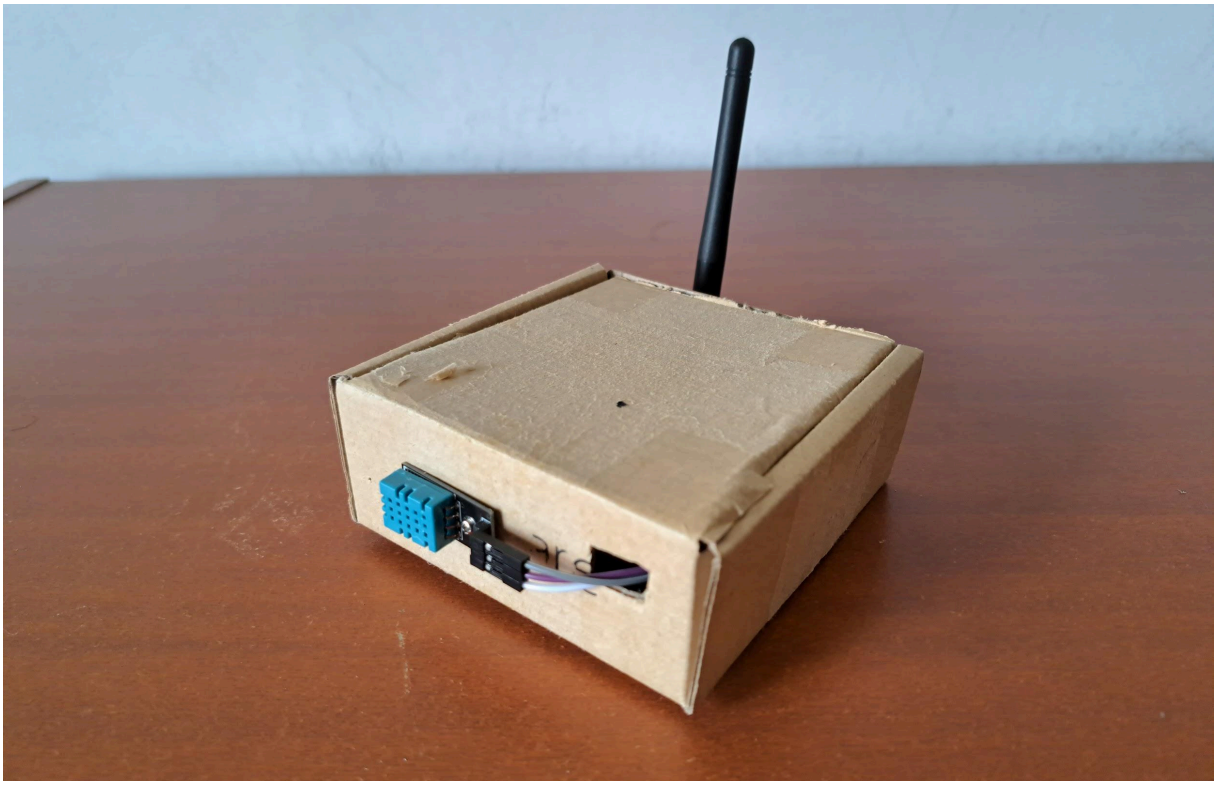
Anexo VII: Imágenes

Raspberry Pi 4B - RAK2287

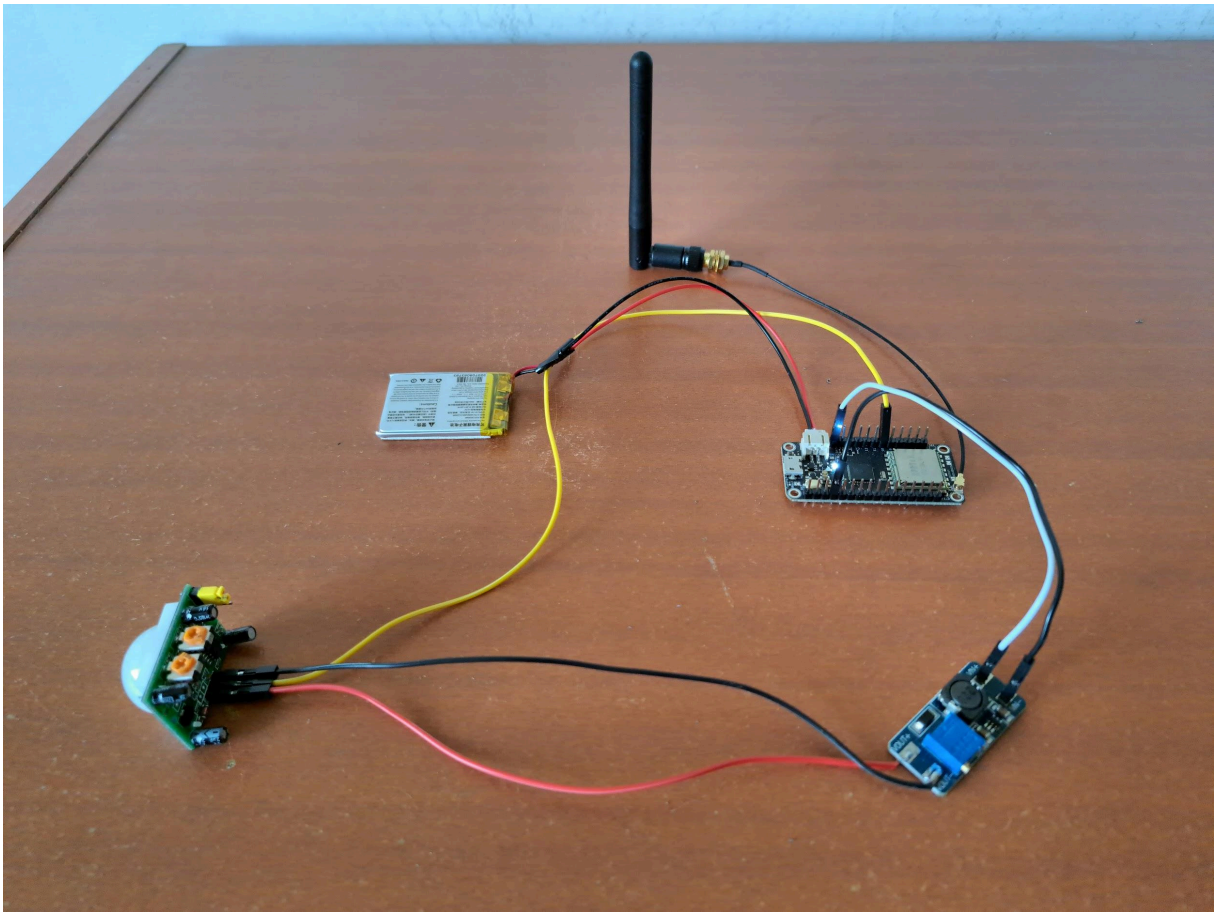


Nodo de Temperatura y Humedad





Nodo de Detección de Movimiento



Nodo de Distancia