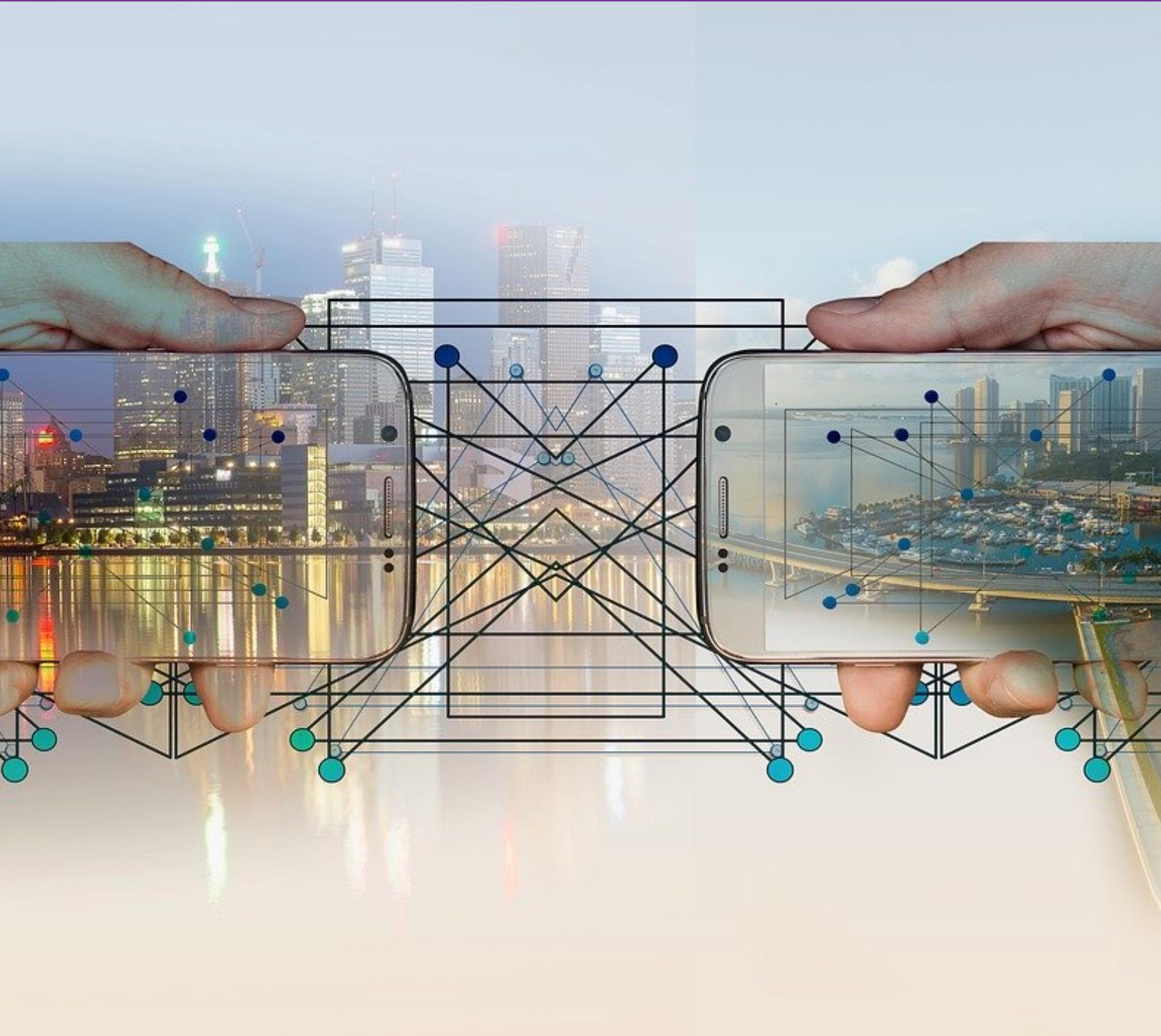




# Desarrollo de prototipo de solución IoT basado en tecnología LoRaWAN

Práctica Profesional Supervisada





**Desarrollo de prototipo de solución IoT  
basado en tecnología LoRaWAN**

Práctica Profesional Supervisada  
Diciembre, 2019

Por  
Bibiana Mollinedo Rivadeneira

Tutor por empresa  
Jonathan Hernán Sánchez

Tutor por universidad  
Damián Héctor Primo

Licencia: Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

Foto de portada: Gerd Altmann

Presentado en: UNRC, Facultad de Ingeniería, Ruta Nac. 36 - Km. 601, X5804BYA  
Río Cuarto, Córdoba Argentina  
<https://github.com/brivadeneira/LoRaWANprototype>

## Aprobación

El presente informe deja sentado los objetivos de la práctica profesional supervisada según las tareas y procedimientos establecidos en el plan de trabajos de la misma, durante el período con inicio 29 de Agosto hasta 6 de Diciembre del corriente año bajo el convenio con la empresa FONEXA S.A. y la supervisión de tutor por empresa, Jonathan Hernan Sanchez y tutor por Universidad, Damián Héctor Primo.

Bibiana Mollinedo Rivadeneira-

.....  
*Firma*

.....  
*Fecha*

## Resumen

Se desarrolla un prototipo de solución IoT (*Internet of Things*) con tecnología de base **LoRaWAN**, conformado por un **módulo de adquisición de datos, uno de comunicación/transmisión inalámbrica de los mismos, así como un módulo de procesamiento y visualización** para el usuario final.

Previo proceso de investigación a nivel de hardware así como de software, respecto a los requerimientos, opciones disponibles en el mercado local, haciendo hincapié en los protocolos de comunicación implicados, recopilando información necesaria para la toma de decisiones de usos e implementación de tecnologías.

Se realiza la puesta en marcha y prueba de concepto, se establecen conclusiones y proyecciones según soluciones comerciales potenciales, relacionadas a *sensado de nivel de llenado en contenedores de residuo para optimización de rutas de recolección y monitoreo de sistemas de riego rurales*.

## Agradecimientos

**Lucas Bellomo**, compañero de los momentos más importantes.  
Por estar a mi lado siempre.

**Jonathan Sánchez**, tutor y amigo.  
Por compartir pasiones, proyectos y mates.

**Damián Primo**, tutor y colega.  
Por el apoyo y guía.

# Índice general

Preface . . . . .	II
Resumen . . . . .	III
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
<b>2. Descripción de la Empresa</b>	<b>3</b>
2.1. Resumen . . . . .	3
2.2. Datos institucionales . . . . .	3
2.3. Contexto . . . . .	3
2.4. Descripción de las tareas . . . . .	4
2.5. Descripción del proyecto . . . . .	5
<b>3. Metodología</b>	<b>6</b>
<b>4. Fundamentos teóricos</b>	<b>7</b>
4.1. Sobre LoRa/LoRaWAN . . . . .	7
<b>5. Análisis de componentes y tecnologías</b>	<b>19</b>
5.1. Requerimientos mínimos . . . . .	19
5.2. Hardware disponible . . . . .	20
5.3. Endpoints . . . . .	20
5.4. End-points y Gateways . . . . .	21
5.5. Network/Application Server . . . . .	21
5.6. Tecnologías definitivas . . . . .	24
5.7. Gateways . . . . .	27
5.8. Servidores de red y aplicación . . . . .	27
5.9. Formato Frame . . . . .	28
<b>6. Procedimiento</b>	<b>29</b>
6.1. Conectividad punto a punto LoRa . . . . .	29
6.2. Implementación gateway LoRa . . . . .	29
6.3. Implementación nodos LoRa . . . . .	29
6.4. Implementación servidores de red y aplicación . . . . .	29
6.5. Simulación de paquetes LoRa . . . . .	29
6.6. Desarrollo e implementación de API MQTT . . . . .	29
6.7. Desarrollo e implementación de dashboard . . . . .	30
<b>7. Desarrollo</b>	<b>31</b>
7.1. Conectividad punto a punto LoRa con ESP32 . . . . .	31
7.2. Implementación gateway LoRa ESP32 . . . . .	32
7.3. Implementación nodos LoRa . . . . .	32
7.4. Implementación servidores de red y aplicación . . . . .	33
7.5. Simulación paquetes LoRa . . . . .	33
7.6. Desarrollo e implementación de API MQTT . . . . .	34
7.7. Desarrollo e implementación de dashboard . . . . .	34

<b>8. Resultados</b>	<b>35</b>
8.1. Red LoRa punto a punto . . . . .	35
8.2. Red LoRaWAN . . . . .	36
8.3. API y dashboard . . . . .	37
8.4. Análisis de los resultados . . . . .	39
<b>9. Conclusiones</b>	<b>40</b>
9.1. Sobre el proyecto . . . . .	40
9.2. Aspectos laborales . . . . .	40
9.3. Aspectos profesionales . . . . .	40
<b>10. Bibliografía</b>	<b>41</b>
10.1. Documentos técnicos . . . . .	41
10.2. Libros . . . . .	41
10.3. Artículos académicos . . . . .	41
10.4. Repositorios digitales . . . . .	42
10.5. Guías, tutoriales y artículos en blogs . . . . .	42

# Índice de figuras

2.1. Logo Fonexa.	3
2.2. Organigrama Fonexa.	4
2.3. Diagrama de bloques actividades.	4
2.4. Diagrama global del proyecto con tecnologías tentativas.	5
4.1. Arquitectura general de una red LoRaWAN.	8
4.2. Ocupación espectral LoRa..	9
4.3. Formato frame LoRa.	10
4.4. Clases de endpoint.	13
4.5. Diagrama comparativo endpoints.	14
4.6. Modos de activación.	15
4.7. Diagrama de bloques módulo LoRa.	17
5.1. Diagrama de tecnologías definitivas.	25
5.2. Red LoRaWAN: esp32-oled, arduino-uno...	26
7.1. Microcontrolador ESP32 oled LoRa.	31
7.2. Simulador de sistemas embebidos.	33
7.3. Paquetes simulados.	34
8.1. Red LoRa punto a punto.	35
8.2. Red LoRaWAN: esp32 oled, arduino uno.	36
8.3. Paquetes de red LoRaWAN.	37
8.4. API datos LoRaWAN.	37
8.5. Gráfico de datos con filtro por dispositivo esp32.	38
8.6. Gráfico de datos con filtro por dispositivo arduino-uno.	38
8.7. Gráfico de datos con filtro por todos los dispositivos.	39

# Índice de tablas

2.1. Días y horarios . . . . .	3
4.1. Tasas de bit . . . . .	12
4.2. Aplicaciones según tipo de nodo LoRaWAN . . . . .	14
4.3. Modos de activación nodos LoRaWAN . . . . .	16
5.1. Cuadro comparativo servidores, parte 1 . . . . .	21
5.2. Cuadro comparativo servidores, parte 4 . . . . .	22
5.3. Cuadro comparativo servidores, parte 2 . . . . .	23
5.4. Cuadro comparativo servidores, parte 4 . . . . .	23

# 1 Introducción

Los elementos de la cotidianidad se transformaron al cabo de muy poco tiempo y de manera exponencial, en dispositivos a conectar a la red, enviando y recibiendo dimensiones gigantescas de datos. Sin embargo existen aún áreas sin acceso a internet, pero con no menos importantes, ni en menor cantidad, datos a aportar.

Las redes LoRaWAN, en su arquitectura híbrida de comunicación, ofrece alcances significativos, bajos requerimientos de potencia, versatilidad en la implementación con bajos costos.

Ante la tarea de diseñar una red de este tipo, y más específicamente prepararse para la toma de decisiones desde una perspectiva técnica, ingenieril y lo más agnóstica posible ante diferentes escenarios y por ende variables a contemplar, se opta por experiencias empíricas de recopilación de datos, análisis y conclusiones en base a una prueba de concepto y prototipo funcional con vista a escalar a un producto comercial.

## 1.1 Motivación

El proyecto conformado por el conjunto de tareas y procesos de **investigación y desarrollo** de las tecnologías en auge de Internet de las cosas y la construcción de un prototipo surge en la empresa FONEXA, impulsado por Jonathan, del área de desarrollo de software.



## 2 Descripción de la Empresa

FONEXA S.A.



Figura 2.1: Logo Fonexa.

### 2.1 Resumen

Empresa joven fundada por ingenieros en telecomunicaciones con espíritu emprendedor. Experiencia en la industria tras haber formado parte de las principales compañías de telecomunicaciones del País, enfocada principalmente en comprender las necesidades de los clientes y de esta forma ofrecerles la solución, servicio o producto que mejor se adapte a cada situación.

### 2.2 Datos institucionales

- **Domicilio:** Estados Unidos N° 770, Banda Norte, Río Cuarto, Córdoba.
- **Teléfono:** (0358)4621093
- **Rubro:** Telecomunicaciones

### 2.3 Contexto

- **Área de desarrollo de práctica profesional supervisada:** I+D, bajo tutoría de encargado de desarrollo de software de la empresa.
- **Días y horas de trabajo:**

Día	Desde	Hasta	Cant de hs
Lun	8.30hs	13.00hs	4,5
Mar	8.30hs	13.00hs	4,5
Jue	8.30hs	13.00hs	4,5
Vie	14.00hs	17.30hs	3,5

Tabla 2.1: Días y horarios

NOTA: Durante la realización de la práctica profesional se flexibilizaron días, horarios y modalidades de trabajo en función de las tareas, al rededor de los pactados inicialmente. Ver anexo: *Planilla de horas y Diagrama de Gantt*

- **Descripción de actividades:** Se implementan soluciones de telecomunicaciones de todo tipo, tanto de infraestructura de redes de diferentes tecnologías, como desarrollo de software. En particular, la práctica profesional se realiza como continuación de la iniciativa de impulsar nuevos proyectos y soluciones de Internet de las cosas.

- **Organigrama:**

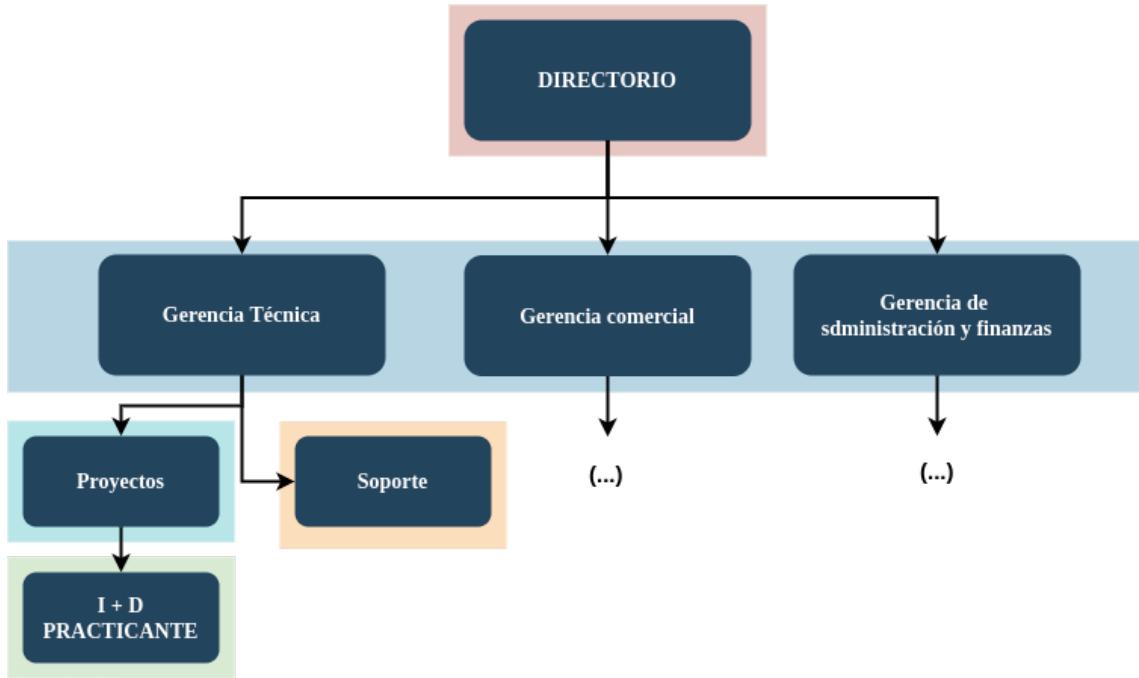


Figura 2.2: Organigrama Fonexa.

## 2.4 Descripción de las tareas

Se llevan a cabo tareas de investigación y desarrollo de un prototipo de solución de Internet de las cosas con tecnología LoRaWAN acorde a plan de trabajo, con dedicación diaria de cuatro horas reloj, y según el siguiente diagrama de procesos y tareas:



Figura 2.3: Diagrama de bloques actividades.

## 2.5 Descripción del proyecto

El proyecto puede ser descripto desde un enfoque general/global. Para su planificación y abordaje, se definen *"módulos"* según las funcionalidades requeridas como sigue:

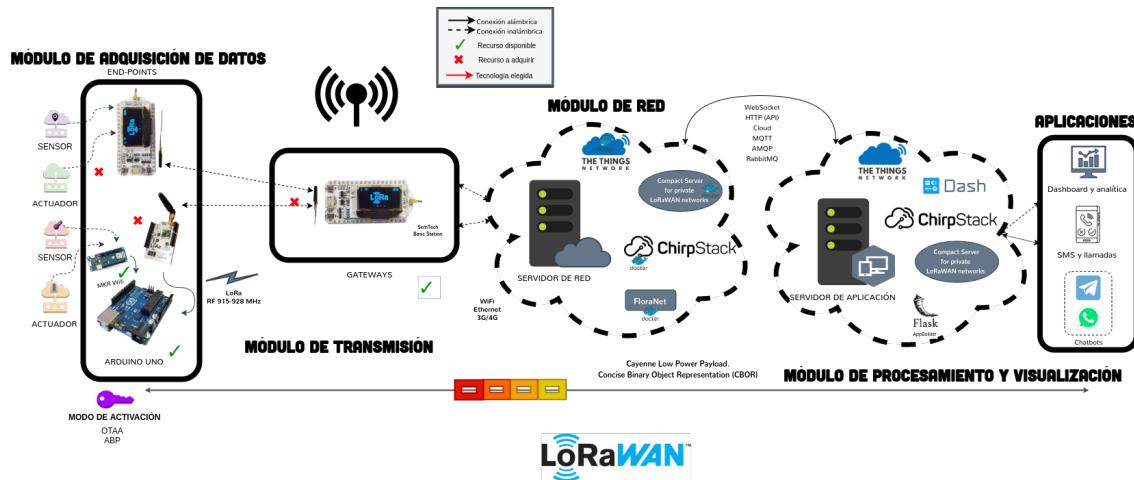


Figura 2.4: Diagrama global del proyecto con tecnologías tentativas.

*Diagrama global del proyecto con tecnologías tentativas* de brivadeneira, jonathan sanchez / CC BY SA 4.0<sup>1</sup>

- **Módulo de adquisición de datos:** Hardware y software necesario para sensar magnitudes físicas de interés, modificar variables de entorno, se conecta con el módulo adyacente, el de transmisión según tecnología LoRaWAN]. *Inicialmente se simula la adquisición con software.*
- **Módulo de comunicación/transmisión:** Hardware necesario para comunicar o transmitir los datos provenientes del módulo de adquisición a través de la red de forma inalámbrica, hasta el servidor.
- **Módulo de red:** (*Servidor de red*). Encargado de administrar la red LoRaWAN, gestionar y administrar los dispositivos activos, y servir los datos al módulo de procesamiento y visualización.
- **Módulo de procesamiento y visualización de datos:** Código de procesamiento de los datos adquiridos, servicio web de visualización dinámica para el usuario final.

<sup>1</sup><<https://creativecommons.org/licenses/by-sa/4.0/deed.en>>

### 3 Metodología

Para cada uno de los módulos descriptos anteriormente, se lleva a cabo el proceso de **investigación** de los fundamentos, especificaciones técnicas, haciendo hincapié en los protocolos de comunicación, así como características económicas y sociales de las tecnologías en cuestión.

Se realiza la **toma de decisiones** sobre la elección de las tecnologías, modos de configuración, dispositivos en función a lo anterior, en primera instancia para el proceso de prototipado y para llevar a cabo la prueba de concepto, para luego proyectar las opciones adecuadas a soluciones comerciales.

Se simulan o adquieren los recursos necesarios para el desarrollo y la **puesta en marcha** del prototipo.

Se obtienen y analizan los resultados para establecer así conclusiones sobre el prototipo y su proyección como potencial producto o solución tecnológica.

# 4 Fundamentos teóricos

A continuación se exponen a modo de síntesis, los datos más relevantes que surgen del proceso de investigación, como base para la toma de decisiones.

## 4.1 Sobre LoRa/LoRaWAN

### 4.1.1 LoRa

(*Long Range*) es una técnica de *modulación* basada en *chirp spread spectrum* (SSC), *espectro ensanchado con chirp*.

### 4.1.2 LoRaWAN

(*Low Power Wide Area Network - Redes de baja potencia y área amplia*) es un protocolo de comunicación y arquitectura de sistema, orientado a **IoT** (*Internet of Things - Internet de las cosas*) que implementa **LoRa**.

Dichas especificaciones adquirieron gran "*popularidad*" en el campo de **IoT**, algunos de los factores de lo anterior son:

- **Largo alcance**, un gateway puede dar cobertura a dispositivos de toda una ciudad, esto es, alcance a distancias en kilómetros.

El 13 de Julio de 2019, se logró un nuevo récord mundial de alcance con LoRaWAN, con **766 km** de distancia y 25mW de potencia de transmisión. Más información.<sup>1</sup>

- **Bajo consumo**, el protocolo de acceso al medio es tal, que los nodos no se sincronizan con los gateways, se energizan e inician la transmisión sólo cuando tienen datos a enviar, esto reporta tiempo de vida de baterías muy superiores a otras tecnologías.

Por ejemplo, para un sensor tomando datos del suelo en un sistema de control de irrigación, se considera que la humedad del suelo no va a cambiar demasiado rápido, se toma una medición por hora, lo que permite cambiar la batería **cada dos años**, comparado con sensar datos cada *cinco minutos*, lo que requeriría cambio de batería *cada dos meses*. Ver más<sup>2</sup>

- **Alta capacidad de la red**, la técnica de modulación empleada permite la comunicación entre un gateway y múltiples nodos a diferentes distancias y tasas de transferencia.
- **Adaptabilidad**, la red soporta diferentes clases de nodos según los requerimientos, permitiendo adaptar la red a diferentes aplicaciones.
- **Bajo costo**, una red funcional puede ser implementada con dispositivos de bajo costo.
- **Seguridad**, implementa algoritmos robustos de encriptación. Implementando encriptación a nivel de red como así también a nivel de aplicación.
- **Gran comunidad activa alrededor del mundo**.
- **Compatible con componentes open source**.

<sup>1</sup><<https://www.thethingsnetwork.org/article/lorawan-distance-world-record>>

<sup>2</sup><<https://www.alliot.uk/achieving-long-battery-life-with-lora-sensors/>>

#### 4.1.3 Arquitectura

La arquitectura general de una red LoRaWAN es como sigue:

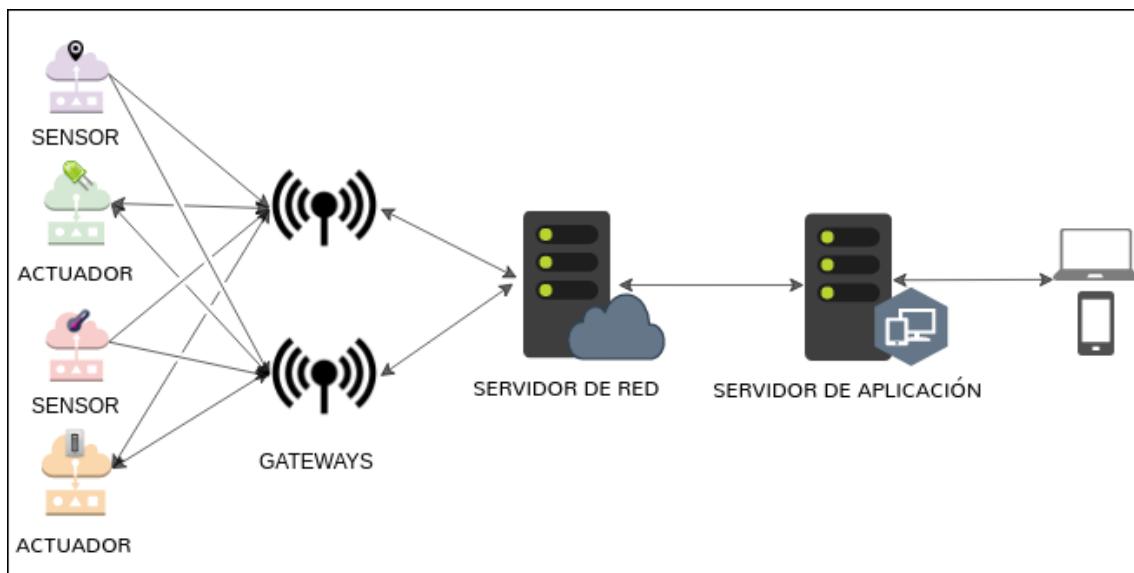


Figura 4.1: Arquitectura general de una red LoRaWAN.

Arquitectura general de una red LoRaWAN de brivadeneira / CC BY SA 4.0<sup>3</sup>

- **End-points:** Se comunican con uno o más gateways según el protocolo LoRa.
  - **Sensor:** (→) Registra medición de una magnitud física y la transmite a los gateways. *Ej: sensor de temperatura, de presión, de nivel.*
  - **Actuador:** (↔) Segundo dato recibido desde la red *modifica* uno o más parámetros del entorno. *Ej: LED, relé.*
- **Gateways:** (↔) Se comunican con end-points según protocolo LoRa, así como con *-uno o más-* servidores de red por *Ethernet, WiFi o 3G/4G.*
- **Servidor de red:** (↔) Se comunica con gateways y servidor de aplicación a través de red de datos.
- **Servidor de aplicación:** (↔) Se comunica con dispositivos del usuario, así como servidor de red. Provee servicios al usuario, autentica dispositivos, etc.

#### 4.1.4 Especificaciones técnicas

A continuación se exponen las especificaciones técnicas de interés de LoRaWAN.

##### Cifrado

LoRaWAN implementa **AES** en el payload del tráfico de la red, así como **AES** en el campo de *MIC, Código de Integridad del Mensaje.*

<sup>3</sup><<https://creativecommons.org/licenses/by-sa/4.0/deed.en>>

## Frecuencia de trabajo

En Argentina, la banda de frecuencias de trabajo para una red LoRaWAN es 902 – 928 MHz, (915-928 MHz usable).

Ver pag. 8, "Table 1: Channel Plan per Country." en Anexo 1: LoRaWAN Regional Parameters (v1.1).

## Normativa Argentina

El rango de frecuencias de trabajo corresponde a la lista de Bandas no licenciadas<sup>4</sup>:

Conviene destacar que el Reglamento de Radiocomunicaciones de UIT ha destinado a nivel mundial (y en algún caso, regional) bandas para uso primario para las aplicaciones Industriales, Científicas y Médicas (ICM). La Nota de Pie 5.150 dice: "Las bandas (...) 902-928MHz en la Región 2 (frecuencia central 915MHz), (...), están designadas para aplicaciones industriales, científicas y médicas (ICM). Los servicios de radiocomunicación que funcionan en estas bandas deben aceptar la interferencia perjudicial resultante de estas aplicaciones."

Según la Cuadro de atribución de bandas de frecuencias de la República Argentina<sup>5</sup>, la banda comprendida entre los 915 MHz y 928 MHz (ver pag 188, nótense que no comienza en 902 MHz), corresponde a **"Servicio TIC para banda de frecuencia de uso compartido"**. \* Tipo de servicio FIJO/MOVIL de *Uso Privado – Prestador*. \* Bajo **resolución 581MM18**<sup>6</sup>, la que establece los siguientes niveles de trabajo: \* Potencia Máxima del Transmisor (Conducida) 30dBm. \* Potencia Máxima del Transmisor (P.I.R.E.) 36dBm.

## Ocupación espectral

La distribución espectral es según la norma **AU915-928 US902-928**, como sigue:

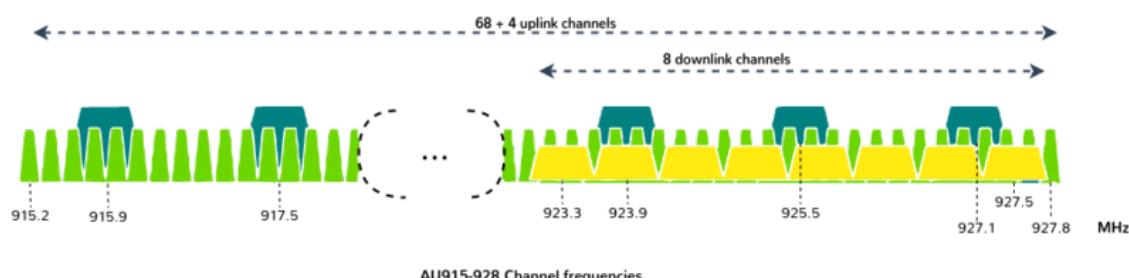


Figura 4.2: Ocupación espectral LoRa..

.^AU915-928 LoRa v1.1" de brivadeneira / CC-BY-SA 4.0.<sup>7</sup>

### ■ Upstream

- 64 canales numerados de 0 a 63 de 125kHz de ancho de banda, y 200kHz entre ellos. El primero en 915.2MHz y el último en 927.1MHz. (**Verde**).
- 8 canales numerados de 64 a 71 de 500kHz de ancho de banda, y 1.6MHz entre ellos. El primero en 915.9MHz y el último en 927.1MHz. (**Azul**).

<sup>4</sup><[https://www.enacom.gob.ar/bandas-no-licenciadas\\_p680](https://www.enacom.gob.ar/bandas-no-licenciadas_p680)>

<sup>5</sup><[https://www.enacom.gob.ar/multimedia/noticias/archivos/201904/archivo\\_20190416044315\\_5617.pdf](https://www.enacom.gob.ar/multimedia/noticias/archivos/201904/archivo_20190416044315_5617.pdf)>

<sup>6</sup><<https://www.enacom.gob.ar/multimedia/normativas/2018/res581MM.pdf>>

<sup>7</sup><<http://creativecommons.org/licenses/by-sa/4.0/>>

## ■ Downstream

- 8 canales numerados de 0 a 7 de 500kHz de ancho de banda, y 600kHz entre ellos. El primero en 923.3MHz y el último en 927.5MHz. (Amarillo).

NOTA: En Argentina existe un consenso de usar los canales 8 a 15 de la especificación AU915 p **AU915 sub-band 2**, Para implementación de la red de código abierto TTN.

## Formato de las tramas

Las tramas (*frames*) del protocolo LoRa posee una estructura formada por las capas **física**, **MAC** y superiores (*aplicación*):

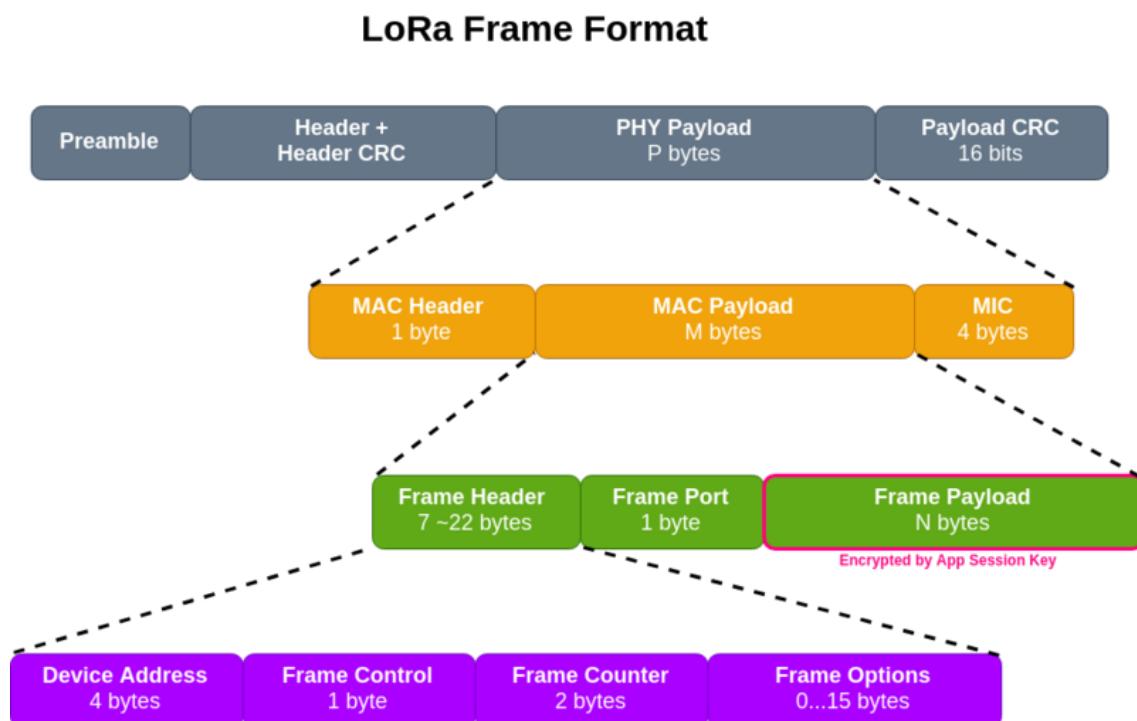


Figura 4.3: Formato frame LoRa.

"LoRa Frame Format" de brivideneira / CC-BY-SA 4.0<sup>8</sup>

- **Capa física -azul- (Physical Layer Frame):** La trama LoRa comienza con un
  - **Preámbulo:** Además de cumplir la función de *sincronismo*, define el *esquema de modulación de paquetes*. Duración típica del preámbulo: 12.25 Ts.
  - **Encabezado + CRC (Header):** Contiene información sobre el *tamaño* del payload, si el *CRC* del Payload-gPayload de 16-bit está o no presente en la trama. *Sólo los frames del enlace de subida contienen el campo CRC*. Tamaño de 20 bits.
  - **Payload: -anaranjado- (PHY Payload)** - 0-96 bits
    - **Capa MAC.**
      - ◊ **MAC Header:** Define la versión del protocolo así como el tipo de mensaje.

<sup>8</sup><<http://creativecommons.org/licenses/by-sa/4.0/>>

- ◊ **MAC Payload:** -verde- Contiene *join-request* o *join-access*, empleados en el proceso de activación de un end-point. El MAC Payload es gestionado por la *capa de aplicación*, está conformado por:
  - ◊ **Frame Header:** -violeta-
  - ◊ **Dirección de dispositivo:** los primeros 8 bits identifican la red, los demás se asignan de forma dinámica durante el inicio de sesión e identifican al dispositivo en la red.
  - ◊ **Frame Control:** 1 byte, contiene información de control de la red como ser implementar o no velocidad de *subida* especificada por el gateway, ACK del mensaje anterior, si el gateway tiene más datos para transmitir.
  - ◊ **Frame counter** para el número de secuencia.
  - ◊ **Frame options:** datos para cambiar configuración como velocidad de transmisión, potencia de transmisión y validación de conexión.
  - ◊ **Frame Port:** Valor determinado según el tipo de aplicación.
  - ◊ **Frame Payload:** Datos a enviar a la red. Se cifra con la AppSKey mediante el algoritmo AES 128.
  - ◊ **MIC (Message Integrity Code):** Resultado de computar el header y payload MAC con una NwkSKey.

Ver Anexo: "Formatos de payload"

#### 4.1.5 Endpoints

El módulo de adquisición de datos tiene como funcionalidad el sensado de magnitudes físicas de interés, (*según aplicación, temperatura, presión, humedad, nivel, etc.*). Se conecta directamente con el de transmisión según tecnología LoRaWAN.

En el diagrama general de la arquitectura de la red LoRa se denominan *end-point*.

#### Factor de ensanchado vs tasa de bits

El factor de ensanchado de las señales enviadas en una red LoRa, definido como:  $SF = \frac{chip\_rate}{symbol\_rate}$ , el cociente entre la cantidad de pulsos por segundo y la cantidad de símbolos por segundo, varía entre 7 y 12 según normativa regional e impacta directamente en la tasa de bits como sigue:

$$bitrate = SF \cdot \frac{Bw}{2^{SF}}$$

A continuación se muestran las tasas de bits, según configuración física correspondiente a normativa AU915-928

DataRate	Configuration	Bit rate [bit/sec]
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	RFU	
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	
15	Defined in LoRaWAN	

Tabla 4.1: Tasas de bit

#### Clases de end-points (A, B y C)

Según los requerimientos para el diseño del sistema y en función del "intercambio" (*trade off*) entre optimizar la **energía necesaria** (y por ende la vida útil de la batería) vs optimizar la **latencia** en el enlace de descarga, existen tres tipos bien definidos de end-point, caracterizados por los tiempos de transmisión y recepción con el gateway, la energía necesaria para ello versus la latencia en la transmisión. random.

Todos los tipos de end-points permiten comunicaciones bi-direccionales, implementan el método de acceso **ALOHA** con *tiempo de uplink* random.

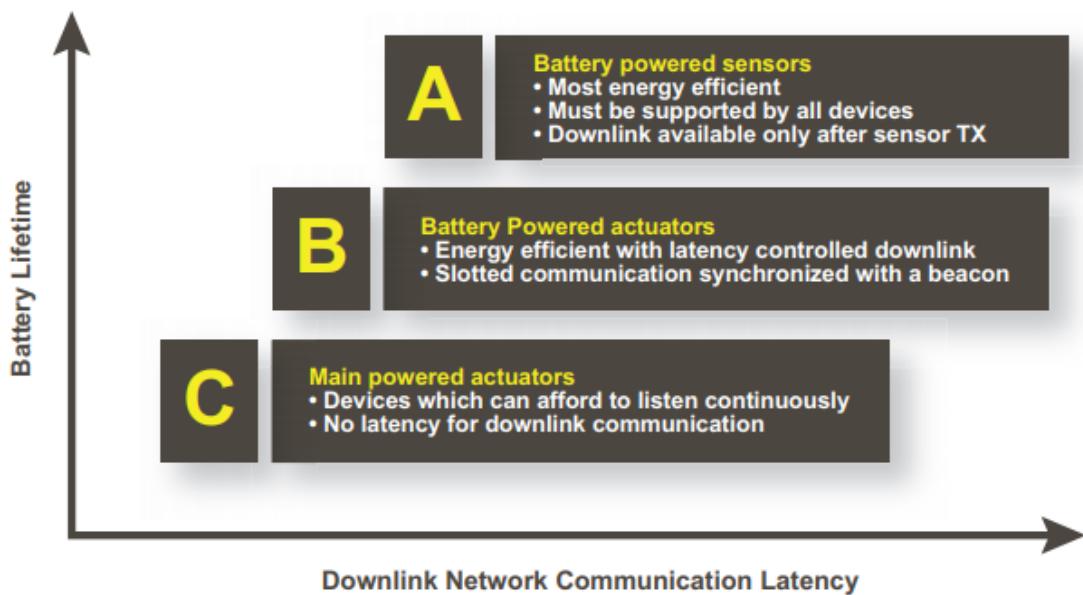


Figura 4.4: Clases de endpoint.

Pag. 10 "What is LoRaWAN?"<sup>9</sup> de LoRa Alliance<sup>10</sup>

### Clase A

Los end-points de clase A abren una ventana de *dowlink* "(escuchan)" por cada ventana de uplink, la que se habilita al cabo de un tiempo azaroso.

No permiten comunicación Full-Duplex, sino Half-Duplex, mientras el end-point transmite información al gateway, éste no puede responder".

El **requerimiento de energía es mínimo** (*y por ende la vida útil de batería máxima*), pero es el tipo de end-point que **más latencia** presenta en la transmisión.

### Clase B

A diferencia de los end-point clase A, por cada ventana de uplink se abren dos pequeñas ventanas de downlink, el requerimiento de energía es mayor al de los end-points de clase A, pero presenta menor latencia.

### Clase C

A diferencia de las clases A y B, la ventana de tiempo para el enlace de downlink está disponible siempre, es decir, permiten comunicación Full-Duplex.

<sup>9</sup><<https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf>>

<sup>10</sup><[lora-alliance.org](http://lora-alliance.org)>

El requerimiento de energía es el máximo, mientras que los tiempos de latencia mínimos. Lo anterior es relevante para aplicaciones críticas, donde se requieren tiempos de respuesta mínimos, como ser alarmas, sistemas de reacción ante emergencias, etc.

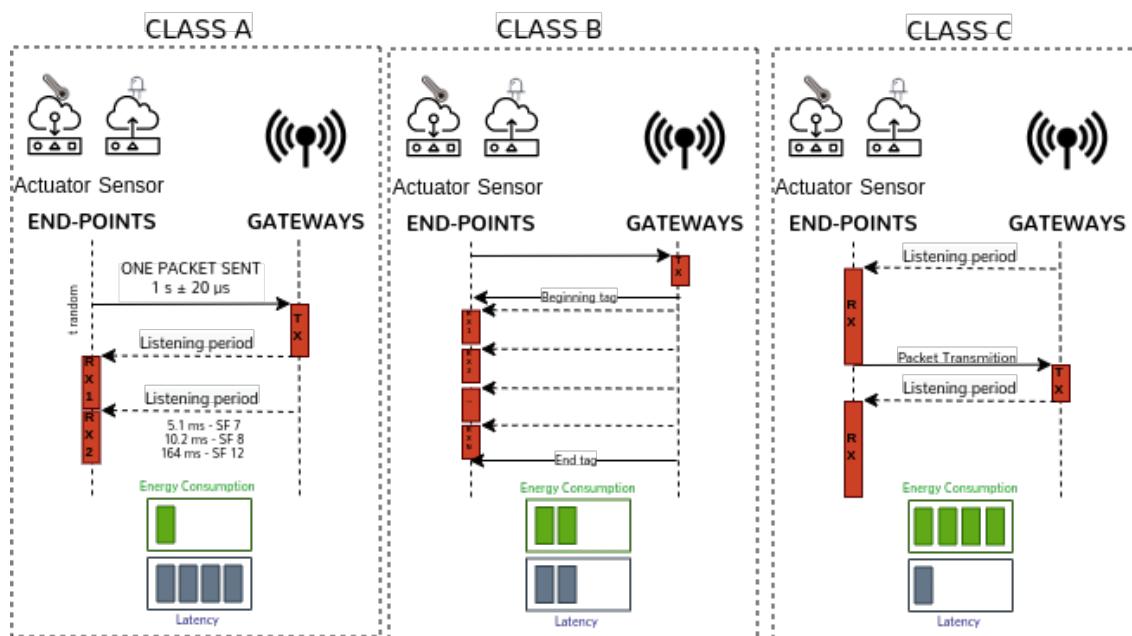


Figura 4.5: Diagrama comparativo endpoints.

"LoRa End Device Classifications" de brivadeneira / CC BY SA 4.0<sup>11</sup>

#### Ejemplos de uso según clase de end-point

Clase	Uso	Energía	Latencia
A	Monitoreo de sistema de riego para estadísticas de consumo, caudal, etc.	Mínima	Máxima
A	Monitoreo de nivel de llenado de contenedores de residuos para optimizar rutas de recolección.	Mínima	Máxima
B	Monitoreo de contenido de gas en sistemas zeppelin para optimizar recorrido de proveedor.	Media	Media
C	Monitoreo crítico y sistema de alarmas ante fallas.	Máxima	Mínima

Tabla 4.2: Aplicaciones según tipo de nodo LoRaWAN

<sup>11</sup><<https://creativecommons.org/licenses/by-sa/4.0/deed.en>>

#### 4.1.6 Modos de activación

Cuando un end-point desea conectarse a la red LoRaWAN, envía al gateway un "join-request" o solicitud de inicio de sesión en la red mediante un mensaje con datos de configuración y abre una ventana de recepción.

El gateway reenvía el mensaje al servidor, éste devuelve un mensaje para confirmar el inicio de sesión del end-point si los datos fueran correctos, así como las claves, o un mensaje de rechazo de lo contrario.

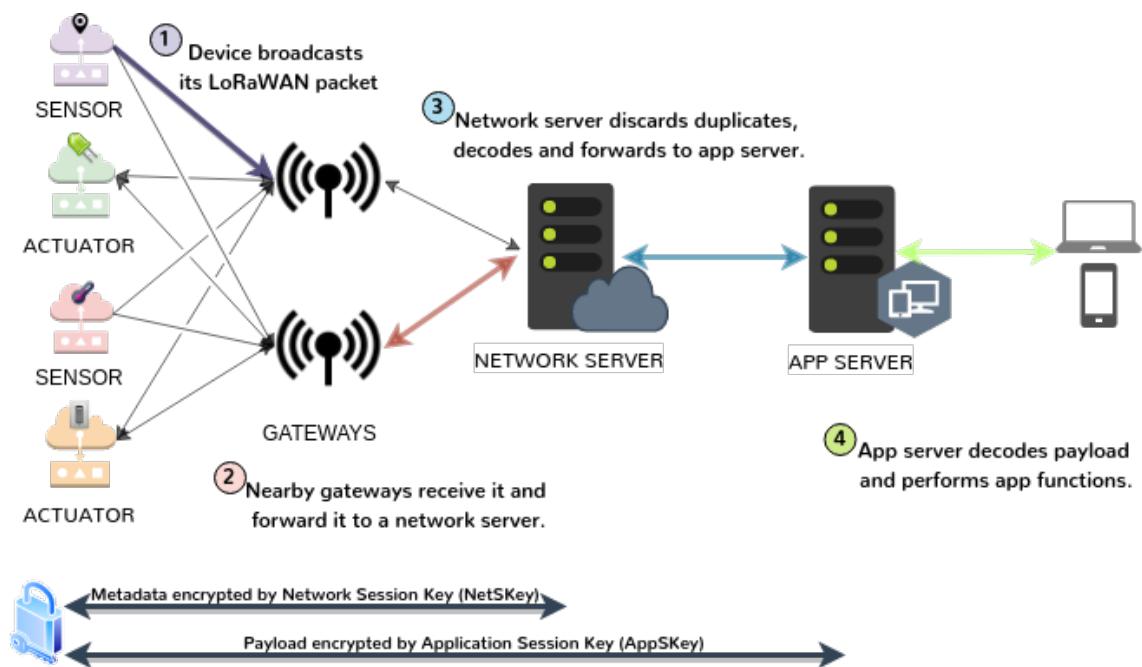


Figura 4.6: Modos de activación.

"LoRa end-devices activation" de brivadeneira / CC BY SA 4.0<sup>12</sup>

Ver Anexo: "Modos de activación"

<sup>12</sup><<https://creativecommons.org/licenses/by-sa/4.0/deed.en>>

## Cuadro comparativo

OTAA	ABP
El end-point puede cambiar de red e iniciar una nueva sesión sin necesidad de re-programar parámetros.	Si el end-point cambia de red, se deben re-configurar los parámetros
Requiere capacidad de trabajar con mensajes de join.	No requiere capacidad de trabajar con mensajes de join.
Los parámetros RxDelay y CFList pueden ser configurados en el momento del inicio de sesión	
La generación de las claves NwkSKey y AppSKey se realiza automáticamente garantizando que éstas son únicas	Se deben generar las claves NwkSKey y AppSKey previo al inicio de sesión y garantizar que sean únicas.
Proceso de inicio de sesión adecuado para redes LoRaWAN con numerosos end-points y con proyección a escalar.	Proceso de inicio de sesión adecuado para prototipar y pruebas de concepto.

Tabla 4.3: Modos de activación nodos LoRaWAN

### 4.1.7 Gateway

En la arquitectura de una red LoRa, los gateways conectan los nodos con la red de datos. Su función principal es **recibir los mensajes de uno o más end-point** a través del enlace de *uplink*, agregar *metadata*, opcionalmente agregar mensajes de estado y **reenviarlos al servidor**, este proceso se lleva a cabo por un "forwarder". Y viceversa.

Un gateway consta de dos *componentes*:

- **LoRa gateway**

Se comunica con los end-point por RF a través redes de baja potencia LoRa. Las señales que llegan al gateway son analógicas, una vez en este punto de la arquitectura, se digitalizan a través de un dispositivo electrónico cuyo diagrama de bloques simplificado es como sigue:

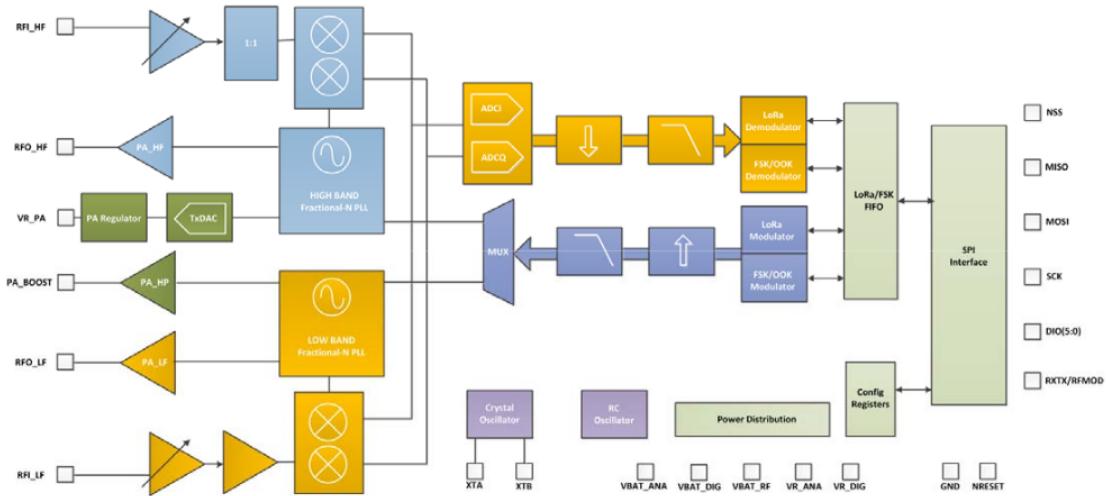


Figure 1. Block Diagram

Figura 4.7: Diagrama de bloques módulo LoRa.

"Simplified Block Diagram", Figure 1, pag. 9, Data Sheet RMF95<sup>13</sup>.

Los datos digitalizados se envían al Microcontrolador/Microprocesador para que este pueda contar con dichos datos para ejecutar sus tareas principales. En el caso del chip RFM95 utiliza la interfaz **SPI** mientras que algunos chips utilizan I<sup>2</sup>C.

### LoRa gateway Bridge

Este componente del gateway usa redes de banda ancha como **WiFi, Ethernet o radio celular** para comunicarse con la red WAN, donde se encuentra el servidor.

Envía los mensajes provenientes de los end-points a través de la red de datos.

El gateway recibe además mensajes a través del enlace de *download* provenientes del servidor con metadata añadida por el mismo, y datos de configuración del gateway o mensajes de configuración destinados a los end-points.

Ver Anexo: "Protocolos Gateway LoRa"

### 4.1.8 LoRa Network Server

El servidor de red en la arquitectura LoRa es el responsable de **enrutar** los datos de la red IoT entre dispositivos y aplicaciones.

Ver Anexo Servidores de red LoRaWAN

<sup>13</sup><[https://cdn.sparkfun.com/assets/learn\\_tutorials/8/0/4/RFM95\\_96\\_97\\_98W.pdf](https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf)>

#### 4.1.9 Integración

##### MQTT

(*Message Queuing Telemetry Transport*) Transporte de mensajes de telemetría por colas.

###### Breve introducción al protocolo

MQTT se diseña para intercambiar grandes cantidades de datos entre dispositivos de una red IoT -(M2M, *embebidos o aplicaciones móviles*)- bajo la pila de protocolos TCP/IP, de manera agnóstica a sus tecnologías y rendimientos, pensado para generar mensajes **ligeros**, en redes **inestables**, de **ancho de banda limitado** y con nodos de **baja potencia**, con rendimiento que roza el **tiempo real**.

El mecanismo de intercambio *bidireccional* de mensajes de tipo suscripción-publicación se basa en un **broker** (*intermediario o negociador*), bajo un paradigma de orientado a **eventos** y de **baja latencia**. Provee seguridad y escalabilidad.

Ver Anexo: "Protocolos de integración, Fundamentos MQTT"

# 5 Análisis de componentes y tecnologías

Se realiza el análisis comparativo de componentes y tecnologías de cada sección de la red LoRa, con enfoque en la toma de decisiones con vista a **prototipar** una solución, esto es, alcanzar los requerimientos mínimos para la *prueba de concepto* del potencial producto.

Se analizan opciones ideales, recomendadas y disponibles. Se toman decisiones en función de dicho análisis, pero con base en hardware disponible.

## 5.1 Requerimientos mínimos

### ▪ End-point

**Sensor/es y Actuador/es:** Es posible simular datos adquiridos por sensores de manera sencilla con un código en Python. Así mismo se podrá simular o conectar actuadores del tipo relé u otro que permitan la modificación del entorno donde se encuentra cada end-point.

- x **Módulo RF LoRa:** Necesario para la comunicación entre end-point y gateway LoRaWAN. *Debe trabajar en el rango de frecuencias 915-928MHz, ver modelo RFM95.*

**Antena RF:** Se requiere para comunicaciones de larga distancia, debe trabajar en el rango de frecuencias mencionado y de radiación omnidireccional. Para el caso del prototipo basta con un conductor a modo de dipolo.

- x **Fuente de corriente:** Necesaria para energizar el módulo RF. Segundo el caso:
  - o Cable USB (se monta circuito de prueba en protoboard).
  - o Shield/microcontrolador.
    - ◊ Arduino.
    - ◊ RaspberryPi.
    - ◊ Otros.
  - o Baterías: Para pruebas de campo que tengan que ver con mediciones de alcance, se analizará la implementación simple de un sistema autónomo de alimentación.

**Módulo WiFi:** Dispositivo adicional que permite comunicar varios sensores a un end-point, únicamente posible en casos de que se tenga conexión a la red eléctrica. No necesaria para el prototipo.

### ▪ Gateway

- x **Módulo RF LoRa:** Necesario para la comunicación entre end-points y gateway LoRaWAN. *Debe trabajar en el rango de frecuencias 915-928MHz, ver modelo RFM95.*

**Antena RF:** Se requiere para comunicaciones de larga distancia, debe trabajar en el rango de frecuencias mencionado y de radiación omnidireccional. Para el caso del prototipo basta con un conductor a modo de dipolo.

- x **Fuente de corriente:** Necesaria para energizar el módulo RF. Segundo el caso:
  - o Cable USB (se monta circuito de prueba en protoboard).
  - o Shield/microcontrolador.
    - ◊ Arduino.
    - ◊ RaspberryPi.

◊ Otros.

**Enlace Backhole:** Dependiendo del dispositivo a utilizar como gateway, la conexión de este hacia internet podrá hacerse vía WiFi, Ethernet, 4G u otro.

■ **Server:**

- x Red.
- x Aplicación. A fin de prototipar, es posible implementar ambos en un mismo host y separando los mismos mediante procesos de virtualización.

## 5.2 Hardware disponible

En este caso, como recurso disponible *a priori* se cuenta con el siguiente hardware

- Microcontrolador **ESP32** (*dos unidades disponibles*).
- Microcontrolador **ESP8266** (*una unidad disponible*).
- Microcontrolador Arduino UNO (*una unidad disponible*)
  - Módulo WiFi para Arduino. (*una unidad disponible*)

## 5.3 Endpoints

Los end-points deben operar bajo la norma **AU915-928** para la comunicación inalámbrica de datos con los gateway.

Se determina qué *módulo RF* emplear, como interface entre la sección de adquisición de datos y gateway, en función de sensores compatibles y disponibles con el módulo RF y viceversa.

### 5.3.1 Ideal

La elección **ideal** de hardware para end-points o nodos corresponde a productos certificados por LoRa Alliance<sup>1</sup>.

### 5.3.2 Recomendado

Acotando el conjunto de posibilidades, se recomienda trabajar con hardware compatible con norma **AU915-928**, y especificación sub-band2 (*frecuencia de trabajo Argentina, y consenso para usar la red TTN si fuera el caso*).

### 5.3.3 Para prototipado

A fin de alcanzar el objetivo de **prototipar** una solución, se acota el conjunto de hardware según el que puede proveer el equipo de trabajo, minimizando los tiempos de adquisición, costos, pero también como **demostración de simpleza y versatilidad de implementaciones de redes IoT con LoRaWAN**.

---

<sup>1</sup><<https://lora-alliance.org/lorawan-certified-products>>

## 5.4 End-points y Gateways

Entre los requerimientos indispensables para montar una red LoRaWAN se encuentran el de transmisión de señales de RF hacia el/los gateways, **implementando el estándar LoRa**, para lo que se necesita un módulo con dichas características que opere en el rango de frecuencias correspondiente a la *normativa argentina* y disponible en el mercado local. El mismo corresponde al Módulo FM95/96/97/98<sup>2</sup>.

## 5.5 Network/Application Server

Ver Anexo: "Servidores de red y aplicación LoRaWAN"

Se realiza un análisis comparativo de las opciones disponibles para servidores de red y de aplicación a través del siguiente cuadro:

Solución	Licencia
LoRaServer	MIT
The Things Network	MIT
Compact server for private LoRaWAN	MIT
FloraNet	MIT

Tabla 5.1: Cuadro comparativo servidores, parte 1

---

<sup>2</sup><[https://cdn.sparkfun.com/assets/learn\\_tutorials/8/0/4/RFM95\\_96\\_97\\_98W.pdf](https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf)>

Solución	Características
LoRaServer	<p>Es parte del proyecto LoRaServer con soluciones listas para usar (incluyendo App Server). Por defecto usa el protocolo MQTT. Soporta configuración de canales de LoRaWAN. Incluye interfáz web. Documentación disponible.</p>
The Things Network	<p>Es parte del stack TTN con soluciones listas para usar (incluyendo App Server). Arquitectura distribuida que requiere Routers, Brokers y Handlers además del servidor de red. Soporta redes privadas, pero requiere conexión con TTN Account Server. Soporta redes privadas intercambiando información con la red pública. Soporta velocidad de datos adaptativa. Incluye interfáz web. Provee productos y servicios para industrias. Comunidad alrededor del mundo (existe comunidad en Río Cuarto). Documentación disponible.</p>
Compact server for private LoRaWAN	<p>Incluye funcionalidades de servidor de red y de aplicación. Pensado para prototipos. Pensado para redes privadas. Incluye interfaz web. Documentación disponible. Brinda asistencia vía lista de e-mails.</p>
FloraNet	<p>Pensado para integrar con Microsoft Azure. Documentación disponible. Guía paso a paso disponible.</p>

Tabla 5.2: Cuadro comparativo servidores, parte 4

Solución	Se integra / compatible con
LoRaServer	End-points clase A/B/C. ABP y OTAA. LoRaWAN 1.0 y 1.1.
The Things Network	Semtech. Basic Station. ADR. gRPC y REST API. MQTT y HTTP.
Compact server for private LoRaWAN	Docker. PostgreSQL. Redis.
FloraNet	End-points clase A/C. ABP y OTAA. LoRaWAN 1.0 y 1.1.

Tabla 5.3: Cuadro comparativo servidores, parte 2

Solución	Características
LoRaServer	Es parte del proyecto LoRaServer con soluciones listas para usar (incluyendo App Server). Por defecto usa el protocolo MQTT. Soporta configuración de canales de LoRaWAN. Incluye interfáz web. Documentación disponible.
The Things Network	Es parte del stack TTN con soluciones listas para usar (incluyendo App Server). Arquitectura distribuida que requiere Routers, Brokers y Handlers además del servidor de red. Soporta redes privadas, pero requiere conexión con TTN Account Server. Soporta redes privadas intercambiando información con la red pública. Soporta velocidad de datos adaptativa. Incluye interfáz web. Provee productos y servicios para industrias. Comunidad alrededor del mundo (existe comunidad en Río Cuarto). Documentación disponible.
Compact server for private LoRaWAN	Incluye funcionalidades de servidor de red y de aplicación. Pensado para prototipos. Pensado para redes privadas. Incluye interfaz web. Documentación disponible. Brinda asistencia vía lista de e-mails.
FloraNet	Pensado para integrar con Microsoft Azure. Documentación disponible. Guía paso a paso disponible.

Tabla 5.4: Cuadro comparativo servidores, parte 4

## 5.6 Tecnologías definitivas

Dada la naturaleza del proyecto, orientada a la investigación y desarrollo, y concibiendo al par *practicante-tutor* como un equipo que lleva adelante el mismo, se seleccionan e implementan herramientas de:

- **Gestión de proyectos**, hojas de cálculo en Google Drive<sup>3</sup>.
- **Registro de horas y tareas**, hojas de cálculo en Google Drive<sup>4</sup>, diagrama de Gantt.
- **Sistema de control de versiones**, git<sup>5</sup>.
- **Servicio de repositorios digitales y abiertos**, Github<sup>6</sup>.
- **Documentación**, markdown, hackmd.io.
- **Confección de diagramas**, draw.io.

El entorno de desarrollo y puesta en marcha se define según los problemas abordados durante los procesos, sin descuidar las buenas prácticas y herramientas conocidas y amenas para el equipo de trabajo:

- **Simulador de sistemas embebidos**, mbed simulator.<sup>7</sup>
- **IDE de desarrollo para microcontroladores**, VS Code<sup>8</sup>.
- **IDE para gestión de proyectos y librerías de sistemas embebidos**, PlatformIO IDE<sup>9</sup>.
- **Entorno de desarrollo para python scripting**, Jupyterlab<sup>10</sup>.

### 5.6.1 Para prototipado

Se opta por montar la red en infraestructura de servicios gratuitos (*y por ende limitados*) en la nube, así como los métodos y formatos más simples.

### 5.6.2 Para potencial solución comercial

Se seleccionan las soluciones acordes a redes privadas, "*sin sacrificar simplicidad*".

---

<sup>3</sup><[https://www.google.com/intl/es\\_ALL/drive/](https://www.google.com/intl/es_ALL/drive/)>

<sup>4</sup><[https://www.google.com/intl/es\\_ALL/drive/](https://www.google.com/intl/es_ALL/drive/)>

<sup>5</sup><<https://git-scm.com/>>

<sup>6</sup><<https://github.com/>>

<sup>7</sup><<https://labs.mbed.com/simulator/>>

<sup>8</sup><<https://go.microsoft.com/fwlink/?LinkID=760868>>

<sup>9</sup><<https://platformio.org/install/ide?install=vscode>>

<sup>10</sup><<https://jupyterlab.readthedocs.io/en/stable/>>

Se muestra a continuación el diagrama simbólico indicando las tecnologías elegidas para cada caso:

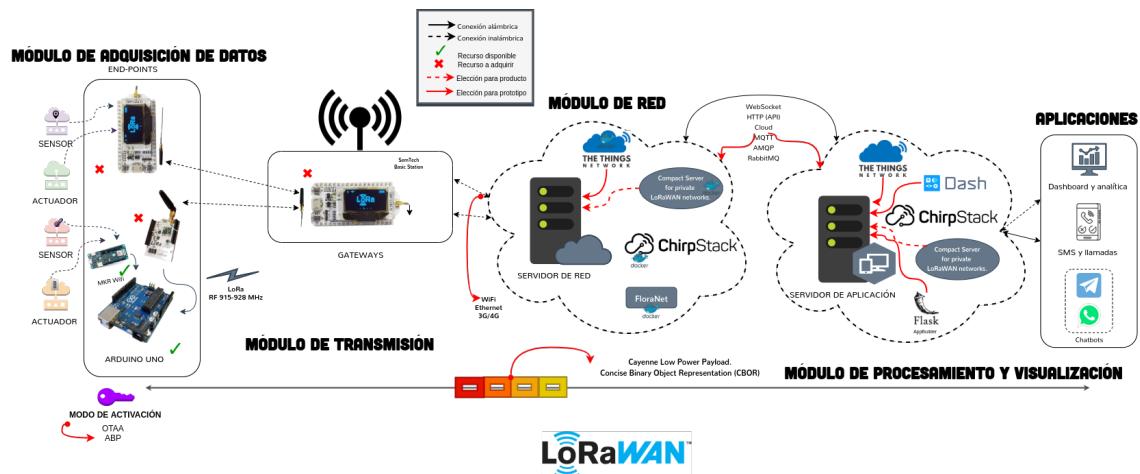


Figura 5.1: Diagrama de tecnologías definitivas.

*Diagrama global del proyecto con tecnologías tentativas y resultado de toma de decisiones de brivadeneira, jonathan sanchez / CC BY SA 4.0<sup>11</sup>.*

<sup>11</sup><<https://creativecommons.org/licenses/by-sa/4.0/deed.en>>

A continuación se describen las opciones definitivas por sección de arquitectura de red LoRaWAN:

- **Gateway:** esp32-oled

- **Nodos:**

- esp32-oled
- arduino-uno

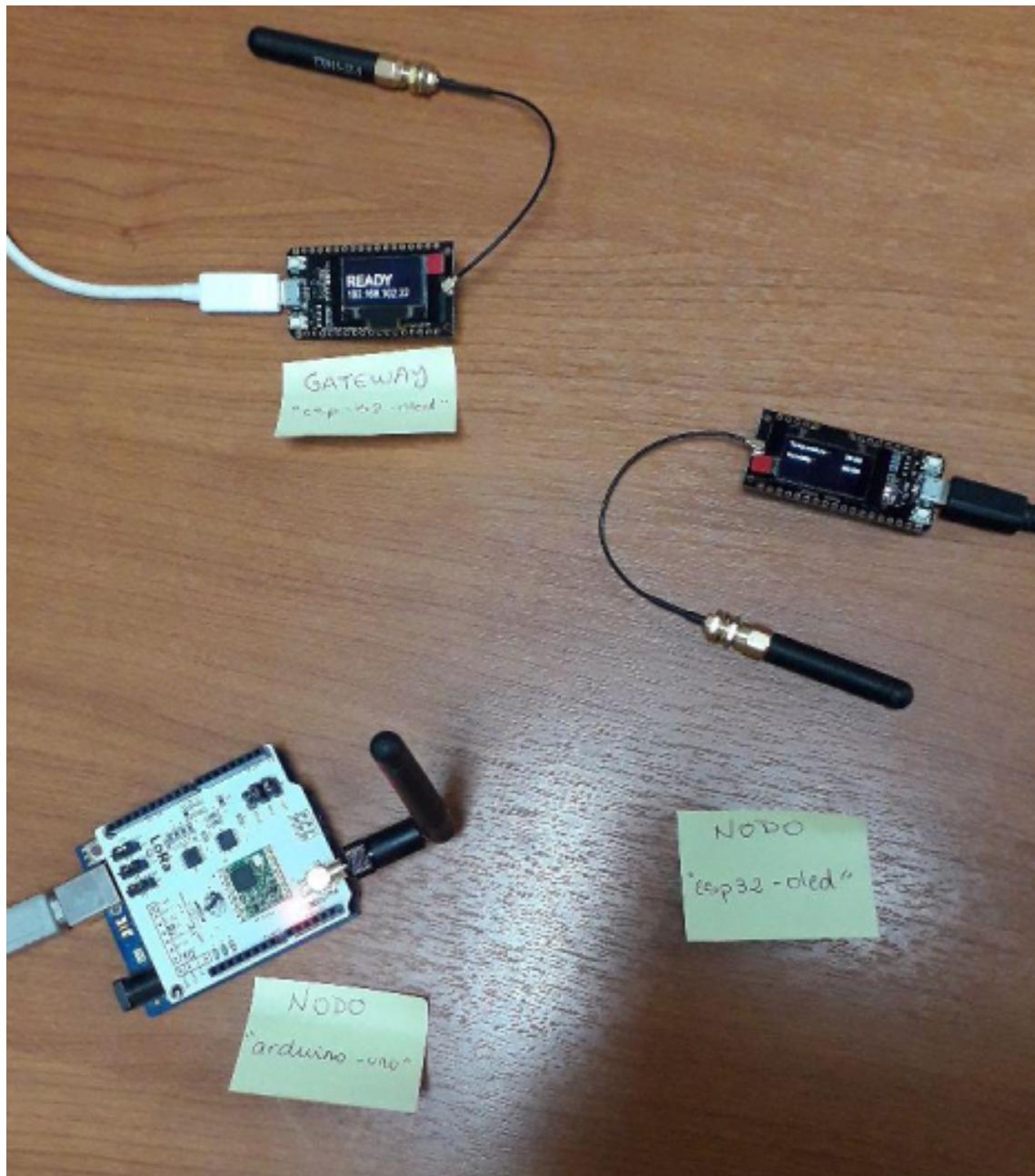


Figura 5.2: Red LoRaWAN: esp32-oled, arduino-uno..

### 5.6.3 End-points

Se seleccionan microcontroladores disponibles y adquiridos de diferentes tecnologías para testear una red agnóstica a las mismas:

- ESP32, oled.
- Arduino uno.

#### Clase de end-point

Se elige la clase de end-point tipo **A**, dado que las aplicaciones que se proyectan como evolución del prototipo no presenta exigencias de tiempo de transmisión, así como se buscan minimizar costos y acotar las opciones de hardware disponibles, lo que impacta en los requerimientos de energía.

#### Modo de activación

Dado que la cantidad de dispositivos de la red a prototipar es controlable, se selecciona el modo de activación **ABP** (*Activation by personalization*) por su simpleza y posibilidad de pre-programar los dispositivos.

## 5.7 Gateways

Se selecciona el microcontrolador con mayor capacidad de performance para cumplir el rol de **gateway** de la red, esto es esp32-oled.

Se elige **Semtech** como software forwarder debido al tiempo y respaldo en el desarrollo del software, apostando a la compatibilidad con mayor cantidad de tecnologías en una potencial red LoRaWAn como evolución del prototipo.

## 5.8 Servidores de red y aplicación

A fin de realizar el prototipado así como la prueba de concepto, se opta por usar la infraestructura de los servicios en la nube dados por thethingsnetwork<sup>12</sup>, suprimiendo así tiempo de desarrollo e implementación.

Se deben contemplar las limitaciones<sup>13</sup> de dichos servicios respecto al flujo de datos en la red pública, para enviar datos desde los nodos durante todo el día, se establecen intervalos de mensaje de aproximadamente **5 minutos**, tiempo estimado para dos nodos.

Proyectando una potencial solución comercial, se elige **Compact server for private LoRaWAN networks** como servidor de red así como servidor de aplicación, con documentación disponible y contacto directo con desarrollador. Además, *a priori* se define montar ambos servidores en la misma computadora, de manera que esta solución se adapta mejor a los recursos disponibles.

Se agrega procesamiento de datos con Python y visualización Flask y Dash, dada la experiencia en dichas tecnologías en el equipo de trabajo, como extensión del prototipo así como muestra de la posibilidad de funciones específicas y personalización a futuras soluciones comerciales.

---

<sup>12</sup><<https://www.thethingsnetwork.org/>>

<sup>13</sup><<https://www.thethingsnetwork.org/forum/t/limitations-data-rate-packet-size-30-seconds-uplink-and-10-messages-downlink-per-day-fair-access-policy-guidelines/1300>>

Como protocolo de comunicación entre los módulos de red y aplicación se elige **MQTT**, por ser estándar para aplicaciones de IoT y como oportunidad para explorar el mismo.

## 5.9 Formato Frame

Manteniendo el criterio de elección a fin de prototipado, procurando simpleza, versatilidad de implementación así como compatibilidad con mayor cantidad de tecnologías y escalabilidad, se selecciona **CBOR** como formato de frame.

# **6 Procedimiento**

Habiendo seleccionado tecnologías definitivas, se describen los procedimientos para implementar los módulos del proyecto desde end-points hasta servidor de aplicación, testeando conectividad y desempeño.

La siguiente lista de tareas describe el proceso de testeo, implementación y puesta en marcha del prototipo luego de la toma de decisiones y adquisición de recursos necesarios:

## **6.1 Conectividad punto a punto LoRa**

Se realiza para corroborar buenas condiciones de hardware, se buscan ejemplos de código abierto, se realizan las modificaciones correspondientes para adaptar al hardware, entorno de desarrollo así como a la frecuencia de transmisión.

## **6.2 Implementación gateway LoRa**

Se busca registrar por primera vez un gateway LoRa en una red pública, se buscan ejemplos de código abierto, se realizan las modificaciones correspondientes para adaptar al hardware, entorno de desarrollo así como a la frecuencia de transmisión, se emplea el microcontrolador ESP32 por su capacidad de procesamiento superior a arduino-uno.

## **6.3 Implementación nodos LoRa**

Se crea una aplicación a la que se añaden dispositivos LoRa en una red pública, se buscan ejemplos de código abierto, se realizan las modificaciones correspondientes para adaptar al hardware, entorno de desarrollo así como a la frecuencia de transmisión, se emplean los microcontroladores ESP32 y arduino-uno.

## **6.4 Implementación servidores de red y aplicación**

Se usa el servicio en la nube de [thethingsnetwork.org](http://thethingsnetwork.org), servicio gratuito y en una red pública, pero con limitaciones de uso.

## **6.5 Simulación de paquetes LoRa**

A fin de avanzar con los módulos de aplicación bajo un escenario controlado de simulación se realiza la búsqueda de herramientas de código abierto, se simulan nodos y gateways LoRa para trabajar sobre los datos que reportan a la salida los servidores de red y de aplicación.

## **6.6 Desarrollo e implementación de API MQTT**

Se emplea el broker público de [thethingsnetworks.org](http://thethingsnetworks.org), se incia un cliente MQTT y se desarrollan algoritmos en Python para servir los payload LoRa en una API.

## **6.7 Desarrollo e implementación de dashboard**

Se desarrolla además una aplicación web para graficar los datos en tiempo real, la que es alimentada por los datos de la API.

# 7 Desarrollo

A continuación se describe el desarrollo de las tareas según lo planificado y el resultado de la toma de decisiones.

El desarrollo de software para el prototipo de red LoRaWAN se encuentra hosteado y documentado con las correspondientes instrucciones que **garantizan su reproducibilidad**, en el siguiente repositorio: LoRaWANprototype<sup>1</sup>.

## 7.1 Conectividad punto a punto LoRa con ESP32

Se comprueba conectividad LoRa entre microcontroladores ESP32, con módulo LoRa integrado, antena de RF y display:

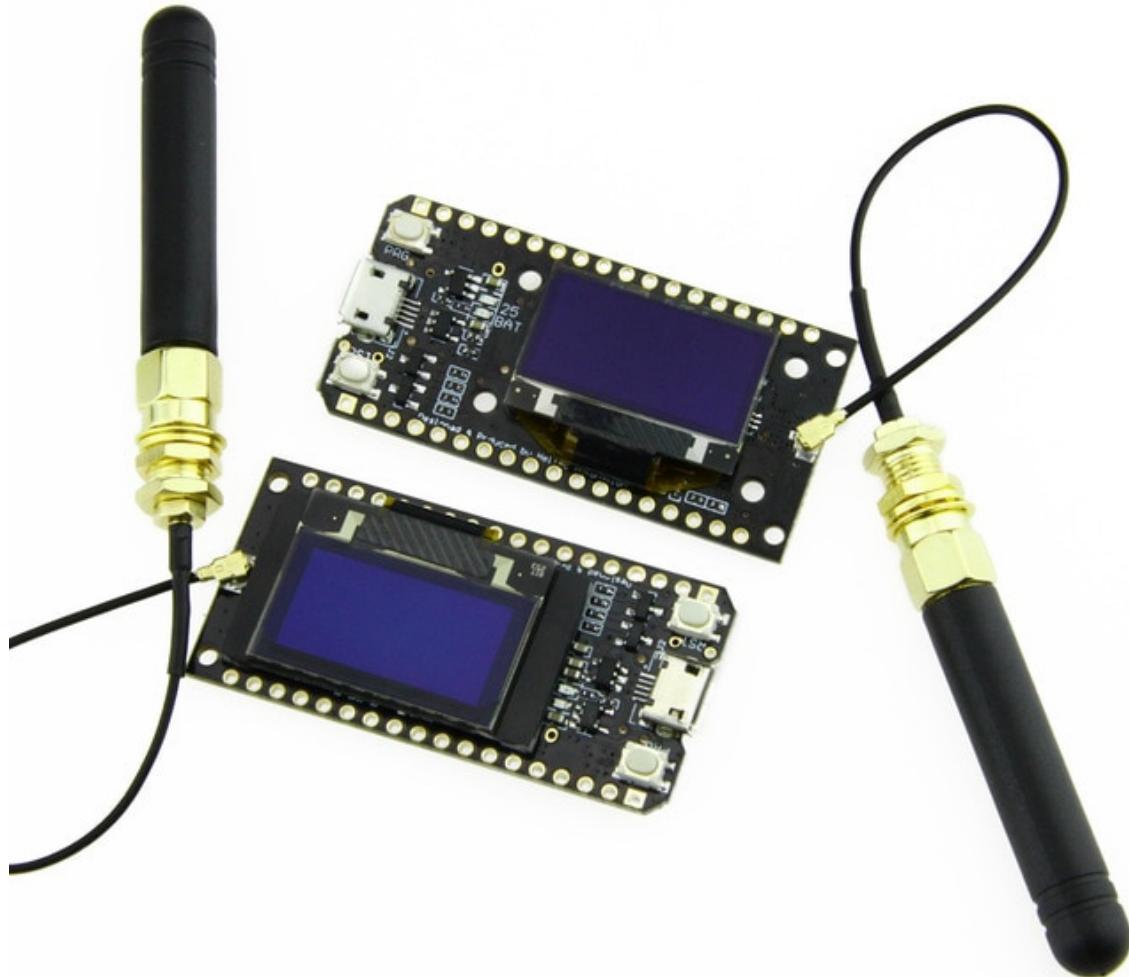


Figura 7.1: Microcontrolador ESP32 oled LoRa.

---

<sup>1</sup><<https://github.com/brivadeneira/LoRaWANprototype>>

Se toman ejemplos de código abierto y sus correspondientes instrucciones para adaptarlo al entorno de desarrollo del equipo de trabajo, realizar modificaciones mínimas en los algoritmos y testear así de manera sencilla que el módulo LoRa y las antenas de RF funcionen correctamente.

Se usan ejemplos de Aaron.Lee from HeiTec AutoMation, ChengDu, China, [www.heltec.cn](http://www.heltec.cn).

Se complementa la puesta en marcha de este ejemplo con observaciones respecto a la frecuencia de transmisión y canal, analizando la evolución de la red con la normativa argentina en mente.

Se siguen las instrucciones principales del proyecto LoRaWANprototype<sup>2</sup>, para a continuación implementar oled lora receiver<sup>3</sup> y oled lora sender<sup>4</sup> en cada uno de los esp32.

## 7.2 Implementación gateway LoRa ESP32

Se toman ejemplos de código abierto y sus correspondientes instrucciones para adaptarlo al entorno de desarrollo del equipo de trabajo, así como la documentación oficial de [thethingnetworks.org](http://thethingnetworks.org), para registrar por primera vez un gateway LoRa en la red pública.

Se usan ejemplos de Maarten Westenberg ([mw12554@hotmail.com](mailto:mw12554@hotmail.com)), based on work done by Thomas Telkamp for Raspberry PI 1-ch gateway and many others.

Se siguen las instrucciones principales del proyecto LoRaWANprototype<sup>5</sup>, para a continuación implementar esp32 single channel lora gateway<sup>6</sup>.

## 7.3 Implementación nodos LoRa

### 7.3.1 Implementación nodo LoRa ESP32

Se toman ejemplos de código abierto y sus correspondientes instrucciones para adaptarlo a la frecuencia de trabajo de la normativa Argentina, entorno de desarrollo del equipo de trabajo, así como la documentación oficial de [thethingnetworks.org](http://thethingnetworks.org), para registrar por primera vez una aplicación y un dispositivo nuevo.

Se usan ejemplos de Thomas Telkamp and Matthijs Kooijman.

Se siguen las instrucciones principales del proyecto LoRaWANprototype<sup>7</sup>, para a continuación implementar esp32 single channel lora node<sup>8</sup>.

### 7.3.2 Implementación nodo arduino uno

Se siguen las instrucciones principales del proyecto LoRaWANprototype<sup>9</sup>, para a continuación implementar arduino uno lora node<sup>10</sup>.

<sup>2</sup><<https://github.com/brivideneira/LoRaWANprototype>>

<sup>3</sup><[https://github.com/brivideneira/LoRaWANprototype/tree/master/1.oled\\_lora\\_receiver](https://github.com/brivideneira/LoRaWANprototype/tree/master/1.oled_lora_receiver)>

<sup>4</sup><[https://github.com/brivideneira/LoRaWANprototype/tree/master/0.oled\\_lora\\_sender](https://github.com/brivideneira/LoRaWANprototype/tree/master/0.oled_lora_sender)>

<sup>5</sup><<https://github.com/brivideneira/LoRaWANprototype>>

<sup>6</sup><[https://github.com/brivideneira/LoRaWANprototype/tree/master/2.esp32\\_single\\_channel\\_lora\\_gateway](https://github.com/brivideneira/LoRaWANprototype/tree/master/2.esp32_single_channel_lora_gateway)>

<sup>7</sup><<https://github.com/brivideneira/LoRaWANprototype>>

<sup>8</sup><[https://github.com/brivideneira/LoRaWANprototype/tree/master/3.esp32\\_single\\_channel\\_lora\\_node](https://github.com/brivideneira/LoRaWANprototype/tree/master/3.esp32_single_channel_lora_node)>

<sup>9</sup><<https://github.com/brivideneira/LoRaWANprototype>>

<sup>10</sup><[https://github.com/brivideneira/LoRaWANprototype/tree/master/4.arduino\\_uno\\_lora\\_node](https://github.com/brivideneira/LoRaWANprototype/tree/master/4.arduino_uno_lora_node)>

## 7.4 Implementación servidores de red y aplicación

El prototipo de red LoRaWAN se monta sobre la infraestructura de thethingsnetworks<sup>11</sup>, tanto en lo que a servidor de red como de aplicación respecta.

Se registra el gateway esp32, se crea una nueva aplicación y se registran los nodos esp32 y arduino uno.

## 7.5 Simulación paquetes LoRa

A fin de construir integraciones de aplicación en un escenario controlado, se emplea el simulador online Mbed OS Simulator<sup>12</sup> de paquetes LoRa, escrito en js y con ejemplos LoRa disponibles.

### 5. Elegir el ejemplo LoRaWAN:

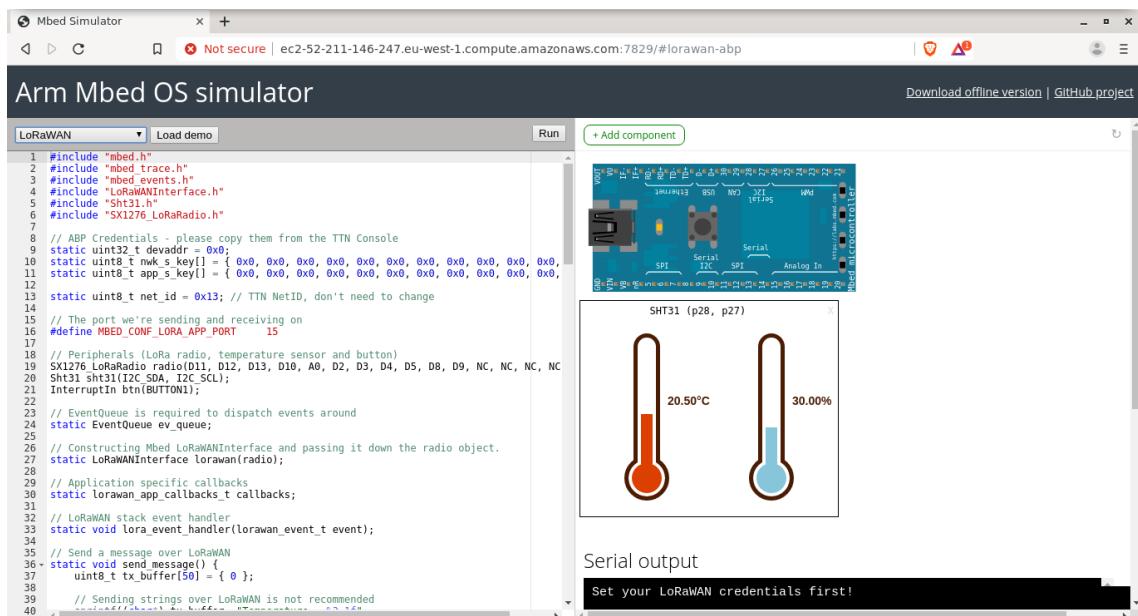


Figura 7.2: Simulador de sistemas embebidos.

### 5.1 Copiar y pegar las claves desde la aplicación (paso 4.2) y reemplazar en el código:

```
““ // Device credentials, register device as OTAA in The Things Network and copy credentials here static uint8t DEVEUI[] = 0x00, 0x37, 0xD5, 0x78, 0x63, 0x16, 0xE6, 0x64 ;
static uint8t APPEUI[] = 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x02, 0x38, 0x0C ; static uint8t APPKEY[] = 0xB3, 0x9B, 0x30, 0x5F, 0xF1, 0xDF, 0xB4, 0x5B, 0xFA, 0xB5, 0x61, 0x3E,
0x4B, 0x30, 0xF7, 0x75 ;
```

““

### 5.2 Modificar el puerto a 1

```
// The port we're sending and receiving on #define MBED_CONF_LORA_APP_PORT
1
```

<sup>11</sup><<https://www.thethingsnetwork.org/>>

<sup>12</sup><<https://labs.mbed.com/simulator>>

## 6. Click en run

6.1 En devices ->sim-1 ->data se pueden observar los paquetes generados:

APPLICATION DATA					II pause	clear
Filters	uplink	downlink	activation	ack	error	
	time	counter	port			
↓	21:23:22			dev addr: 26 01 28 DD app eui: 70 B3 D5 7E D0 02 38 0C dev eui: 00 37 D5 78 63 16 E6 64		
↓	21:22:59			dev addr: 26 01 22 AC app eui: 70 B3 D5 7E D0 02 38 0C dev eui: 00 37 D5 78 63 16 E6 64		
↓	21:22:41			dev addr: 26 01 29 E4 app eui: 70 B3 D5 7E D0 02 38 0C dev eui: 00 37 D5 78 63 16 E6 64		
↓	21:22:28			dev addr: 26 01 28 65 app eui: 70 B3 D5 7E D0 02 38 0C dev eui: 00 37 D5 78 63 16 E6 64		

Figura 7.3: Paquetes simulados.

7. Repetir los pasos 4 a 6 tantas veces como dispositivos se deseen simular.

## 7.6 Desarrollo e implementación de API MQTT

Se llevan adelante tareas de *desarrollo* y puesta en marcha de una API en Python que reciba los mensajes de los nodos LoRa a través del protocolo MQTT y los *sirva* en formato json.

Se usan las librerías flask para crear una aplicación web - cliente de MQTT con un socket que permite procesamiento de varios hilos, uno para recibir los datos del cliente y otro para publicar los datos en la API.

## 7.7 Desarrollo e implementación de dashboard

Se desarrolla además un *dashboard* dinámico y en tiempo real que grafica los datos de los nodos, este servicio *consume* la API.

Se procesan y formatean los datos y se crea una aplicación web con la librería dash.

Cabe destacar que lo anterior es de código abierto y se encuentra publicado en un repositorio digital abierto: LoRaWAN prototype<sup>13</sup>.

<sup>13</sup><<https://github.com/brivideneira/LoRaWANprototype>>

# 8 Resultados

Se muestran a continuación los resultados de la investigación, desarrollo, configuración y puesta en marcha de la red LoRa punto a punto y posterior red LoRaWAN integrada a API y dashboard en tiempo real de los datos sensados.

## 8.1 Red LoRa punto a punto

Se descargan e implementan códigos de ejemplo de un transmisor y un receptor LoRa, se corrobora el funcionamiento del módulo y antena para dicha transmisión:

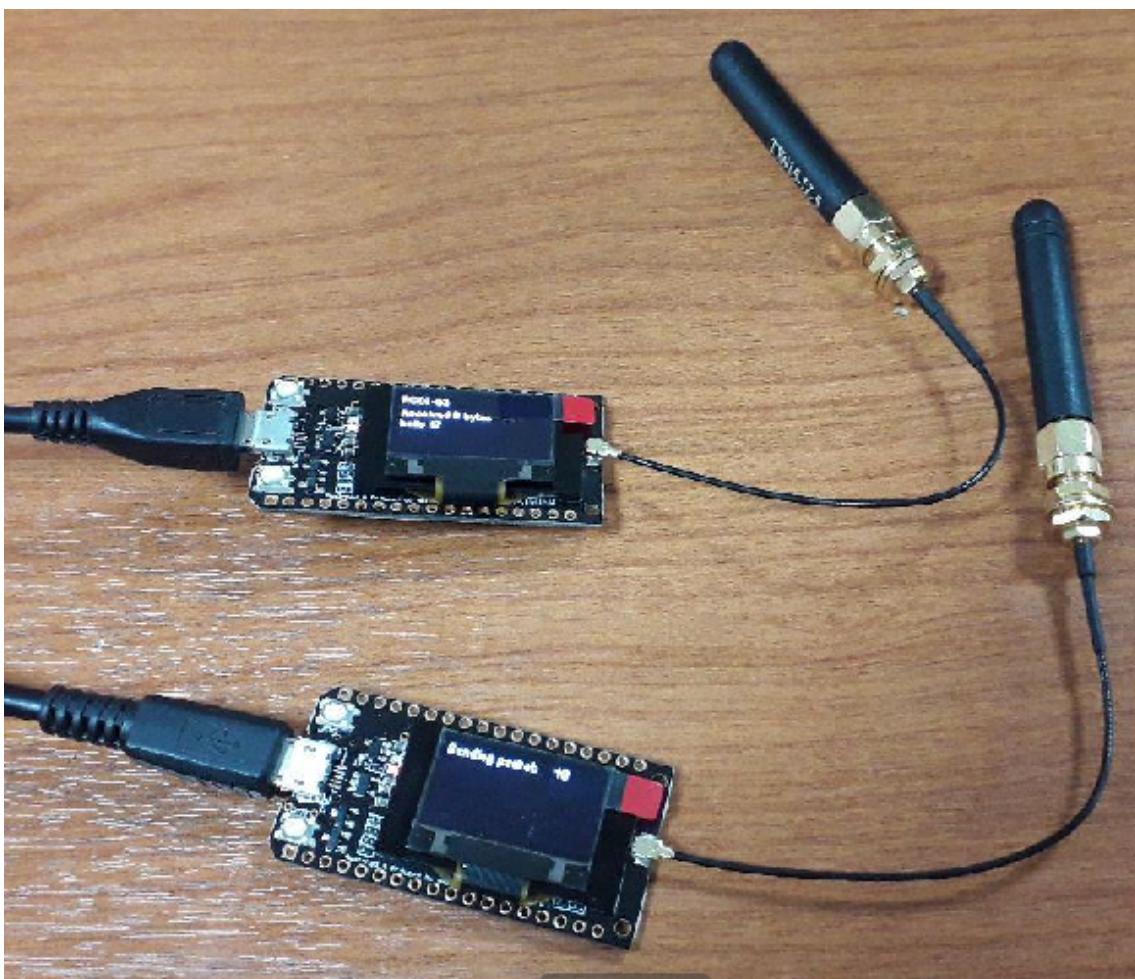


Figura 8.1: Red LoRa punto a punto.

El hardware elegido, con display, permite testear rápidamente el funcionamiento.

## 8.2 Red LoRaWAN

Se descargan e implementan códigos en los microcontroladores, se registra el gateway en [thethingsnetworks.org](http://thethingsnetworks.org), así como los nodos en una aplicación:

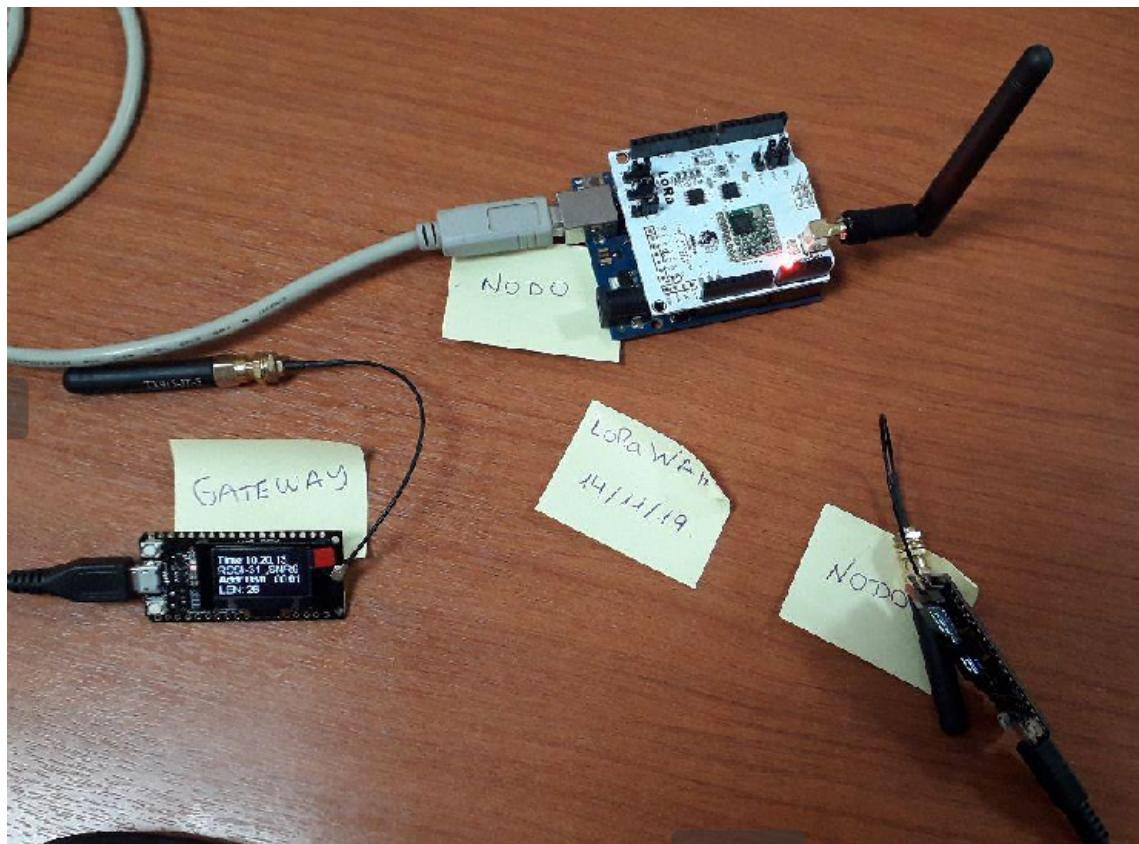


Figura 8.2: Red LoRaWAN: esp32 oled, arduino uno.

Los nodos envían mensajes LoRa al gateway, recepción que se puede apreciar en el display, el gateway se conecta con los servidores de red y aplicación, se pueden apreciar además los datos en la aplicación:

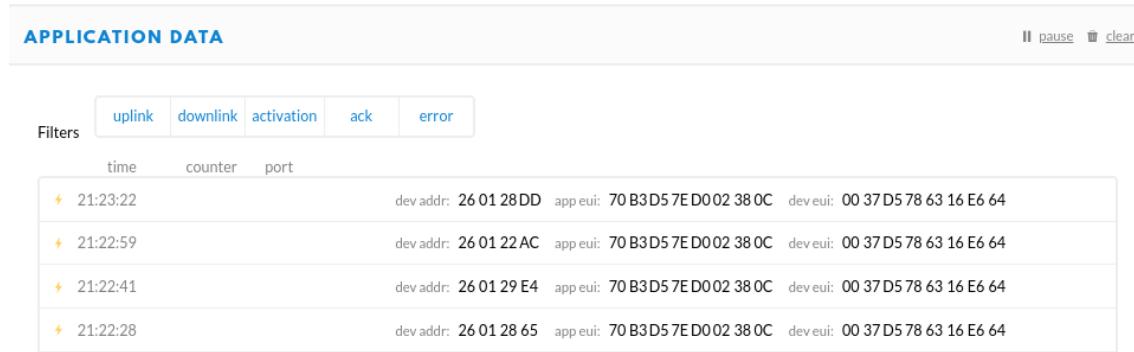


Figura 8.3: Paquetes de red LoRaWAN.

## 8.3 API y dashboard

Para el desarrollo de la API y dashboard se emplea una herramienta de **simulación**<sup>1</sup> de sistemas embebidos con conectividad LoRa, reemplazando los nodos y gateway.

### 8.3.1 API

Se agrega a los servidores de red y aplicación **integración** de un **broker MQTT** disponibles en [thethingsnetwork.org](https://thethingsnetwork.org). Se desarrolla e implementa un script en Python que inicia un **cliente MQTT** y sirve los datos en formato json en una API, en tiempo real:



Figura 8.4: API datos LoRaWAN.

Se pueden observar payloads LoRa servidos en tiempo real.

<sup>1</sup><<https://labs.mbed.com/simulator/>>

### 8.3.2 Dashboard

Se desarrolla una aplicación interativa que grafica en tiempo real los datos de *humedad* y *temperatura*.

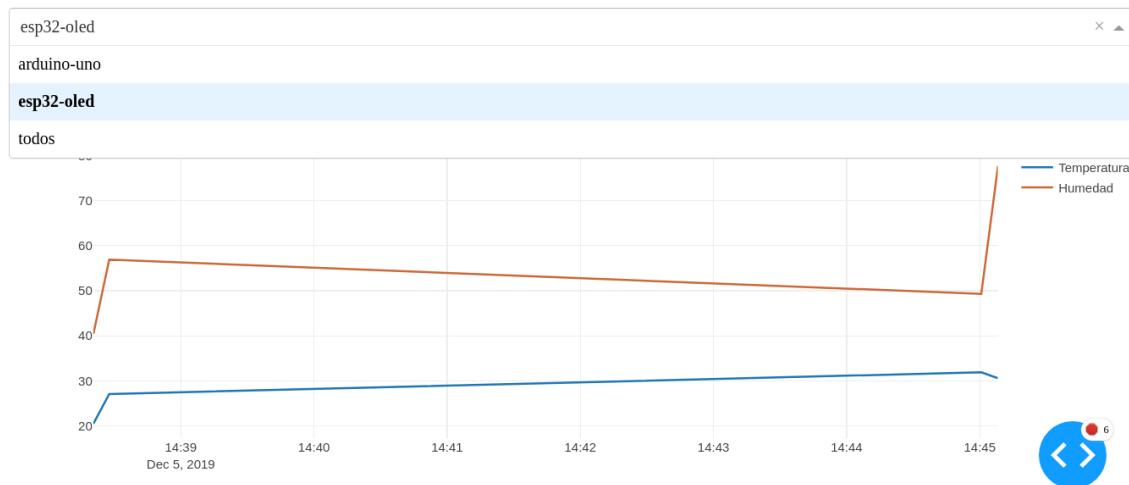


Figura 8.5: Gráfico de datos con filtro por dispositivo esp32.

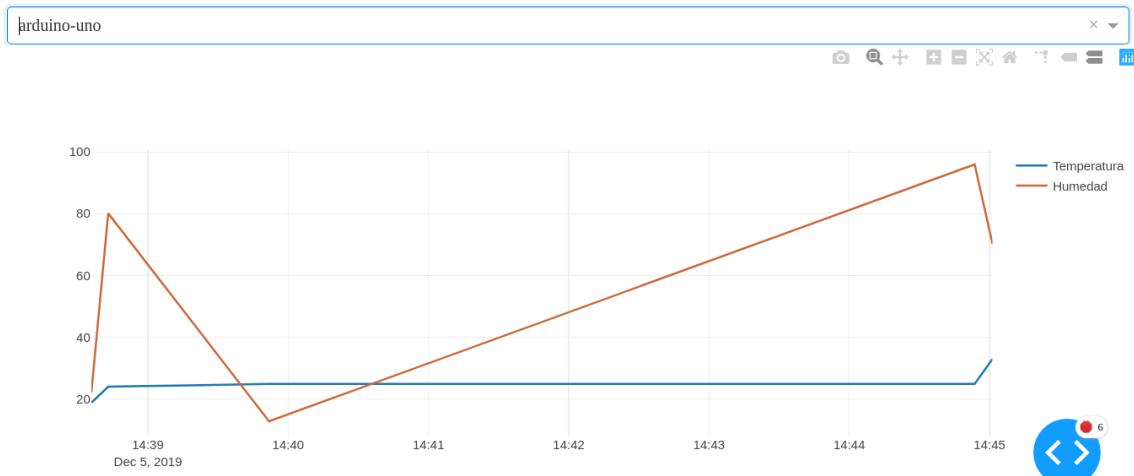


Figura 8.6: Gráfico de datos con filtro por dispositivo arduino-uno.

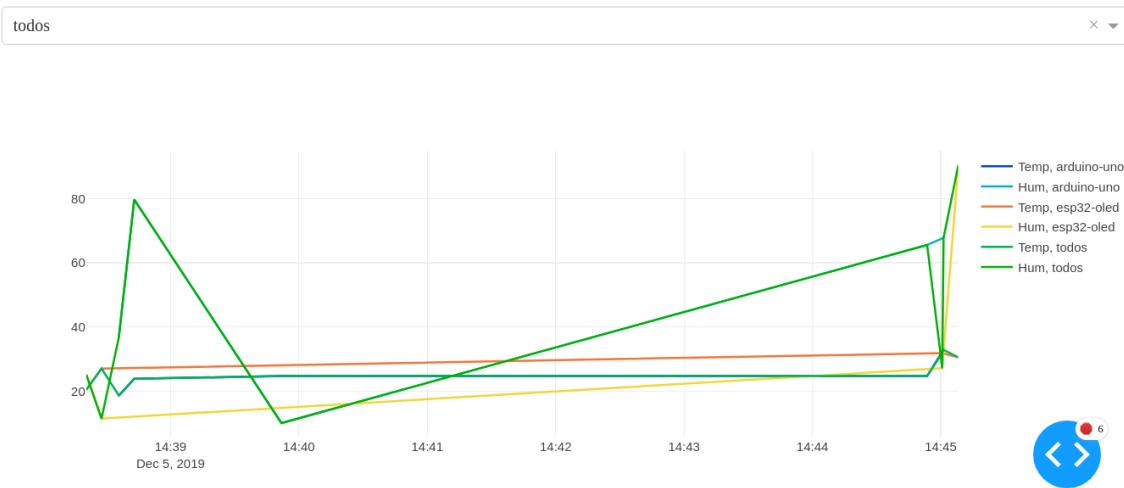


Figura 8.7: Gráfico de datos con filtro por todos los dispositivos.

## 8.4 Análisis de los resultados

El desarrollo, implementación y puesta en marcha del prototipo LoRaWAN se obtiene producto de la búsqueda bibliográfica, confirmar y profundizar conceptos de telecomunicaciones, analizar entre las numerosas opciones, tecnologías, códigos, dispositivos, soluciones, etc, desde una perspectiva ingenieril y con vistas a una evolución que inicia en una prueba de concepto, pero se proyecta a una potencial solución comercial.

# **9 Conclusiones**

## **9.1 Sobre el proyecto**

Se alcanzaron los objetivos de investigación y desarrollo así como de implementación y puesta en marcha de la prueba de concepto y prototipo esencial para mostrar solución de internet de las cosas, con respaldo de tecnologías y decisiones para potencial solucion comercial y para escalar el mismo. Respetando las tareas asignadas y el plan de trabajo, con algunas modificaciones en las tecnologías elegidas y entorno de desarrollo.

## **9.2 Aspectos laborales**

En el equipo practicante-tutor se supieron establecer planificaciones, tareas a realizar y cumplirlas de manera seria y profesional y con el uso de herramientas de gestión de proyectos sin inconveniente alguno. El tutor optó por la flexibilidad horaria, lo que benefició la correcta adaptación a actividades laborales y académicas en el día a día. Medió correspondientemente en la solicitud de adquisición de equipamientos, con la menor demora posible conforme al plan de trabajo.

La modalidad presencial resulta en muchas ocasiones innecesaria, sin embargo la adaptación al lugar de trabajo y personal de la empresa fue natural.

## **9.3 Aspectos profesionales**

Las actividades en el marco de la práctica profesional se llevaron a cabo en un escenario lo más cercano posible a una experiencia laboral real, la relación interpersonal practicante-tutor se desenvolvió naturalmente y como un verdadero equipo de trabajo, mientras que el vínculo con el resto del personal ameno, relaciones interpersonales, en algunas ocasiones la comunicación fuera del equipo practicante-tutor podría haber sido más fluída y en un contexto más organizado. Las responsabilidades asumidas respetaron el plan de trabajo pactado sin eventualidad ni inconveniente alguno.

# 10 Bibliografía

## 10.1 Documentos técnicos

Technical Marketing Workgroup 1.0, "LoRaWAN. What is it?"<sup>1</sup>, LoRa Alliance<sup>2</sup>, Noviembre 2015.

LoRa Alliance<sup>3</sup>, "LoRaWAN Regional Parameters v1.0.3revA"<sup>4</sup>, LoRa Alliance<sup>5</sup>, Inc., 2017.

Cuadro de Atribución de Bandas de Frecuencias de la República Argentina<sup>6</sup>, CABFRA, Julio 2019.

Resolución 581MM18<sup>7</sup>, Poder Ejecutivo Nacional, Septiembre 2018.

Carsten Bormann<sup>8</sup>, RFC 7049 - Concise Binary Object Representation (CBOR)<sup>9</sup>, Octubre de 2013.

Documentación oficial Semtech UDP<sup>10</sup>, Semtech-Cycleo.

Documentación oficial Basic Station Forwarder<sup>11</sup>, Semtech Corp.

Fette Melnikov, RFC N° 6455 - WebSocket protocol<sup>12</sup>, Diciembre de 2011.

## 10.2 Libros

G. C. Hillar,

## 10.3 Artículos académicos

"LoRaWAN - OTA or ABP?"<sup>13</sup>, Newie Ventures.

.^nálisis para Despliegue de una Red de Sensores Heterogénea"<sup>14</sup>, Medina Santiago, Romero Fernando, De Giusti Armando, Tinetti Fernando G.

.^ Study of LoRa: Long Range Low Power Networks for the Internet of Things"<sup>15</sup>, Aloÿs Augustin, Jiazi Yi Thomas Clausen and William Mark Townsley.

---

<sup>1</sup><<https://lora-alliance.org/resource-hub/what-lorawanr>>

<sup>2</sup><<https://lora-alliance.org/>>

<sup>3</sup><<https://lora-alliance.org/>>

<sup>4</sup><<https://lora-alliance.org/resource-hub/lorawanr-regional-parameters-v11rb>>

<sup>5</sup><<https://lora-alliance.org/resource-hub/lorawanr-regional-parameters-v103revA>>

<sup>6</sup><[https://www.enacom.gob.ar/cuadro-de-atribucion-de-bandas-de-frecuencias-de-la-republica-argentina-cabfra\\_\\_p1588](https://www.enacom.gob.ar/cuadro-de-atribucion-de-bandas-de-frecuencias-de-la-republica-argentina-cabfra__p1588)>

<sup>7</sup><<https://www.enacom.gob.ar/multimedia/normativas/2018/res581MM.pdf>>

<sup>8</sup><<mailto:cabo@tzi.org?subject=cbor.io>>

<sup>9</sup><<https://cbor.io/>>

<sup>10</sup><[https://github.com/LoRa-net/packet\\_forwarder/blob/master/PROTOCOL.TXT](https://github.com/LoRa-net/packet_forwarder/blob/master/PROTOCOL.TXT)>

<sup>11</sup><<https://doc.sm.tc/station/>>

<sup>12</sup><<https://tools.ietf.org/rfc/rfc6455.txt>>

<sup>13</sup><[https://static1.squarespace.com/static/560cc2c2e4b01e842d9fac18/t/5a938d38ec212d9451fbecf8/1519619387035/OTAA\\_or\\_AB Pv3.pdf](https://static1.squarespace.com/static/560cc2c2e4b01e842d9fac18/t/5a938d38ec212d9451fbecf8/1519619387035/OTAA_or_AB Pv3.pdf)>

<sup>14</sup><[http://sedici.unlp.edu.ar/bitstream/handle/10915/73360/Documento\\_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y](http://sedici.unlp.edu.ar/bitstream/handle/10915/73360/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y)>

<sup>15</sup><<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038744/>>

## 10.4 Repositorios digitales

"Single Channel LoRaWAN Gateway"<sup>16</sup>, Maarten Westenberg.

"Heltec ESP32 ESP8266 Series Arduino Develop Environment"<sup>17</sup>, Aaron.Lee from Hel-Tec AutoMation, ChengDu, China ☎ 000000000000 www.heltec.cn.

.^Arduino-LMIC library"<sup>18</sup>, Thomas Telkamp and Matthijs Kooijman.

.^ESP32 Single Channel Gateway"<sup>19</sup>, Vergil Cola.

## 10.5 Guías, tutoriales y artículos en blogs

Admin (10 de Octubre de 2018). "LoRa- (Long Range) Network and Protocol Architecture with Its Frame Structure"<sup>20</sup> [*Entrada en un blog*]. Recuperado de Techplayon<sup>21</sup>.

akirasan (13 de Mayo de 2018). "Nodo LoRaWAN con ESP32"<sup>22</sup> [*Entrada en un blog*]. Recuperado de Techplayon<sup>23</sup>.

akirasan (12 de Mayo de 2018). "MiniGateway LoRa monocanal con ESP32"<sup>24</sup> [*Entrada en un blog*]. Recuperado de Techplayon<sup>25</sup>.

---

<sup>16</sup> <<https://github.com/platenspeler/ESP-1ch-Gateway-ver-2.0>>

<sup>17</sup> <[https://github.com/Heltec-Aaron-Lee/WiFi\\_Kit\\_series](https://github.com/Heltec-Aaron-Lee/WiFi_Kit_series)>

<sup>18</sup> <<https://github.com/tftelkamp/arduino-lmic>>

<sup>19</sup> <<https://github.com/vpcola/ESP32SingleChannelGateway>>

<sup>20</sup> <<http://www.techplayon.com/lora-long-range-network-architecture-protocol-architecture-and-frame-formats>>

<sup>21</sup> <<http://www.techplayon.com>>

<sup>22</sup> <<http://akirasan.net/nodo-lorawan-con-esp32/>>

<sup>23</sup> <<http://akirasan.net/>>

<sup>24</sup> <<http://akirasan.net/minigateway-lora-monocanal-con-esp32/>>

<sup>25</sup> <<http://akirasan.net/>>



# Anexo

---

- Anexo
  - Glosario
    - ACK
    - ADR
    - AES
    - ALOHA
    - API
    - Coseno alzado
    - Chirp
    - CRC
    - CUPS
    - FullDuplex
    - gPRC
    - HalfDuplex
    - Hash
    - HTTP
    - IoT
    - I<sup>2</sup>C
    - JSON
    - LNS
    - Open Source
    - RF
    - Payload
    - SPI
    - CSSC
    - TLS
    - UDP
    - Web socket
  - Investigación
    - Formato de Payload
    - Modos de activación (nodos)
    - Protocolos Gateway LoRa
    - Servidores de red y aplicación LoRaWAN
    - Protocolos de integración

- Fundamentos REST API
- Análisis de delay y overhead MQTT vs HTTP
  - Análisis MQTT
  - Análisis HTTP
  - HTTP
- Planilla de horas y diagrama de Gantt

## Glosario

---

### ACK

(**A**CKnowledgement), “acuse de recibo”, mensaje que el destino de la comunicación envía al origen de esta para confirmar la recepción de un mensaje.

Más información: Artículo Wikipedia (<https://es.wikipedia.org/wiki/ACK>)

### ADR

(**A**daptative **D**ata **R**ate), velocidad de datos adaptable. Es un mecanismo que optimiza las velocidades de datos, los tiempos de transmisión y el consumo de energía de la red. El mismo debe estar disponible en cuanto un dispositivo posea condiciones de RF estables.

Más información: Documentación TheThingsNetwork

(<https://www.thethingsnetwork.org/docs/lorawan/adr.html>)

### AES

(**A**dvanced **E**nryption **S**tandard - Estándar avanzado de cifrado), algoritmo de cifrado simétrico.

Más información: Artículo de Wikipedia ([https://es.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](https://es.wikipedia.org/wiki/Advanced_Encryption_Standard))

### ALOHA

Protocolo del nivel de enlace de datos para redes de área local con topología de difusión.

Más información: Artículo de Wikipedia ([https://es.wikipedia.org/wiki/ALOHAnet#El\\_protocolo\\_ALOHA](https://es.wikipedia.org/wiki/ALOHAnet#El_protocolo_ALOHA))

### API

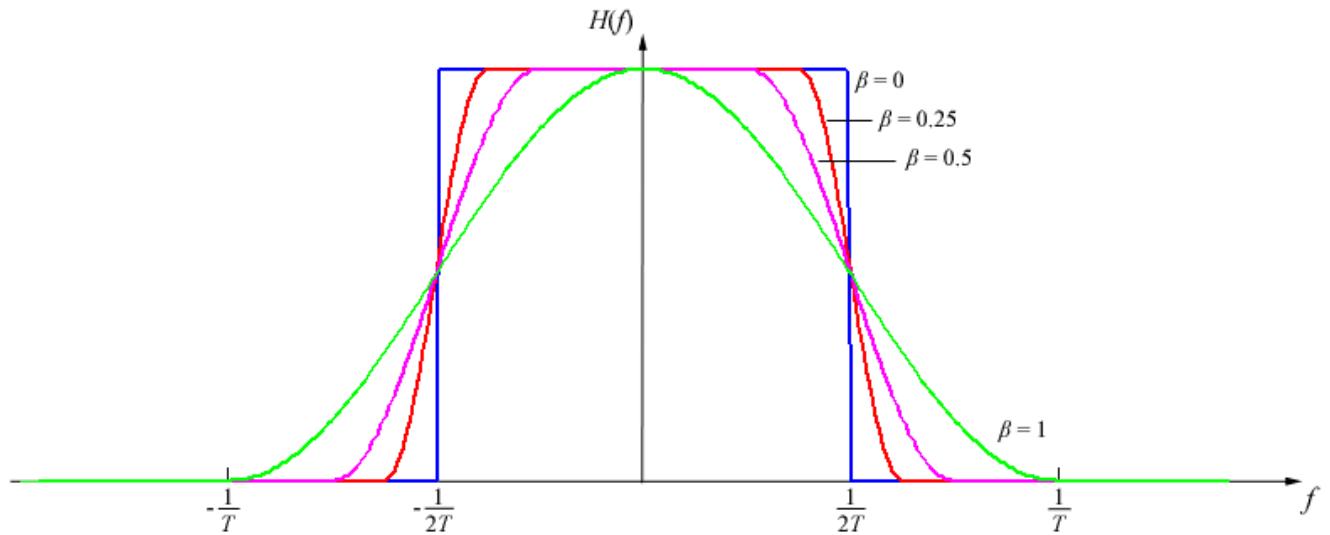
(**A**pplication **P**rogramming **I**nterface), interfáz de programación de aplicaciones, conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Más información: Artículo de Wikipedia.

([https://es.wikipedia.org/wiki/Interfaz\\_de\\_programaci%C3%B3n\\_de\\_aplicaciones](https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones))

## Coseno alzado

Tipo de filtro electrónico a fin de reducir al mínimo la interferencia entre símbolos (ISI).

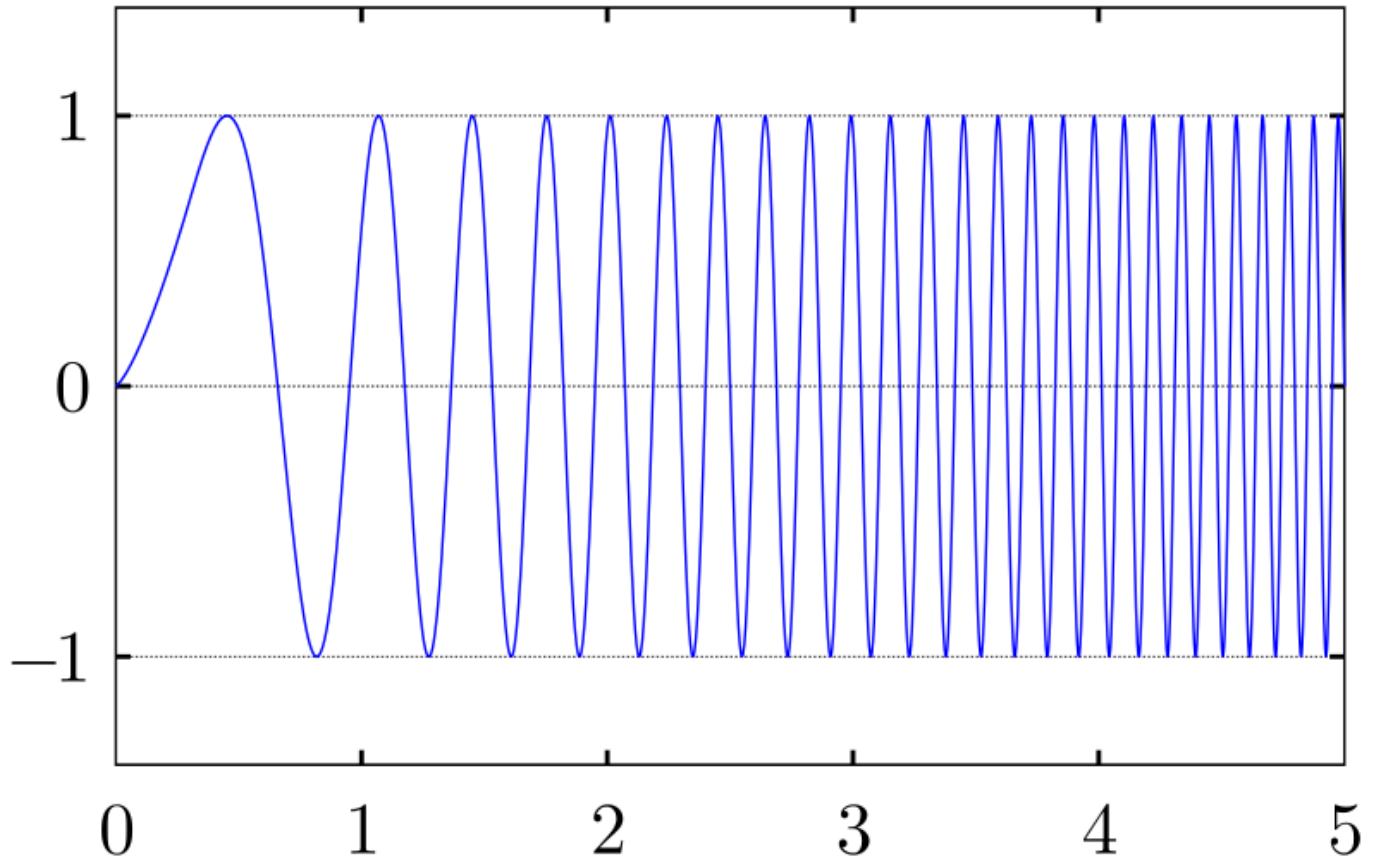


*Raised Cosine Filter Response* de Wdwd (<https://commons.wikimedia.org/wiki/User:Wdwd>) / CCby-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)

Más información: Artículo de Wikipedia. ([https://es.wikipedia.org/wiki/Filtro\\_de\\_coseno\\_alzado](https://es.wikipedia.org/wiki/Filtro_de_coseno_alzado))

## Chirp

Señal cuya frecuencia varía (*aumenta y disminuye*) con el tiempo.



(<https://en.wikipedia.org/wiki/Chirp#/media/File:Linear-chirp.svg>)

*“A linear chirp waveform; a sinusoidal wave that increases in frequency linearly over time”*  
de Georg-Johann (<https://commons.wikimedia.org/wiki/User:Georg-Johann>) / CC BY-SA 3.0

(<https://creativecommons.org/licenses/by-sa/3.0/>)

Más información: Artículo de Wikipedia. (<https://en.wikipedia.org/wiki/Chirp>)

## CRC

**(Cyclic Redundancy Check)**, verificación por redundancia cíclica, es un código de detección de errores usado frecuentemente en redes digitales y en dispositivos de almacenamiento para detectar cambios accidentales en los datos.

Más información: Artículo de Wikipedia.

([https://es.wikipedia.org/wiki/Verificaci%C3%B3n\\_de\\_redundancia\\_c%C3%ADcica](https://es.wikipedia.org/wiki/Verificaci%C3%B3n_de_redundancia_c%C3%ADclica))

## CUPS

*(En el contexto de una red LoRaAN)*. Protocolo que implementa requests HTTP/REST a las estaciones para proveer información de estado y “actualizaciones”.

Más información: Documentación Semtech. (<https://lora-developers.semtech.com/resources/tools/basic-station/the-cups-protocol/>)

## FullDuplex

Sistema de comunicación que admite envío y recepción de información de manera **simultánea**.

Más información: Artículo de Wikipedia.

([https://es.wikipedia.org/wiki/D%C3%BAplex\\_\(telecomunicaciones\)#D%C3%BAplex\\_\(d%C3%BAplex\\_completo\\_o\\_full\\_duplex\)](https://es.wikipedia.org/wiki/D%C3%BAplex_(telecomunicaciones)#D%C3%BAplex_(d%C3%BAplex_completo_o_full_duplex)))

## **gRPC**

Framework RPC ([https://es.wikipedia.org/wiki/Llamada\\_a\\_procedimiento\\_remoto](https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto)) (*Remote Procedure Call, ejecuta código de manera remota.*) open-source.

## **HalfDuplex**

Sistema de comunicación en el que la información se transmite en un sólo sentido, se envía o se recibe, y no de manera simultánea.

Más información: Artículo de Wikipedia.

([https://es.wikipedia.org/wiki/D%C3%BAplex\\_\(telecomunicaciones\)#Semid%C3%BAplex\\_\(half\\_duplex\)](https://es.wikipedia.org/wiki/D%C3%BAplex_(telecomunicaciones)#Semid%C3%BAplex_(half_duplex)))

## **Hash**

Resultado de aplicar una función de cifrado a un dato.

Más información: Artículo de Wikipedia.

([https://es.wikipedia.org/wiki/Funci%C3%B3n\\_hash\\_criptogr%C3%A1fica#targetText=Las%20funciones%20hash%20criptogr%C3%A1ficas%20son,y%20son%20f%C3%A1ciles%20de%20calcular.](https://es.wikipedia.org/wiki/Funci%C3%B3n_hash_criptogr%C3%A1fica#targetText=Las%20funciones%20hash%20criptogr%C3%A1ficas%20son,y%20son%20f%C3%A1ciles%20de%20calcular.))

## **HTTP**

(*Hypertext Transfer Protocol*), protocolo de transferencia de hipertexto para transferencia de información en la web.

Más información: Artículo de Wikipedia.

([https://es.wikipedia.org/wiki/Protocolo\\_de\\_transferencia\\_de\\_hipertexto#targetText=El%20Protocolo%20de%20transferencia%20en%20la%20WorldWide%20Web.](https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto#targetText=El%20Protocolo%20de%20transferencia%20en%20la%20WorldWide%20Web.))

## **IoT**

(*Internet of Things - Internet de las cosas*). Paradigma referido a la interconexión de objetos a la red (internet), que, previo a su definición, no lo estaban.

Más información: Artículo de Wikipedia. ([https://es.wikipedia.org/wiki/Internet\\_de\\_las\\_cosas](https://es.wikipedia.org/wiki/Internet_de_las_cosas))

## **I<sup>2</sup>C**

(*Inter-Integrated Circuit*)

**Circuito inter-integrado** es un bus serie de datos desarrollado en 1982 por Philips (<https://es.wikipedia.org/wiki/Philips>) Semiconductors (hoy NXP Semiconductors ([https://es.wikipedia.org/wiki/NXP\\_Semiconductors](https://es.wikipedia.org/wiki/NXP_Semiconductors)), parte de Qualcomm (<https://es.wikipedia.org/wiki/Qualcomm>)<sup>1</sup> ([https://es.wikipedia.org/wiki/I%C2%B2C#cite\\_note-1](https://es.wikipedia.org/wiki/I%C2%B2C#cite_note-1))). Se utiliza principalmente internamente para la comunicación entre diferentes partes de un circuito, por ejemplo, entre un controlador y circuitos periféricos integrados.

Más Información: Artículo de Wikipedia. (<https://es.wikipedia.org/wiki/I%C2%B2C>)

## JSON

(*JavaScript Object Notation*), notación de objeto de JavaScript, formato de texto sencillo para el intercambio de datos.

Ejemplo:

```
{"usuario":"user", "password":"123456"};
```

## LNS

(*En el contexto de una red LoRaWAN*). Protocolo que permite comunicación entre dispositivos mediante websocket, para transmportar e intercambiar texto en formato JSON.

Más información: Documentación Semtech. (<https://lora-developers.semtech.com/resources/tools/basic-station/the-lns-protocol/>)

## Open Source

Código Abierto, es un modelo de desarrollo de software basado en la colaboración abierta. Si un software es de código abierto, entonces el código se encuentra disponible, garantizando ciertas libertades.

Más información: Artículo de Wikipedia ([https://es.wikipedia.org/wiki/C%C3%B3digo\\_abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto))

## RF

**Radio Frecuencia**, rango de frecuencias entre 3 Hz y 300 GHz del espectro electromagnético.

Más información: Artículo de Wikipedia (<https://es.wikipedia.org/wiki/Radiofrecuencia>)

## Payload

*Carga útil*, datos enviados en un canal de comunicación excluyendo cabecera o metadatos.

Más información: Artículo de Wikipedia  
([https://es.wikipedia.org/wiki/Carga\\_%C3%BAtil\\_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Carga_%C3%BAtil_(inform%C3%A1tica)))

## SPI

**(Serial Peripheral Interface).** Interfaz de periféricos en serie, estándar de comunicaciones sincrónicas para la transferencia de información entre circuitos integrados en equipos electrónicos en serie.

Más información: Artículo de Wikipedia. ([https://es.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](https://es.wikipedia.org/wiki/Serial_Peripheral_Interface))

## CSSC

**(Chirp Spread Spectrum),** espectro ensanchado con chirp, técnica de modulación de gran ocupación espectral y cuyas señales varían en frecuencia con el tiempo.

Más información: Artículo de Wikipedia. ([https://en.wikipedia.org/wiki/Chirp\\_spread\\_spectrum](https://en.wikipedia.org/wiki/Chirp_spread_spectrum))

## TLS

**(Transport Layer Security),** seguridad en capa de transporte, conjunto de protocolos de cifrado para seguridad en comunicaciones de redes de datos.

Más información: Artículo de Wikipedia. ([https://es.wikipedia.org/wiki/Transport\\_Layer\\_Security](https://es.wikipedia.org/wiki/Transport_Layer_Security))

## UDP

**(User Datagram Protocol),** protocolo de datagramas de usuario, protocolo del nivel de transporte basado en el intercambio de datagramas, correspondiente a capa 4 del modelo OSI. No implementa corrección de errores.

Más información: Artículo de Wikipedia. ([https://es.wikipedia.org/wiki/Protocolo\\_de\\_datagramas\\_de\\_usuario](https://es.wikipedia.org/wiki/Protocolo_de_datagramas_de_usuario))

## Web socket

Tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

Más información: Artículo de Wikipedia. (<https://es.wikipedia.org/wiki/WebSocket>)

## Investigación

---

A continuación se anexan los resultados de investigación para toma de decisiones por sección

# Formato de Payload

(*Frame Payload*)

A continuación se describen los formatos de payload bien conocidos:

## CBOR

*Concise Binary Object* (RFC 7049 (<https://cbor.io>))

Formato diseñado para un tamaño de código extremadamente pequeño, por ende un tamaño de mensaje pequeño y extensibilidad sin la necesidad de negociar la versión.

- **Estructura de datos JSON:** Soporta objetos de tipo `number`, `string`, `array`, `map`, `bool`. No requiere un esquema en particular. Pero requiere codificar objetos como claves, gráficos o valores sensados, usualmente se realiza en `base64`, también es posible la codificación binaria (*procesamiento más veloz, implementación simple*).
- **"tags":** Definidos como mecanismo de identificación de **información adicional** al modelo básico. Tanto futuras versiones del estándar como *terceros* pueden definir tags.

Se puede implementar con numerosos lenguajes de programación incluyendo *Python*, *C* y hasta *Scratch*.

Ejemplo simple:

```
{"Fun": true, "Amt": -2}
```

## Cayenne LPP

*Cayenne Low Power Payload* (<https://github.com/myDevicesIoT/CayenneLPP>)

El formato de payload para redes LoRaWAN. Es compatible con reglas de **restricción de tamaño** por debajo de `11bytes`, lo que permite a un dispositivo enviar múltiples datos a la vez.

Además, permite enviar diferentes datos sensados en *diferentes frames*. Para ello, cada dato debe acompañarse de un prefijo de `2bytes`:

- **Data Channel:** Identifica únicamente a un sensor en los frames. Por ejemplo: “*indoor sensor*”
- **Data Type:** Identifica el *tipo de dato* en el frame, por ejemplo “*temperature*”.
- **Estructura del payload:**

1 Byte	1 Byte	N Bytes	1 Byte	1 Byte	M Bytes	...
Data1 Ch.	Data1 Type	Data1	Data2 Ch.	Data2 Type	Data2	...

- **Tipos de datos (Data type)** Corresponde a los lineamientos de la *IPSO Alliance Smart Objects*, identificando los tipos con un ID de objeto.

- **LPP Data Type**: Identificador de `1byte` que se corresponde con el identificador IPSO y puede convertirse de manera sencilla. Por ejemplo:

```
LPP_DATA_TYPE = IPSO_OBJECT_ID - 3200
```

Donde `3200` es el ID IPSO, mientras que el ID LPP es `0` y se corresponde con una entrada digital.

Ejemplo de payload enviando *dos datos sensados a la vez* :

Payload (Hex)	03 67 01 10 05 67 00 FF	
Data Channel	Type	Value
03 ⇒ 3	67 ⇒ Temperature	0110 = 272 ⇒ 27.2°C
05 ⇒ 5	67 ⇒ Temperature	00FF = 255 ⇒ 25.5°C

Compatible con Arduino .

## Modos de activación (nodos)

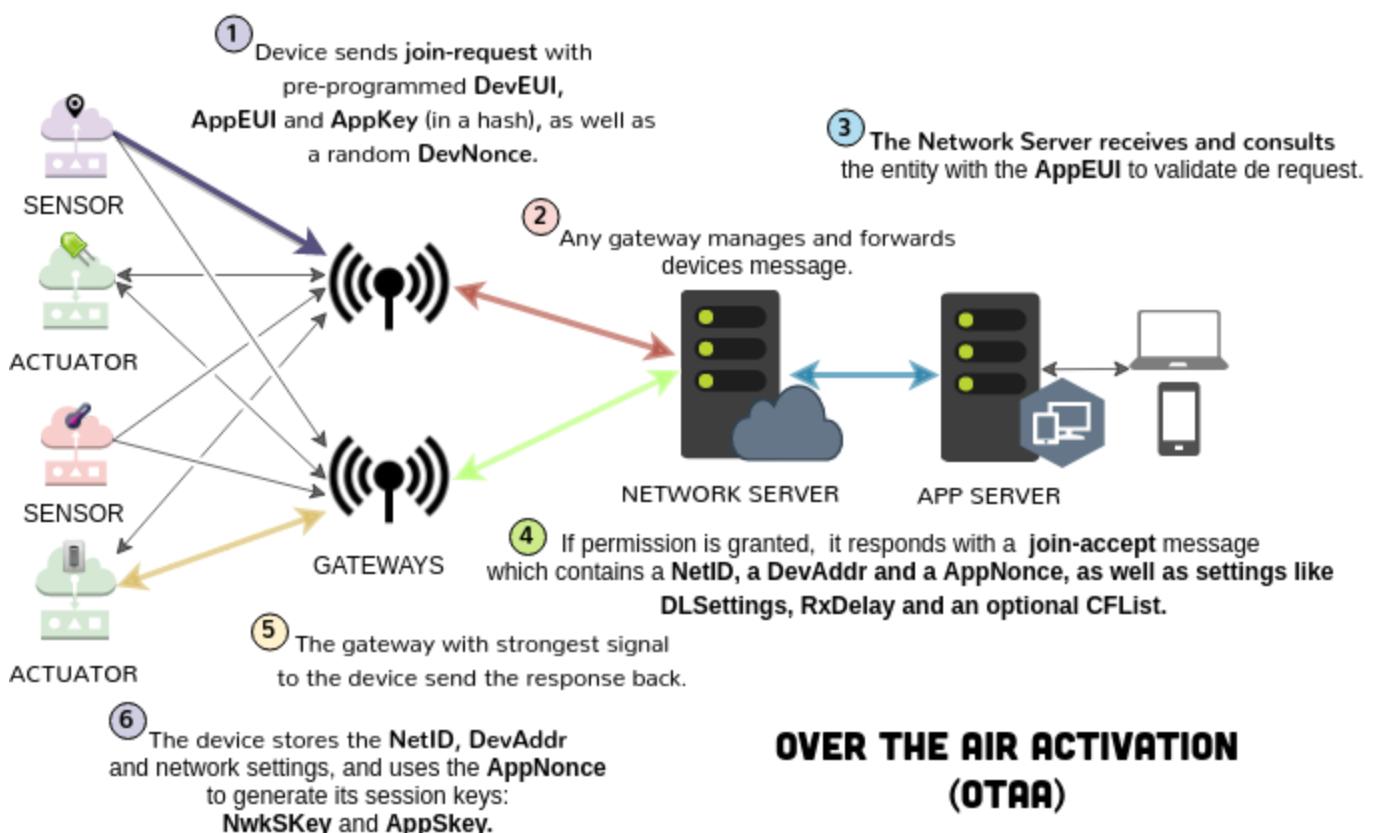
Parametros para la activación:

- **AppEUI**: Identificación única de aplicación que agrupa objetos *por aplicación*. (*Configurable*).
- **AppKey**: Clave secreta AES de 128bits compartida entre el dispositivo periférico y la red. Se utiliza para determinar las claves de sesión. (*Configurable*).
- **AppSKey**: *Application Sesion Key*. Clave para cifrar el payload.
- **AppNonce**: Número único enviado desde la red a los end-points durante el inicio de sesión para generar la clave de sesión.
- **CFLIST**: *Channel Frequency List* Configuración de frecuencias para cada canal.
- **DevAddr**: Dirección del dispositivo, lo identifica en una red.
- **DevEUI**: Número de identificación de fábrica, único para cada dispositivo. (*Configurable*).
- **DevNonce**: Número al azar de uso único que envía un end-point para evitar que un dispositivo no autorizado replique un mensaje de solicitud de inicio de sesión.

- **DLSettings** Datos de configuración para el enlace de bajada, velocidad velocidad de datos.
- **EUI**: Identificador único extendido, ID global.
- **Nonce** Número que se usa por única vez durante el cifrado de mensajes.
- **NetIDNetwork**: Identificador único de red.
- **NwkSKey**: *Network Session Key*. Llave para cifrar el paquete con metadatos de la sesión.
- **RxDelayReceive Delay**: Tiempo entre transmisión y recepción.

A continuación se describe el procedimiento correspondiente a los dos modos de activación posibles:

### OTAA (Over The Air Activation) {#dOTAA}



([https://commons.wikimedia.org/wiki/File:Over\\_The\\_Air\\_Activation\\_LoRaWAN.png](https://commons.wikimedia.org/wiki/File:Over_The_Air_Activation_LoRaWAN.png))

“Over The Air Activation LoRaWAN” de brivideneira / CC BY SA 4.0

(<https://creativecommons.org/licenses/by-sa/4.0/deed.en>)

1. El **end-point** a conectar envía un **join-request** con los parámetros **DevEUI**, **AppEUI** y **AppKey** (*se envía en un hash*) previamente configurados. Además genera un **DevNonce**.
2. El **gateway** reenvía el mensaje al *servidor de red*. Si recibiera mensajes duplicados, reenvía sólo uno.
3. El **servidor de red** consulta la entidad asociada al **AppEUI** recibido, si se autoriza el inicio de sesión, se envía un **join-accept**, o mensaje de aceptación al inicio de sesión

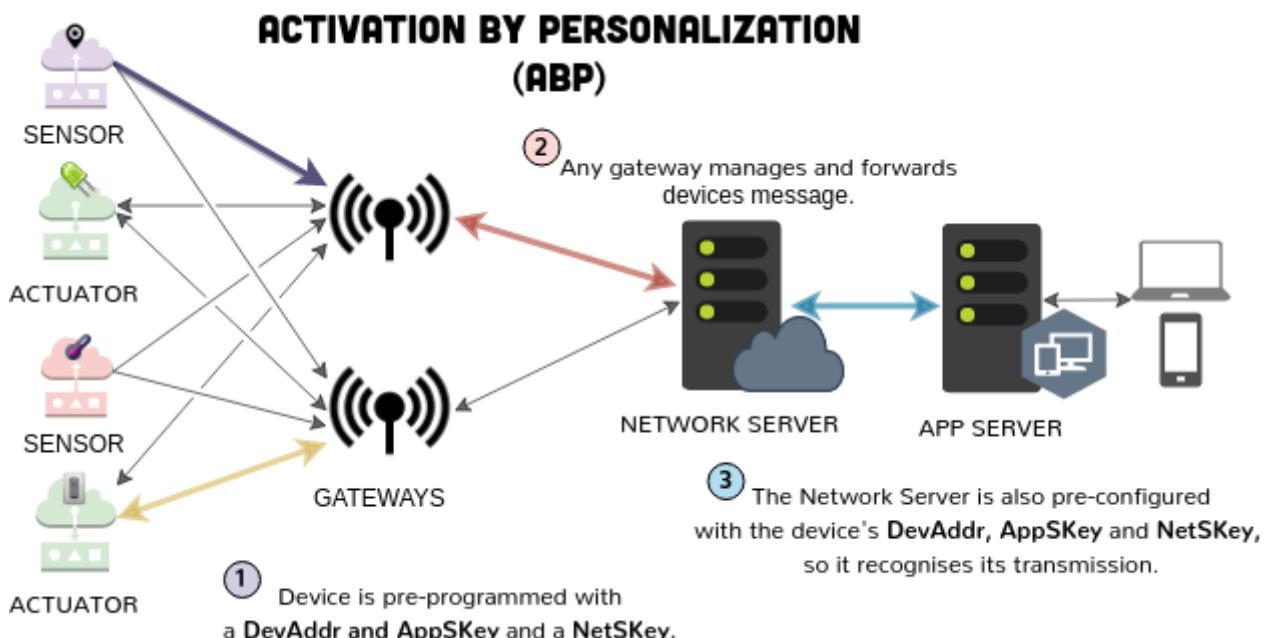
con los siguientes parámetros:

- **NetID**
- **DevAddr**
- **AppNonce**
- **DLS****ettings**
- **RxDelay**
- (opcional) **CFL****ist**.

4. El **gateway** reenvía el **join-accept** al end-point.

5. El **end-point** almacena los parámetros de **NetID** y **DevAddr**, y usa el **AppNonce** para generar las claves **NwkSKey** y **AppSKey** de la sesión.

## ABP (Activation by personalization) {dABP}



(<https://commons.wikimedia.org/wiki/Special:UploadWizard>)

“Activation By Personalization LoRaWAN” de brivadeneira / CC BY SA 4.0

(<https://creativecommons.org/licenses/by-sa/4.0/deed.en>)

1. El **end-point** a conectar posee los parámetros **DevAddr**, las claves **NwkSKey** y **AppSKey** de la sesión pre-configuradas, con las que envía lo que desea transmitir.
2. El **gateway** reenvía el mensaje al *servidor de red*. Si recibiera mensajes duplicados, reenvía sólo uno.
3. El **servidor** posee, pre-configurados los parámetros **DevAddr**, las claves **NwkSKey** y **AppSKey**, de manera que reconoce o no la transmisión.

Más información “OTAA or ABP v3”

([https://static1.squarespace.com/static/560cc2c2e4b01e842d9fac18/t/5a938d38ec212d9451fbecf8/1519619387035/OTA\\_A\\_or\\_AB Pv3.pdf](https://static1.squarespace.com/static/560cc2c2e4b01e842d9fac18/t/5a938d38ec212d9451fbecf8/1519619387035/OTA_A_or_AB Pv3.pdf))

## Protocolos Gateway LoRa

Los mensajes recibidos desde los end-points deben ser reenviados al servidor empleando un *protocolo de redes de datos*. A continuación se detallan los protocolos compatibles con redes LoRa.

### Semtech UDP

*Protocolo básico de comunicación entre LoRa gateway y LoRa server.*

Crea datagramas UDP para enviar mensajes a los end-points (vía RF), para reenviar mensajes desde los end-points hacia el servidor (vía IP).

*No hay autenticación de gateways o servidor y los mensajes de ACK tienen como objeto medir calidad de la red y no corregir o reenviar datagramas UDP.*

Más información: Documentación repo oficial ([https://github.com/LoRa-net/packet\\_forwarder/blob/master/PROTOCOL.TXT](https://github.com/LoRa-net/packet_forwarder/blob/master/PROTOCOL.TXT))

### Basic Station forwarder

Características generales:

- Centraliza la gestión de actualizaciones y configuración (CUPS), mediante el protocolo HTTP/REST.
- Usa TLS para autenticación de cliente y servidor.
- Implementa LNS (*Layer 2 Tunneling Protocol*) para intercambiar mensajes con el servidor a través de un web socket

Más información: Documentación oficial Basic Station Forwarder (<https://doc.sm.tc/station/>)

Ambos protocolos son open source e intercambian mensajes entre gateways y servidor de red en formato JSON.

## Servidores de red y aplicación LoRaWAN

### LoRaServer {dLoRaServer}

Componente de solución Open Source para puesta en marcha de redes LoRaWAN, se complementa con solución para servidor de aplicación.

Se encuentra bajo la licencia MIT, por lo que permite ser usado con fines comerciales.

Se llevan a cabo tareas de **monitoreo** del estado de la red, de **administración de dispositivos**, su estado antes de unirse a la red y durante la transmisión. El estado de las baterías y los enlaces.

Elimina mensajes duplicados. Además, *encola* los mensajes a enviar a gateways o end-points hasta que un gateway pueda recibirlas.

Soporta **end-points clase A/B/C** y modos de activación ABP así como OTAA. Sincroniza el clock de los dispositivos con el clock del servidor. Es compatible con dispositivos **LoRaWAN 1.0 y 1.1** de manera simultánea.

Implementa **ADR Adaptive data-rate**, adaptando la velocidad de datos para reducir el tiempo de transmisión y el consumo de energía de los dispositivos.

Soporta **Channel (re)configuration** re configuración de canales automáticamente en los dispositivos para usar un plan pre-definido.

Registra estadísticas, localización y re configuración de los canales para los gateways. Soporta geolocalización integrando *LoRa Geolocation Server*.

Envía mensajes de *multicast*. Implementa además funciones de ruteo ante dos o más dispositivos a transmitir.

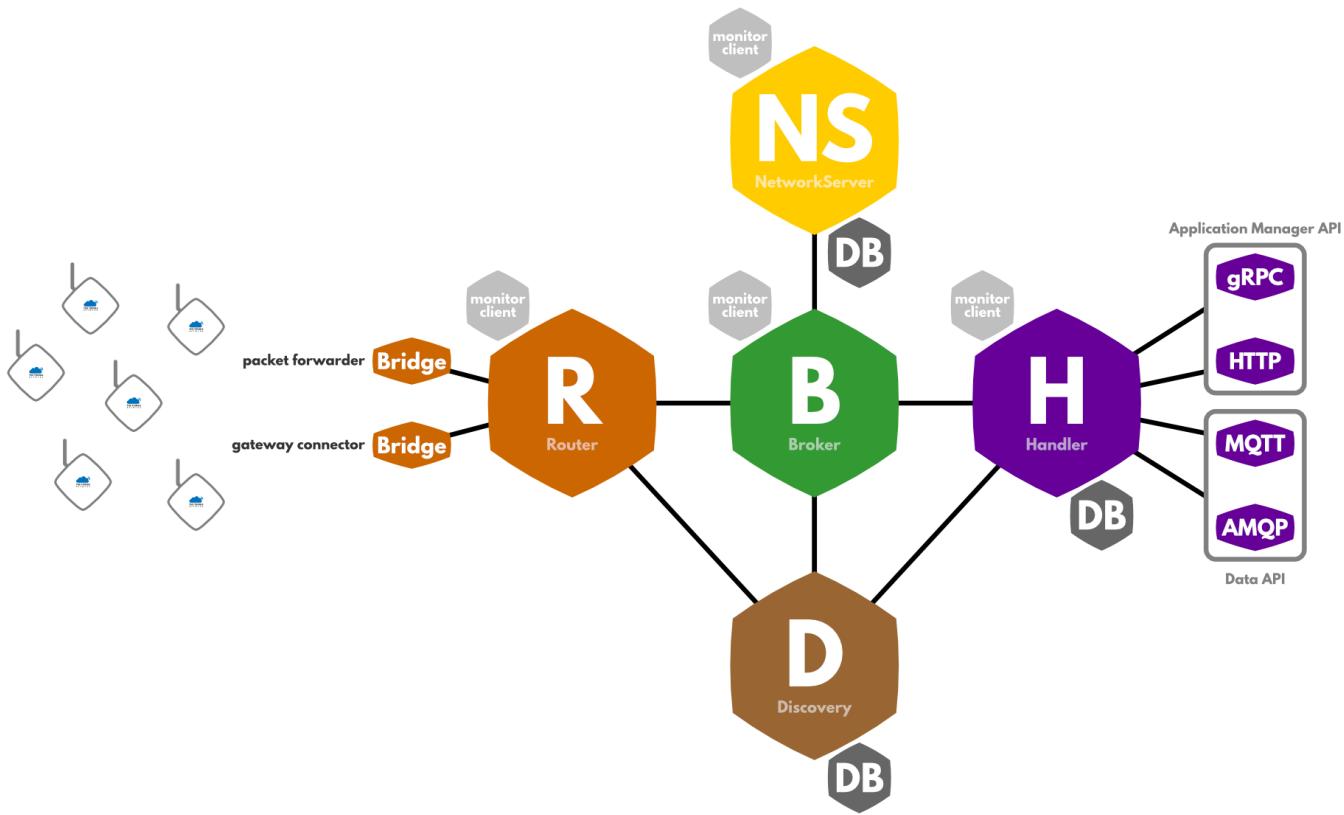
El servidor de red LoRa provee además una API para integrar con otros componentes de red, una solución para el *servidor de aplicación* y un complemento para control de la red.

*System architecture* (<https://www.loraserver.io/overview/architecture>), **LoRaServer**.

### The Things Network

The Thinkgs Network se implementa en una **red pública** en la que las tareas de enrutado se realizan de forma **descentralizada y distribuída**, en la que cada componente “aporta” al backend, formando parte de una comunidad de red.

Su arquitectura es como sigue:



*Network Architecture* (<https://www.thethingsnetwork.org/docs/network/architecture.html>), The Things Network.

Funciona bajo una filosofía de distribuir y **separar** funciones bien definidas. Se describen a continuación los dispositivos que se encuentran entre los **gateways** y el **servidor de red**:

- **Router**: Es responsable de todas las funcionalidades relacionadas con los **gateways** y los parámetros regionales.
- **Broker**: Administra el rango de direcciones de un dispositivo y es el responsable de *encontrar el Handler* correcto al que debe reenviar los mensajes.
- **Network Server (Servidor de red)**: Mantiene el estado de cada dispositivo individual.
- **Handler**: Es el responsable de *cifrar y descifrar* los mensajes así como de reenviarlos a las aplicaciones.

La solución provista por TTN está pensada para formar parte de la *red pública*, sin embargo, es posible la puesta en marcha de *redes privadas*, pero se requiere el uso del **servidor de cuentas TTN** para tareas de **autenticación y autorización**.

*Otro escenario posible es el de una red privada intercambiando datos con la red pública.*

#### Compact server for private LoRaWAN networks

Servidor LoRaWAN Open-source que integra tanto el **servidor de red** como el **servidor de aplicación**, pensado para implementación de redes LoRaWAN privadas.

Soporta **todos los estandares regionales de LoRaWAN 1.0.3**.

Se consultó a desarrollador sobre soporte de norma regional argentina:

“Yes, both AU915-928 and US902-928 regions are supported.

Cheers,  
Petr”

Compatible con un número indefinido **end-points clase A/C** y modos de activación ABP así como OTAA.

Desempeña todos los cifrados y testeos de integridad requeridos. Soporta contadores de frames como testeo para activación **ABP**.

Compatible con **LoRaWAN 1.0.3**. Soporta **gateways basados** en *Packet Forwarder*, como *the Semtech LoRa demo kit*, *LoRa Lite gateway*, *LORANK-8*, *MultiConnect Conduit*, or *Kerlink Wirnet Stations*, así como gateways bajo el protocolo **Basic Station LNS**.

La cantidad de nodos y gateways queda limitada por el hardware empleado.

Cuenta con módulos internos de aplicaciones lógicas, con ejemplos para:

- **Semtech/IMST LoRaMote**
- **Microchip LoRa Technology Mote** (usando un plug-in).

Parsea automáticamente formatos de payloads bien conocidos:

- **Cayenne Low Power Payload.**
- **Concise Binary Object Representation (CBOR).**

Es capaz de almacenar datos directamente a una base de datos de **MongoDB**.

Cuenta con **aplicaciones** externas a través:

- **WebSocket** protocol RFC6455.
- HTTP/1.1 and HTTP/2 protocol (**REST API**)
- **MQTT** v3.1/v3.1.1
- Aplicaciones cloud hosteadas en:
  - **Amazon AWS IoT**
  - **IBM Watson IoT Platform**
  - **MathWorks ThingSpeak**
  - **Microsoft Azure IoT Hub**
  - **ThingsBoard Open-source IoT Platform**
  - **Adafruit IO**
  - **Orange Live Objects**

Soporta los protocolos de capa de aplicación **AMQP 0-9-1** a **RabbitMQ**.

Compatible con datos *confirmados* y *no confirmados* para enlace de subida y de bajada.  
Soporta mensajes de multicast.

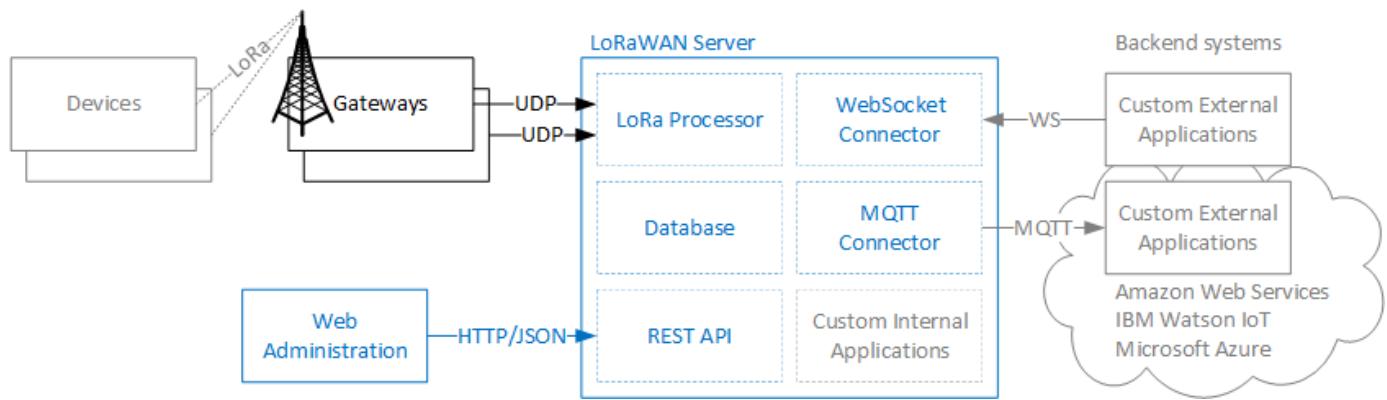
Provee **interfaz web de gestión**. Soporta configuración *manual* así como *automática* (ADR) de la tasa de datos.

**Monitorea** el estado, batería y calidad de conexión del *servidor*, *gateways* y *nodos*. Puede enviar **alertas** vía e-mail o Slack sobre el estado de los dispositivos.

Se integra con los siguientes Sistemas Operativos: *Windows*, **GNU/Linux**, *OS X*, *Solaris*, **Raspbian**, *mLinux*, *Yocto/OpenEmbedded*, *OpenWrt*, **Docker**. Compatible con Clusters para alta performance. *Compatible también con Erlang*.

*Brinda asistencia y recibe solicitudes de funcionalidades para la API*.

Pensado para redes LoRaWAN privadas, requiere instalar *Erlang/OTP 20.3+*.



*The main components of the lorawan-server de petr gotthard* (<mailto:petr.gotthard@centrum.cz>) /  
MIT License (<https://github.com/gotthardp/lorawan-server/blob/master/LICENSE>)

## FloraNet

Servidor de red LoRaWAN Open Source para puesta en marcha de redes LoRaWAN, incluye interfaz de usuario web, gRPC, REST API.

Se encuentra bajo la licencia MIT, por lo que permite ser usado con fines comerciales.

Compatible con normas regionales US 902-928 MHz y **AU 915-928 MHz**.

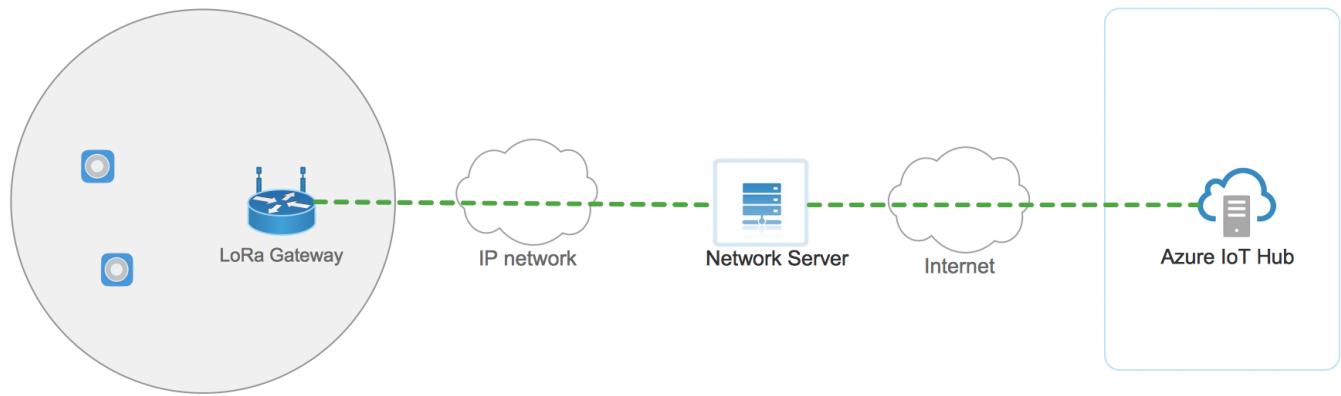
Soporta **end-points clase A/C** y modos de activación ABP así como OTAA.

Elimina mensajes duplicados proveniente de gateways.

Implementa **ADR**.

Compatible con **Azure IoT Hub**, emplea los protocolos **HTTPS** y **MQTT**.

Documentación (wiki floranet (<https://github.com/Fluent-networks/floranet/wiki>)) disponible.



*FloraNet architecture de Frank Edwards (<https://github.com/frankedward>) / MIT License*  
*(<https://github.com/gotthardp/lorawan-server/blob/master/LICENSE>)*

Requiere **Python**.

### LoRa Application Server

Componente de la red LoRaWAN desde el que se dan de alta los dispositivos (*trabaja con join-request*), se realizan tareas de cifrado y descifrado, se reciben los datos de los endpoints y se realizan tareas de interés para el usuario final como *visualización, estadísticas, comunicación por mensajería instantánea*, etc.

### LoRa App Server

Servidor de aplicación open source, parte de la solución LoRa Server (<https://www.loraserver.io/>), incluye interfaz de usuario web, gRPC, REST API. Permite además la administración de usuarios con diferentes permisos. Monitorea estado de los gateways. Se integra con MQTT, HTTP.

Más información Features LoRa Server (<https://www.loraserver.io/lora-app-server/overview/features/>).

### The Things Network

El servidor de red permite crear aplicaciones de manera sencilla, además, se integra con las siguientes tecnologías:

- Go (<https://www.thethingsnetwork.org/docs/applications/golang/>)
- Java (<https://www.thethingsnetwork.org/docs/applications/java/>)
- Node-RED (<https://www.thethingsnetwork.org/docs/applications/nodered/>)
- Node.js (<https://www.thethingsnetwork.org/docs/applications/nodejs/>)
- Python (<https://www.thethingsnetwork.org/docs/applications/python/>)

### Compact server for private LoRaWAN networks

Descripto en sección de servidor de red.

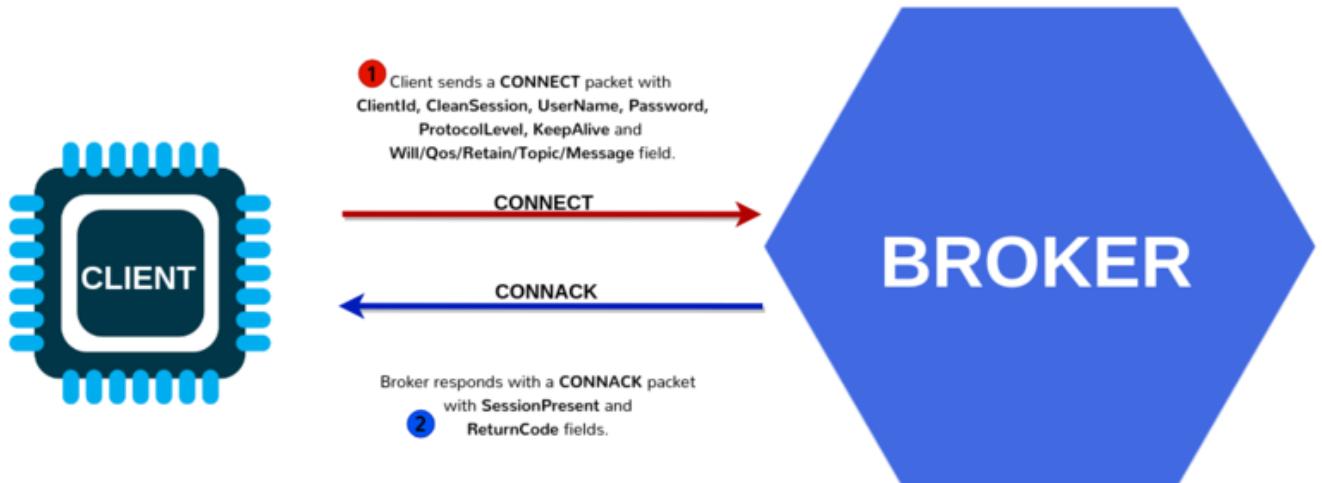
## Protocolos de integración

### Fundamentos MQTT

Todos los clientes se comunican con el **broker** o **servidor**, éste se encarga de la autenticación y autorización de los mismos. Un cliente puede **publicar** un mensaje, enviandolo al servidor, o puede “*estar interesado en recibir cierto tipo de mensajes*”, por lo que se **suscribe** para hacerlo. El broker recibe los mensajes publicados y los distribuye a los clientes suscriptos a ese tipo de mensajes.

#### Autenticación

El cliente envía un paquete **CONNECT**.



([https://commons.wikimedia.org/wiki/File:MQTT\\_autenticacion.png](https://commons.wikimedia.org/wiki/File:MQTT_autenticacion.png))

“MQTT authentication” de brivadeneira / CC BY SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/deed.en>)

- **ClientID:** string único que identifica a los clientes, si el campo está vacío, el servidor genera uno.
- **CleanSession:** bool , indica cómo proceder cuando el cliente se desconecta, si está seteado en 1 o True , se descarta toda información relacionada con la sesión. Si se setea en 0 o False , el servidor almacena los datos de la sesión y cuando el cliente se reconecta, gozade una sesión persistente.
- **UserName:** bool , indica si el cliente desea especificar un nombre de usuario o no con True / 1 o False / 0 correspondientemente y especificarlo en el campo Userfield .
- **password:** bool , indica si el cliente desea especificar una contraseña de usuario o no con True / 1 o False / 0 correspondientemente y especificarlo en el campo password .

- **ProtocolLevel:** Indica la versión de MQTT que debe usar el broker.
- **KeepAlive:** Tiempo en segundos al cabo del cual el cliente debe enviar mensajes de control al servidor. Si el parámetro es 0, el cliente debe enviar un PINGREQ o ping request para indicar al servidor que continúa conectado, al que responde el servidor con un PINGRESP. Si nada de lo anterior sucede, la conexión se cierra.
- **Will, WillQoS, WillRetain, WillTopic y WillMessage:** Indican al servidor almacenar o no el último mensaje de la sesión, la calidad de servicio del último mensaje, si se debe mantener el último mensaje o no, correspondientemente. Si will es 1 o True, se deben especificar WillMessage y WillTopic, si el cliente se desconecta, el servidor publica el mensaje en el tópico correspondiente, con la calidad de servicio especificada.

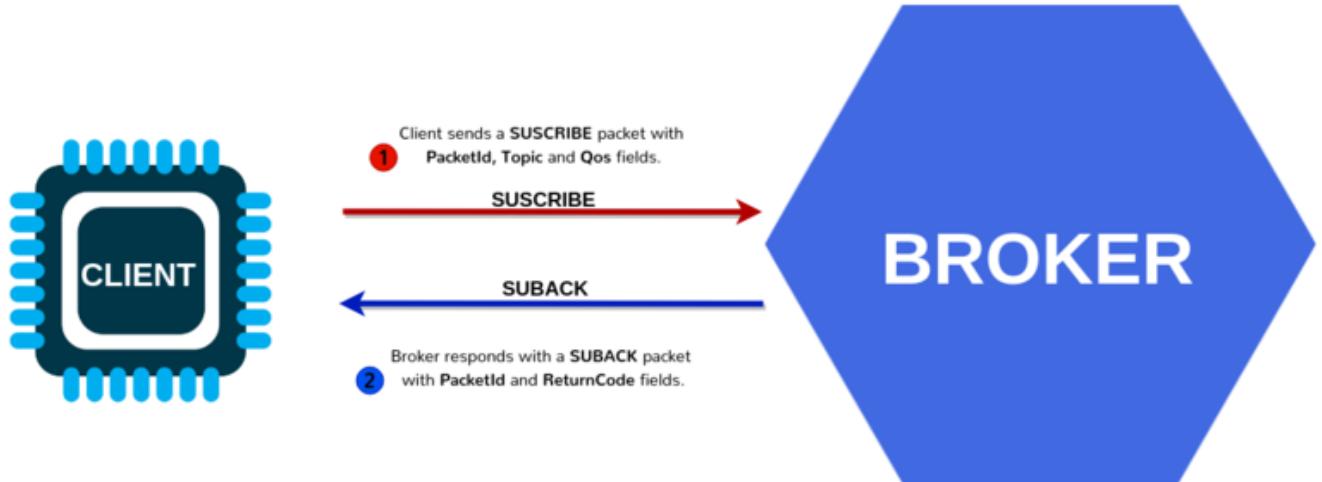
El servidor responde con un paquete **CONNACK** con los siguientes campos:

- **SessionPresent:** (Según CleanSession, 1 / True → 0 / False (no requiere almacenar sesión), 0 / False → 1 / True (sesión persistente))
- **ReturnCode:** Indica resultado de la autenticación según:

ReturnCode	Descripción
0	Conexión aceptada.
1	Conexión rechazada por no soportar versión de MQTT solicitado.
2	Conexión rechazada por haber sido denegado el ClientId solicitado.
3	Conexión rechazada, si bien la red está disponible, no lo está MQTT.
4	Conexión rechazada por usuario o contraseña incorrectos.
5	Conexión rechazada por haber fallado la autenticación.

## Suscripción

El cliente envía un paquete **SUSCRIBE** para suscribirse a uno o más tópicos con un PacketId en el header y uno o más pares Topic, QoS → "topic/1", 0.



([https://commons.wikimedia.org/wiki/File:MQTT\\_autenticacion.png](https://commons.wikimedia.org/wiki/File:MQTT_autenticacion.png))

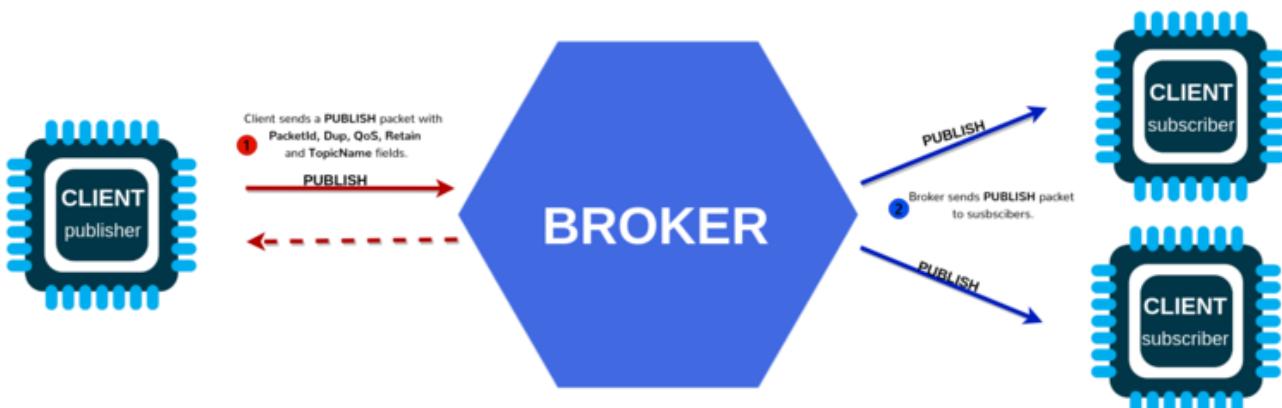
“MQTT suscribe” de brivadeneira / CC BY SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/deed.en>)

El broker responde con un paquete **SUBACK** con el mismo **PacketId** y un **ReturnCode** según:

ReturnCode	Descripción
0	Suscripción exitosa con un máximo QoS de 0
1	Suscripción exitosa con un máximo QoS de 1
2	Suscripción exitosa con un máximo QoS de 2
128	Falló suscripción

## Publicación

Una vez que el cliente se conectó exitosamente con el broker, puede comenzar a **publicar** mensajes, enviando paquetes **PUBLISH**.



([https://commons.wikimedia.org/wiki/File:MQTT\\_publish.png](https://commons.wikimedia.org/wiki/File:MQTT_publish.png))

- **PacketId:** Según QoS , 0 → 0 / null , 1/2 → !=0
- **Dup:** Según QoS , 0 → 0 / null , 1/2 → !=0 , en este último caso el servidor reenvía el mensaje publicado si no se recibió ACK de los clientes suscriptos.
- **QoS:** Nivel de QoS para el mensaje. De ser 0 , ante un mensaje de publicación, el broker no responde, lo envía a los clientes suscriptos. Para valores distintos de 0 , se realiza interacción extra entre cliente que publica y broker.
  - 0: Máximo una entrega, ofrece la misma garantía que TCP, el servidor envía el paquete una vez, el cliente no devuelve ACK, el servidor no almacena el paquete para reenvío ni planifica ninguna otra comunicación luego de la entrega. *Baja sobrecarga, no hay garantía.*
  - 1: Mínimo de una entrega, el servidor envía el paquete, pero lo almacena a la espera un ACK del cliente, hasta entonces, reenvía el paquete. *Garantiza entrega, paquetes duplicados.*
  - 2: Exactamente una entrega, el servidor envía un paquete, recibe un paquete PUBREC , al que responde con un PUBREL y espera un PUBCOMP considerado como ACK, y garantía de exactamente una entrega del paquete. *Garantía de entrega, no hay sobrecarga, datos críticos.*
- **Retain:** 1/True , 0/False , retiene o no el mensaje, si se retiene, es enviado a los clientes que se suscriban al tópico en el futuro.
- **TopicName:** string que indica el nombre del tópico, responden a una jerarquía y usan “/” como delimitador, por ejemplo: "sensors/drone01/altitude" .

Los datos del payload son agnósticos al formato, es decir que puede contener un JSON , un XML o un binario .

El cliente puede enviar un paquete **UNSUSCRIBE** para desuscribirse a uno o más tópicos con un `PacketId` en el header y uno o más pares `Topic` , `QoS` → "topic/1" , 0 .

Recomendaciones para nombres de tópicos:

Crear directorios jerárquicos separados por “/”, por ejemplo:

sensors/drone01/altitude .

NO usar: “+”, “#” ni “\$”.

## Seguridad

Se pueden implementar técnicas de seguridad para la comunicación bajo MQTT en tres diferentes capas:

- **Red (3):** VPN (Virtual Private Network).
- **Transporte (4):** TLS (Transport Layer Security), las comunicaciones MQTT viajan por TCP, por defecto no están cifradas, se puede usar TLS para asegurar cifrado e integridad, se denomina **MQTTS**.

- **Aplicación (5):** Se pueden implementar funcionalidad de autenticación y autorización empleando el `clientId`.

## Mosquitto

Broker o Server Open source compatible con MQTT.

## Fundamentos Web socket

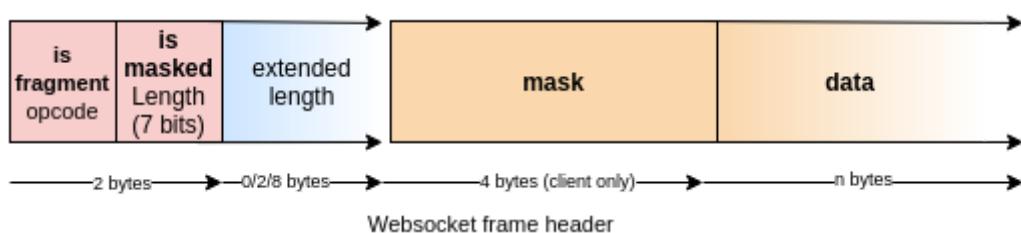
### Handshake de apertura

Toda conexión por websocket comienza con un `request HTTP`, como cualquier otro, pero incluyendo la cabecera `upgrade`, que indica que el cliente desea cambiar a un protocolo diferente, esto es, websocket. Este proceso se conoce como *WebSocket opening handshake* e incluye los siguientes campos:

Header	Descripción
<code>Sec-WebSocket-Key</code>	Se envía por única vez de cliente a servidor en el <code>request HTTP</code> , para evitar ataques “cross-protocol”.
<code>Sec-WebSocket-Accept</code>	Se envía por única vez de servidor a cliente en el <code>response HTTP</code> , para confirmar que el servidor conoce el protocolo websocket.
<code>Sec-WebSocket-Extensions</code>	Puede aparecer más de una vez en el <code>request HTTP</code> , pero solo una en el <code>response</code> , estos datos sirven de acuerdo entre cliente y servidor respecto al uso de extensiones a nivel de protocolo durante la conexión.
<code>Sec-WebSocket-Protocol</code>	Se especifican los protocolos a usar tanto de cliente a servidor como en el sentido contrario.
<code>Sec-Websocket-Version</code>	Se especifica la versión del protocolo websocket, por defecto 13, si el servidor no soporta el protocolo especificado por el cliente, le envía una lista de versiones compatibles.

### Frame websocket

Cada mensaje de la comunicación websocket tiene uno o más frames con el siguiente header:



- **Opcode:** Valor numérico que corresponde con un **tipo de payload**:

Opcode	Descripción
1	Texto
2	Binario
8	Handshake de cierre de sesión
9	Ping
10 (hex 0xa )	Pong (respuesta a ping)

Con cuatro bits, son posibles 16 combinaciones de opcodes, las 11 restantes se reservan para futuros opcodes.

- **Length:** Indica el tamaño del payload:
  - 0-126 : tamaño del payload.
  - 127 : el tamaño se encuentra en el length extendido.
- **Mask:** Indica si se enmascara el frame con un 1 .
- **Data.**

Si se envían mensajes fragmentados en varios frames, el último bit debe setarse en 1 para indicar que se trata del fragmento de un mensaje.

### Handshake cierre

Para diferenciar entre los cierres de sesión que son accidentales y los que no, es posible especificar un código que describa la razón:

Código	Descripción	Cuándo usarlo
0-999	Prohibido	No usar.
1000	Cierre normal	La sesión se completó con éxito.
1001	Cierre aplicación	La aplicación se pierde y no se espera una reconexión.
1002	Error de protocolo	La conexión se rompe por un error de protocolo.
1003	Tipo de dato inesperado	La aplicación recibe un mensaje con un tipo de error inesperado y no se puede manejar.
1004/5/6/7	Reservado	No usar, se definirá en el futuro.
1008	Mensaje viola las políticas	La aplicación cerró la sesión por una razón que no corresponde a los otros códigos, o se recibió un mensaje que no se puede manejar.
1009	Mensaje muy largo	Tamaño inmanejable.
1010	Se requiere extensión	El cliente (navegador) requiere extensiones específicas no negociadas por el servidor.
1011	Condición inesperada	La aplicación no puede seguir manejando la conexión.
1015	TLS (reservado)	Se usa para indicar que TLS falló antes del handshake.
3000-3999	Requiere registro	
4000-4999	Privado	Uso personalizado.

## Fundamentos REST API

Una REST API usa URIs (Uniform Resource Identifiers) como dirección del recurso que deben respetar el siguiente formato:

URI = scheme “://” authority “/” path [ “?” query ] [ “#” fragment ]

- / debe separar jerarquías y no debe estar al final de una URI.
- - se debe usar para facilitar la lectura de la URI y no se debe usar \_
- Se deben usar minúsculas.
- Las URIs no deben incluir la extensión de los archivos.

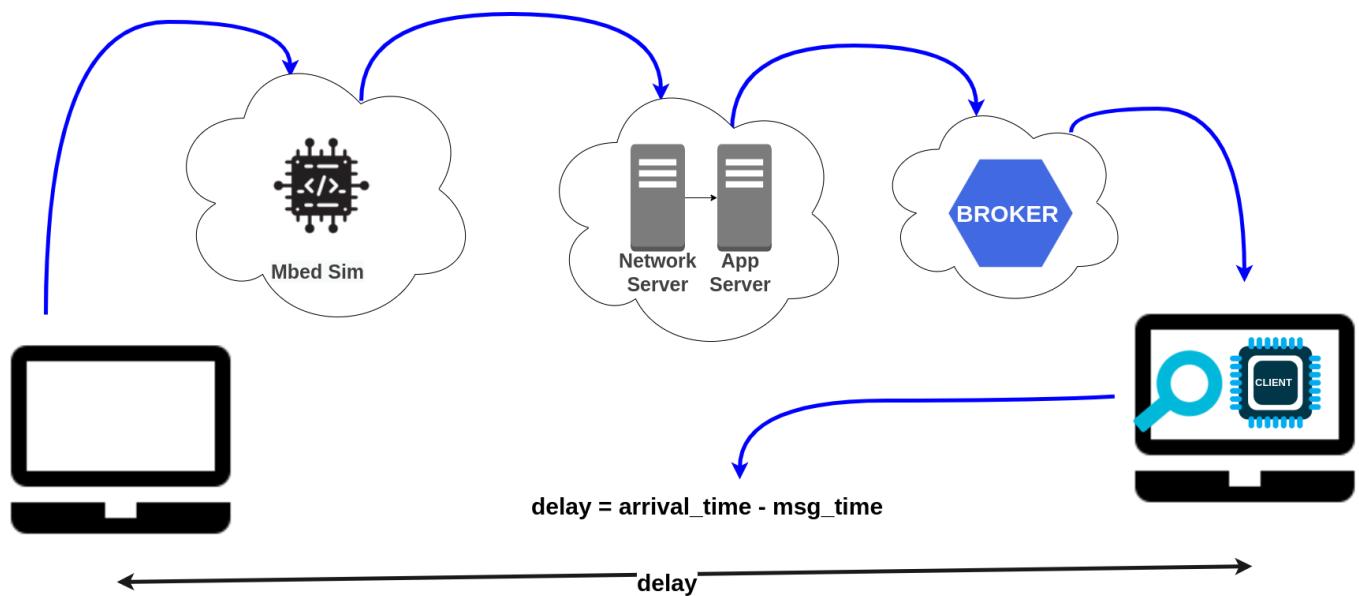
REST API implementa el protocolo HTTP (v1.1), incluyendo los métodos, códigos de respuesta y cabeceras de mensajes.

## Análisis de delay y overhead MQTT vs HTTP

### Análisis MQTT

#### Datos simulados

A continuación se muestra el diagrama de red de análisis de MQTT para la red LoRa simulada:



Los servidores de red y aplicación así como el broker MQTT son servicios provistos por thethingsnetwork.

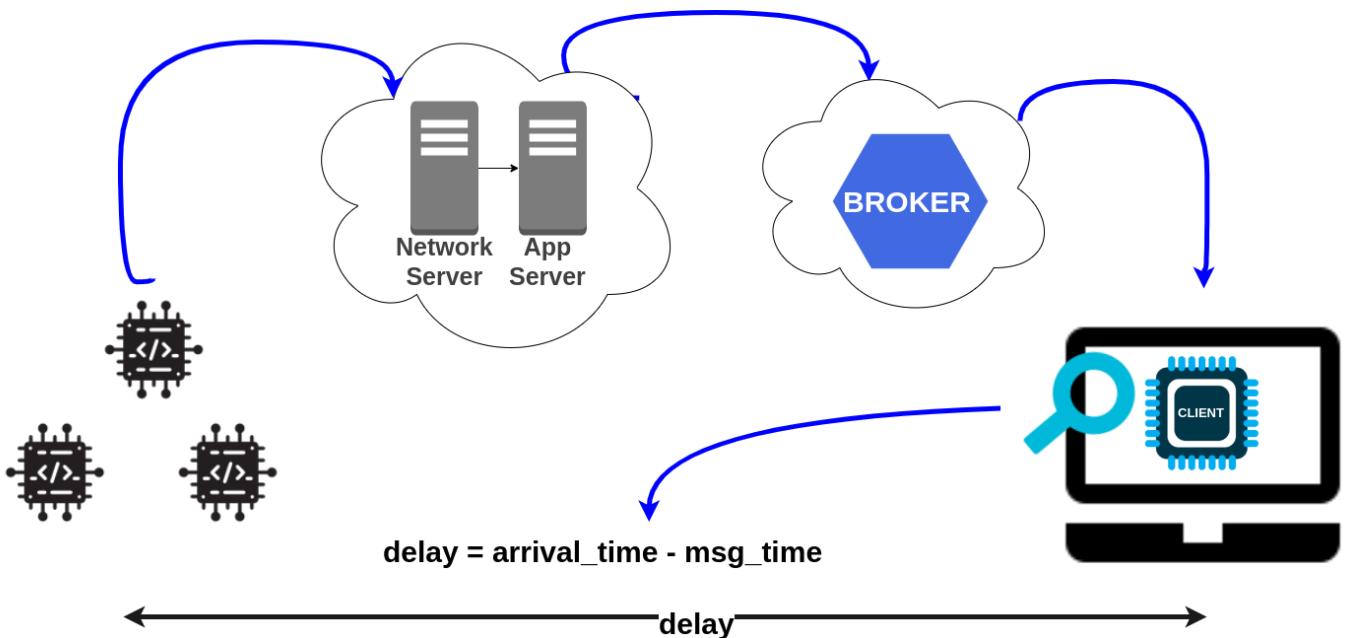
Los íconos de lupa indican en qué punto de la red se capturan los paquetes de análisis.

El comando para iniciar el cliente MQTT es como sigue:

```
mosquitto_sub -h eu.thethings.network -t '+/devices/+events/activations' -u 's
```

### Datos reales

A continuación se muestra el diagrama de red de análisis de MQTT para la red LoRa real:



El comando para iniciar el cliente MQTT es como sigue:

```
mosquitto_sub -h eu.thethings.network -t "+/devices/+" -u "fonexa" -P "ttn-acco
```

*Enviar un mensaje desde el gateway simulado.*

Se puede observar en la consola de mosquitto :

```
simulacion-lora/devices/sim-1/events/activations
{"app_eui":"70B3D57ED002380C",
"dev_eui":"0037D5786316E664",
"dev_addr":"2601210B",
"metadata":{"time":"2019-10-09T23:30:59.946537221Z",
"frequency":868.3,
"modulation":"LORA",
"data_rate":"SF8BW125",
"airtime":53760000,"coding_rate":"4/6",
"gateways":[{"gtw_id":"eui-0242ee000084ee70",
"timestamp":1322089000,"time":"2019-10-09T23:30:59.939Z","channel":2,
"rss":-35,
"snr":5,
"rf_chain":0}]}}
```

Datos reales

```
{
  "time": "2019-11-28T13:30:58.715386532Z",
  "frequency": 902.3,
  "modulation": "LORA",
  "data_rate": "SF7BW125",
  "coding_rate": "4/5",
  "gateways": [
    {
      "gtw_id": "eui-240ac4fffff308f78",
      "timestamp": 577252293,
      "time": "",
      "channel": 0,
      "rss": -60,
      "snr": 9,
      "latitude": -33.11341,
      "longitude": -64.32847,
      "altitude": 440
    }
  ]
}
```

## Instalar e iniciar wireshark

```
sudo apt install wireshark
sudo wireshark
```

## Escribir MQTT en filtros

No.	Time	Source	Destination	Protocol	Length	Info
35	1.800446844	192.168.0.20	52.169.76.255	MQTT	180	Connect Command
85	2.067832587	52.169.76.255	192.168.0.20	MQTT	68	Connect Ack
87	2.068092766	192.168.0.20	52.169.76.255	MQTT	103	Subscribe Request (id=1) [+/devices/+events/activations]
90	2.311285991	52.169.76.255	192.168.0.20	MQTT	69	Subscribe Ack (id=1)
562	12.959310846	52.169.76.255	192.168.0.20	MQTT	498	Publish Message [simulacion-lora/devices/sim-1/events/activations]

## Captura MQTT

## Sobre los fundamentos de MQTT

### Jerarquía de protocolos

Statistics -> Protocol hierarchy

Wireshark - Protocol Hierarchy Statistics - wlp6s0								
Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	5	100.0	918	658	0	0	0
Ethernet	100.0	5	7.6	70	50	0	0	0
Internet Protocol Version 4	100.0	5	10.9	100	71	0	0	0
Transmission Control Protocol	100.0	5	81.5	748	536	0	0	0
MQ Telemetry Transport Protocol	100.0	5	64.8	595	426	5	595	426

## Protocol Hierarchy Statistics, Wireshark

## Conversación entre Broker y cliente

Statistics -> Conversation

```

---  

-  

- Address A  

- Port A  

- Address B  

- Port B  

- Packets  

- Bytes  

- Packets A → B  

- Bytes A → B  

- Packets B → A  

- Bytes B → A  

- Rel Start  

- Duration  

- Bits/s A → B  

- Bits/s B → A  

-  

- 192.168.0.20  

- 55794  

- 52.169.76.255  

- 1883  

- 5  

- 918  

- 2  

- 283  

- 3  

- 635  

- 1.800446844  

- 11.158864002  

- 202.8880358784034  

- 455.2434727306931

```

## Statistics -> Flow Diagram



## Flow Diagram, Wireshark

## Latencia

### Filtrar paquetes PUBLISH MQTT

En el host cliente MQTT, se filtran los paquetes de publish MQTT. El **filtro en wireshark** es `mqtt.msgtype == 3`.

Se exporta la captura con el filtro aplicado en un `json`, (*File -> Export packet dissections -> as JSON*).

El delay, dado por **el tiempo entre el momento en el que se genera el mensaje LoRa en el nodo de la red, hasta que arriva al host del cliente**, se determina calculando el delta de tiempo entre el tiempo de arrivo que se extrae desde la captura de paquetes, y el tiempo dentro del payload LoRa, (*habiendo hecho la correspondiente conversión por diferentes timezones*).

El script de análisis es como sigue:

```

import datetime as dt
import requests
import json
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from bs4 import BeautifulSoup

def parse_date_time(date_time_str, option):
    """
    Returns a datetime object,
    if option is 0, expects a string like: '2019-12-01T14:19:17.285087186Z'
    If option is 1, expects a string like 'Dec 1, 2019 11:19:17.639483558 -03'
    but uses just 17.639483 information of seconds
    """
    if option:
        split_s = date_time_str.split()
        #split_s looks like ['Dec', '1', '2019', '11:19:17.639483558', '-03']
        split_s[-1] += '00'
        if len(split_s[1]) == 2:
            # if day is equal to one digit and a comma
            split_s[1] = '0' + split_s[1]
        # next line solves problem with too much digits for microseconds
        split_s[-2] = split_s[-2][:-3]
        date_time_s = ' '.join(split_s)
        return dt.datetime.strptime(date_time_s, '%b %d, %Y %H:%M:%S.%f %z')
    else:
        return dt.datetime.strptime(date_time_str[:-4] + date_time_str[-1], '%Y

#-----#
#----- MQTT -----#
#-----#


def mqtt_delay_from_json_capture(json_capture_path):
    """
    Returns 4 lists: 'gen_times', 'publish_times',
    'deltas' and 'deltas_milisec' according to publish mqtt packets
    """
    with open(json_capture_path) as f:
        json_capture = json.load(f)

    gen_times = []
    publish_times = []
    deltas = []

    for packet in json_capture:
        try:
            gen_time_str = json.loads(packet['_source']['layers']['mqtt']['mqtt'])
            publish_time_str = packet['_source']['layers']['frame']['frame.time']
            gen_time = parse_date_time(gen_time_str, 0).astimezone()
            publish_time = parse_date_time(publish_time_str, 1).astimezone()
            delta = publish_time - gen_time
        except KeyError:

```

```

    pass
gen_times.append(gen_time)
publish_times.append(publish_time)
deltas.append(delta)

deltas_milisec = [delta.total_seconds()*1000 for delta in deltas]
return gen_times, publish_times, deltas, deltas_milisec

```

## Sobrecarga (overhead)

Para analizar el **overhead** se aplican como columnas Frame length (*primer capa, física*) y Msg length , (MQTT)

## Simulación

No.	Time	Source	Destination	Protocol	Frame length	Msg Len
11515	450.705904874	52.169.76.255	192.168.0.20	MQTT	497	431
11196	447.225952867	52.169.76.255	192.168.0.20	MQTT	497	431
10539	428.247426286	52.169.76.255	192.168.0.20	MQTT	497	431
10513	427.581292395	52.169.76.255	192.168.0.20	MQTT	497	431
10089	419.439069665	52.169.76.255	192.168.0.20	MQTT	497	431
9893	417.730883412	52.169.76.255	192.168.0.20	MQTT	497	431
9171	401.349911517	52.169.76.255	192.168.0.20	MQTT	497	431
8997	399.269968262	52.169.76.255	192.168.0.20	MQTT	497	431
1227	33.420492555	52.169.76.255	192.168.0.20	MQTT	495	429
1004	31.292874104	52.169.76.255	192.168.0.20	MQTT	475	409
151	6.698840007	192.168.0.20	52.169.76.255	MQTT	180	112
163	7.009540852	192.168.0.20	52.169.76.255	MQTT	103	35
170	7.247151946	52.169.76.255	192.168.0.20	MQTT	69	3
161	7.009216652	52.169.76.255	192.168.0.20	MQTT	68	2
10518	427.711906399	52.169.76.255	192.168.0.20	MQTT	67	0
10506	427.446604585	192.168.0.20	52.169.76.255	MQTT	68	0
7905	367.215587252	52.169.76.255	192.168.0.20	MQTT	67	0
7898	366.973615203	192.168.0.20	52.169.76.255	MQTT	68	0
6719	307.916524998	52.169.76.255	192.168.0.20	MQTT	67	0
6716	307.678489980	192.168.0.20	52.169.76.255	MQTT	68	0
5466	247.625788996	52.169.76.255	192.168.0.20	MQTT	67	0
5463	247.345592593	192.168.0.20	52.169.76.255	MQTT	68	0
4301	187.286204121	52.169.76.255	192.168.0.20	MQTT	67	0

Se puede observar una sobre carga (*dada por la diferencia entre la longitud del mensaje y la longirud de frame*) que varía entre 66 y 69 bytes.

## Datos reales

5 192.168.102... MQTT	0.000085836 ✓	609	543 Publish Message [fonexa/devices/arduino-uno/up]
5 192.168.102... MQTT	0.000952303 ✓	608	542 Publish Message [fonexa/devices/arduino-uno/up]
5 192.168.102... MQTT	0.000373635 ✓	593	527 Publish Message [fonexa/devices/arduino-uno/up]
5 192.168.102... MQTT	0.000043549 ✓	592	526 Publish Message [fonexa/devices/arduino-uno/up]
5 192.168.102... MQTT	-0.000288948 ✓	584	518 Publish Message [fonexa/devices/esp32-oled-nodo/up]
5 192.168.102... MQTT	0.001339284 ✓	584	518 Publish Message [fonexa/devices/esp32-oled-nodo/up]
5 192.168.102... MQTT	0.000842789 ✓	583	517 Publish Message [fonexa/devices/esp32-oled-nodo/up]
5 192.168.102... MQTT	0.001171837 ✓	568	502 Publish Message [fonexa/devices/esp32-oled-nodo/up]
52.169.76.255 MQTT	0.000595147 ✓	148	80 Connect Command
52.169.76.255 MQTT	0.0006514932 ✓	87	19 Subscribe Request (id=2) [+/devices/+/up]
192.168.102... MQTT	0.000032184 ✓	69	3 Subscribe Ack (id=2)
192.168.102... MQTT	0.000178767 ✓	68	2 Connect Ack
5 192.168.102... MQTT	0.000142064 ✓	67	0 Ping Response
5 192.168.102... MQTT	5.006298545 ✓	68	0 Ping Request
5 192.168.102... MQTT	0.000136345 ✓	67	0 Ping Response
52.169.76.255 MQTT	5.007211391 ✓	68	0 Ping Request

Se puede observar una sobre carga (*dada por la diferencia entre la longitud del mensaje y la longirud de frame*) de 66 bytes.

## Seguridad

Para un paquete MQTT con un mensaje lora, es posible mirar el contenido del mensaje:

```
Frame 562: 498 bytes on wire (3984 bits), 498 bytes captured (3984 bits) on interface 0
Ethernet II, Src: CiscoSpv_8d:91:ec (34:bd:fa:8d:91:ec), Dst: HonHaiPr_d4:07:95 (9c:2a:70:d4:07:95)
Internet Protocol Version 4, Src: 52.169.76.255, Dst: 192.168.0.20
Transmission Control Protocol, Src Port: 1883, Dst Port: 55794, Seq: 13, Ack: 152, Len: 432
[3 Reassembled TCP Segments (435 bytes): #560(1), #561(2), #562(432)]
MQ Telemetry Transport Protocol, Publish Message
  Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
  Msg Len: 432
  Topic Length: 48
  Topic: simulacion-lora/devices/sim-1/events/activations
Message [truncated]: {"app_eui":"70B3D57ED002380C", "dev_eui":"0037D5786316E664", "dev_addr": "26012979", "metadata":
```

```
{"app_eui":"70B3D57ED002380C",
"dev_eui":"0037D5786316E664",
"dev_addr": "26012979",
"metadata": {"time": "2019-10-10T01:22:38.970294643Z",
"frequency": 868.5,
"modulation": "LORA",
"data_rate": "SF7BW125",
"airtime": 26880000,
"coding_rate": "4/6",
"gateways": [{"gtw_id": "eui-0242ee000084ee70", "timestamp": 1623747000,
"time": "2019-10-10T01:22:38.962Z",
"channel": 2, "rss": -35,
"snr": 5, "rf_chain": 0}]}}
```

Dado que MQTT, por defecto, no implementa cifrado.

## Análisis HTTP

En la aplicación creada en *thethingsnetwork*, click en `Integrations -> Add integrations`, seleccionar `HTTP Integration`:

## ADD INTEGRATION



## HTTP Integration (v2.6.0)

The Things Industries B.V.

Sends uplink data to an endpoint and receives downlink data over HTTP.

[documentation](#)

## Process ID

The unique identifier of the new integration process

simulacion-http



## Access Key

The access key used for downlink

default key devices messages



## URL

The URL of the endpoint

sim-http



## Method

The HTTP method to use

POST



## Authorization

The value of the Authorization header



## Custom Header Name

An optional custom HTTP header that you would like to add to the request

sim



## Custom Header Value

The value of the custom Header



Cancel

Add integration

## HTTP integration

Crear URL de API, ingresar en requestbin (<https://requestbin.com>), Create a Request bin .

Editar la URL del endpoint en configuración de aplicación.

## HTTP

## Escribir HTTP en filtros

No.	Time	Source	Destination	Protocol	Frame length	Content-Length	Time since previous frame	Info
2783	77.316889601	192.168.0.20	52.211.146.2...	HTTP	567		0.020729513	GET /socket.io/?EIO=3&transport=polling&t=Msppl9Y&sid=gVuX-8s1YnaP2C-6A2...
2810	77.502094821	52.211.146.2...	192.168.0.20	HTTP	557		0.000511021	HTTP/1.1 200 OK (application/javascript)
2812	77.507994789	192.168.0.20	52.211.146.2...	HTTP	773	136	0.005540924	POST /api/lora/send HTTP/1.1 (application/json)
2813	77.508992319	192.168.0.20	52.211.146.2...	HTTP	675	44	0.243397943	POST /timesync HTTP/1.1 (application/json)
2815	77.543431027	192.168.0.20	52.211.146.2...	HTTP	525		0.799453175	GET /out/user_1570679741659.wasm HTTP/1.1
2818	77.570995307	192.168.0.20	52.211.146.2...	HTTP	723		0.000355701	GET /socket.io/?EIO=3&transport=websocket&sid=gVuX-8s1YnaP2C-6AZu1
2820	77.754377130	52.211.146.2...	192.168.0.20	HTTP	332	31	0.245474811	HTTP/1.1 200 OK (application/json)
2822	77.757750449	52.211.146.2...	192.168.0.20	HTTP	294	2	0.249755651	HTTP/1.1 200 OK (text/html)
2845	77.824066015	52.211.146.2...	192.168.0.20	HTTP	241		0.000511556	HTTP/1.1 101 Switching Protocols
2873	78.182750482	52.211.146.2...	192.168.0.20	HTTP	296	3	0.576184961	HTTP/1.1 200 OK (text/plain)
2938	78.762879614	192.168.0.20	52.211.146.2...	HTTP	675	44	0.539230568	POST /timesync HTTP/1.1 (application/json)
2966	79.033993578	52.211.146.2...	192.168.0.20	HTTP	332	31	0.000541098	HTTP/1.1 200 OK (application/json)
3051	79.665142780	52.211.146.2...	192.168.0.20	HTTP	360		0.035676955	HTTP/1.1 200 OK (application/wasm)
3054	79.871131638	192.168.0.20	52.211.146.2...	HTTP	774	137	0.205269541	POST /api/lora/send HTTP/1.1 (application/json)
3057	80.045695292	192.168.0.20	52.211.146.2...	HTTP	675	44	1.011638424	POST /timesync HTTP/1.1 (application/json)
3058	80.123499476	52.211.146.2...	192.168.0.20	HTTP	294	2	0.252367834	HTTP/1.1 200 OK (text/html)
3062	80.307233710	52.211.146.2...	192.168.0.20	HTTP	332	31	0.261538418	HTTP/1.1 200 OK (application/json)
3077	80.726275973	192.168.0.20	52.211.146.2...	HTTP	675	44	0.418931830	POST /timesync HTTP/1.1 (application/json)
3080	80.984811066	52.211.146.2...	192.168.0.20	HTTP	332	31	0.258530593	HTTP/1.1 200 OK (application/json)
3105	81.861440407	192.168.0.20	52.211.146.2...	HTTP	675	44	0.876568773	POST /timesync HTTP/1.1 (application/json)
3106	81.99462181	192.168.0.20	52.211.146.2...	HTTP	675	44	1.870957510	POST /timesync HTTP/1.1 (application/json)
3107	82.122408782	52.211.146.2...	192.168.0.20	HTTP	322	34	0.270068276	HTTP/1.1 200 OK (application/json)

## Captura HTTP

Con vista en un *POST lora* (*columna Info*), en la pestaña HTTP, y método POST seleccionar el post lora como filtro:

```

▶ Frame 3054: 774 bytes on wire (6192 bits), 774 bytes captured (6192 bits) on interface 0
▶ Ethernet II, Src: HonHaiPr_d4:07:95 (9c:2a:70:d4:07:95), Dst: CiscoSpv_8d:91:ec (34:bd:fa:8d:91:ec)
▶ Internet Protocol Version 4, Src: 192.168.0.20, Dst: 52.211.146.247
▶ Transmission Control Protocol, Src Port: 42374, Dst Port: 7829, Seq: 2026, Ack: 131421, Len: 708
▼ Hypertext Transfer Protocol
  ▼ POST /api/lora/send HTTP/1.1\r\n
    ▶ [Expert Info (Chat/Sequence): POST /api/lora/send HTTP/1.1\r\n]
      Request Method: POST
      Request URI: /api/lora/send
      Request Version: HTTP/1.1
      Host: ec2-52-211-146-247.eu-west-1.compute.amazonaws.com:7829\r\n
      Connection: keep-alive\r\n
    ▶ Content-Length: 137\r\n
    ▶ Origin: http://ec2-52-211-146-247.eu-west-1.compute.amazonaws.com:7829\r\n
0040  29 24 50 4f 53 54 20 2f 61 70 69 2f 6c 6f 72 61  )$POST / api/lora
0050  2f 73 65 6e 64 20 48 54 54 50 2f 31 2e 31 0d 0a  /send HT TP/1.1..
0060  48 6f 73 74 3a 20 65 63 32 2d 35 32 2d 32 31 31  Host: ec 2-52-211

```

*Captura POST lora api*

## Análisis HTTP

### Jerarquía de protocolos

Statistics -> Protocol hierarchy

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	9	100.0	7000	1,259	0	0	0
Ethernet	100.0	9	1.8	126	22	0	0	0
Internet Protocol Version 4	100.0	9	2.6	180	32	0	0	0
Transmission Control Protocol	100.0	9	95.6	6694	1,204	1	741	133
Hypertext Transfer Protocol	88.9	8	81.4	5697	1,024	0	0	0
JavaScript Object Notation	88.9	8	16.1	1129	203	8	1129	203

*Protocol Hierarchy Statistics, Wireshark*

### Handshake

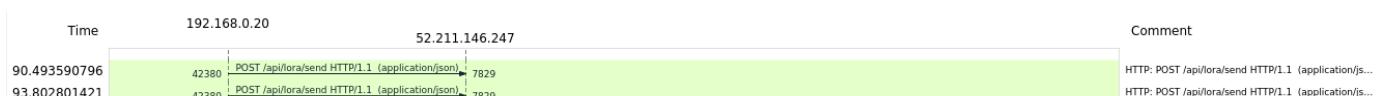
Statistics -> Conversation

```

-- 
-
- Address A
- Address B
- Packets
- Bytes
- Packets A → B
- Bytes A → B
- Packets B → A
- Bytes B → A
- Rel Start
- Duration
- Bits/s A → B
- Bits/s B → A
-
- 52.211.146.247
- 192.168.0.20
- 9
- 7000
- 0
- 0
- 9
- 7000
- 49.331159418
- 44.471642003
- 0
- 1259.2294207671107

```

## Statistics -> Flow Diagram



## Flow Diagram, Wireshark

### Latencia

El delay usando HTTP como protocolo de integración se realiza mediante un script que hace *scraping* sobre la página en HTML de la API hosteada en requestbin.net (<http://requestbin.net>).

```

import datetime as dt
import requests
import json
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from bs4 import BeautifulSoup

def parse_date_time(date_time_str, option):
    """
    Returns a datetime object,
    if option is 0, expects a string like: '2019-12-01T14:19:17.285087186Z'
    If option is 1, expects a string like 'Dec 1, 2019 11:19:17.639483558 -03'
    but uses just 17.639483 information of seconds
    """
    if option:
        split_s = date_time_str.split()
        #split_s looks like ['Dec', '1', '2019', '11:19:17.639483558', '-03']
        split_s[-1] += '00'
        if len(split_s[1]) == 2:
            # if day is equal to one digit and a comma
            split_s[1] = '0' + split_s[1]
        # next line solves problem with too much digits for microseconds
        split_s[-2] = split_s[-2][:-3]
        date_time_s = ' '.join(split_s)
        return dt.datetime.strptime(date_time_s, '%b %d, %Y %H:%M:%S.%f %z')
    else:
        return dt.datetime.strptime(date_time_str[:-4] + date_time_str[-1], '%Y

#-----#
#----- HTTP -----#
#-----#


def http_delay_from_api_url(url):
    #url = 'http://requestbin.net/r/10ojyvy1?inspect'
    #url = 'http://requestbin.net/r/11a0l9y1?inspect'
    response = requests.get(url)
    html_doc = response.text

    gen_times = []
    publish_times = []
    deltas = []

    soup = BeautifulSoup(html_doc, 'html.parser')

    timestamp_divs = soup.findAll("div", {"class": "timestamp"})
    for div in timestamp_divs:
        publish_time_str = div.find("span")['title']
        publish_time_s = 'T'.join(publish_time_str.split()) + 'Z'
        publish_time = parse_date_time(publish_time_s, 0)
        publish_times.append(publish_time)

    requestbody_divs = soup.findAll("div", {"class": "request-body"})
    for div in requestbody_divs:

```

```

payload_str = div.find("pre").find(text=True)
payload_json = json.loads(payload_str)
gen_time_str = payload_json['metadata']['time']
gen_time = parse_date_time(gen_time_str, 0)
gen_times.append(gen_time)

for i, gen_time in enumerate(gen_times):
    delta = publish_times[i] - gen_time
    deltas.append(delta)

deltas_milisec = [delta.total_seconds()*1000 for delta in deltas]
return gen_times, publish_times, deltas, deltas_milisec

```



## Sobrecarga (overhead)

### Datos

Para analizar el **overhead** se aplican como columnas **Frame length** (*primer capa, física*) y **Content length**, (*HTTP*).

No.	Time	Source	Destination	Protocol	Frame length	Content-Length	Time since previous frame ir Info
2382	74.376771835	52.211.146.2...	192.168.0.20	HTTP	1006	0.249158731	HTTP/1.1 200 OK (text/html)
926	37.464388338	52.211.146.2...	192.168.0.20	HTTP	1006	0.253126723	HTTP/1.1 200 OK (text/html)
818	34.62859236	52.211.146.2...	192.168.0.20	HTTP	1006	0.254090187	HTTP/1.1 200 OK (text/html)
3254	98.493596796	192.168.0.20	52.211.146.2...	HTTP	796 159	0.091632673	POST /api/lora/send HTTP/1.1 (application/json)
3321	93.862601421	192.168.0.20	52.211.146.2...	HTTP	790 153	0.998801609	POST /api/lora/send HTTP/1.1 (application/json)
2076	62.863447166	192.168.0.20	52.211.146.2...	HTTP	775 138	0.249350671	POST /api/lora/send HTTP/1.1 (application/json)
3854	79.871131638	192.168.0.20	52.211.146.2...	HTTP	774 137	0.205269541	POST /api/lora/send HTTP/1.1 (application/json)
1786	49.331159418	192.168.0.20	52.211.146.2...	HTTP	774 137	0.000386373	POST /api/lora/send HTTP/1.1 (application/json)
2812	77.507994789	192.168.0.20	52.211.146.2...	HTTP	773 136	0.005940924	POST /api/lora/send HTTP/1.1 (application/json)
2044	61.870984501	192.168.0.20	52.211.146.2...	HTTP	773 136	0.760461792	POST /api/lora/send HTTP/1.1 (application/json)
1822	50.132014773	192.168.0.20	52.211.146.2...	HTTP	770 133	0.002158859	POST /api/lora/send HTTP/1.1 (application/json)
1005	38.693537288	192.168.0.20	52.211.146.2...	HTTP	753	0.002353298	GET /img/controller_mbed.svg HTTP/1.1
2818	77.570905307	192.168.0.20	52.211.146.2...	HTTP	723	0.000355701	GET /socket.io/?EIO=3&transport=websocket&id=gVuX-8s1YnaP2C-6AzU1 HTTP/
2503	75.459435898	192.168.0.20	52.211.146.2...	HTTP	723	0.000674858	GET /socket.io/?EIO=3&transport=websocket&id=v-DzhIg5Gb8z2m-dAzU0 HTTP/
1070	39.315331866	192.168.0.20	52.211.146.2...	HTTP	723	0.006562756	GET /socket.io/?EIO=3&transport=websocket&id=P_3czUhVv3ffXVVf0AZuz HTTP/
1002	38.668429362	192.168.0.20	52.211.146.2...	HTTP	723	0.000448132	GET /socket.io/?EIO=3&transport=websocket&id=bl2_uYd76Pjt-JdFAZuy HTTP/
868	35.858389883	192.168.0.20	52.211.146.2...	HTTP	678 47	0.132617385	POST /timesync HTTP/1.1 (application/json)
789	33.888703284	192.168.0.20	52.211.146.2...	HTTP	678 47	0.024974314	POST /timesync HTTP/1.1 (application/json)
786	33.771813546	192.168.0.20	52.211.146.2...	HTTP	678 47	0.002719246	POST /timesync HTTP/1.1 (application/json)
753	32.323760328	192.168.0.20	52.211.146.2...	HTTP	678 47	0.218731146	POST /timesync HTTP/1.1 (application/json)
729	31.858459541	192.168.0.20	52.211.146.2...	HTTP	678 47	0.719841775	POST /timesync HTTP/1.1 (application/json)
723	30.858162398	192.168.0.20	52.211.146.2...	HTTP	678 47	0.384983237	POST /timesync HTTP/1.1 (application/json)

Se puede observar una sobre carga (*dada por la diferencia entre la longitud del mensaje y la longirud de frame*) promedio es de 635 bytes.

Dado que se usa un servicio en la nube (*gratuito*), el método `POST` no se genera en la LAN, por lo que no es posible sniffear con wireshark.

Se descarga el contenido HTML de la URL de *requests bin*, se “escrapean” los datos de fecha y hora con `Python` como sigue:

```

def http_overhead(json_capture_path, url):
    #url = 'http://requestbin.net/r/10ojyvy1?inspect'
    #url = 'http://requestbin.net/r/11a0l9y1?inspect'
    response = requests.get(url)
    html_doc = response.text
    soup = BeautifulSoup(html_doc, 'html.parser')

    frame_lengths = []
    msg_lengths = []

    # Next code gets msg lenghts from json capture
    with open(json_capture_path) as f:
        json_capture = json.load(f)

    for packet in json_capture:
        try:
            msg_length_str = json.loads(packet['_source']['layers']['mqtt']['mq'])
        except KeyError:
            pass
        msg_length = int(msg_length_str)
        msg_lengths.append(msg_length)

    # Next code gets frame lenghts from url
    span6_divs = soup.findAll("div", {"class": "span6 content"})
    for div in span6_divs:
        texto = div.text.strip()
        if '\n' in texto:
            bytes_str = texto.split('\n')[1].split(' ')[0]
        else:
            bytes_str = texto.split(' ')[0]
        bytes = int(bytes_str)
        frame_lengths.append(bytes)

    mean_msg_length = sum(msg_lengths) / len(msg_lengths)
    mean_frame_length = sum(frame_lengths) / len(frame_lengths)
    return mean_frame_length - mean_msg_length

```

## Seguridad

Para el caso de un POST de mensaje lora por HTTP (API), el contenido del mensaje **no es visible**, dado que está cifrado.

```

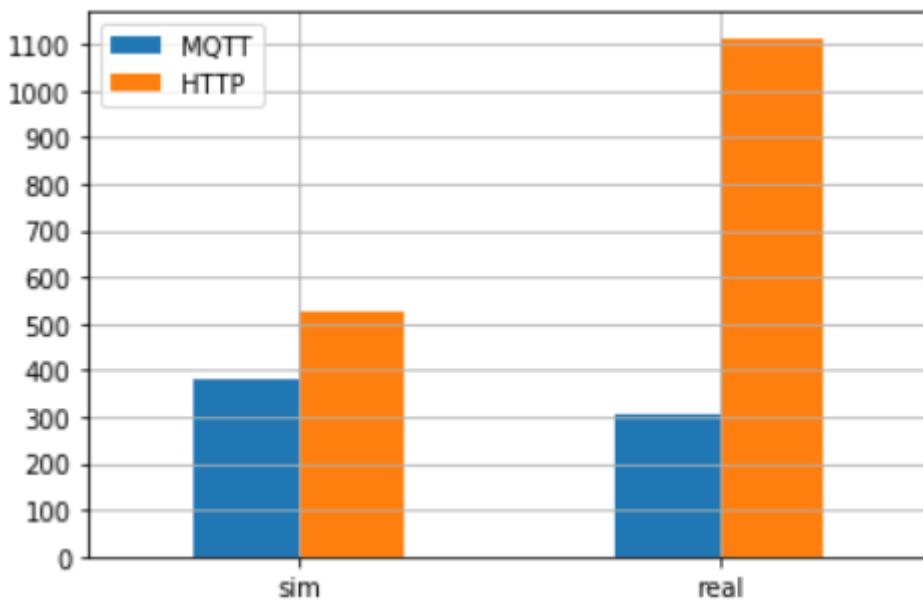
▶ Frame 3254: 796 bytes on wire (6368 bits), 796 bytes captured (6368 bits) on interface 0
▶ Ethernet II, Src: HonHaiPr_d4:07:95 (9c:2a:70:d4:07:95), Dst: CiscoSpv_8d:91:ec (34:bd:fa:8d:91:ec)
▶ Internet Protocol Version 4, Src: 192.168.0.20, Dst: 52.211.146.247
▶ Transmission Control Protocol, Src Port: 42380, Dst Port: 7829, Seq: 610, Ack: 267, Len: 730
└ Hypertext Transfer Protocol
  ▶ POST /api/lora/send HTTP/1.1\r\n
    Host: ec2-52-211-146-247.eu-west-1.compute.amazonaws.com:7829\r\n
    Connection: keep-alive\r\n
    Content-Length: 159\r\n
    Origin: http://ec2-52-211-146-247.eu-west-1.compute.amazonaws.com:7829\r\n
    User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36\r\n
    Content-Type: application/json\r\n
    Accept: */*\r\n
    Referer: http://ec2-52-211-146-247.eu-west-1.compute.amazonaws.com:7829/view/user_1570679741650\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.9\r\n
    Cookie: io=gVuX-8s1YnaP2C-6AZu1\r\n
  \r\n
  [Full request URI: http://ec2-52-211-146-247.eu-west-1.compute.amazonaws.com:7829/api/lora/send]
  [HTTP request 2/36]
  [Prev request in frame: 3244]
  [Response in frame: 3255]
  [Next request in frame: 3262]
  File Data: 159 bytes
└ JavaScript Object Notation: application/json
  └ Object
    ▶ Member Key: payload
    ▶ Member Key: freq
    ▶ Member Key: bandwidth
    ▶ Member Key: datarate

```

## Resultados

Parámetro	MQTT (sim)	MQTT (real)	HTTP (sim)	HTTP (real)
latencia	379.298500 [ms]	305.594625 [ms]	524.574300 [ms]	1113.238350[ms] *
overhead	69 bytes	69 bytes	110.5 bytes	168.924 bytes
cifrado	no	no	si	si

\* Dato no significativo para el análisis.



**(tiempo en milisegundos).** Gráfico comparativo *delay* en protocolos MQTT y HTTP para una red LoRaWAN simulada y una real.

## Análisis de los resultados

Ambos protocolos de aplicación, pero en particular de *integración* en el contexto de una red LoRaWAN, cumplen con el objetivo de servir los datos de nodos LoRa.

Para el relevamiento de los dos parámetros de interés, se observa una diferencia muy significativa que favorece a MQTT, sin embargo para este último protocolo las credenciales de autorización son transparentes.

Se puede implementar una capa de seguridad con MQTTS.

Desde el enfoque comparativo de los resultados de la red LoRaWAN *simulada* versus la red “real”, se observa una cohesión del comportamiento.

Cabe aclarar que el resultado obtenido para delay empleando HTTP en la red real, no resulta comparable y su desproporción se debe al uso de un servicio gratuito y en la nube.

## Planilla de horas y diagrama de Gantt

Planilla de horas					Horas Totales	Total hs cumplida
Fecha	Hora Inicio	Hora Fin	Horas cumplidas	Tareas		
29/08/2019	8:30	13:00	4:30	Análisis de diagrama general del proyecto. Esquema general informe. Redacción de introducción y objetivos. Se comienza investigación sobre LoraWAN.	310:00:00	310:30:00
30/08/2019	14:00	17:30	3:30			
31/08/2019	14:00	19:00	5:00	Investigación sobre LoraWAN. Investigación sobre módulos adquisición de datos.		
02/09/2019	14:00	17:30	3:30			
03/09/2019	8:30	13:00	4:30	Investigación sobre especificaciones técnicas LoraWAN Investigación sobre end-points		
05/09/2019	8:30	13:00	4:30	Investigación sobre especificaciones técnicas LoraWAN Investigación sobre end-points, compatibilidad con hardware disponible Investigación sobre gatways		
06/09/2019	14:00	17:30	3:30			
07/09/2019	14:00	19:00	5:00	Investigación sobre gatways		
06/09/2019	14:00	17:30	3:30			
09/09/2019	14:00	16:00	2:00			
10/09/2019	8:30	13:00	4:30	Investigación sobre LoRa Server.		
12/09/2019	8:30	13:00	4:30			
13/09/2019	8:30	13:00	4:30			
14/09/2019	14:00	19:00	5:00	Investigación sobre LoRa Server. Sobre frames y payloads. Detalles borrador informe.		
16/09/2019	14:00	17:30	3:30			
17/09/2019	8:30	13:00	4:30			
18/09/2019	8:30	13:00	4:30	Diagrama solución, toma de decisiones, documentación y diagramas.		
19/09/2019	8:30	13:00	4:30	Investigación sobre propagación LoRa, detalles informe (bibliografía, diagramas)		
20/09/2019	8:30	13:00	4:30			
21/09/2019	14:00	19:00	5:00	Toma de decisiones, documentación		
22/09/2019	8:30	13:00	4:30			
23/09/2019	8:30	13:00	4:30	Presupuesto gateways, server		
27/09/2019	14:00	17:30	3:30			
28/09/2019	14:00	18:00	4:00	Toma de decisiones, documentación		
30/09/2019	8:30	13:00	4:30	Investigación MQTT, organización hardware.		
01/10/2019	8:30	13:00	4:30	Investigación MQTT y plataforma		
03/10/2019	8:30	13:00	4:30	Investigación MQTT, websocket y plataforma		
04/10/2019	14:00	17:30	3:30	Investigación websocket, HTTP API, plataforma y simulación de paquetes LoRa		
05/10/2019	16:00	21:00	5:00			
06/10/2019	15:00	20:00	5:00	Instalación servidores de red y aplicación, simulación paquetes LoRa		
07/10/2019	14:00	17:30	3:30			
08/10/2019	8:30	13:00	4:30	Testeo conectividad LoRa		
11/10/2019	14:00	17:30	3:30			
12/10/2019	14:00	18:00	4:00			
14/10/2019	8:30	13:00	4:30			
15/10/2019	8:30	13:00	4:30	Implementacion gw ESP32		
17/10/2019	8:30	13:00	4:30			
18/10/2019	14:00	17:30	3:30			
19/10/2019	14:00	19:00	5:00			
21/10/2019	8:30	13:00	4:30	Documentación testeo LoRa, gateway con ESP32		
22/10/2019	8:30	13:00	4:30	Implementación nodo ESP32		
24/10/2019	8:30	13:00	4:30	Implementación gateway + nodo		
25/10/2019	14:00	17:30	3:30			
26/10/2019	14:00	19:00	5:00	Documentación gateway + nodo		
29/10/2019	8:30	13:00	4:30			
31/10/2019	8:30	13:00	4:30	Entorno de desarrollo		
01/11/2019	14:00	17:30	3:30			
02/11/2019	14:00	19:00	5:00	Implementación arduino + esp32		
04/11/2019	8:30	13:00	4:30	Entorno de desarrollo		
05/11/2019	8:30	13:00	4:30	Entorno de desarrollo		

07/11/2019	8:30	13:00	4:30	Entorno de desarrollo e Implementación arduino + esp32		
08/11/2019	14:00	17:30	3:30			
09/11/2019	9:00	14:00	5:00	Entorno de desarrollo, mantenimiento pc		
12/11/2019	8:30	13:00	4:30	aplicación ttn + mqtt		
13/11/2019	8:30	13:00	4:30			
14/11/2019	13:00	17:30	4:30			
15/11/2019	8:30	13:00	4:30			
18/11/2019	13:00	17:30	4:30			
19/11/2019	8:30	13:00	4:30	mqtt api		
21/11/2019	8:30	13:00	4:30			
22/11/2019	8:30	13:00	4:30			
23/11/2019	21:30	2:30	5:00	mqtt api + gráfico		
24/11/2019	9:00	13:00	4:00			
25/11/2019	8:30	13:00	4:30	dashboard plot		
26/11/2019	8:30	13:00	4:30			
27/11/2019	8:30	13:00	4:30			
28/11/2019	8:30	13:00	4:30	integracion mqtt api y dashboard plot		
29/11/2019	8:30	13:00	4:30			
30/11/2019	8:00	14:00	6:00	Analisis delay y overhead MQTT vs HTPP		
01/12/2019	14:00	19:00	5:00			
02/12/2019	8:00	14:00	6:00			
03/12/2019	8:00	13:00	5:00	Repository		
04/12/2019	8:00	13:00	5:00			
05/12/2019	8:00	13:00	5:00			
05/12/2019	15:00	18:00	3:00			
06/12/2019	8:00	13:00	5:00	Informe y documentación a presentar		

Nº TAREA	TAREA	INICIO	FIN	DURACIÓN	% COMPLETADO	SEMANA 1			
						LU	MA	JUE	VIE
1	Investigación								
1,1	Investigar sobre LoraWAN	29/08/2019	13/09/2019	15	100%			1	1
1,2	Sobre los elementos de LoraWAN				100%		1	1	1
1,3	End points				100%			1	1
1,4	Gateways				80%				
1,5	Plataforma				100%				
1,6	Formas de integración				100%				
1,7	Casos de uso				100%				
2	Diseño de solución								
2,1	Diagramar solución y sus partes	29/08/2019	13/09/2019	15	100%		1	1	1
2,2	Componentes a utilizar				100%				
2,3	Especificaciones técnicas				90%				
2,4	Seleccionar dispositivos y plataformas				100%				
3	Implementación de endpoints								
3,1	Definir endpoints a usar	23/09/2019	27/09/2019	4	100%				
3,2	Presupuestar los mismos				100%				
3,3	Adquirir los endpoints				80%				
3,4	Implementar endpoints				100%				
4	Implementación de gateways								
4,1	Definir gateways a usar	23/09/2019	4/10/2019	11	100%				
4,2	Presupuestar los mismos				100%				
4,3	Adquirir los gateways				80%				
4,4	Implementar gateways				100%				
4,3	Lograr comunicación half duplex entre endpoints y gateways				100%				
4,4	Lograr comunicación full duplex entre endpoints y gateways				100%				
5	Implementación dashboard								
5,1	Servir los datos adquiridos	7/10/2019	11/10/2019	4	100%				
5,2	Montar servidor red				100%				
5,3	Desarrollar módulo de procesamiento de datos				100%				
5,4	Desarrollar dashboard				100%				
5,5	Integrar al sistema				100%				
6	Pruebas de campo								
6,1	Montar solución en un escenario real	10/11/19	10/25/19	14	100%				
6,2	Puesta en marcha del sistema	10/26/19	10/31/19	5	100%				
6,3	Pruebas	11/1/19	12/31/19	60	100%				
6,4	Proyección	12/1/19	12/6/19	5	100%				
6,5	Conclusiones	12/1/19	12/6/19	5	100%				
7	Documentación	8/29/19	12/6/19	99	100%		1	1	1



Nº TAREA	TAREA	ANA 5		SEMANA 6					SEMANA 7			SEMANA 8			
		VIE	LU	MA	JUE	VIE	SA	DO	LU	MA	JUE	VIE	LU	MA	JUE
1	Investigación														
1,1	Investigar sobre LoraWAN														
1,2	Sobre los elementos de LoraWAN														
1,3	End points														
1,4	Gateways	■													
1,5	Plataforma	■													
1,6	Formas de integración	■													
1,7	Casos de uso														
2	Diseño de solución														
2,1	Diagramar solución y sus partes	■													
2,2	Componentes a utilizar	■	■						■	■	■				
2,3	Especificaciones técnicas	■	■						■	■	■		■		
2,4	Seleccionar dispositivos y plataformas	■	■										■		
3	Implementación de endpoints														
3,1	Definir endpoints a usar	■													
3,2	Presupuestar los mismos	■													
3,3	Adquirir los endpoints		■							■	■				
3,4	Implementar endpoints									■	■		■	■	
4	Implementación de gateways														
4,1	Definir gateways a usar	■													
4,2	Presupuestar los mismos	■													
4,3	Adquirir los gateways		■							■	■				
4,4	Implementar gateways									■	■		■	■	
4,3	Lograr comunicación half duplex entre endpoints y gateways														
4,4	Lograr comunicación full duplex entre endpoints y gateways														
5	Implementación dashboard														
5,1	Servir los datos adquiridos														
5,2	Montar servidor red		■							■	■				
5,3	Desarrollar módulo de procesamiento de datos														
5,4	Desarrollar dashboard														
5,5	Integrar al sistema														
6	Pruebas de campo														
6,1	Montar solución en un escenario real														
6,2	Puesta en marcha del sistema									■	■		■	■	
6,3	Pruebas														
6,4	Proyección														
6,5	Conclusiones														
7	Documentación	■							■	■	■	■	■	■	

Nº	TAREA		SEMANA 9					SEMANA 10					SEMANA 11					LUNES
			VIE	LU	MA	JUE	VIE	LU	MA	JUE	VIE	LU	MA	JUE	VIE	LU		
1	Investigación																	
1,1	Investigar sobre LoraWAN																	
1,2	Sobre los elementos de LoraWAN																	
1,3	End points																	
1,4	Gateways																	
1,5	Plataforma																	
1,6	Formas de integración																	
1,7	Casos de uso																	
2	Diseño de solución																	
2,1	Diagramar solución y sus partes																	
2,2	Componentes a utilizar																	
2,3	Especificaciones técnicas																	
2,4	Seleccionar dispositivos y plataformas																	
3	Implementación de endpoints																	
3,1	Definir endpoints a usar																	
3,2	Presupuestar los mismos																	
3,3	Adquirir los endpoints																	
3,4	Implementar endpoints		■					■■■■■										
4	Implementación de gateways																	
4,1	Definir gateways a usar																	
4,2	Presupuestar los mismos																	
4,3	Adquirir los gateways							■■■■■										
4,4	Implementar gateways		■															
4,5	Lograr comunicación half duplex entre endpoints y gateways																	
4,6	Lograr comunicación full duplex entre endpoints y gateways														■■■■■			
5	Implementación dashboard																	
5,1	Servir los datos adquiridos																	
5,2	Montar servidor red																	
5,3	Desarrollar módulo de procesamiento de datos																	
5,4	Desarrollar dashboard																	
5,5	Integrar al sistema																	
6	Pruebas de campo																	
6,1	Montar solución en un escenario real																	
6,2	Puesta en marcha del sistema																	
6,3	Pruebas																	
6,4	Proyección																	
6,5	Conclusiones																	
7	Documentación		■					■■■■■							■■■■■			



Nº TAREA	TAREA	5	SEMANA 16					SEMANA 17					SEMANA 18				
			VIE	LU	MA	JUE	VIE	LU	MA	JUE	VIE	LU	MA	JUE	VIE		
1	Investigación																
1,1	Investigar sobre LoraWAN																
1,2	Sobre los elementos de LoraWAN																
1,3	End points																
1,4	Gateways																
1,5	Plataforma																
1,6	Formas de integración																
1,7	Casos de uso																
2	Diseño de solución																
2,1	Diagramar solución y sus partes																
2,2	Componentes a utilizar																
2,3	Especificaciones técnicas																
2,4	Seleccionar dispositivos y plataformas																
3	Implementación de endpoints																
3,1	Definir endpoints a usar																
3,2	Presupuestar los mismos																
3,3	Adquirir los endpoints																
3,4	Implementar endpoints																
4	Implementación de gateways																
4,1	Definir gateways a usar																
4,2	Presupuestar los mismos																
4,3	Adquirir los gateways																
4,4	Implementar gateways																
4,5	Lograr comunicación half duplex entre endpoints y gateways																
4,6	Lograr comunicación full duplex entre endpoints y gateways																
5	Implementación dashboard																
5,1	Servir los datos adquiridos																
5,2	Montar servidor red																
5,3	Desarrollar módulo de procesamiento de datos																
5,4	Desarrollar dashboard		■	■	■	■	■										
5,5	Integrar al sistema								■	■	■	■	■				
6	Pruebas de campo																
6,1	Montar solución en un escenario real																
6,2	Puesta en marcha del sistema								■	■	■	■	■				
6,3	Pruebas																
6,4	Proyección																
6,5	Conclusiones																
7	Documentación		■	■	■	■	■	■	■	■	■	■	■				

Lo más importante sobre quien aspira a ser un profesional en la **Universidad Pública** no es su promedio, su cantidad de publicaciones, o cargos políticos alcanzados. Lo más importante es no corromper los valores propios y los universales, no quebrarse ante las injusticias y la corrupción, ante la imprecisión del camino fácil, defender los recursos públicos como si fuera tal: un profesional.

Lo segundo más importante es nunca perder la imaginación y la creatividad, hacer de ellas, un plan de estudios propio.

Universidad  
Nacional de  
Río Cuarto

Ruta Nac. 36 - Km. 601  
X5804BYA Río Cuarto, Córdoba

<https://github.com/brivadeneira/LoRaWANprototype>