Universitat Oberta de Catalunya

uoc.edu

Diseño y simulación de un sistema de alquiler de bicicletas e integración en red LoRaWAN

Juan Luis Mayor Puyol
Máster Universitario en Ingeniería de Telecomunicación
Smart Cities

Víctor Monzón Baeza
Carlos Monzo Sánchez

Junio 2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño y simulación de un sistema de alquiler de bicicletas e integración en red LoRaWAN*
Nombre del autor:	*Juan Luis Mayor Puyol*
Nombre del consultor/a:	*Víctor Monzón Baeza*
Nombre del PRA:	*Carlos Monzo Sánchez*
Fecha de entrega :	06/2021
Titulación:	*Máster de Ingeniería de Telecomunicación*
Área del Trabajo Final:	*Smart Cities*
Idioma del trabajo:	*Castellano*
Palabras clave	*Smart bike* *LPWAN* *LoRaWAN*

Resumen del Trabajo:

Uno de los objetivos de las Smart Cities es contribuir a la mejora de la movilidad en las ciudades usando la tecnología con la misión de reducir las emisiones de monóxido de carbono fomentando, por ejemplo, el uso de bicicletas como medio transporte. Para ello, en las ciudades existen servicios de alquiler de bicicletas gestionado por los ayuntamientos, donde es posible usar la tecnología para la gestión del servicio. Además, actualmente es posible usar dispositivos IoT de bajo consumo y bajo coste y poder transmitir los datos de diferentes sensores usando redes LPWAN.

El objetivo de este trabajo es diseñar un sistema de alquiler de bicicletas, que fomente su uso premiando el buen comportamiento y la seguridad vial de los trayectos (se respeta las intersecciones señalizadas, semáforos, etc.), con descuento en el coste del alquiler. Los datos obtenidos durante el trayecto se pueden usar para la gestión del sistema de alquiler, conociendo por parte de la empresa el número de bicicletas en circulación, así como, donde se encuentran estacionadas y mejorar la gestión del servicio. Para ello, las bicicletas dispondrán de diferentes sensores, que a través de dispositivos IoT transmitirán los datos del trayecto como son posición, velocidad, así como datos de la seguridad vial, a través de una red LPWAN.

Usando la plataforma “The Things Network” y el simulador de dispositivos IoT Mbed se realiza una simulación de la comunicación entre el dispositivo y el servidor de aplicaciones a través de esta red.

Abstract:

One of the goals of Smart Cities is to contribute to the improvement of mobility in cities using technology, one of the missions is reducing carbon monoxide emissions by promoting, for example, bicycles as a means of transport. To do this, in cities we find bicycle shared services managed by city councils, where it is possible to use technology to manage this service. In addition, it is currently possible to use low-consumption and low-cost IoT devices and to be able to transmit data from different sensors using LPWAN networks.

The objective of this work is to design a bicycle sharing system, which encourages its use by rewarding good behavior and safety of the routes (signed intersections, traffic lights), with a discount on the rental cost. The data obtained during the route can be used to manage the rental system, knowing by the company the number of bicycles in circulation, as well as where they are parked and improving the management of the service. For this, the bicycles will have different sensors, which through IoT devices will transmit the data of the route such as position, speed, as well as road safety data, through an LPWAN network.

Using the platform "The Things Network" and the IoT Mbed emulator, the air interface is simulated, and it is possible to check the communication between the device and the application server via TTN network.

Índice

1. Introducción 1
1.1. Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 2
1.3 Enfoque y método seguido 2
1.4 Planificación del Trabajo 3
1.5 Breve sumario de productos obtenidos 5
1.6 Breve descripción de los otros capítulos de la memoria 5
2. Estado del arte 6
2.1. *Smart Cities*. Sector movilidad 6
2.1.1. Movilidad no motorizada 7
2.1.2. Transporte público 7
2.2. Sistema Alquiler bicicletas (BSS) 8
2.2.1. Características del sistema BSS de nueva generación 9
2.3. IoT (Internet of Things) 10
2.4. Tecnología LPWAN 11
2.4.1. NB-IoT 12
2.4.1.1. Modo de operación 13
2.4.1.2. Arquitectura de red 13
2.4.2. LoRa 14
2.4.2.1. Arquitectura de red 14
2.4.2.2. Capa física 15
2.4.2.3. Tipos de dispositivos 16
2.5. Sistemas de información geográfica SIG 19
2.5.1. Open Street Map 19
2.5.1.1. Formato de los datos 19
2.5.2. Base de datos georreferenciada 20
2.5.4. MongoDB 21
3. Diseño de sistema de alquiler 22
3.1.1. Bicicleta 22
3.1.2. Red LoRaWAN 23
3.2. Procesos 24
3.2.1. Alquiler bicicletas 24
3.2.2. Obtención de datos 25
3.2.3. Comprobación comportamiento 25
3.2.4. Fin de trayecto 26
3.3. Alarmas 27
4. Diseño red LoRaWAN 28
4.1. Estructura interfaz radio 28
4.2. Diseño paquete de carga útil 29
4.2.1. Mensaje Uplink 30
4.2.2. Mensaje de Downlink 31
4.3. Time of Air (ToA) 31
4.4. Estructura base de datos geolocalización 33
5. Arquitectura del sistema de simulación 35
5.1. Arquitectura de red simulación 35

5.1.1. Simulador Mbed....35
5.1.1.1. Web del simulador Mbed....36
5.1.1.2. Emulador Gateway....37
5.1.2. Servidor de aplicaciones....38
5.2. Creación e instalación de aplicaciones....39
5.2.1. Instalación simulador Mbed....39
5.2.2. Servidor de aplicaciones....41
5.2.2.1. Instalación servidor....41
5.2.2.2. Instalación Software necesario....42
5.2.2.3. Instalación Base de datos mongodb....42
5.2.2.4. Instalación Servidor Web y PHP....44
5.2.3. Definición aplicación en servidores TTN....45
5.2.3.1. Registro Gateway....45
5.2.3.2. Registro de la aplicación....46
5.2.3.2.1. Añadir dispositivo....47
5.2.3.2.2. Añadir conexión al servidor de aplicación....49
6. Simulación de BSS propuesto con Mbed y análisis de los resultados....52
6.1.1. Módulo simulador Mbed....52
6.1.2. Módulo Servidor de aplicaciones....55
6.1.3. Análisis simulación....58
7. Conclusiones....60
8. Glosario....61
9. Bibliografía....63
10. Anexos....66
I. Cálculo Time of Air (ToA) con Matlab....66
II. Código Mbed emulator....67
III.A. Código PHP “/bike/”....72
III.B. Código PHP “/message/”....72
IV. Código aplicación Python....73

Lista de figuras

Figura 1 Sistema multimodal de alquiler [4] 9
Figura 2 Comunicaciones inalámbricas [5] 11
Figura 3 Modos de operación NB-IoT [38] 13
Figura 4 Pila de protocolo LoRa [39] 14
Figura 5 Arquitectura red LoRaWAN [8] 15
Figura 6 Tipos de dispositivos LoRaWAN [8] 17
Figura 7 Red LoRaWAN TTN [12] 18
Figura 8 Esquema de bloques IoT [13] 22
Figura 9 Gateways y cobertura 23
Figura 10 Procesos del BSS 24
Figura 11 Proceso inicio alquiler 25
Figura 12 Procesos obtención de datos y comportamiento 26
Figura 13 Interfaz radio [22] 28
Figura 14 Payload [23] 32
Figura 15 Arquitectura simulación 35
Figura 16 Pila de protocolo Dispositivo IoT con SO MBed [41] 36
Figura 17 Pila de protocolo Dispositivo IoT con simulador [41] 36
Figura 18 Simulación MBED 37
Figura 19 Log Gateways 38
Figura 20 Maquina virtual simulador Mbed 39
Figura 21 V2 proxy 40
Figura 22 V3 proxy 40
Figura 23 “Simulator.html” 40
Figura 24 Máquina virtual servidor de aplicaciones 41
Figura 25 Comprobación versión mongodb 43
Figura 26 Importación datos desde OSM 43
Figura 27 Configuración No-IP 44
Figura 28 EUI Gateway 45
Figura 29 Configuración básica Gateway 46
Figura 30 Configuración capa de red Gateway 46
Figura 31 Overview aplicación 47
Figura 32 Configuración básica dispositivo 48
Figura 33 Configuración unión del dispositivo 48
Figura 34 Configuración capa de red dispositivo 49
Figura 35 Configuración capa de aplicación del dispositivo 49
Figura 36 Creación conexión webhook 50
Figura 37 Creación API Key 50
Figura 38 Simulación sistema BSS 52
Figura 39 Establecimiento conexión con simulador Mbed 53
Figura 40 Mensajes en TTS 53
Figura 41 Envío y recepción de mensaje en el simulador Mbed 54
Figura 42 Flujo de mensajes plataforma TTN 54
Figura 43 Configuración rutas HTTP POST 55
Figura 44 Mensaje Uplink en servidor de aplicaciones 56
Figura 45 Mensajes Downlink y señalización 57
Figura 46 Log aplicación de búsqueda de cruces 58
Figura 47 Log Gateway 58

Lista de tablas

Tabla 1 Comparación tecnologías LPWAN [8] 12
Tabla 2 Sensibilidad [10] 16
Tabla 3 Comparativa bases de datos georrefenciadas [40] 20
Tabla 4 Campos Payload [22] 29
Tabla 5 Tipos de datos [24] 29
Tabla 6 Cálculo ToA Uplink 33
Tabla 7 Cálculo ToA Downlink 33
Tabla 8 Comparación valores ToA 59

1. Introducción

1.1. Contexto y justificación del Trabajo

Smart Cities (Ciudad Inteligente) se puede definir como “Una ciudad innovadora que utiliza las TIC y otros medios para mejorar la calidad de vida, la eficiencia de la operación y los servicios urbanos y la competitividad, al tiempo que se asegura de satisfacer las necesidades de las generaciones presentes y futuras con respecto a los aspectos económicos, sociales y ambientales” [17]. Actualmente, en la gestión de las ciudades hay que incluir otro desafío como es la sostenibilidad que cubren un amplio espectro de problemas ambientales. Estos problemas incluyen el tráfico, contaminación, desechos o consumo energético.

Dentro del campo de la movilidad, el crecimiento urbano implica un aumento del tráfico rodado y, por tanto, de la contaminación, siendo un área donde las soluciones de Smart Cities pueden ser un elemento clave.

Aparte de las mejoras del transporte urbano, el fomento de un sistema de bicicletas compartida seguro y asequible se ha convertido en un elemento clave en la política de transporte. En este contexto, el uso de las tecnologías actuales se puede usar para realizar una gestión inteligente del servicio y de la seguridad en los trayectos.

Los sistemas de alquiler de bicicletas permiten a los ciudadanos desplazarse por la ciudad sin necesidad de usar el coche personal, preservando el medio ambiente. En trayecto largos es un excelente complemento al transporte público.

En la actualidad, es posible obtener un dispositivo IoT (Internet of Things) de bajo coste y bajo consumo que se pueden instalar en las bicicletas y que junto con diferentes sensores permitan trasmitir datos sobre la posición de la bicicleta, velocidad, distancia recorrida, posibles caída y comportamiento del usuario.

En este trabajo, se propone y diseña un sistema de alquiler de bicicletas (Bycicle Share System BSS) que fomente el uso de estos sistemas premiando el uso responsable del sistema durante el trayecto. Para ellos a partir de los datos obtenidos durante el trayecto ofrecer descuento en el coste del alquiler según el comportamiento del usuario de esta manera fomentar el buen uso del sistema.

Por otro lado, las características de los datos obtenido por estos sistemas lo hacen atractivo en la investigación de la planificación urbana sostenible. Esto sería posible gracias a que los sensores instalados nos facilitan la recopilación de datos sobre el estado de las vías y permitirá tener una red virtual con capacidad de conocer el estado de la movilidad en tiempo real y usándose para mejorar la seguridad de las vías.

1.2 Objetivos del Trabajo

En este trabajo se plantea cumplir los siguientes objetivos:

- Diseño sistema alquiler bicicletas: Partiendo del modelo de un sistema actual alquiler de bicicletas, adaptarlo a los requerimientos de este trabajo definiendo los requisitos que debe cumplir para la adquisición de los datos del comportamiento del usuario durante el trayecto, con el objetivo de poder calcular el coste del servicio.

- Desarrollar sistema de adquisición de datos geográficos: Se pretende diseñar un sistema que a partir de los datos de localización del usuario se pueda analizar si se circula de forma segura. Para ello, las bicicletas deben ser capaz de conocer la posición de las intersecciones señalizadas de las vías por la que circula el. Esta información se transmite al servidor donde se ha instalado la aplicación que a partir de la posición del usuario se obtiene la posición de cruces señalizados y se transmiten al dispositivo IoT. Por otro lado, los datos recibidos se podrán usar para obtener los datos del estado de trayecto y al finalizar el tiempo de alquiler calcular el coste a cobrar al usuario teniendo en cuenta la seguridad del trayecto.

- Estudio simulador de dispositivos IoT: Se estudiará como adaptar un simulador de dispositivos IoT para poder ser usado en entornos de *Smart Cities.*

- Integración y simulación entorno: Se pretende probar las aplicaciones diseñadas simulando el dispositivo IoT diseñado e integrarlo en una red real de LoRaWAN (Low Radio Wide Area Network) como es la plataforma “The Things Network” y conectarlo al servidor donde se encuentra instalada la aplicación y la base de datos georreferenciada.

1.3 Enfoque y método seguido

Para la realización de este trabajo se va ha realizar un diseño teórico del sistema de alquiler de bicicletas, desde el comienzo del trayecto al seleccionar la bicicleta, así como del seguimiento de la ruta seguida y envío de la información necesaria para obtener los datos de posicionamiento de las intersecciones y velocidad máxima de la vía por la que se circula, así como la posición de la estación más próxima. Al finalizar el trayecto, los datos obtenidos se almacenan en los servidores de aplicaciones y pueden ser usados para la mejora del servicio y conocer el estado de las vías.

Posteriormente, se diseñará el dispositivo IoT que se integrarán en la bicicleta y el servidor para la gestión de los datos de posicionamiento de estos elementos, así como los datos a enviar, a través de LoRaWAN necesarios para obtener los datos necesarios según la ruta que se está realizando.

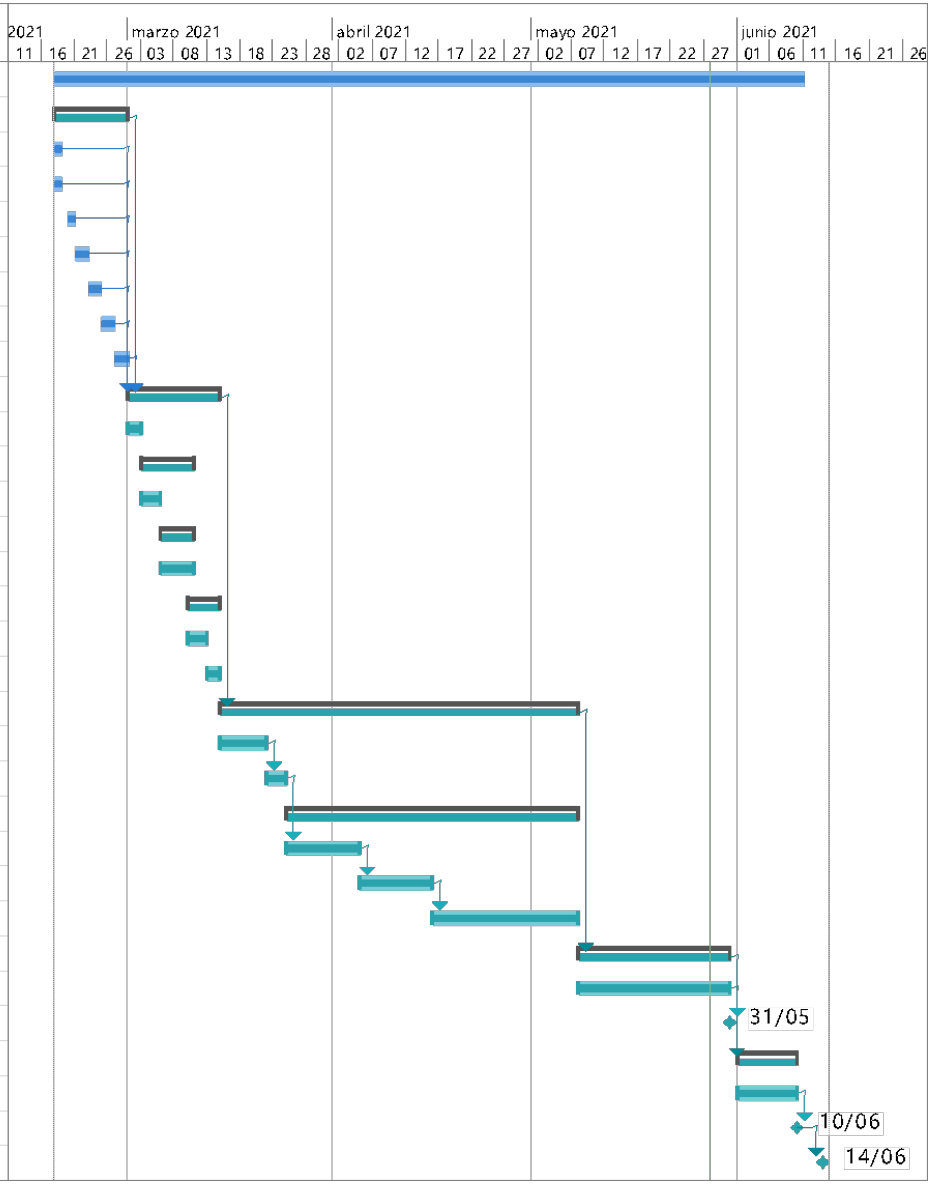
Para finalizar, con la ayuda del emulador de dispositivos IoT se simulará el dispositivo diseñado y se integrará, por un lado, en la plataforma “The Thing

Networks”, por otro lado, el servidor de aplicaciones que enviará los datos necesarios al dispositivo IoT.

1.4 Planificación del Trabajo

En el diagrama siguiente se expone la planificación temporal del trabajo a realizar, así como los hitos a completar para finalizar el trabajo con éxito

Id	Nombre de tarea	Duración	Comienzo	Fin
1	Trabajo Fin de Master	113 días	jue 18/02/21	jue 10/06/21
2	**PEC 1**	**11 días**	**jue 18/02/21**	**dom 28/02/21**
3	Propuesta de titulo	1 día	jue 18/02/21	jue 18/02/21
4	Resumen	1 día	jue 18/02/21	jue 18/02/21
5	Palabra Clave	1 día	sáb 20/02/21	sáb 20/02/21
6	Motivación	2 días	dom 21/02/21	lun 22/02/21
7	Objetivo del proyecto	2 días	mar 23/02/21	mié 24/02/21
8	Enfoque y método seguido	2 días	jue 25/02/21	vie 26/02/21
9	Planificación del proyecto	2 días	sáb 27/02/21	dom 28/02/21
10	**PEC 2 State of Art**	**14 días**	**lun 01/03/21**	**dom 14/03/21**
11	Sistema alquiler bicicletas	2 días	lun 01/03/21	mar 02/03/21
12	**Tecnología LPWAN**	**8 días**	**mié 03/03/21**	**mié 10/03/21**
13	MB-IoT	3 días	mié 03/03/21	vie 05/03/21
14	**LoRaWAN**	**5 días**	**sáb 06/03/21**	**mié 10/03/21**
15	The Things Stack	5 días	sáb 06/03/21	mié 10/03/21
16	**Sistema geolocalización**	**5 días**	**mié 10/03/21**	**dom 14/03/21**
17	OpenStreetView	3 días	mié 10/03/21	vie 12/03/21
18	Base de datos geolocalización	2 días	sáb 13/03/21	dom 14/03/21
19	**PEC 3**	**54 días**	**lun 15/03/21**	**vie 07/05/21**
20	Diseño sistema alquiler	7 días	lun 15/03/21	dom 21/03/21
21	Diseño red LoRaWAN	3 días	lun 22/03/21	mié 24/03/21
22	**Simulación**	**44 días**	**jue 25/03/21**	**vie 07/05/21**
23	Diseño simulador aplicación IoT	11 días	jue 25/03/21	dom 04/04/21
24	Diseño servidor aplicación	11 días	lun 05/04/21	jue 15/04/21
25	Integración simulador en The Thing Netwo	22 días	vie 16/04/21	vie 07/05/21
26	**PEC 4**	**23 días**	**sáb 08/05/21**	**lun 31/05/21**
27	Realizacion Memoria	23 días	sáb 08/05/21	dom 30/05/21
28	Entrega memoria	0 días	lun 31/05/21	lun 31/05/21
29	**PEC 5**	**9 días**	**mar 01/06/21**	**jue 10/06/21**
30	Realización presentación	9 días	mar 01/06/21	mié 09/06/21
31	Entrega presentación	0 días	jue 10/06/21	jue 10/06/21
32	Defensa	0 días	lun 14/06/21	lun 14/06/21



1.5 Breve sumario de productos obtenidos

- Diseño sistema de alquiler de bicicletas.
- Diseño de dispositivo IoT.
- Diseño de servidor de aplicación con los datos de localización.
- Diseño de arquitectura de red de simulador IoT para aplicaciones de *Smart Cities*.
- Integración del diseño en plataforma LoRaWAn “The Thing Network”

1.6 Breve descripción de los otros capítulos de la memoria

Capitulo 2. Estado del arte. En este capitulo se hará una investigación documental de las necesidades a tener en cuenta en el diseño del sistema de alquiler, así como de la tecnología LPWAN (Low-Power Wide-Area Network) elegida en este trabajo. Por último, se incluye una explicación de las plataformas de geolocalización que existe y de las soluciones de bases de datos adaptada a la geolocalización.

Capitulo 3. Diseño del sistema de alquiler. Se describirá los diferentes módulos que forman parte del sistema diseñado de alquiler de bicicletas y partir de este diseño se describirá los requerimientos de los diferentes los elementos que forman parte de la red a diseñar.

Capitulo 4. Diseño red LoRaWAN. A partir de los requerimientos del sistema de red diseñado en este capitulo se diseñará el formato de la carga útil a transmitir y recibir teniendo en cuenta las características de diseño de este tipo de redes.

Capitulo 5. Arquitectura de sistema de simulación. En este capitulo se definirá la arquitectura del sistema diseñado para poder probar las aplicaciones desarrolladas, también se incluye la descripción de la creación e integración de los elementos que forman parte la arquitectura.

Capitulo 6. Simulación y análisis resultados. Se analiza la puesta en servicio de la simulación analizando su funcionamiento y se comprobará que se cumple los requisitos diseñados en los capítulos anteriores.

Capitulo 6. Conclusiones. Análisis de los resultados obtenidos y lecciones aprendidas, así como análisis de los objetivos planteados y líneas a completar en futuros trabajos.

2. Estado del arte

Es este capitulo se realiza un repaso del estado actual del sector de la movilidad de las *Smart Cities*, en donde se expondrá como las nuevas tecnologías pueden ayudar en el objetivo de mejorar la movilidad urbana en las grandes ciudades. Este trabajo está centrado en los sistemas de bicicletas compartidas, por lo que, se presenta un ejemplo de como puede ser un sistema multimodal incluyendo un sistema de alquiler de bicicletas y cuales son las necesidades de este tipo de sistema.

Por otro lado, se realiza un análisis de los sistemas de comunicación LPWAN NB-IoT (Narrow Band IoT) y LoRaWAN centrándonos en está ultima tecnología que es la que se va a utilizar en este trabajo, permitiendo conectar los dispositivos instalados en las bicicletas con los servidores.

Por último, se realizará un breve estudio de los sistemas de información geográficas centrándose en Open Street Map (OSM) y las bases de datos que existen para manejar los datos georreferenciados que permitirá obtener la información geográfica necesaria en este trabajo

2.1. *Smart Cities*. Sector movilidad

Teniendo en cuenta que el concepto de *Smart Cities* surge como la búsqueda de soluciones tecnológicas para conseguir una ciudad más eficiente a la hora de manejar sus recursos. Actualmente, en la Unión Europea (UE) el 70% de la población vive en el entorno de la ciudad [1], por lo que se hace necesario hacer un uso eficiente de los recursos disponible para la movilidad urbana. No solo por gestionar el uso del transporte, sino que el aumento de la población provoca problemas como son la contaminación, cambio climático y congestión del tráfico.

Por esto, desde la UE existe acciones para mejorar la movilidad urbana, como es la iniciativa CIVITAS [2], cuyo objetivo es fomentar el uso de transporte ecológico.

Dentro del plan estratégico de movilidad urbana de la UE, se ha identificado los retos principales a los que las ciudades deben hacer frente, entre los que se encuentra:

- Movilidad no motorizada: Debe potenciarse el uso del transporte no motorizado, como es el uso de las bicicletas, facilitando el acceso a partir de un sistema de alquiler sencillo, barato, haciendo que sea atractivo y sobre todo que sea seguro.

- Transporte publico: Donde es necesario llegar a un sistema eficiente, de alta calidad, seguro y accesible.

- Sistemas de movilidad multimodal: Se debe fomentar el uso de los sistemas públicos existente facilitando la posibilidad de cambiar de un medio de transporte a otro.

Para alcanzar estos objetivos se puede usar soluciones tecnológicas por lo que la *Smart Cities* se presenta como una solución para el desarrollo de soluciones sostenibles de movilidad en las ciudades. Por otro lado, el uso de la información para procesar y analizar grandes volúmenes de datos permite conocer el estado de la movilidad urbana en tiempo real y tomar decisiones que permita mejorar los sistemas de movilidad y hacer un uso eficiente de los sistemas de transportes, además de conseguir aumentar la seguridad de los usuarios de estos sistemas de transporte [19].

2.1.1. Movilidad no motorizada

Uno de los objetivos de la movilidad en las *Smart Cities* en la ciudad consiste en fomentar el uso de las opciones no motorizadas y limpias, fomentando el uso de desplazamiento en bicicleta. Desde UE [16] se quiere fomentar el uso de las bicicletas en las ciudades en los trayectos cortos para reducir la contaminación y congestión de tráfico en las ciudades con campañas de sensibilización [18] y proporcionando financiación al través del programa CIVITAS.

Además, un mayor uso de la bicicleta en las ciudades reduce el número de accidente en las ciudades [3], debido a que un mayor número de ciclista por habitantes ha demostrado que reduce la tasa de accidentes. Varios factores pueden explicar esta reducción de la tasa de accidentes. En primer lugar, a medida que cada ciclista acumula más kilómetros, se vuelve más experimentado y consciente de los peligros del tráfico. En segundo lugar, cuando los ciclistas se vuelven más numerosos en el tráfico, los conductores de vehículos de motor se vuelven más conscientes de la presencia de ciclistas y pueden comportarse de manera más considerada con ellos.

También se pude observar lo efectos beneficiosos del ciclismo en la salud reduciendo el riesgo cardiovascular. Se calcula que los años de vida ganados en bicicleta superan los años de vida perdidos en choques en 20 a 1 [3].

Otro factor de mejora son los efector medioambientales donde se estima que los efectos del uso de esto sistemas en las ciudades es la siguiente [3]:

- 30% menos cogestión de tráfico.
- 36% menos de emisión de monóxido de carbono (CO2).
- 25% menos del consumo de gasolina en los coches particulares.
- 9% menos de contaminación acústica.

2.1.2. Transporte público

En las grandes ciudades se enfrenta a problemas causado por el transporte y el tráfico, por lo que interesa buscar soluciones para mejorar la movilidad, consiguiendo reducir la congestión, accidentes y contaminación, para ello se necesita un transporte publico eficiente y no contaminante, por ello desde EU existe planes de mejoras del transporte urbano [35].

En esta área el uso de las nuevas tecnologías de las *Smart Cities* ayuda en el fomento del uso del transporte urbano adaptándolo a las necesidades del

usuario. Estas soluciones incluyen el uso de planificadores de viajes, servicios de información del servicio en tiempo real, esquemas de uso compartido, o servicios bajo demanda [36].

2.1.3. movilidad multimodal

La movilidad urbana esta cambiando debido a que existe un aumento de la oferta de diferentes modos de transporte ya sea publico y privado que se puede usar de forma conjunta, siendo uno de los principales retos de la movilidad inteligente de las *Smart Cities*. En este campo los vehículos inteligentes y las comunicaciones entre dispositivos M2M (Machine to Machine) abre un amplio campo de desarrollo de diferentes aplicaciones multimodales fomentando el uso eficiente de los diferentes modos de transportes disponibles, permitiendo de esta manera reducir el tráfico y, por tanto, reducir el consumo energético, la congestión de las ciudades y la contaminación [36]. La información en tiempo real y planificadores de viaje permiten obtener un beneficio en el esquema multimodal. La oferta combinada permite competir con el coche privado en términos de flexibilidad y costes. También es posible, tener una oferta completa al combinar eficientemente la movilidad no motorizada y el transporte público.

2.2. Sistema Alquiler bicicletas (BSS)

Los sistemas de alquiler de bicicletas (BSS), para trayectos cortos, es una opción sostenible, eficiente y no contaminante que fomenta el uso de las bicicletas en las ciudades, mejorando la movilidad y reduciendo la contaminación. Existen múltiples soluciones de sistemas BSS en el mercado, en este apartado se describe un sistema de alquiler integrado en una red multimodal [4].

En este articulo [4], se describe un sistema multimodal que incluye en la planificación de los trayectos el poder reservar la bicicleta dependiendo del trayecto realizado en el trasporte urbano, para ello, el sistema debe conocer en todo momento en tiempo real es estado de las estaciones y su distribución, así como el estado del transporte público. Esta información no solo se usa para reservar la bicicleta sino también para conocer las necesidades de las estaciones y poder mover elemento de una estación a otra según las necesidades de cada momento, ya sea, por el histórico de reservas o por estado en cada momento de las peticiones de reserva, tanto de las bicicletas como el del transporte urbano.

En la figura 1 se muestra los diferentes elementos y sistemas que forman parte del sistema descrito el articulo, que incluye las bicicletas inteligentes, con sensores instalados y con conexión a los servidores del sistema a través dispositivos IoT. Estos dispositivos, envían información de su estado, estado del usuario y disponibilidad para el siguiente cliente. Además, usando comunicación M2M soporta bloqueo/desbloqueo de la bicicleta para evitar robos y vandalismo. La aplicación del sistema es capaz de conocer el estado a tiempo real, suministrando la información al área de operación y mantenimiento y al usuario a la hora de conocer las posibilidades que tienen a la hora de alquilar la bicicleta.

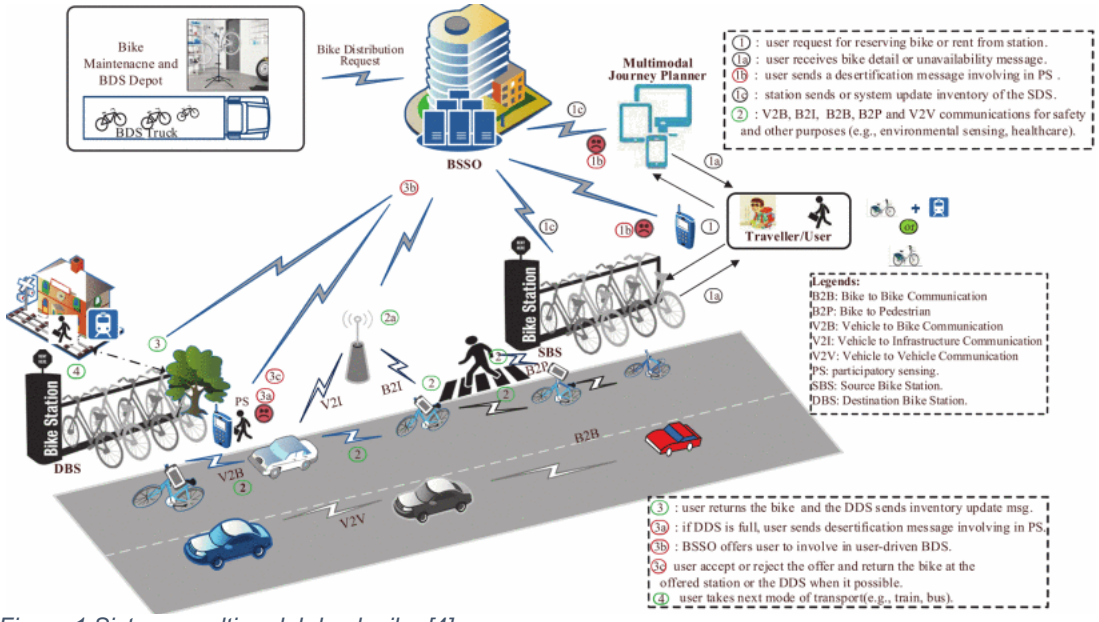


Figura 1 Sistema multimodal de alquiler [4]

2.2.1. Características del sistema BSS de nueva generación

En el sistema descrito y gracias a los dispositivos IoT, es posible desarrollar sistemas más completos, seguros y que responda dinámicamente en tiempo real. El autor propone que un sistema de alquiler de nueva generación debería incluir los siguientes servicios:

- Servicio de reserva (RS): Este servicio permite al usuario realizar la reserva y el pago online usando su móvil o usando el quiosco instalado en las estaciones de bicicletas o en las estaciones modales donde se comienza el trayecto.

- Conectividad multimodal (MC): La mayoría de los trayectos en bicicleta son cortos y pueden ser combinado con el transporte urbano. En los futuros sistemas de alquiler es interesante incluir la posibilidad de combinar los trayectos, por lo que, unos de los servicios que debe incluir el sistema es poder conocer el estado de los diferentes sistemas de transporte de la ciudad. De esta manera, el sistema es capaz de ofrecer opciones incluyendo la conexión multi-modal.

- Comprobación de disponibilidad de bicicleta y estacionamiento (BPAC): El Sistema debe conocer en tiempo real el número de bicicletas disponibles y el esta de las estaciones antes de aceptar las reservas de los usuarios. Si existe disponibilidad el sistema permite realizar la reserva, en caso contrario verifica el calendario de redistribución para comprobar si es posible realizar la reserva o en caso contrario ofrecer posibles estaciones próximas en la que se puede realizar la reservar.

- Servicio de distribución de Bicicletas (BDS): Uno de los objetivos de sistema de alquiler es evitar que las estaciones no dispongan de

bicicletas, para ello, el servicio BDS debe monitorizar continuamente la demanda y se capaz de adelantarse a las necesidades dinámicamente.

- Robo y vandalismo: El robo y vandalismo es uno de los principales problemas en estos sistemas, por lo que, para la supervivencia y sostenibilidad de los BSS deben poseer tecnologías avanzadas de bloqueo y seguimiento.

- Servicios de salud: El uso principal de las bicicletas es la realización de actividad física y el impacto que tiene en la salud de los usuarios. Lo sistemas de alquiler deben ser capaz de ofrecer información de la cantidad de ejercicio físico realizado.

- Seguridad de los usuarios: Los BSS deberían ofrecer servicios mejorados de seguridad para los ciclistas utilizando las TIC y el sistema de transporte inteligente.

- Informes: El sistema debe ser capaz de ofrecer información sobre el estado del sistema y de los trayectos, parar ofrecer un mejor servicio y ayudar a las ciudades a mejorar el estado de las carreteras.

2.3. IoT (Internet of Things)

IoT se describe como la capacidad de los dispositivos de conectarse a Internet y entre si, sin la necesidad, por parte de los usuarios de interactuar con estos dispositivos, permitiendo transmitir información y toma decisiones.

Los requisitos básicos que debe cumplir un dispositivo para ser considerado IoT son:

- Bajo consumo de energía. Deben estar diseñado para que el consumo de energía sea eficiente, ya que, van a trabajar con baterías y en entornos donde no es posible un cambio regular.

- Tamaños reducidos. Los dispositivos de conexión deben ser pequeños para poder ser utilizado con cualquier tipo de sensores.

- Movilidad en las comunicaciones. Debe asegurarse que las redes de comunicación funcionan correctamente con dispositivo de movilidad total.

- Servicios de localización. Las redes de transporte deben ser capaz de permitir la localización de los dispositivos.

- Comunicaciones seguras. Debe asegurarse la confidencialidad de los datos trasmitidos.

- Redes de conexión de amplio alcance. Las redes de comunicación deben de tener un amplio alcance y ser capaz de aceptar un elevado número de dispositivos.

- Bajo coste. Tanto los dispositivos como los elementos deben de tener un bajo coste de producción.

Estos requisitos son básicos porque Los dispositivos IoT se usan en redes de sensores que trabajan en un entorno donde existe un gran número de elementos, con poco ancho de banda, trabajando con baterías y e instalados en zona aisladas. Con estas restricciones los protocolos de comunicación que se usen deben ser capaz de manejar eficientemente estas comunicaciones. En la figura 2 observamos las diferentes redes de comunicaciones que existen en la actualidad donde se sitúan según el área de cobertura y el ancho de banda, en el caso de las redes móviles, WFI y Bluetooth se tienen un amplio ancho de banda, pero no cubren el área de cobertura que necesitaría los dispositivos IoT. El grupo de redes de comunicaciones que cubrirían una amplia área de cobertura y con suficiente ancho de banda para estos dispositivos sería LPWAN.

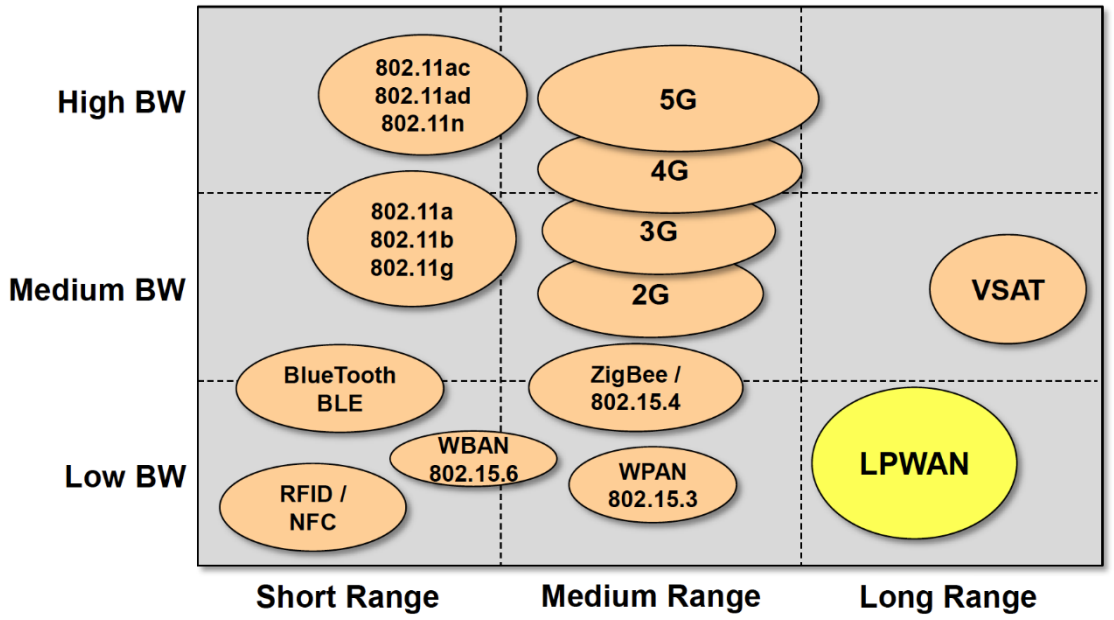


Figura 2 Comunicaciones inalámbricas [5]

2.4. Tecnología LPWAN

LPWAN es un conjunto de tecnologías que nos permite conectar los dispositivos a larga distancia con poco ancho de banda y por tanto bajo consumo de energía. Con estas características permiten que pueda ser usado en sistemas IoT.

En Europa es posible usar la banda ISM (Industrial, Scientific and Medical), la cual representa un espectro que se reserva al uso no comercial asociado con la industria, la ciencia y los servicios médicos. Esta banda puede ser usada sin licencia siempre que se respeten las restricciones de potencia transmitida y uso del interfaz aire.

Los diferentes sistemas que forman parte del LPWAN tienen en común las siguientes características.

- Ultra baja potencia de operación: Al tener una baja potencia de transmisión implica un bajo consumo de energía, por tanto, las baterías de dispositivos instalados en las bicicletas duren más.

- Amplia cobertura: LPWAN permite la comunicación entre dispositivos en un rango de kilómetros, reduciendo el número de estaciones necesario a desplegar en las ciudades con el sistema de alquiler y facilitando el diseño de la red.

- Baja tasa de datos: Se tiene una baja tasa de datos que va desde los pocos bps a 2 Mps, con una tasa media de unos ciento kbps.

Dentro del conjunto de tecnologías LPWAN la principal diferencia son las que ofrecen el servicio en el espectro licenciado (NB-IoT, LTE-M) y aquellas que usan el espectro no licenciado (Sigfox, LoRaWAN).

Las tecnologías licenciadas, los operadores usan parte de las frecuencias asignadas para dar servicio a los dispositivos IoT, estos operadores cobran el servicio según el número de Kbytes. En cambio, las tecnologías no licenciadas están apoyadas por operadores alternativo y resulta más baratas, tanto en conectividad como en el coste de los dispositivos. La ventaja que presentan es que se pueden desplegar rápidamente a bajo coste y sin necesidad de disponer de un espectro licenciado, de ahí que resulten mas atractivas

En la tabla 1 se ofrece una comparativa entre las diferentes tecnologías donde observamos que se cumplen las principales características de esta tecnología con una amplia área de cobertura, poco consumo de energía permitiendo una larga vida de las baterías, debido a que tiene un pequeño ancho de banda.

Feature	LoRaWAN	Sigfox	NB-IoT	LTE-M
Modulation	SS Chirp	GFSK/DBPSK	UNB/GFSK/ DBPSK	OFDMA
Data Rate	290bps - 50kbps	100bps 128bytes Max	100bps 12/8bytes Max	200kbps - 1Mbps
Link Budget	154 dB	146 dB	151 dB	146 dB
Battery lifetime	8 _ 10 years	7 _ 8 years	7 _ 8 years	1_ 2 years
Power Efficiency	Very High	Very High	Very High	Medium
Security/ Authentication	Yes(32 bits)	Yes(16 bits)	No	Yes(32 bits)
Range	2-5km urban 15km suburban 45km rural	3-10km urban - 30-50km rural	1.5km urban - 20-40km rural	35km - 2G 200km - 3G 200km - 4G
Interference Immunity	Very High	Low	Low	Medium
Scalability	Yes	Yes	Yes	Yes
Mobility/ Localization	Yes	No	Limited, No Loc	Only Mobility

Tabla 1 Comparación tecnologías LPWAN [8]

2.4.1. NB-IoT

Narrow Band IoT (NB-IoT) [6][7] es una de las tecnologías LPWAN que usa la red de acceso móvil. Aparece por primera vez en la Versión 13 del estándar 3GPP [37]. Ha sido crea para para poder cumplir con los requisitos de las

comunicaciones IoT. Ofrece dispositivos de bajo coste, bajo consumo y baja latencia. Utiliza la infraestructura móvil ya existente usando un ancho de banda de 200 kHz. Dispone de 2 modos de ahorro de energía para maximizar el rendimiento y eficiencia: recepción discontinua extendida (eDRX) y modo de ahorro de energía (PSM). PSM permite que los dispositivos entre en un estado profundo de hibernación de hasta 310 horas.

2.4.1.1. Modo de operación

En la figura 3 aparece los diferentes modos de operación de este sistema y que serían:

- In-band: en este modo, el canal de NB-IoT usa un bloque en la portadora de LTE.

- Guard band: NB-IoT utiliza un bloque de los recursos no usado de banda de guarda de LTE.

- Stand alone: en este modo, se usa una portadora dedicada que puede uno de los canales de GSM.

Los modos In-Band y Guard permite usar los servicios de autenticación, autorización y seguridad de LTE.

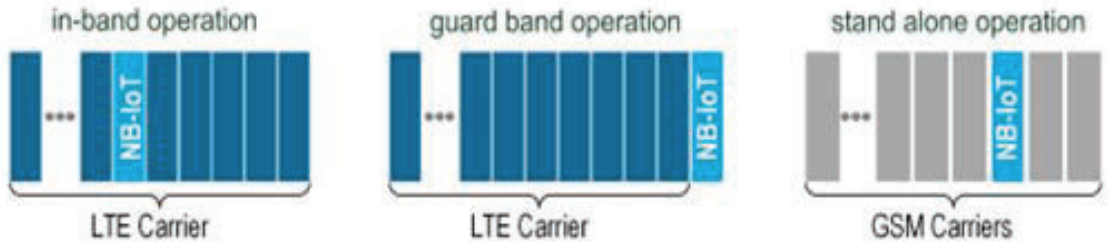


Figura 3 Modos de operación NB-IoT [38]

2.4.1.2. Arquitectura de red

La arquitectura de NB-IoT coincide con la red Evolved Packet Core (EPC) de l LTE, pero adaptada para una red IoT, es decir, capaz de admitir un alto número de dispositivos con baja tasa de datos.

La arquitectura estaría forma por los siguientes elementos

- NB-IoT UE: NB-IoT UE (User Equipment) establece conexión y se comunica con el eNodeB.

- eNodeB: El propósito del eNodeB es llevar a cabo el procesamiento de acceso del interfaz aire. El eNodeB se comunica a través del Interfaz S1-lite con el IoT Evolved Packet Core (EPC).

- IoT EPC: interfaces de IoT EPC con el NAS del UE y reenvía los datos de IoT a la plataforma de IoT para su procesamiento.

- Plataforma de IoT: la plataforma de IoT recopila los datos de varias redes de acceso de IoT y los reenvía al servidor de aplicaciones correspondiente.

- Servidor de aplicaciones: el servidor de aplicaciones es el destinatario final de los datos de los dispositivos IoT, los procesa de acuerdo con los requisitos del cliente.

2.4.2. LoRa

LoRa es una tecnología LPWA, de código abierto y está gestionada por LORA-alliance. Es una tecnología de capa física que modula la señal en la banda ISM (industriales, científicas y médicas) usando una técnica de espectro ensanchado llamada CSS (Chirp Spread Spectrum). Es una tecnología de baja potencia de transmisión, baja tasa de transmisión y amplia cobertura.

En la figura 4 se muestras la capa que lo forman, la capa física que cuya modulación permite la comunicación a larga distancia y la capa MAC (LoRaWAN) que proporciona el acceso a los servidores de red y que ha sido desarrollado para reducir el consumo de baterías de los dispositivos, así como la capacidad de la res, la calidad de servicio y la seguridad en las conexiones.

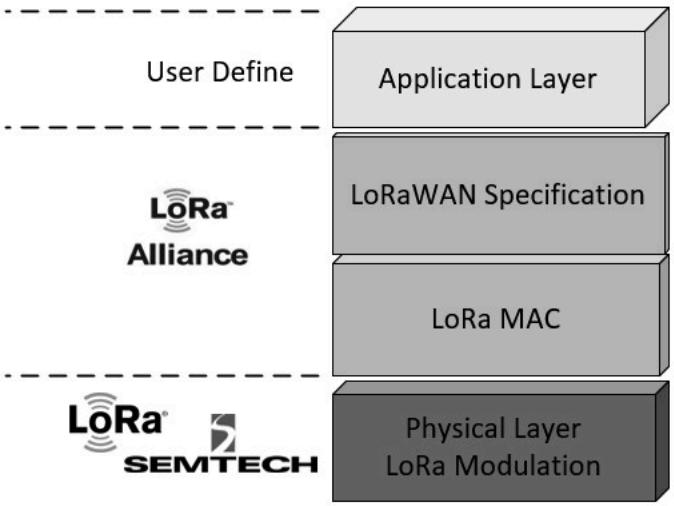


Figura 4 Pila de protocolo LoRa [39]

LoRa usa el esquema ADR (Adaptive Data Rate) en la infraestructura de red para gestionar el bit rate individualmente y maximizar la duración de las baterías de los dispositivos, normalmente este esquema se usa en dispositivos con condiciones de radio estable, es decir, para elemento estáticos.

2.4.2.1. Arquitectura de red

La arquitectura de red está formada por los siguientes elementos:

- Dispositivos: Dispositivo IoT con sistema de comunicación de bajo consumo integrado y que se comunica son los Gateways.

- Gateway: antenas que reciben y envía las señales a los dispositivos y se conecta con los servidores a través de interfaces de alta capacidad.

- Servidores de red: Servidores que descodifica los mensajes recibidos por los dispositivos, realiza verificaciones de seguridad y de ADR. Distribuye los mensajes desde los dispositivos finales a la aplicación correcta y viceversa.

- Aplicación: Pieza de software que se ejecuta en un servidor que recibe los datos de los servidores de red. Descodifica los paquetes recibidos y decide las acciones según la aplicación.

En la figura 5 se muestra la arquitectura típica que usa una topología “star-of-star”.

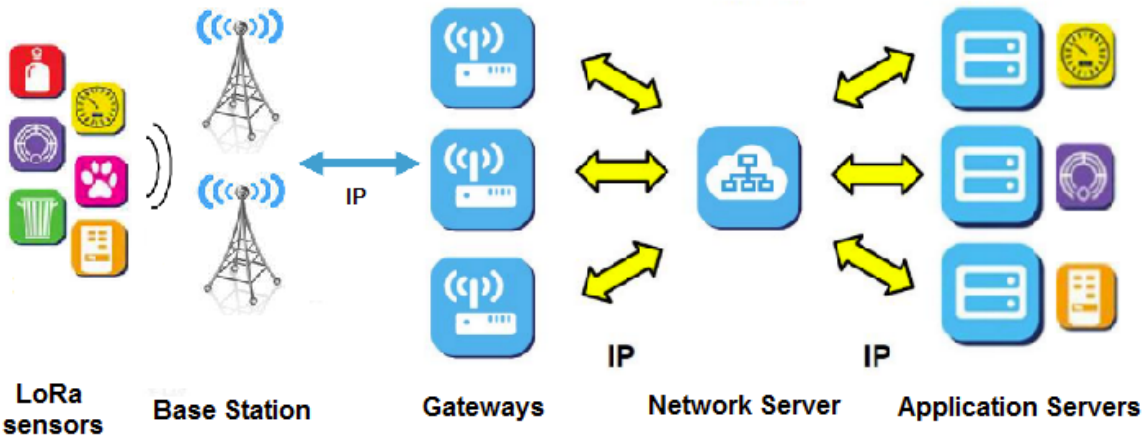


Figura 5 Arquitectura red LoRaWAN [8]

2.4.2.2. Capa física

LoRa usa una técnica de modulación de espectro ensanchado patentado por Semtech y mantenido por LoRa Alliance. Modula las señales en la banda sin licencia Industrial, científica y médica (ISM). LoRa utiliza el pulso de radar comprimido de alta intensidad (chirp), con técnicas de modulación de espectro ensanchado (CSS) y con técnicas de corrección de errores FEC (Foward Error Correction) que ayuda a reducir las interferencias causadas por diferentes velocidades de datos. También permite la recuperación de información en el caso de errores de transmisión. LoRa también admite comunicación bidireccional. La velocidad de datos de LoRa varía de 300 bps
a 37,5 kbps Los parámetros de transmisión para LoRa son los siguientes:

- Potencia de transmisión (TP): Se puede ajustar en un rango de entre -4 dB y 20 dB, para potencias mayores de 17 dB usa un 1% del ciclo de trabajo. El ciclo de trabajo es el porcentaje máximo del tiempo de duración en el que un dispositivo puede ocupar un canal.

- Frecuencia portadora: Se puede programar entre 137 MHz a 1020 MHz con pasos de 61Hz. En Europa las bandas disponibles están entre 863 MHz y 870 MHz.

- Ancho de banda (BW): En rango de ancho de banda varia entre los 7.8 kHz y los 500 kHz, lo típico es que se elija entre 500 kHz, 250 kHz o 125 kHz.

- Spreading Factor (SF): SF es la relación entre la tasa de símbolos y la tasa de chip. El rango de SF varía según las ubicaciones geográficas, en Europa se usa entre 6 y 12. Representados como [SF6, SF7, SF8,,, SF12]. Cada aumento en el SF de la tasa de transmisión duplica la duración de la transmisión aumenta los consumos de energía para LoRa. Tal y como se aprecia en la tabla 2 el aumento de SF también aumenta la sensibilidad, y, por tanto, la relación señal / ruido (SNR), como consecuencia aumenta el tiempo aire del paquete. LoRa usa un factor de difusión ortogonal que permite la transmisión de múltiples paquetes con diferentes SF en el mismo canal al mismo tiempo.

Spreading factor	Sensitivity (dBm)
7	-130.0
8	-132.5
9	-135.0
10	-137.5
11	-140.0
12	-142.5

Tabla 2 Sensibilidad [10]

- Tasa de codificación (CR): LoRa se enfrenta a una ráfaga de interferencia mientras transmite los datos. Esta interferencia se controla seleccionando la tasa de codificación (CR). Las configuraciones posibles para CR son 4/5, 4/6, 4/7 o 4/8. La protección será más alta con un CR más alto, pero implica un tiempo mayor del paquete en el aire.

- Tasa de datos adaptativa (ADR): En LoRa, las funciones de ADR habilitan los dispositivos y servidores para seleccionar la configuración de los parámetros de transmisión automáticamente. La velocidad de datos se puede gestionar de dos formas a través de el dispositivo en sí o a través de equipos de red. Ayuda a conservar la energía y nos da un mejor rendimiento seleccionando automáticamente la configuración según requisitos

2.4.2.3. Tipos de dispositivos

En los dispositivos, existe un compromiso entre la latencia en el enlace descendente y la cantidad de energía utilizada. LoRaWAN es un protocolo asincrónico basado en ALOHA, significa que un dispositivo puede "despertar" a

intervalos programables para comprobar el enlace descendente, reduciendo así la latencia y el consumo de batería.

En LoRaWAN existe tres clases, que se muestran en la figura 6. La principal diferencia entre estas clases es la latencia del enlace descendiente, por tanto, la vida útil de la batería del dispositivo, se elijará el modo de trabajo dependiendo de las necesidades de la aplicación.

- Clase A (bidireccional): Los dispositivos admiten la comunicación bidireccional entre un dispositivo y una puerta de enlace. Los mensajes de enlace ascendente (desde el dispositivo al servidor) se pueden enviar en cualquier momento (aleatoriamente). A continuación, el dispositivo abre dos ventanas de recepción en momentos específicos, después de una transmisión de enlace ascendente. Si el servidor no responde en ninguna de estas ventanas de recepción, la próxima oportunidad será después de la próxima transmisión de enlace ascendente desde el dispositivo. El servidor puede responder en la primera ventana de recepción o en la segunda ventana de recepción, pero no debe utilizar ambas ventanas. Tiene bajo consumo de energía y alta latencia.

- Clase B: Los dispositivos de Clase B agregar ventanas de recepción programadas para los mensajes de enlace descendente desde el servidor. Utilizando balizas de sincronización enviadas por el Gateways. los dispositivos abren periódicamente ventanas de recepción. Tiene un consumo de energía medio, baja latencia.

- Clase C: mantienen abiertas las ventanas de recepción a menos que estén transmitiendo. Esto permite una comunicación de baja latencia, pero consume. Tiene alto consumo de energía. Y la latencia más baja de los tres tipos.

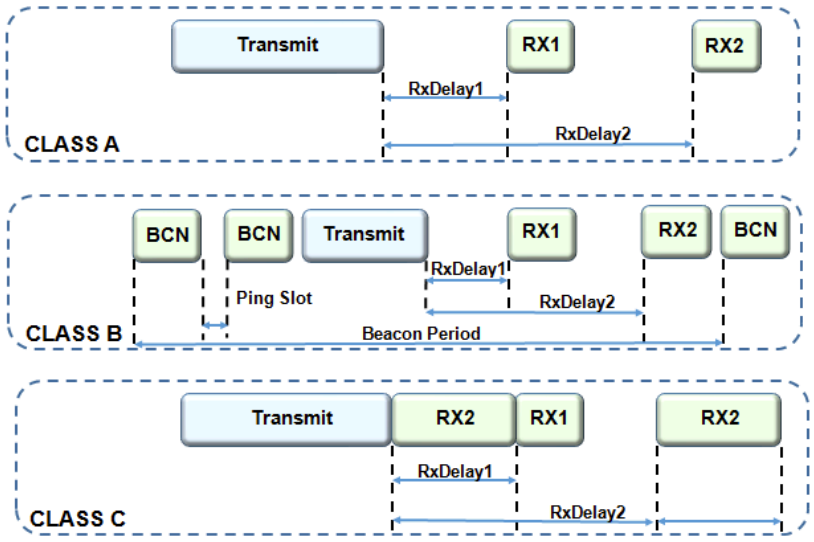


Figura 6 Tipos de dispositivos LoRaWAN [8]

2.4.2.4. The Thing Network

The Things Stack (TTS) es una pila de red LoRaWAN de código abierto adecuada para redes públicas y privadas grandes, globales y distribuidas geográficamente, así como para redes más pequeñas. La arquitectura sigue el modelo de referencia de red LoRaWAN para el cumplimiento de estándares y la interoperabilidad.

The Things Network (TTN) [12], es una red de IoT global, descentralizada, gratuita, abierta y en la que cualquier persona puede aportar sus conocimientos acerca de IoT con la finalidad de mejorar cada vez más la plataforma. El objetivo es facilitar el diseño y la creación de redes, para lo cual daba soporte a los gateways distribuidos por el mundo y además ofrecía la infraestructura de back-end a sus usuarios, es decir, una implementación de las funcionalidades del servidor de red y del servidor de aplicación. Este backend se enfrenta a duplicidades de mensajes, gestiona la transmisión de mensajes descendentes, gestiona integraciones con plataformas, como HTTP o MQTT y ofrece una serie de APIs en diferentes lenguajes de programación como Java, Node.js o Node-Red. En la figura 7 se muestra la arquitectura de TTN, donde los dispositivos IoT se conecta vía radio a los Gateways y estos a los servidores de red a través de redes de banda ancha. Los servidores de red pueden estar conectado a varios servidores de aplicaciones.

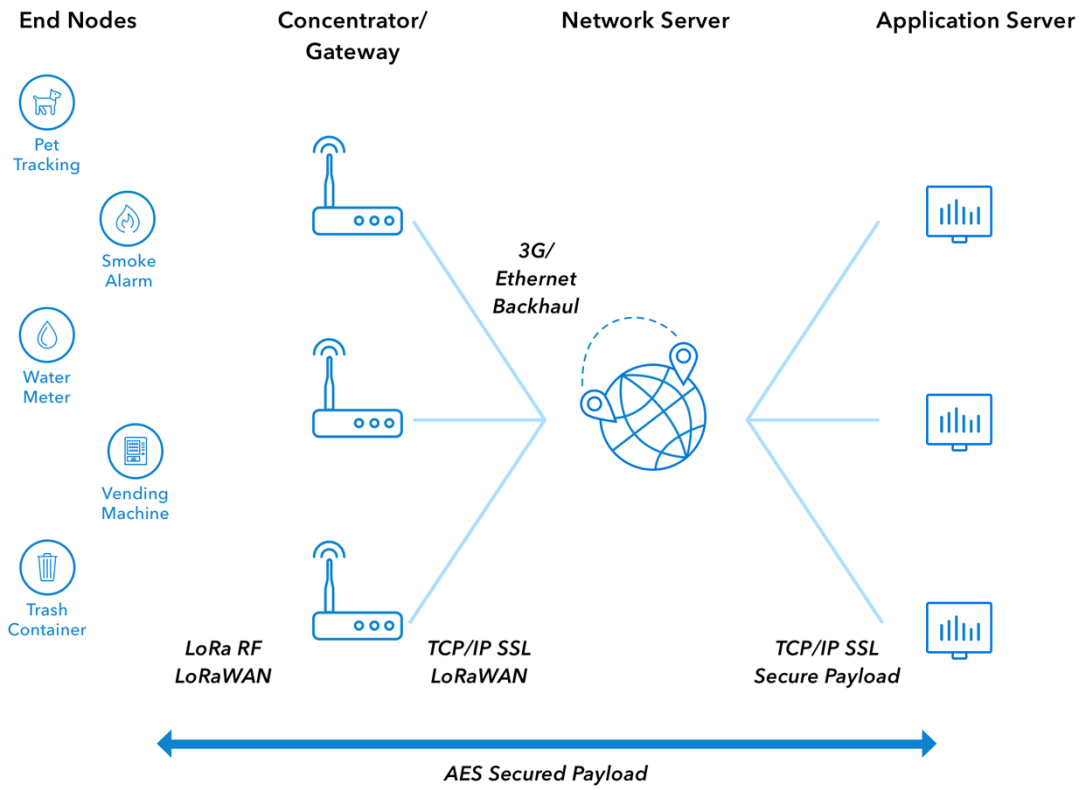


Figura 7 Red LoRaWAN TTN [12]

2.5. Sistemas de información geográfica SIG

Un SIG se define como un conjunto de métodos, herramientas y datos que están diseñados para actuar coordinada y lógicamente en la captura, almacenamiento, análisis, transformación y presentación de toda la información geográfica y sus atributos, con el fin de satisfacer múltiples propósitos [13].

Los SIG son una tecnología que permite gestionar y analizar la información espacial y surgió de la necesidad de disponer rápidamente de información, para resolver problemas y contestar a preguntas de modo inmediato.

Los sistemas de información geográfica utilizan la información geográfica (base de datos georreferenciada) almacenada en un sistema de información puede servir de soporte para la toma de decisiones y para ayudar en la planificación.

La razón fundamental para utilizar un SIG es la gestión de información espacial. El sistema permite separar la información en diferentes capas temáticas y las almacena independientemente, permitiendo trabajar con ellas de manera rápida y sencilla, y facilitando la posibilidad de relacionar la información existente a través de la topología de los objetos, con el fin de generar otra nueva que no podríamos obtener de otra forma.

2.5.1. Open Street Map

Open Street Map (OSM) [14] es un proyecto que proporciona datos de mapas para sitios web, aplicaciones móviles y dispositivos hardware. OSM lo crea una comunidad de colaboradores que aportan y mantienen los datos sobre puntos de interés (POI, Point Of Interest) a lo largo de todo el mundo y el objetivo es la creación de mapas libres y editables. Normalmente, los usuarios suelen subir sus coordenadas desde el GPS para crear localizaciones nuevas o para corregir datos vectoriales ya existentes.
Lo mapas pueden usarse libremente para cualquier propósito.

2.5.1.1. Formato de los datos

OpenStreetMap utiliza una estructura de datos topológica. Los datos se almacenan usando el estándar WGS84, formado por latitud y longitud de proyección de Mercator. Los elementos básicos de la cartografía OSM son los listados a continuación.

- Nodos: son puntos que recogen una posición geográfica dada.

- Vías (ways): son una lista de nodos que representa una polilínea o polígono.

- Relaciones: son grupos de nodos, caminos y otras relaciones a las que se pueden asignar determinadas propiedades.

- Etiquetas (tags): se pueden asignar a nodos, caminos o relaciones y constan de una clave (key) y de un valor (ej. highway=trunk).

Para obtener datos directamente en formatos SIG, desde OpenStreetMap es posible descargar los datos en formato OpenStreetMap XML Data (.osm). Desde la pestaña Export, se selecciona la zona a exportar, así como el formato de salida Google Maps.

2.5.2. Base de datos georreferenciada

Una base de datos georreferenciada, en su forma más básica, viene a ser una base de datos que tiene las coordenadas de sus elementos en uno de sus campos. Se pueden georreferenciar tres tipos básicos de elementos:

- Elementos puntuales
- Elementos lineales
- Elementos poligonales

También tenemos tipos compuestos, que son variantes de los anteriores: multipunto, multilínea o multipolígono, o geometrías que den cabidas a estos 3 elementos conjuntamente dentro de un mismo campo de la base de datos. En este caso, como no estamos hablando de una base de datos espacial, podríamos almacenar estos elementos en un campo de texto, con un formato tipo GEOJSON o WKT (Well Known Text).

2.5.3. Sistemas de gestión de bases de datos

Existen varias opciones en el mercado para el manejo de base de datos, en la tabla 3 se muestra una comparativa con los pros y contra de cada una, para este trabajo se ha optado por usar mongoDB, en la cual no es necesario cargar ningún módulo adicional para el manejo de datos geoespaciales.

Base de datos	Beneficios	Desventajas
PostgreSQL	Puede manejar grandes conjuntos de datos. La extensión PostGIS permite el uso de extensiones geográficas	Es necesario instalar un servidor de base de datos, con una sobrecarga administrativa asociada
MySQL	Puede manejar grandes conjuntos de datos	No dispone de extensiones geográficas. Es necesario instalar un servidor de base de datos, con una sobrecarga administrativa asociada
SQLite	Pequeña, no necesita servidor de base de datos	Puede tener problemas con grandes conjuntos de datos
MongoDB	Se pueden realizar consultas geoespaciales de forma nativa	
Hadoop/ Hive	Puede manejar grande conjunto de datos. Extensión disponible para consultas geoespaciales.	Es necesario instalar un cluster Hadoop, con una carga administrativa asociada

Tabla 3 Comparativa bases de datos georrefenciadas [40]

2.5.4. MongoDB

Para este trabajo se ha decidido escoger MongoDB, entre otras razones, como se puede observar en la tabla 2 no es necesario instalar módulos adicionales para manejar datos geoespaciales como es necesario hacer en el resto de las bases de datos. Además, los ingenieros de MongoDB han desarrollado el controlador de Python Pymongo [29], que nos facilita la integración de los ficheros de datos de OSM [30] en las bases de datos y en el desarrollo de la aplicación desarrollada en este trabajo.

MongoDB es una base de datos no-sql orientada a documentos. Es decir, en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección pueden tener esquemas diferentes y capaz de almacenar los datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse con el tiempo

Gracias a MongoDB [15], si nuestra aplicación almacena las coordenadas geográficas (longitud, latitud), podremos realizar búsquedas de información en base a su localización geográfica. Nos permite realizar los siguientes tipos de consultas basadas en el posicionamiento:

- Consultas de proximidad, en donde obtendremos los documentos ordenados por proximidad (de más cercano a más distante) a un punto geográfico.

- Consultas de acotadas, en donde obtendremos los documentos que están dentro de un área (rectángulo, círculo o polígono).

3. Diseño de sistema de alquiler

3.1. Requisitos

El objetivo del sistema de alquiler de bicicletas (BSS) que se propone en este trabajo está orientado a fomentar el uso del sistema basándose en el buen comportamiento durante el trayecto a través de un descuento en el coste del alquiler con el análisis de los datos recibidos durante el uso.

3.1.1. Bicicleta

La bicicleta debe tener un dispositivo IoT con capacidad de transmisión para la red LoRaWAN y que este formado por un receptor GPS, un velocímetro y acelerómetro, todo alimentado por una batería que se cargará con una dinamo instalada en la bicicleta.

En las bicicletas es necesario optimizar el consumo de energía y es también posible recolectar energía durante los trayectos en el articulo [13] se aconseja usar la energía cinética, usando transductores de energía de vibración electromagnética para la carga de baterías, en este articulo se propone el diagrama de bloque que se muestra en la figura 8. Como se puede observar el dispositivo dispone de una batería que se carga usando la energía cinética a través del USB de entrada. Esta unidad es capaz de suministrar la energía tanto al módulo de transmisión, que en el caso de este diseño estaría formado por el módulo de radio LoRa y el módulo GPS.

Por otro lado, en el caso del diagrama de bloques se suministra energía al módulo Bluetooth (BLE, Bluetooth Low Energy), que este caso se usa para comunicarse con el móvil del usuario, así como los sensores externos y el microcontrolador del sistema (MCU). En el caso del sistema que se está diseñando no se ha incluido el módulo BLE y los sensores instalados en la bicicleta serían el velocímetro, un detector de caídas o botón de alarma.

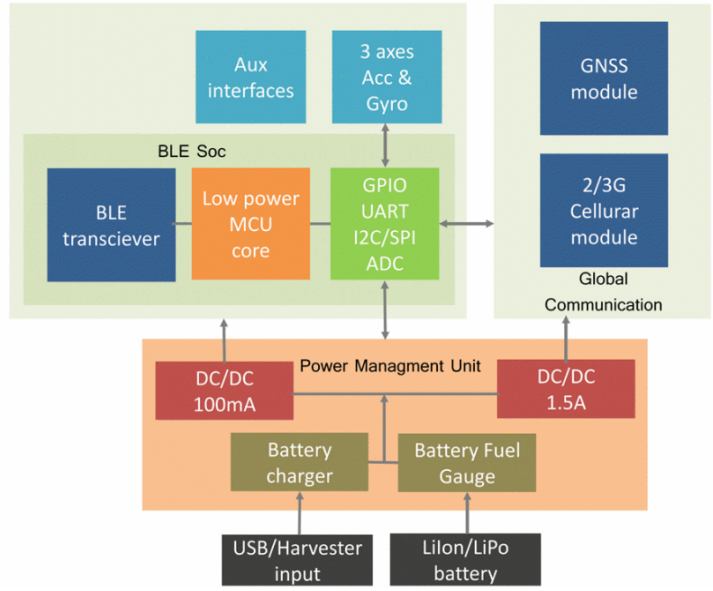


Figura 8 Esquema de bloques IoT [13]

3.1.2. Red LoRaWAN

La red de comunicación estaría formada por los siguientes elementos:

- Gateways: distribuido por la ciudad, sería los equipos que enlazarían vía radio con los dispositivos IoT y vía enlaces de alta capacidad con los servidores de red. Estos elementos son capaces de dar cobertura a zonas de entre 2 y 5 km en zonas urbanas y hasta 40 km en zonas rulares.

  En la figura 9 se muestra el ejemplo de una distribución de los Gateways suponiendo el peor caso, es decir, una cobertura de 2 km, en este caso con 5 gateways se podría dar cobertura a una ciudad de unos 100 km^2 [41].

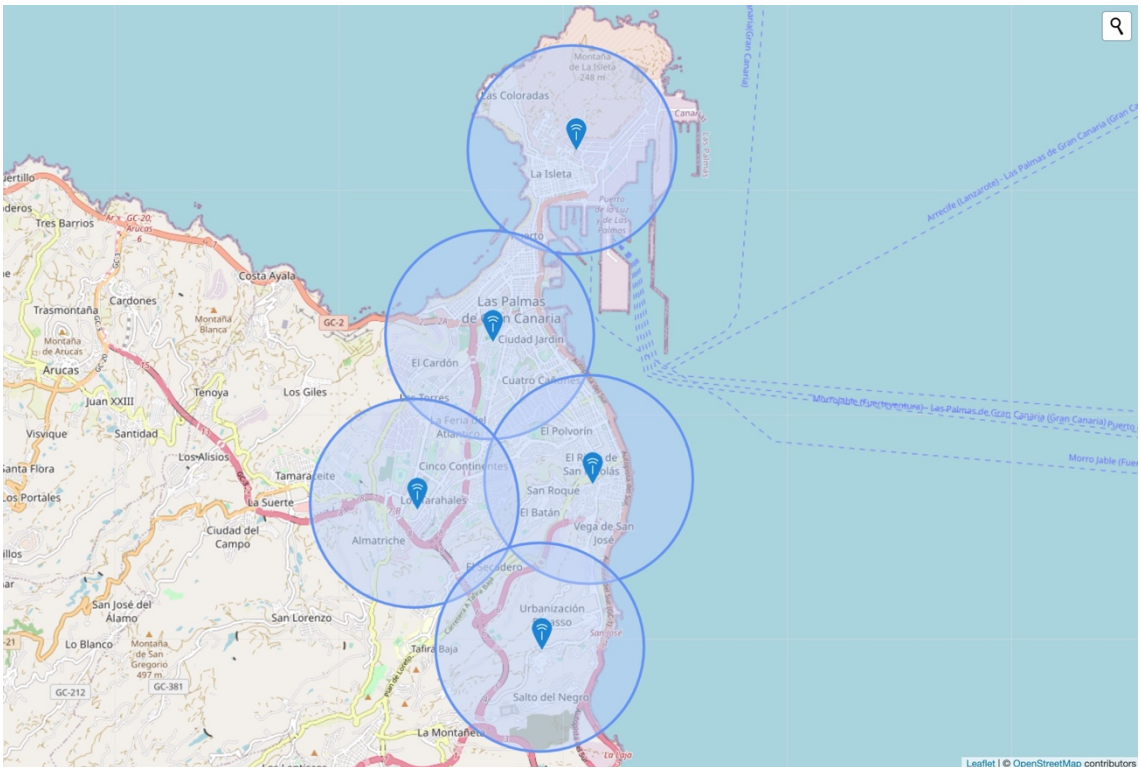


Figura 9 Gateways y cobertura

- Servidores de red. Su función es el control de los Gateways, así como los dispositivos conectados incluyendo el proceso de activación del dispositivo en la red. También reenrutar los mensajes procedentes/hacia los dispositivos finales o los servidores de aplicaciones.

- Servidores de aplicaciones. Las aplicaciones del sistema estarán conectado a la red LoRaWan, recibirá los mensajes procedentes de los dispositivos finales y generará los mensajes a enviar. Dispondrá de la base de datos de usuarios, donde se guardará los datos asociados al usuario, es decir, identificación e historial de alquileres. Por otro lado, tendrá otra base de datos con los datos de posición de los cruces asociados a las calles. Este servidor una vez finalizado el alquiler

procederá a conectarse con la pasarela de pago para proceder con el cobro del trayecto según los datos de comportamiento del usuario.

3.2. Procesos

Los requisitos anteriormente presentados hacen referencia a los siguientes procesos que se realizan en el sistema. Estos procesos se muestran en la figura 10, donde empezamos seleccionando la bicicleta, durante el trayecto obtenemos los datos del comportamiento según los datos obtenidos durante el trayecto y al final se procederá al cobro del trayecto en base al comportamiento y el tiempo de uso

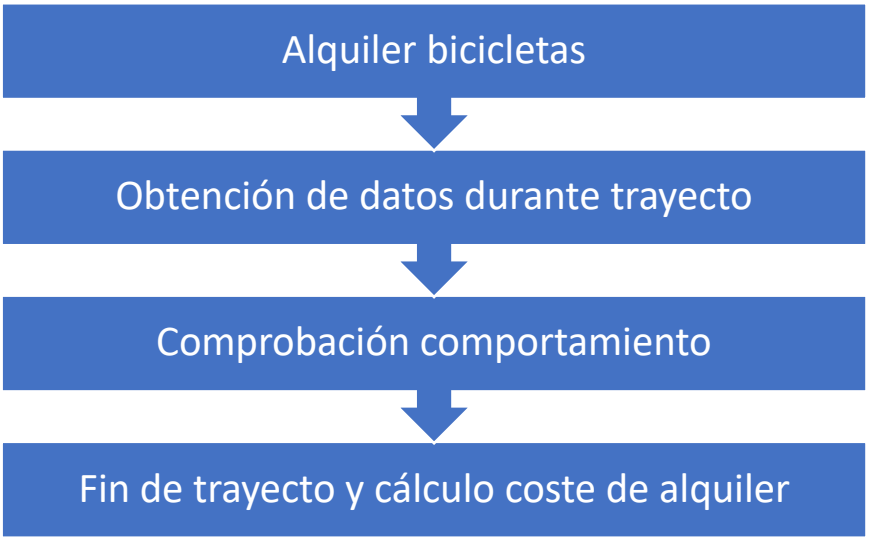


Figura 10 Procesos del BSS

3.2.1. Alquiler bicicletas

En este proceso el usuario selecciona la bicicleta que desea alquilar a través del móvil introduciendo el código de identificación de la bicicleta. La aplicación recibirá la petición, comprobará que el usuario está dado de alta y que la bicicleta está disponible. Si es así, enviará un código de desbloqueo al móvil del usuario. El usuario introducirá dicho código en la bicicleta y se procede al envío del código con la petición de desbloqueo al servidor de aplicaciones, si es correcto se asocia la bicicleta al usuario junto con la hora y el código de la estación donde se encuentra y se iniciará un registro en la base de datos del trayecto incluyendo los datos de hora, usuario, bicicleta y estación de inicio. Además, enviará la señal del desbloqueo a la bicicleta y el usuario podrá iniciar el trayecto.

En la figura 11 corresponde al diagrama de bloques de este proceso, donde se observa la interacción entre el usuario y el sistema a la hora de comenzar con el alquiler de la bicicleta.

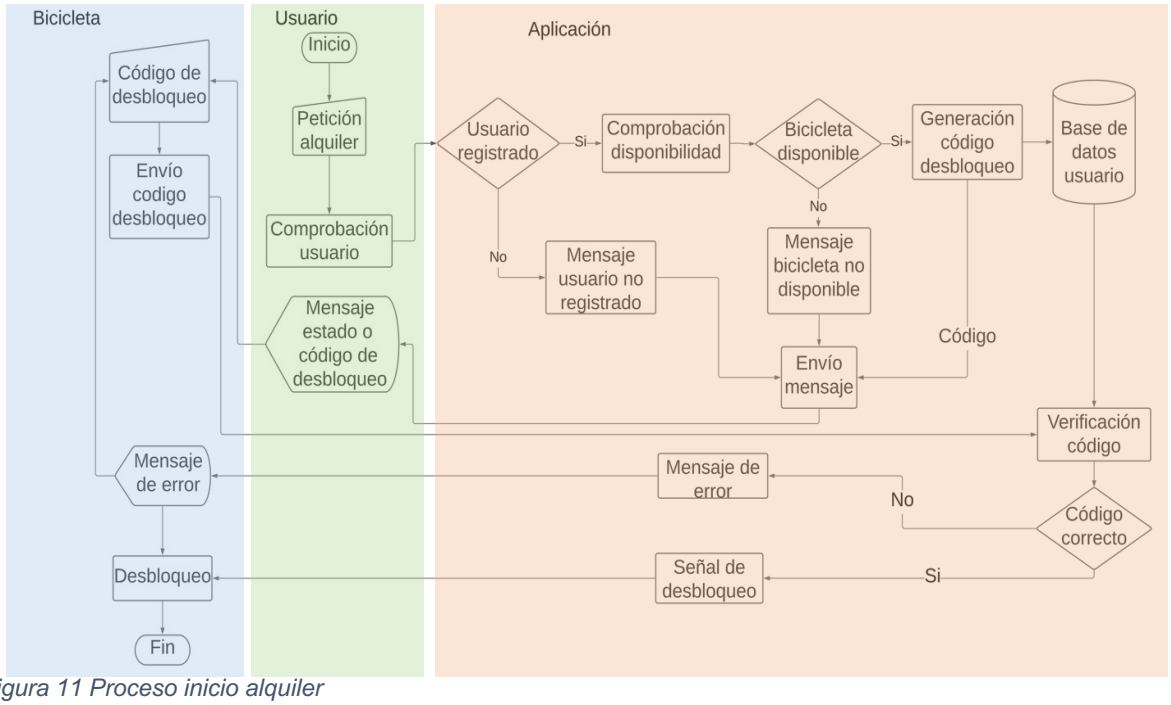


Figura 11 Proceso inicio alquiler

3.2.2. Obtención de datos

Al comienzo del trayecto y cuando ya este en movimiento la bicicleta enviará la posición obtenida al servidor donde se encuentra la aplicación. Dicha aplicación busca la vía por la que está circulando y obtendrá la lista de los cruces señalizados, de que tipo son (STOP, ceda el paso, semáforo) y las estaciones próximas. Enviará dichos datos a la bicicleta, además se guardará en la base de datos el registro del trayecto la vía por la que está circulando.

La bicicleta no volverá a enviar la posición a no se que cambie de vía. La forma que la bicicleta sabrá que se ha cambiado de vía es cuando se esté alejando de todos los cruces recibidos, en este caso enviará la nueva posición a la aplicación para obtener los siguientes cruces.

3.2.3. Comprobación comportamiento

Con los datos de la posición de los cruces cercanos, el servicio correspondiente instalado en la bicicleta buscará el cruce más cercano al cual se este acercando y si la distancia es menor de 20 metros comprueba la aceleración, junto con la velocidad hasta que pase el cruce. En el caso del STOP, si reduce la velocidad hasta pararse, si respeta los cruces entonces sumará un descuento del precio, en caso contrario no obtendrá descuento. En el caso de un Ceda el Paso, obtendrá descuento cuando reduzca la velocidad, aunque no hace falta que pare.

Cada vez que la bicicleta envíe la posición al servidor también enviará el valor del descuento que ha acumulado en el tramo de la vía correspondiente. Si en la ciudad existiera un sistema de información de semáforo [20], la bicicleta podría obtener el estado y realizar el descuento si no se salta el semáforo en rojo.

En la figura 12 observamos el diagrama de bloques de estos dos procesos, donde en el servidor de aplicaciones existen dos bases de datos, una con los datos de usuario donde se almacenan los datos del trayecto y otra con los datos de longitud y latitud de las calles de la ciudad y la posición de los cruces de cada vía. En la bicicleta el dispositivo IoT que tiene instalado realiza continuamente la comprobación del trayecto y cada vez que cambia de vía realiza una petición de la lista de nuevos cruces, también estudia el comportamiento y de esta manera realizar el descuento correspondiente.

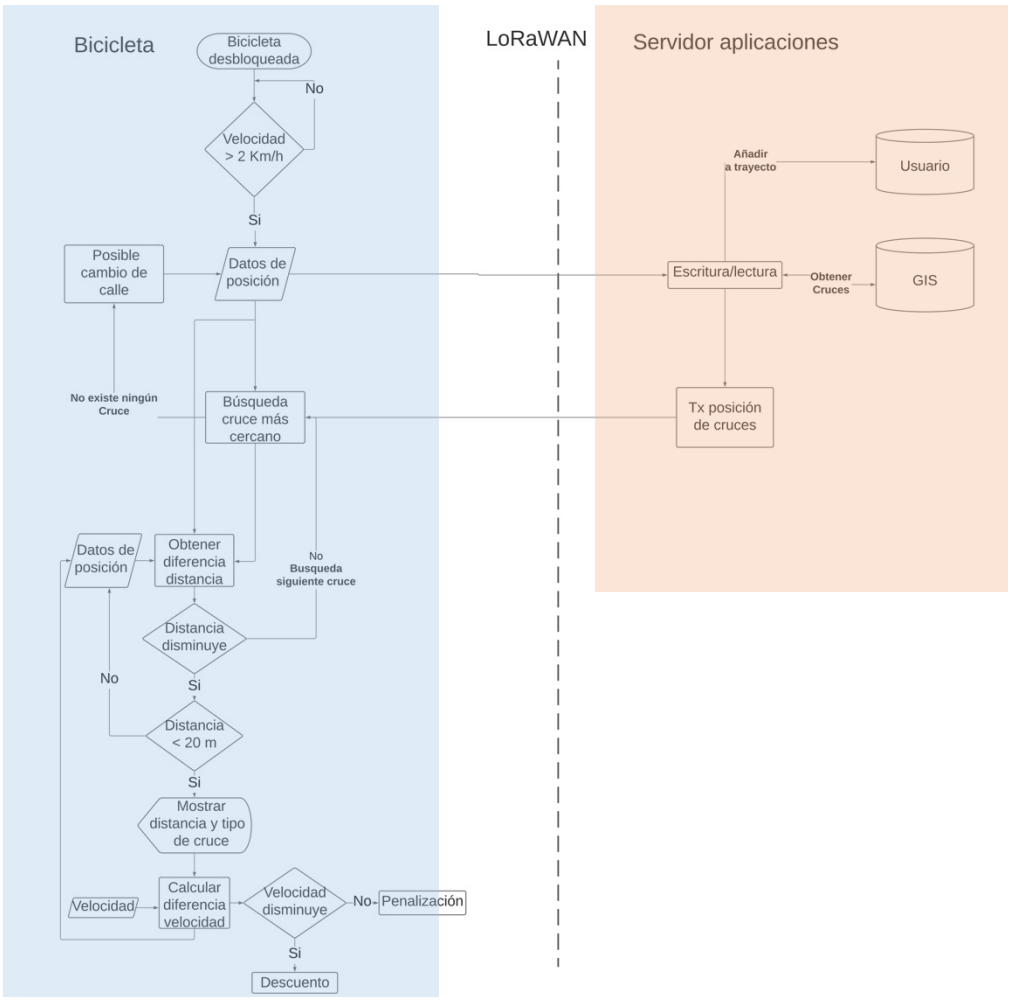


Figura 12 Procesos obtención de datos y comportamiento

3.2.4. Fin de trayecto

Cuando el usuario esté cerca de una estación de estacionamiento se ofrecerá dicha información al usuario, en el caso que el usuario decidiera finalizar el trayecto en el momento en que llega al estacionamiento más cercano hace la petición de bloqueo a través de la bicicleta, la bicicleta enviaría el fin de trayecto a la aplicación. La aplicación calcularía el coste del trayecto a partir del tiempo y de los descuentos acumulado e informaría al usuario a través del móvil. La aplicación enviaría la señal de bloqueo a la bicicleta, finalizaría el registro del trayecto y marcaría la bicicleta como disponible en la estación de entrega.

3.3. Alarmas

El sistema debe ser capaz de transmitir alarmas sobre el estado de los diferentes módulos que lo componen, así como las propias del funcionamiento de la aplicación diseñada.

Las alarmas estarían divididas en los siguientes tipos:

- Sistema. Serian las alarmas relacionadas con el hardware del dispositivo IoT. Un listado de alarmas sería:
    - Estado defectuoso de las baterías.
    - Malfuncionamiento del cargador.
    - Malfuncionamiento de los sensores.
    - Señal transmitida por debajo de los limites.
    - Malfuncionamiento del módulo GPS.
    - Alarmas relacionadas con MCU.

- Seguridad. Estaría formada por alarmas de mayor severidad, que estaría relacionada con intento de robo o problemas con el usuario durante el trayecto. Un posible listado sería.
    - Intento de desbloqueo no autorizado de la bicicleta.
    - Detección de movimiento de la bicicleta no autorizado.
    - Detección de caída durante el trayecto.
    - Activación de emergencia por parte del usuario

Las alarmas estarían clasificadas por severidad, siendo la de mayor severidad las relacionadas con la seguridad, siendo necesario una intervención inmediata, y las relacionadas con el mantenimiento tendrían una severidad menor, no siendo necesaria una intervención inmediata.

4. Diseño red LoRaWAN

A la hora de diseñar la red de comunicación entre los dispositivos IoT y los servidores de red de LoRaWAN, se debe tener en cuenta las limitaciones de uso impuestas en la regulación EN300.220 de la ETSI (European Telecommunications Standards Institute), que fija el tiempo máximo de transmisión de los dispositivos que usen la banda no licenciada ISM (Industrial, Scientific and Medical), a 1% del tiempo total, es decir, que en una hora los dispositivos pueden ocupar el interfaz aire como máximo 36 segundos [21]. Esta limitación se aplica tanto a los dispositivos instalados en las bicicletas como a los Gateways.

4.1. Estructura interfaz radio

La restricción del uso del interfaz aire limita la cantidad de información que se puede transmitir y recibir en el sistema que estamos diseñando. Para ello, debemos de conocer la estructura del interfaz aire que según las especificaciones de LoRaWAN [22], ya que, el tiempo de ocupación no solo incluye la carga útil sino también la información que forma parte de la cabecera de los paquetes enviados. Esta cabecera incluye lo campos que se muestren en la figura 13, donde los bytes que se tienen en cuenta en el cálculo son los forman parte del campo PHYPayload, este campo tiene tres tipos posibles:

- MACPayload: Carga útil.
- Join Request: Petición de unión al servidor.
- Join Accept: Aceptación de unión al servidor.

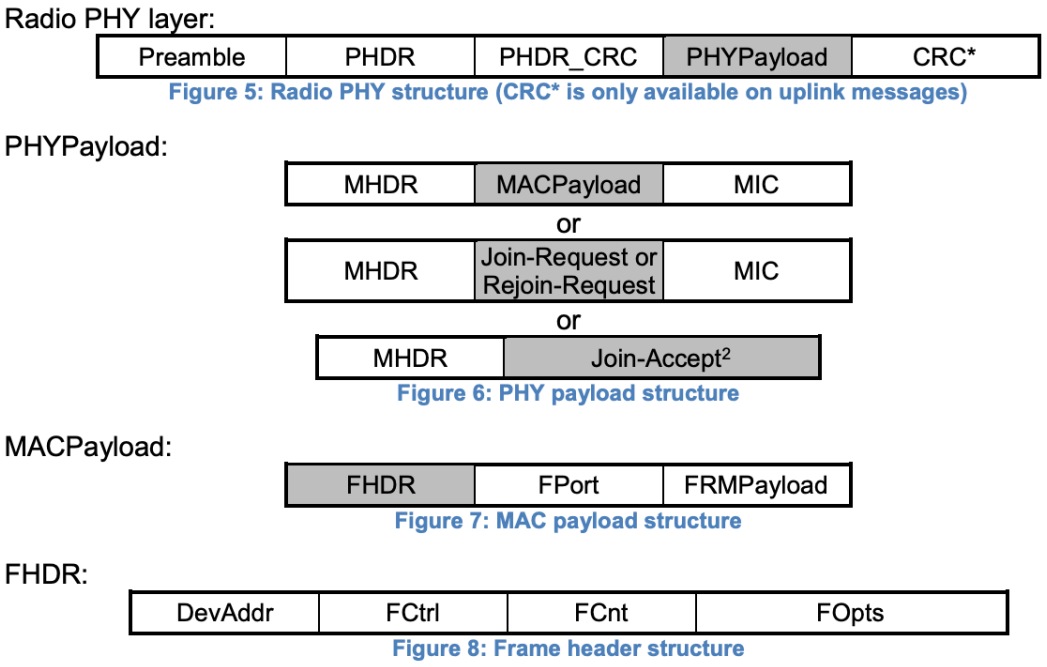


Figura 13 Interfaz radio [22]

La carga útil utiliza el campo MACPayload, el cual su vez está formado por los campos que aparece en la tabla 4, donde DevAddr correspondería a la dirección de red asignada al dispositivo instalado en la bicicleta y Fport es el puerto usado para transmitir y recibir la carga útil en el dispositivo final

Campo	Tamaño (bytes)	Descripción
MHDR	1	MAC Header
DevAddr	4	address of the end-device
FCtrl	1	Frame control octet
FCnt	2	Frame counter
Fport	1	Port field
MIC	4	message integrity code

Tabla 4 Campos Payload [22]

Es decir, a los bytes de carga útil incluida en el campo FRMPayload hay que sumarle los 13 bytes mostrado en la tabla 4 que corresponde a la cabecera del mensaje transmitido y que se tienen en cuenta en la limitación de ocupación del 1%.

4.2. Diseño paquete de carga útil

Los principales datos que se va a transmitir son los datos de longitud y latitud, sabiendo que la longitud puede variar entre -180º y 180^0 y los datos de latitud entre -90^0 y 90^0, además sabiendo que los equipos de recepción de GPS tienen una precisión de 7 decimales, por lo que, en el peor caso, el dato a trasmitir en la carga útil será el dato de la longitud que puede estar formada por una cifra menor de 180 con 7 decimales.

A la hora de codificar los datos tenemos presente los diferentes tipos de datos disponibles, tal y como se representa en la tabla 5

Tipos de datos	Tamaño en memoria	Rango de valores
byte	1 byte / 8 bits	0..255
char (con signo)	1 byte / 8 bits	(7 bits) -128..127
word	2 bytes / 16 bits	0.. 65.535
int (con signo)	2 bytes / 16 bits	(15 bits) -32.768.. 32.767
unsignedlong	4 bytes / 32 bits	0.. 4.294.967.295
long	4 bytes / 32 bits	(31 bits) -2,147,483,648..2,147,483,647

Tabla 5 Tipos de datos [24]

En nuestro caso, con una previa eliminación de lo decimales se puede usar los datos de tipo long. Es decir, para cada dato de posición de longitud y latitud tendríamos 8 bytes de payload.

4.2.1. Mensaje Uplink

A continuación, se va a definir el formato de los mensajes que se van a enviar desde la bicicleta a la aplicación, a través, de la red LoRaWAN. Para este diseño se ha decidido que en el Uplink habrá cuatro tipos de mensajes.

- Inicio. Al inicio del alquiler se enviará el código recibido por el usuario a través del móvil.

- Trayecto. Durante el trayecto se transmitirá la posición del usuario además de los datos de comportamiento y velocidad media.

- Finalización. Al finalizar el alquiler se enviará la petición del bloqueo de la bicicleta.

- Alarmas. Existe una cuarta opción en la cual se enviará el código de la alarma cuando se active, si es critica se enviará en cualquier momento y si no se enviará cuando la bicicleta este estacionada.

Para controlar estos datos se propone usar 2 bytes con la siguiente estructura de datos, donde los dos primeros bits (b_1b_0) se utilizar para identificar el tipo de mensaje que se enviará en los bits de 2 al 15.

- Comienzo alquiler

$b_{15}…b_2$	$b_1\,b_0$
Código desbloqueo	00

- Trayecto

$b_{15}…b_9$	$b_8…b_2$	$b_1\,b_0$
Puntuación	Velocidad	01

- Finalización Alquiler

$b_{15}…b_2$	$b_1\,b_0$
xx	10

- Alarmas

$b_{15}…b_2$	$b_1\,b_0$
Código alarma	11

Por lo que, el tamaño del paquete transmitido será de 10 Bytes, 8 Bytes para los datos de posicionamiento y 2 bytes de control.

4.2.2. Mensaje de Downlink

En este caso la información que se enviará la bicicleta consistirá en la posición de los cruces, según el tipo y la posición de la estación de estacionamiento más próxima. En este caso se usará 2 bytes de control donde cada 4 bits se indicará el número de puntos de interés (cruce o estación) POI con la siguiente estructura:

- Bits de 0 al 3: Número de Stop que se incluye la longitud y latitud en el mensaje de Downlink.

- Bits del 7 al 4: Número de Ceda el paso que se incluye la longitud y latitud en el mensaje de Downlink.

- Bits del 8 al 11: Número de semáforos que se incluye la longitud y latitud en el mensaje de Downlink.

- Bits del 12 al 15: Número de estaciones de estacionamiento cercanas que se incluye la longitud y latitud en el mensaje de Downlink.

La división gráfica de los bytes sería

$b_{15}…b_{12}$	$b_{11}…b_8$	$b_7…b_4$	$b_3…b_0$
Nº Estaciones	Nº de Semáforos	Nº de Cedas el paso	Nº de STOPs

Los datos de la longitud y latitud van a partir del bit 16 en el mismo orden descrito anteriormente.

Se fijará un máximo de 5 POI en el Downlink, por lo que el tamaño del payload será de 42 bytes, 40 bytes correspondiente a los bytes de longitud y latitud de los cruces y las estaciones de estacionamiento y los 2 bytes descritos.

4.3. Time of Air (ToA)

Una vez que conocemos el tamaño de los paquetes a enviar y recibir incluida la cabecera, en este apartado se procede a calcular el tiempo de ocupación del interfaz aire y de esta manera conocer cuantos mensajes por hora se podrá transmitir en el sistema diseñado. Para ello hay que tener las siguientes ecuaciones obtenidas del documento AN1200.13 "LoRa Modem Designer's Guide"[23].

Partiendo de la definición de la duración del símbolo, T_{sym}, que viene a ser el tiempo que se tarda en enviar 2^{SF} chips, donde SF (Spread Factor) es el número de bits transmitidos, por lo que, recordando que el ancho de banda (BW) define la velocidad de chip, tenemos que

$$T_{sym} = \frac{2^{SF}}{BW}$$

Tal y como se ha comentado en el apartado anterior el paquete está formado por los elementos de la figura 14

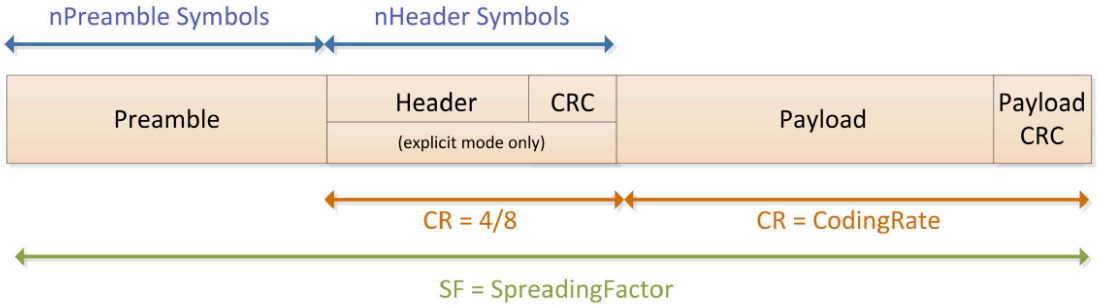


Figura 14 Payload [23]

Sabiendo que la duración del paquete Preamble viene dada por

$$T_{preamble} = (n_{preamble} + 4.25)T_{sym}$$

Donde n_{preamble} en el caso de LoRa es 8.

Del resto de elementos el número de símbolos viene dada por

$$payloadSymNb = 8 + \max\left(ceil\left(\frac{8PL - 4SF + 28 + 16 - 20H}{4(SF - 2DE)}\right)(CR + 4), 0\right)$$

Donde:

- PL es el número de bytes incluida la cabecera
- SF es el spread factor
- H = 0 si existe cabecera y 1 si no existiera
- DE = 1 si la optimización de baja tasa de datos estuviera activada
- CR coding rate (velocidad de codificación), que en el caso de LoRa va de 1 a 4.

Una vez obtenido el número de símbolo obtenemos que la duración de la carga es

$$T_{payload} = payloadSymNbT_{sym}$$

Y, por tanto, el ToA es

$$ToA = T_{preamble} + T_{payload}$$

En este punto del diseño, se procede a calcular el ToA para los diferentes SF y anchos de bandas (BW). Para realizar los cálculos se ha usado Matlab, en el anexo 1 se incluye el código creado. En la tabla 6 correspondiente a Uplink y la tabla 7 correspondiente al Downlink, aparte del tiempo de transmisión también tenemos el número de paquetes por hora (PL/hora) que puede transmitir el dispositivo instalado en las bicicletas y por tanto cada cuanto puede transmitir los datos (Sg.).

En el caso del Uplink, el payload sería los 10 bytes junto con los 13 bytes correspondiente a la cabecera, es decir, PL=23.

SF	BW(khz)	CR	T_{sym}(msg.)	$T_{preamble}$	payloadSymNb	$T_{payload}$	ToA	PL/hora	Sg.
7	125	1	1	12.5	48	49.1	61.7	583	6.1
8	125	1	2	25	43	88	113.1	318	11.3
9	125	1	4	50.2	38	155.6	205.8	174	20.7
10	125	1	8.2	100.4	33	270.3	370.7	97	37.1
11	125	1	16.4	200.7	38	622.6	823.3	43	83.7
12	125	1	32.76	401.4	33	1081.3	1482.8	24	150

Tabla 6 Cálculo ToA Uplink

En el caso del Downlink, el payload sería los 42 bytes junto con los 13 bytes correspondiente a la cabecera, es decir, PL= 55 bytes.

SF	BW(khz)	CR	T_{sym}(msg.)	$T_{preamble}$	payloadSymNb	$T_{payload}$	ToA	PL/hora	Sg.
7	125	1	1	12.5	93	95.2	107.8	334	10.8
8	125	1	2	25	83	170	195	184	20
9	125	1	4	50.2	73	299	349.2	103	35
10	125	1	8.2	100.4	68	557	657.4	54	66
11	125	1	16.4	200.7	73	1196	1396.7	25	144
12	125	1	32.76	401.4	63	2064.4	2465.8	14	257.1

Tabla 7 Cálculo ToA Downlink

4.4. Estructura base de datos geolocalización

En el servidor de aplicaciones se procede a instalar la base de datos Mongodb donde se importará los datos procedentes de Open Street Map (OSM), aprovechando la capacidad de MongoDB de realizar consultas de datos geoespaciales, donde se almacenan como objetos GEOSON o como pares de coordenadas. En este trabajo se va a utilizar los pares de coordenadas para calcular las distancias en un plano euclidiano, de esta manera, en nuestro caso dada la posición del dispositivo instalado en la bicicleta podemos conocer la posición de los cruces de la vía por la que se vaya circulando.

OSM utiliza el sistema de coordenadas WGS84 para almacenar la latitud y longitud de los elementos que forman parte del modelo de datos. Estos elementos son de tres tipos:

- Nodos. Representa un punto especifico en la superficie definido por su longitud y latitud. Cada nodo está formado por un número de identificación y un par de coordenadas.

- Vía. Es una lista ordenada de nodos que definen una polilínea y se usan para representar características lineales tales como calles.

- Relación. Es una estructura de datos multipropósito que permite relacionar varios elementos (nodos, vías).

Todos los elementos pueden tener etiquetas, las etiquetas de un elemento describen el significado del elemento. Esta etiqueta está formada por dos campos de texto de formato libre, siendo la clave y el valor.

En nuestro sistema, a partir de la posición geográfica buscará el elemento de tipo vía por la que circula la bicicleta y una vez conocida el id de la vía, se procederá a buscar los cruces que forman parte de esta vía a partir de los nodos que forman parte de la vía, así como las estaciones próximas, dichos datos se enviará a través de la red LoRaWAN al dispositivo.

En la base de datos un elemento de tipo vía, está formado por la lista de elemento que se encuentran en la vía, la longitud y latitud de dicha vía y las etiquetas correspondientes, en nuestro caso, la etiqueta que nos interesa es “highway”.

Los elementos de tipo nodo, esta formado por la longitud y latitud de dicho elemento y las etiqueta que describe el tipo de nodo, en nuestro caso las etiquetas que nos interesan son:

- Semáforos. la etiqueta es “highway” con valor “traffic_signals”.

- Stop. la etiqueta es “highway” con valor “Stop”.

- Ceda el paso. la etiqueta es “highway” con valor “give_way”.

- Estacionamiento. la etiqueta es “amenity” con valor “bicycle_rental”.

5. Arquitectura del sistema de simulación

En este capitulo se define la arquitectura diseñada para comprobar el funcionamiento de la aplicación IoT diseñada una red LoRaWAN. La simulación va a permitir analizar los mensajes entre el servidor de aplicaciones y los dispositivos IoT que pasan a través de los servidores de red de TTN.

5.1. Arquitectura de red simulación

En nuestro caso, como podemos ver en la Figura 15, usaremos el simulador de dispositivos IoT que usa el sistema operativo Mbed [25], dicho simulador incluye el nodo final y el Gateway, simulando el interfaz radio entre estos dos elementos. En la misma figura se aprecia que el simulador se conecta a los servidores de red de TTN a través de Internet. Por último, se desarrolla el servidor que aplicaciones que incluye la base de datos georreferenciada. Para la conexión entre el servidor de red y el servidor de aplicaciones se definirá un interfaz HTTP que permite intercambiar los mensajes de Uplink y Downlink de los dispositivos IoT.

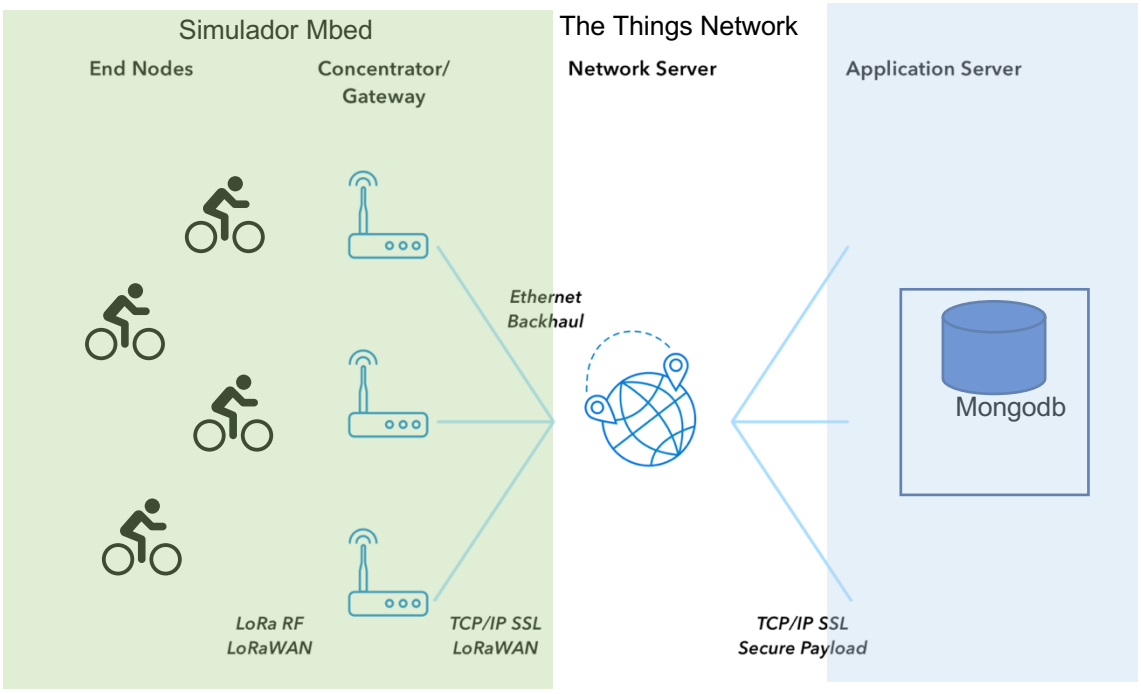


Figura 15 Arquitectura simulación

5.1.1. Simulador Mbed

Mbed es un sistema SO para dispositivos conectados a internet basado en controladores ARM Cortex-M de 32 bits. Es decir, es un sistema operativo para dispositivo IoT de código abierto. Este SO se desarrolla junto con ARM.

Dentro de MBED existen varias herramientas para el desarrollo de aplicación y su compilación en el hardware correspondiente, pero a la hora de probar el desarrollo la compilación es lenta, por lo que, dificulta la realización de pruebas. Por ello Mbed Labs [26] ha creado un simulador basado en la versión del protocolo LoRaWAN MAC V1.0.1, que permite probar los desarrollos rápidamente sin la necesidad de compilar la aplicación en el hardware cada vez que se realiza una modificación.

5.1.1.1. Web del simulador Mbed

Mbed Simulator no simula completamente un dispositivo Cortex-M sino que realizar una compilación cruzada de Mbed Operative System (OS) 5 usando Emscripten. Emscripten es un compilador que coge un proyecto de C++ y genera como salida un archivo en JavaScript que puede ser procesado en navegadores web.

En este trabajo, se usa la API LoRa del sistema operativo, como se observa en la figura 16 y 17 el simulador sustituye la capa de radio LoRa por una simulación, y cuando la pila de LoRaWAN quiere enviar un paquete vía radio el simulador intercepta el paquete cifrado, junto con los datos radio (data rate, canal, frecuencia) y lo reenvía directamente al servidor TTN, lo mismo ocurre con los paquetes recibidos, es decir, los paquetes junto con la información radio es reenviada directamente a la pila de protocolo de LoRaWAN del sistema operativo para ser procesada.

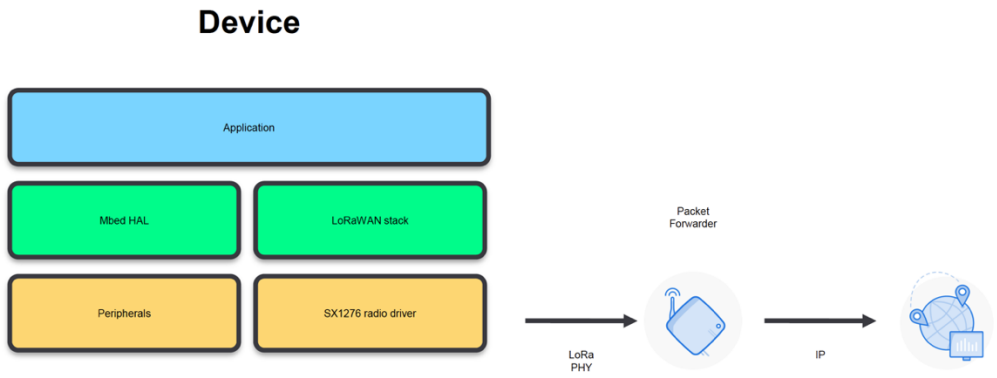


Figura 16 Pila de protocolo Dispositivo IoT con SO MBed [41]

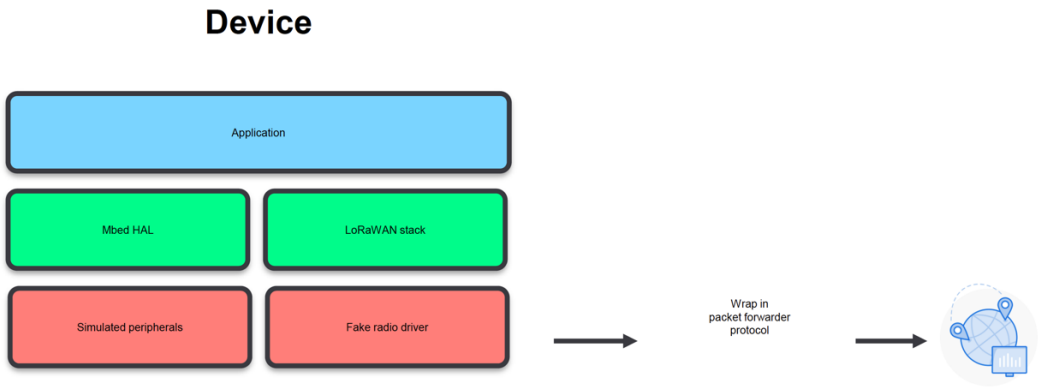


Figura 17 Pila de protocolo Dispositivo IoT con simulador [41]

En la figura 18, se muestra la web del simulador, en el lado izquierdo encontramos el editor del código en C++ de la aplicación que se pretende simular, donde se puede hacer los cambios que desee y compilar haciendo clic en Run.

En la parte derecha superior se incluye el diagrama del microcontrolador con el sistema operativo Mbed, con una serie de puertos de entrada y salida, así como con un botón que usaremos en nuestra simulación para iniciar la codificación del paquete a transmitir y su posterior envío.

Por último, en la parte inferior nos encontramos con la salida de los eventos de la aplicación, que incluye los mensajes de la pila de LoRaWAN (conexión, paquetes a enviar y paquetes recibidos) así como los eventos del código desarrollado.

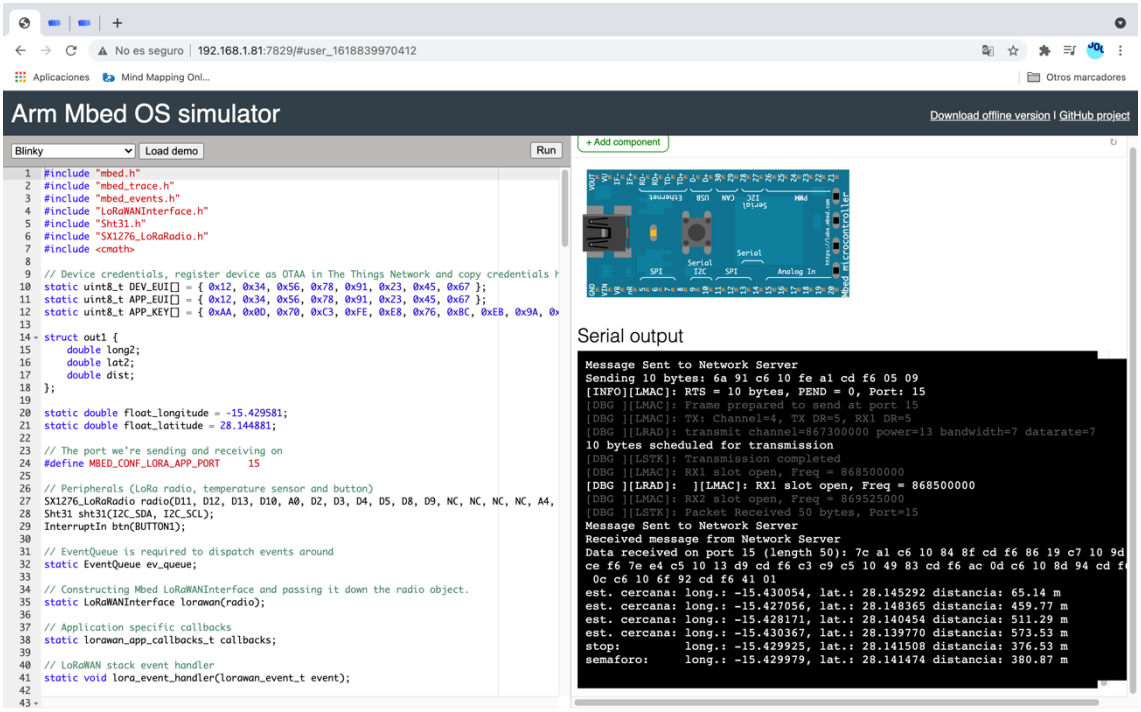


Figura 18 Simulación MBED

5.1.1.2. Emulador Gateway

El simulador incluye un proxy de red que implementa dos funciones que no pueden ser simuladas por el navegador:

- La API NetworkInterface, que se corresponde con la interfaz de red IP genérica para Mbed OS. Que permite abrir sockets UPD y TCP.

- La API LoRaWAN. Los mensajes de la pila LoRaWAN se envían al servidor de red de TTN utilizando el protocolo Semtech UDP, que no está permitido desde el navegador.

Este proxy de red permite abrir sockets UDP y TCP sin procesar y transmitir mensajes LoRaWan.

Cuando se define el Gateway en los servidores de TTN hay que tener en cuenta que el simulador usará como identificador del Gateway la MAC del equipo donde se encuentre instalado el simulador y que hay que definirlo como “legacy packet forwarder”, indicándole que estamos usando el protocolo Semtech UDP.

En la figura 19 se muestra los eventos del Gateway simulado, donde se observa los paquetes enviados y recibidos.

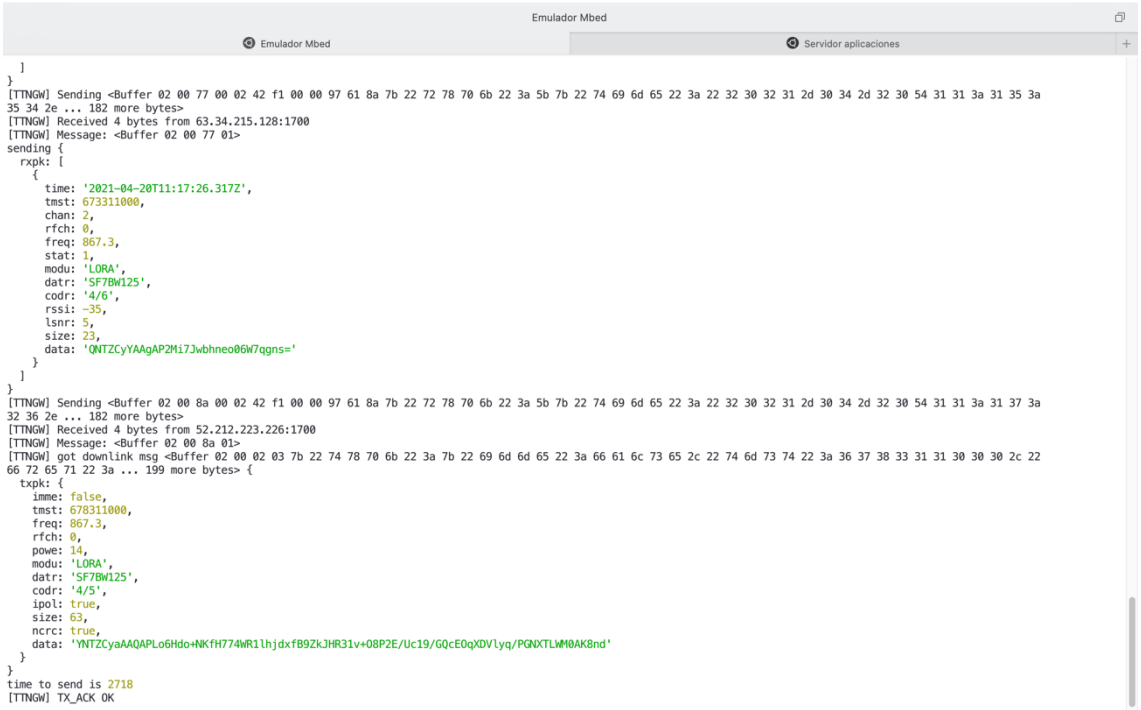


Figura 19 Log Gateways

5.1.2. Servidor de aplicaciones

En esta simulación, el servidor de aplicaciones estará formado por la base de datos georeferenciada Mongodb, con los datos de los nodos, vías y relaciones de la ciudad elegida para realizar esta simulación. También se instalará un servidor web con PHP que se usará para la comunicación con TTN. Para el desarrollo de la aplicación se usa el lenguaje de programación Python, al cual se añadirá el controlador Pymongo para comunicar Python y MongoDB. Además, para poder importar los datos de OSM se usará el paquete software Node-mongosm que a partir de los ficheros “.osm” de OSM, creará las colecciones correspondientes a los nodos, vías y relaciones y guardará las entradas en Mongodb.

5.2. Creación e instalación de aplicaciones

Una vez descrita la arquitectura de sistema que vamos a utilizar para simular la aplicación diseñada, es este apartado se va a describir los pasos seguidos para poder crear los diferentes elementos necesarios.

5.2.1. Instalación simulador Mbed

El simulador se va a instalar en una maquina virtual de Linux para ello usaremos la aplicación Oracle VM VirtualBox. En este software libre de virtualización y de código abierto [27], se crea una máquina virtual Linux usando la distribución Ubuntu server 20.10 de 64-bit, donde se asigna una memoria base de 1024 MB y un disco duro virtual de 50 GB de almacenamiento reservado dinámicamente, la conexión de red será de tipo puente con lo que la máquina virtual tendrá su propia IP en la red interna del laboratorio y podrá conectarse al router de la red de forma independiente. En la figura 20 se muestra dicha configuración

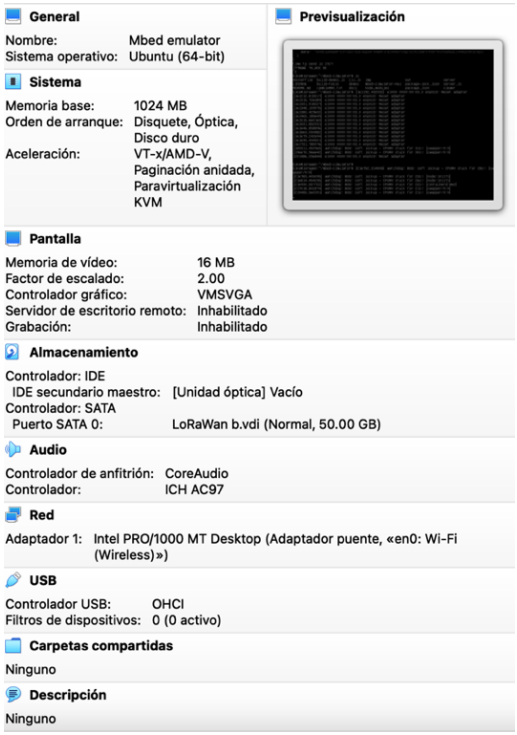


Figura 20 Maquina virtual simulador Mbed

Una vez creada la máquina virtual, se procede a la instalación de la versión local del simulador y del software necesario para su funcionamiento, tal y como se explica en el repositorio de GitHub del simulador [28], donde hay que tener en cuenta que el proxy local del simulado tiene definida la dirección de los servidores de red de TTN correspondiente a la versión 2. En la actualidad, los servidores de red se han actualizado a la versión 3 cambiando su dirección. Por lo que, es necesario actualizarlo en el código del simulador. Para ello, en la carpeta de la aplicación “/server/” editamos “launch-serv.js” y modificamos en la línea 20 en nombre del Host de la versión 2 por el de la versión 3, tal y como se muestra en las Figuras 21 y 22.

```
const { exists } = require('../build-tools/helpers');
const promisify = require('es6-promisify').promisify;
const udp = require('dgram');
const ttnGwClient = udp.createSocket('udp4');
const mac = require('getmac');
const timesyncServer = require('timesync/server');
const version = JSON.parse(fs.readFileSync(Path.join(__dirname, '..', 'package.json'), 'utf-8')).version;
const compression = require('compression');

const LORA_PORT = process.env.LORA_PORT || 1700;
const LORA_HOST = process.env.LORA_HOST || 'router.eu.thethings.network';

let startupTs = Date.now();
```

Figura 21 V2 proxy

```
const { exists } = require('../build-tools/helpers');
const promisify = require('es6-promisify').promisify;
const udp = require('dgram');
const ttnGwClient = udp.createSocket('udp4');
const mac = require('getmac');
const timesyncServer = require('timesync/server');
const version = JSON.parse(fs.readFileSync(Path.join(__dirname, '..', 'package.json'), 'utf-8')).version;
const compression = require('compression');

const LORA_PORT = process.env.LORA_PORT || 1700;
const LORA_HOST = process.env.LORA_HOST || 'eu1.cloud.thethings.network';

let startupTs = Date.now();
```

Figura 22 V3 proxy

Para poder usar el código desarrollado en la web del emulador, se crea el subdirectorio “/bss_lorawan/” debajo del subdirectorio “/demos/” del simulador, bajo este directorio el código de la aplicación se guarda en el fichero “main.cpp”. Además, para que sea visible en la web del simulador se debe editar el fichero “simulator.html” que se encuentra en el subdirectorio “viewer”, añadiendo la línea que se muestra en la Figura 23.

```
<section id="project">
  <section id="editor-container">
    <div id="editor-topbar">
      <select id="select-project">
        <option disabled>-- Basic demos --</option>
        <option selected name="blinky">Blinky</option>
        <option name="interrupts">Interrupts</option>
        <option name="events">Events</option>
        <option name="pwmout">PwmOut</option>
        <option name="busout">Blinky with BusOut</option>
        <option name="trace">Tracing</option>
        <option name="bd">Block Device</option>
        <option disabled>-- Peripherals --</option>
        <option name="lcd">LCD Display</option>
        <option name="touchscreen">Touch screen</option>
        <option name="temperature">Temperature / humidity</option>
        <option disabled>-- Networking --</option>
        <option name="tcpsocket">TCP Socket</option>
        <option name="tlssocket">TLS Socket</option>
        <option name="http">HTTP</option>
        <option name="https">HTTPS</option>
        <option name="coap">CoAP</option>
        <option name="ntp">Time over UDP</option>
        <option name="lorawan">LoRaWAN</option>
        <option name="lorawan-abp">LoRaWAN (ABP)</option>
        <option name="bss_lorawan">BSS </option>
      </select>
```

Figura 23 “Simulator.html”

Para que tengan efecto estos cambios es necesario actualizar el paquete NPM (node package manager) de la web del emulador.

```
$ npm run build-demos
```

E iniciar el servidor web con

```
$ node server.js
```

Una vez iniciado el servidor web nos conectamos al emulador abriendo la dirección IP de la máquina virtual y al puerto 7829 (http:/(ip maquina virtual):7829).

5.2.2. Servidor de aplicaciones

En este apartado se procede a crear el servidor de aplicaciones y los diferentes elementos que los conforman que son:

- Python y PIP. En el servidor se instala los lenguajes de programación que se van a usar para el desarrollo de la aplicación.

- Base de datos MongoDB y creación de base de datos. Se instala la base de datos donde se importarán los datos georrefenciados de la ciudad que se usa en la simulación.

- Servidor Web y PHP. Se instala los elementos necesarios para establecer la comunicación entre el servidor de red de TTN y el servidor de aplicaciones.

5.2.2.1. Instalación servidor

El servidor se instala en como una máquina virtual de Linux usando la aplicación Oracle VM VirtualBox, en la cual creamos una máquina virtual Linux usando la distribución Ubuntu server 20.10 de 64-bit. Al igual que la máquina virtual del emulador asignamos una memoria base de 1024 MB y un disco duro virtual de 50 GB de almacenamiento reservado dinámicamente y la configuración de la tarjeta de red es de tipo puente. En la figura 24 se muestra la configuración descrita

Figura 24 Máquina virtual servidor de aplicaciones

5.2.2.2. Instalación Software necesario

Una vez creada la máquina virtual se procede a instalar los paquetes necesarios para poder configurar y poner en servicio la aplicación a diseñar.

- Python y pip: Aunque por defecto Python 3.0 viene instalado en la distribución de Ubuntu 20.10, es necesario instalar el administrador de paquetes pip, que nos permitirá instalar Node-mongosm y Pymongo. Para ellos instalaremos la distribución correspondiente a Python 3.0.

  Verificar versión Python

  ```
  $ python3 -V
  ```

  Instalar pip

  ```
  $ sudo apt install python3-pip
  ```

  Comprobar versión

  ```
  $ pip3 –versión
  ```

- Instalar conector pymongo: Pymongo es el controlador Python nativo de MongoDB y está desarrollado y mantenido por ingenieros de MongoDB asegurándose que Python y MongoDB funcionen sin problemas [29]. Se instala pymongo usando pip.

  ```
  $ pip3 install pymongo
  ```

- Instalación scripts Node-mongosm: Como se ha comentado anteriormente para poder importar los datos de OSM en la base de datos MongoDB es necesario instalar Node-mongosm. Para ello, instalamos la última compilación estable de npm [30].

  ```
  $ npm install mongosm
  ```

5.2.2.3. Instalación Base de datos mongodb

Siguiendo el procedimiento oficial descrito en la web de mongodb [31], en este trabajo se ha instalado la versión 4.4.4. Una vez instalado la aplicación comprobamos su versión, tal y como se muestra en la figura 25

```
bike@aplicacion:~$ mongo -version
MongoDB shell version v4.4.4
Build Info: {
    "version": "4.4.4",
    "gitVersion": "8db30a63db1a9d84bdcad0c83369623f708e0397",
    "openSSLVersion": "OpenSSL 1.1.1f  31 Mar 2020",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "ubuntu2004",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

Figura 25 Comprobación versión mongodb

En este punto se importará los datos de cruces, calles de la ciudad donde se va a realizar la simulación, para este trabajo se va a utilizar la ciudad de Las Palmas de Gran Canaria. OSM nos ofrece varias opciones para importar los datos, pero como la cantidad de nodos a importar es elevado desde la misma web nos indican que la mejor opción es usar Overpass API, que realiza la descarga desde una replica de la base de satos de OpenStreetMap. En la figura 26, se muestra el área elegida.

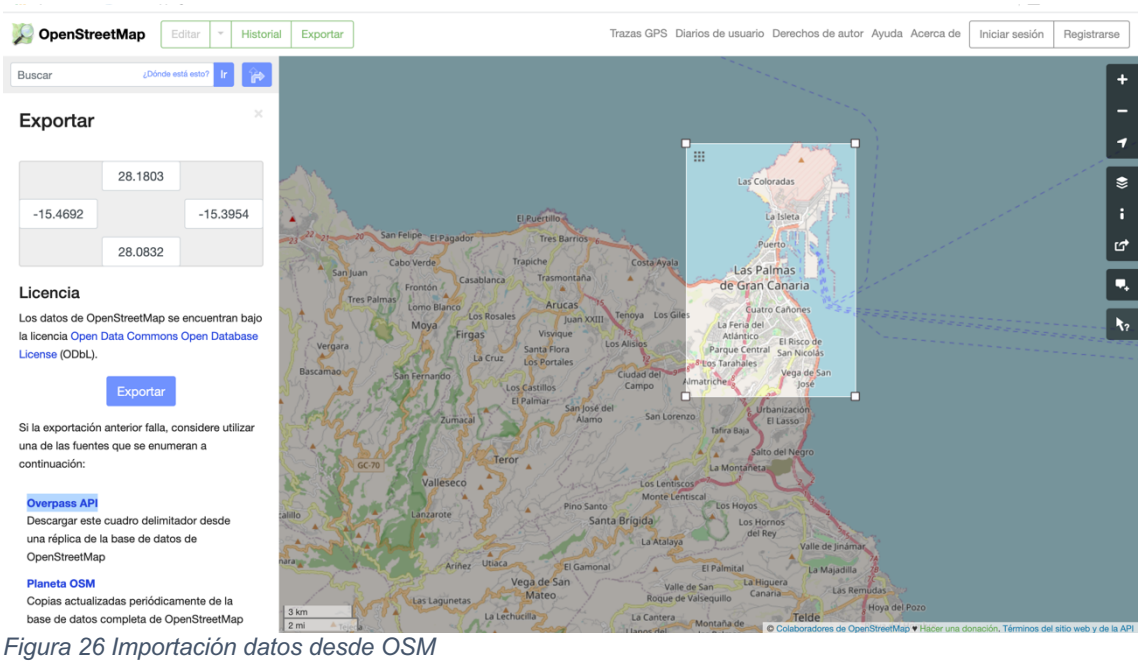


Figura 26 Importación datos desde OSM

Una vez obtenido el fichero “map.osm”, con los nodos, vías y relaciones del área seleccionada se procede a la importación usando el script Node-mongosm con el comando

```
$ ./mongosm -v -f map.osm
```

Este comando crea dentro de Mongodb la base de datos OSM con tres colecciones correspondiente a los elementos, Nodos, vías y relaciones que forman parte de OSM.

5.2.2.4. Instalación Servidor Web y PHP

Para este trabajo se instalará el servidor web Apache2 y el lenguaje de programación PHP.

- Instalación Apache 2

```
$ sudo apt-get install apache2
```

- Comprobación si el servidor está funcionando.

```
$ sudo service apache2 status
```

- Instalación PHP

```
$ sudo apt-get install php libapache2-mod-php
```

Para poder acceder a la web de la aplicación se va a usar el servicio de DDNS (Dinamic DNS) No-IP [32], el cual permite tener un subdominio propiedad de No-IP apuntando a la IP dinámica del servidor de aplicaciones.

Para ello, se crea una cuenta en la web de este servicio y se añade el dominio correspondiente. Para este trabajo se crea el dominio *bikebss.ddns.net*, en la figura 27 se muestra la página de configuración del subdominio, donde en Hostname se introduce el dominio y en Domain se selecciona el dominio propiedad de No-IP a usar, en el campo Record Type se selecciona DNS Host(A) que corresponde con el servicio de DDNS para IPv4, en el campo IPv4 Address muestra la IP dinámica del router donde se encuentra instalado el servidor.

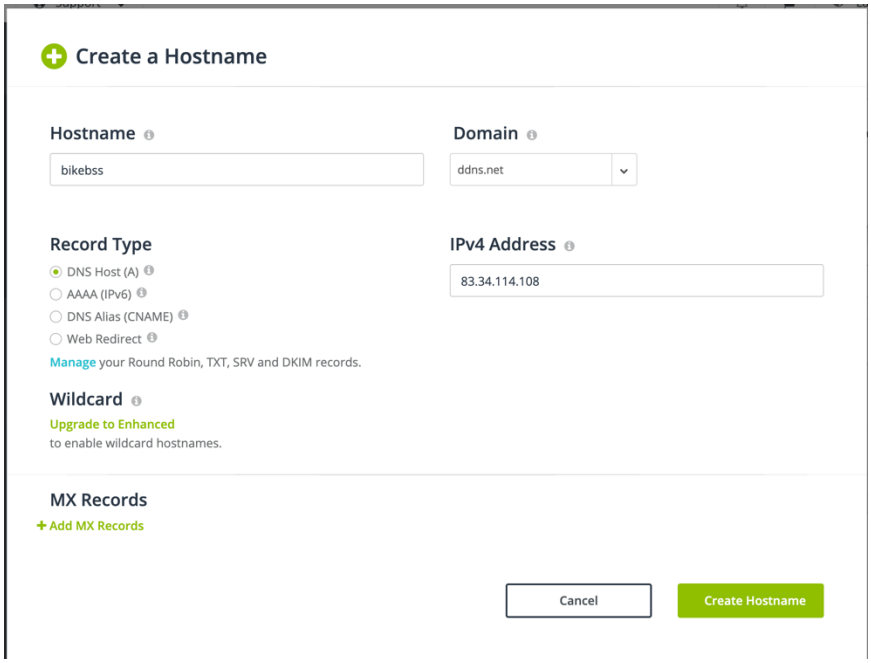


Figura 27 Configuración No-IP

Una vez creado el subdominio, es necesario instalar y configurar el cliente DUC (Dynamic Update Client) en el servidor de aplicaciones. Este cliente permite actualizar la IP en el servidor DNS de servicio cada vez que el router cambie de IP, para ello se sigue el procedimiento aplicado a los servidores Ubuntu [32], una vez instalado, se configura con los datos de la cuenta creada y el nombre del subdominio elegido.

5.2.3. Definición aplicación en servidores TTN

Una vez creada el simulador y el servidor de aplicaciones, queda conectar ambos elementos a través de la red LoRaWan de TTN. Para este trabajo se va a usar The Things Stack Community Edition (TTSCE) que es una de las distribuciones de The Things Stack (TTS). TTS es la tercera versión (v3) del servidor de red LoRaWan desarrollada y mantenida por The Things Industries [33].

Una vez dado de alta en el servidor se procede a definir los diferentes elementos necesarios, en este caso serian:

- Gateway: En la red LoRaWAN, el Gateway es el elemento que conecta los dispositivos finales con los servidores de la red. En esta arquitectura el Gateway forma parte del simulador Mbed.

- Aplicación: El servidor de aplicaciones controla los mensajes de la capa de aplicación de LoRaWAN, ademad de controlar los mensajes de Uplink y Downlink. Y nos permite crear y gestionar los nodos finales.

5.2.3.1. Registro Gateway

El registro del Gateway se realiza en la consola de TTSCE. En el overview del usuario se selecciona “Go to Gateway” y se añade un nuevo Gateway (+ Add Gateway). En el campo de ajustes básico hay que definir el ID del Gateway que debe ser único, puede tomar cualquier valor y no podrá ser usado de nuevo, en este caso se ha usado (eui-0242f1000097618a) que se corresponde con la dirección MAC del Gateway. Por otro lado, se debe indicar el EUI y que en el caso del emulador viene fijado cuando se inicia la aplicación tal y como se muestra en la figura 28.

```
bike@lorawan:~/mbed-simulator$ node server.js
Mbed Simulator v1.9.7
Web server listening on port 7829!
LoRaWAN information:
        Gateway ID:             02:42:f1:00:00:97:61:8a
        Packet forwarder host:  eu1.cloud.thethings.network
        Packet forwarder port:  1700
        Make sure the gateway registered in the network server running the *legacy packet forwarder*
```

Figura 28 EUI Gateway

En el campo “Gateway Server Address” es necesario poner la dirección del servidor al que el Gateway se conecta. En la figura 29 se muestra la configuración básica del Gateway.

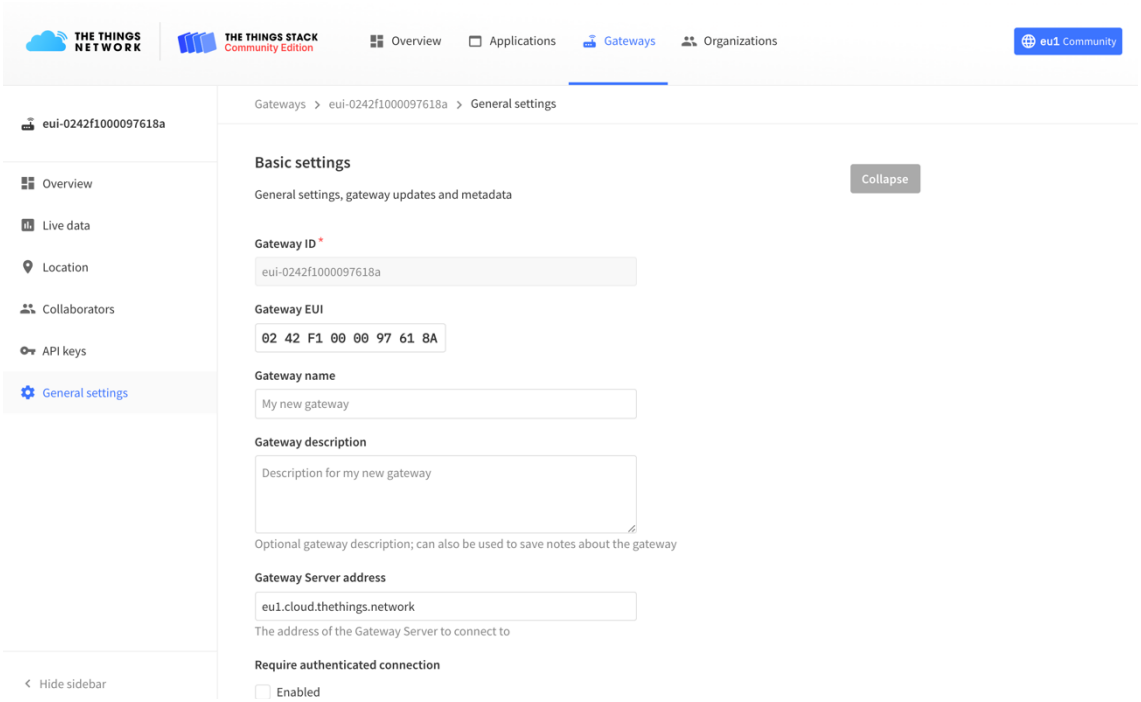


Figura 29 Configuración básica Gateway

Además, se debe definir la banda de frecuencia de trabajo en la capa de red, en nuestro caso, EU863-870, tal y como se muestra en la figura 30.

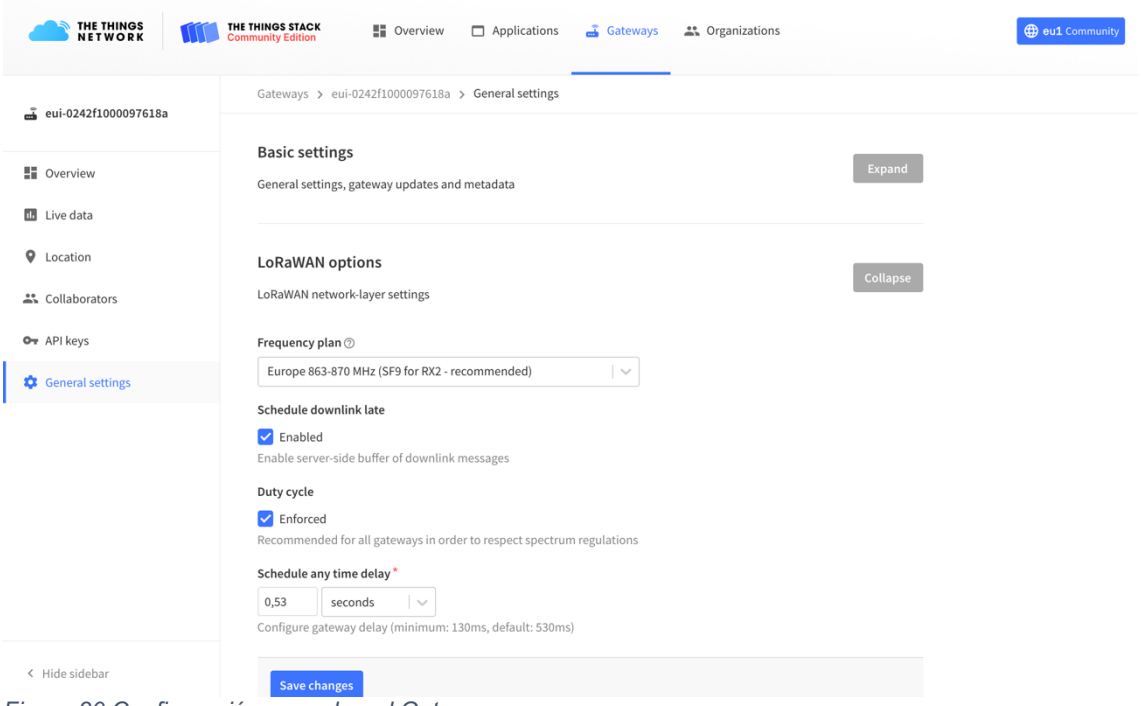


Figura 30 Configuración capa de red Gateway

5.2.3.2. Registro de la aplicación

Una vez creado el Gateway, el siguiente paso es crear dentro del servidor de aplicaciones de TTSCE, la aplicación que conecta el simulador Mbed con el

servidor de aplicaciones creado para este trabajo, permitiendo ver los mensajes entre los diferentes elementos de la red.

En la pagina principal de TTSCE se selecciona “Go to Application” y se añade una nueva aplicación, el único parámetro necesario para la creación es el ID que debe ser único y puede se puede usar cualquiera. En la figura 31, se muestra el “Overview” de la aplicación creada donde aparece los dispositivos creados se puede observar los mensajes entre los elementos de la aplicación creada.

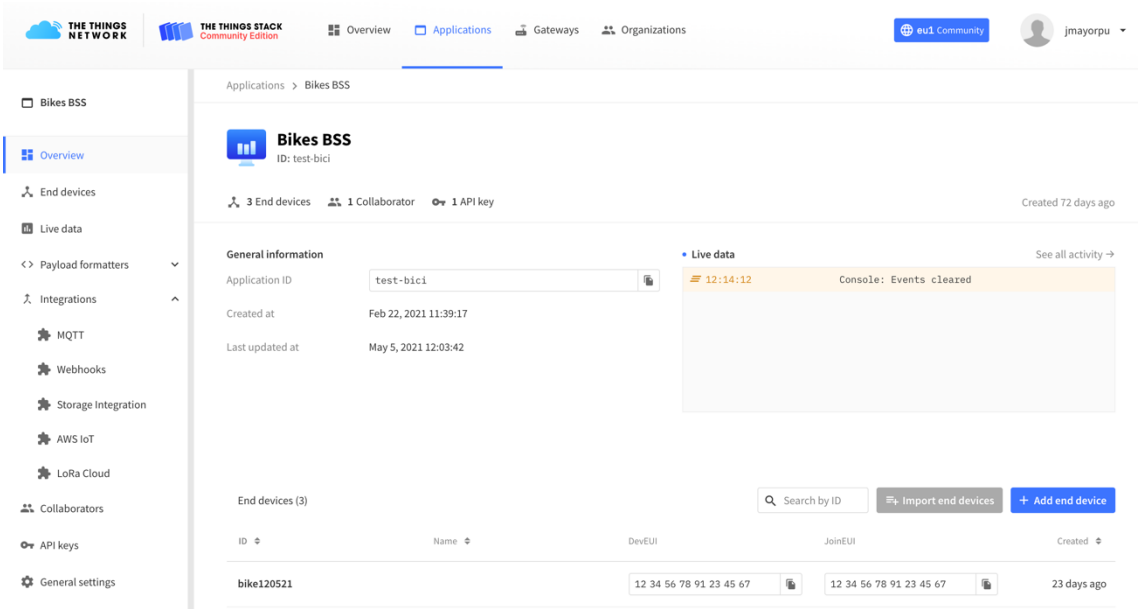


Figura 31 Overview aplicación

5.2.3.2.1. Añadir dispositivo

Una vez creada la aplicación el siguiente paso es añadir el dispositivo final que en este caso corresponde al simulador Mbed. Existe dos tipos de modos para conectarse a la red por parte de los dispositivos finales. ABP (Activation By Personalization) donde las claves de inicio de sesión se guardan el dispositivo final y OTAA (Over The Air Activation) donde las claves de sesión se renuevan cada vez que se inicia la sesión. En este trabajo se va a usar el modo tipo OTAA, para ello, tanto en el dispositivo como a la hora de añadir el dispositivo en la red hay que tener definir los siguientes parámetros.

- DevEUI: Es un identificador único de 32 bits que suministra el fabricante y que en caso del emulador se pude definir y que en este caso es “1234567891234567”.

- AppEUI: Es un identificador único de 64 bits que se utiliza para identificar el servidor de unión durante la activación. Se ha definido con el valor “1234567891234567”.

- AppKey: Corresponde con la clave de cifrado AES de 128 bits usados para proteger los mensajes que están incluidos en el payload y que en

este caso se ha generado a la hora de definir el dispositivo y es “2AA0D70C3FEE876BCEB9A3FB98346F1A8”.

Con esto datos se añade el dispositivo en la aplicación anteriormente creada. En la figura 32 tenemos los datos de DevEUI y AppEUI, junto con el identificador del dispositivo que es único

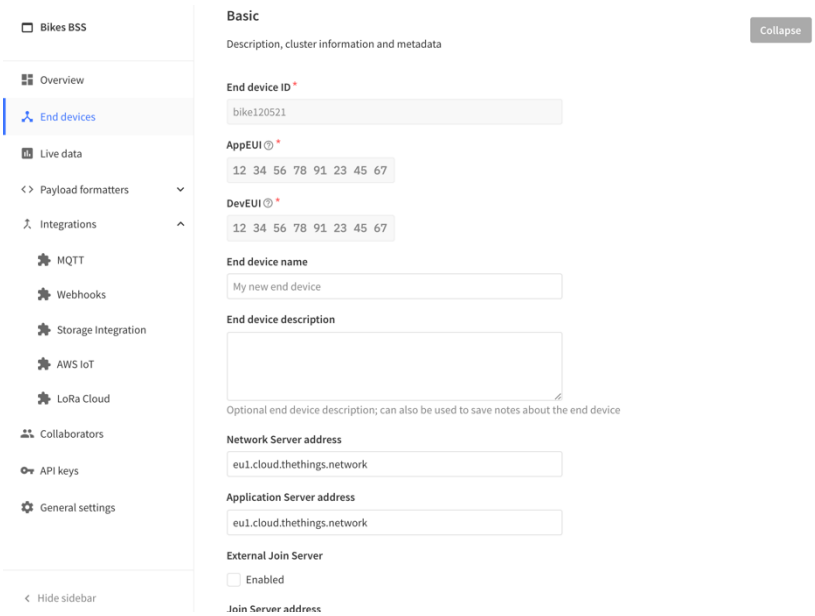


Figura 32 Configuración básica dispositivo

En la figura 33 tenemos la configuración de la unión con el AppKey definido

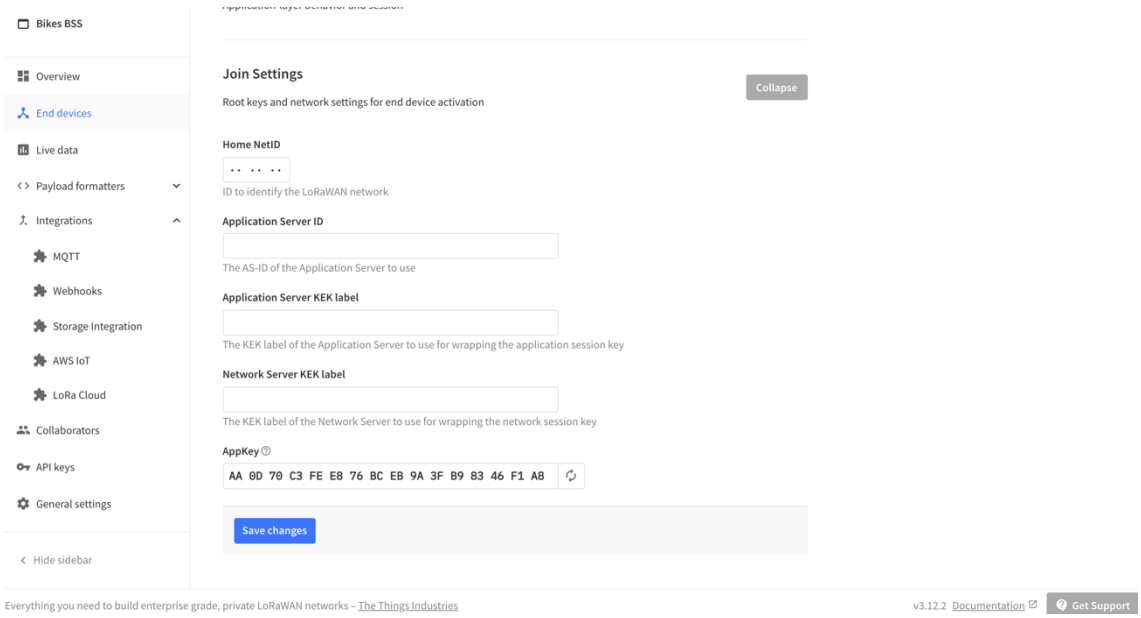


Figura 33 Configuración unión del dispositivo

En el apartado de configuración de la capa de red se selecciona la versión del simulado que es MAC V1.0.1, la banda de trabajo en Europa. El campo NwkSKey corresponde a la clave de cifrado usada después de la activación para proteger

los mensajes que no tienen carga útil, el cual se puede definir o dejar que se genere de forma aleatoria, en este caso se opta por que se genere aleatoriamente, en la figura 34 corresponde a la configuración de la capa de red.

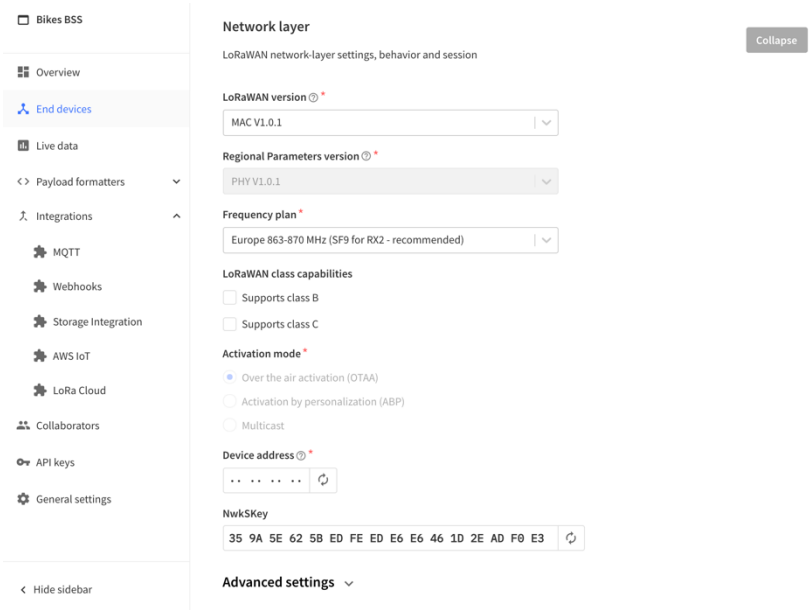


Figura 34 Configuración capa de red dispositivo

Por último, en la capa de aplicación el parámetro AppSKey corresponde a la clave de cifrado usada después de la activación del dispositivo para proteger los mensajes con carga útil y también se deja que el servidor genere la clave de forma aleatoria. En la figura 35 se observa la clave generada para el dispositivo creado.

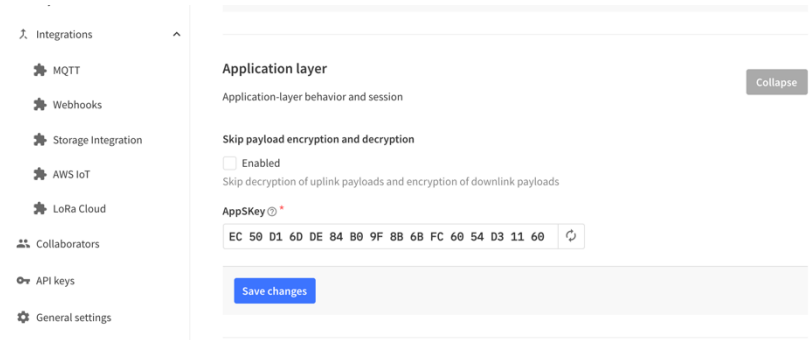


Figura 35 Configuración capa de aplicación del dispositivo

5.2.3.2.2. Añadir conexión al servidor de aplicación.

El servidor de TTSCE permite enviar los mensajes del Uplink hacia el servidor web creado junto con la aplicación usando Webhooks [43]. Este servicio permite reenviar los mensajes procedentes del dispositivo usando HTTP POST. Estos mensajes incluyen la información de conexión a los equipos, incluyendo tanto los mensajes con carga útil así como los mensajes de señalización. Para ello, en la pestaña “Integrations” se selecciona “Webhooks” y de las diferentes opciones se elije “Custom webhook”. En el diseño de la aplicación el formato de los datos

enviados al servidor son de tipo JSON, el ID de la conexión debe ser único y la URL base es la definida en el servicio No-IP, en nuestro caso http://bikebss.ddns.net/. En la figura 36 tenemos la configuración de la conexión.

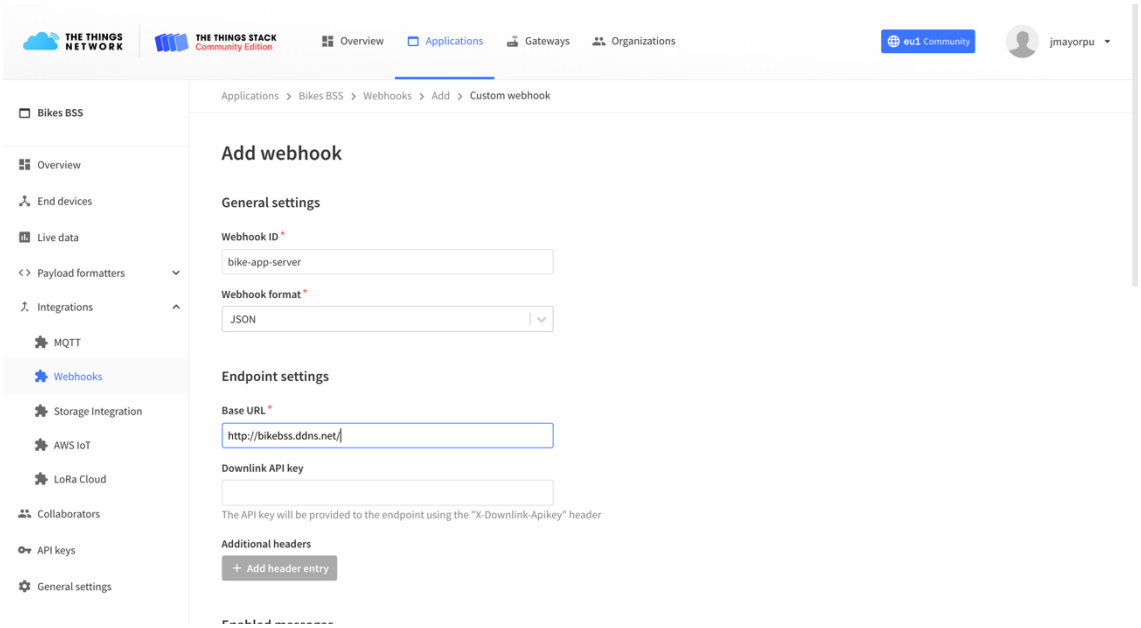


Figura 36 Creación conexión webhook

Una vez creada la conexión de los mensajes de Uplink, el siguiente paso consiste en vincular la aplicación creada en TTSCE para los mensajes de Downlink para ello se debe crear un API key con permiso de escritura para los mensajes de Downlink. En el menú de la aplicación creada se selecciona “API key” y se elije la opción “Write Downlink application traffic”. Una vez creado se guarda la clave generada para usarla en el servidor de aplicaciones. En la figura 37 se muestra configuración del API key de esta aplicación.

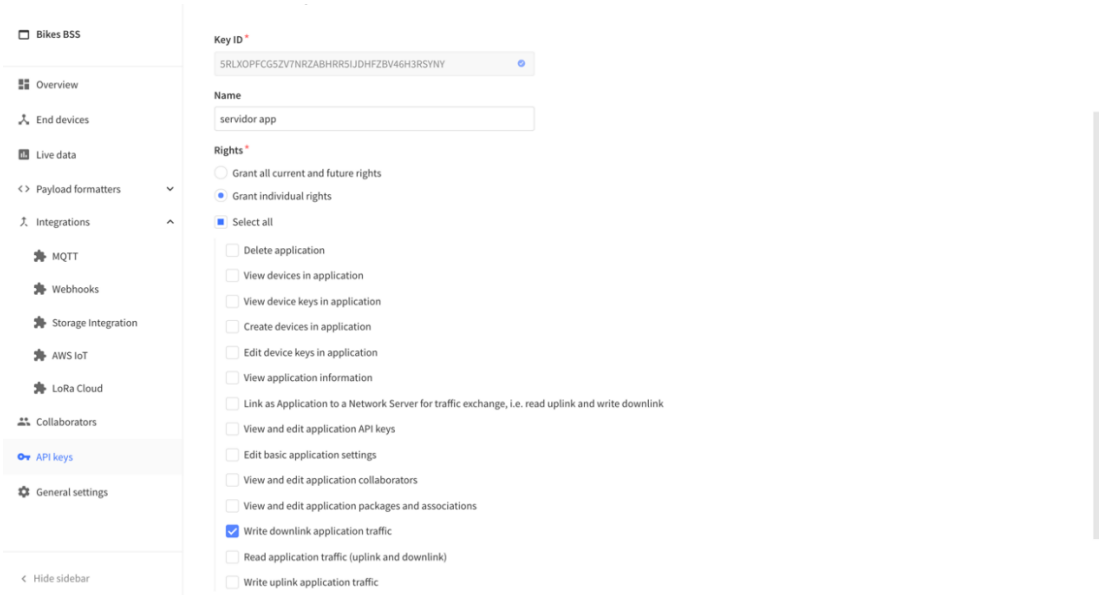


Figura 37 Creación API Key

La aplicación creada en el servidor de aplicaciones debe ser capaz de enviar Push con el siguiente formato para poder identificar la aplicación y el dispositivo destino

/api/v3/as/applications/{application_id}/webhooks/{webhook_id}/devices/{device_id}/down/push

En la cabecera del mensaje enviado debe incluir el API Key en campo “Authorization”.

Por ultimo, el mensaje incluido en el Push que incluye la carga útil debe cumplir con el siguiente formato de tipo JSON.

```
{
  "downlinks":[
    {
      "frm_payload":"xxxxxx",
      "f_port":15,
      "priority":"NORMAL"
    }
  ]
```

6. Simulación de BSS propuesto con Mbed y análisis de los resultados

Del sistema de alquiler propuesto en el apartado 3.2 se va a simular el proceso "Obtención de datos durante trayecto", en concreto la transmisión de la posición del usuario, a través de la red TTSCE, hacia el servidor de aplicación y el envío de la repuesta hacia el dispositivo.

Partiendo del diagrama de bloque presentado en la figura 12, la simulación diseñada incluirá los módulos de envío y recepción de los datos y la consulta en la base de datos GIS de los cruces de la vía por la que circula el usuario y las estaciones más cercanas, tal y como se muestra en la figura 38.

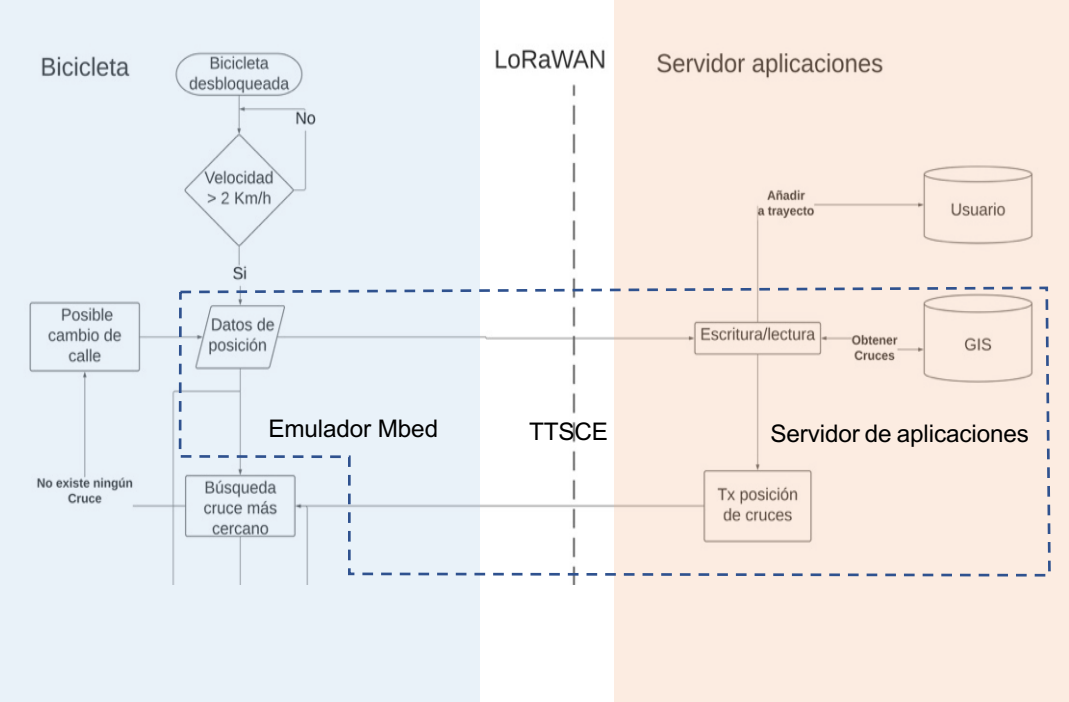


Figura 38 Simulación sistema BSS

6.1.1. Módulo simulador Mbed

El compilador del sistema operativo Mbed se basa en C++, por tanto, el simulador usado tiene un editor basado en este lenguaje de programación. Para el código de la simulación hemos partido del ejemplo incluido en el emulador para dispositivos LoRaWAN con modo de conexión OTAA. En el anexo II aparece el código creado, donde se puede observar que las variables globales definidas corresponden a los parámetros DevEUI, AppEUI y AppKey. En dicho código tiene como variable se incluye la longitud y latitud del usuario. En la parte principal del código se habilitar el traceo de la pila LoRa del sistema operativo Mbed que se muestra en la ventada de salida y se procede a la conexión con el servidor TTSCE. Una vez establecida la conexión, esperamos a que se pulse el botón para enviar los datos de posición y esperar la repuesta por parte de la aplicación.

En la figura 39 tenemos el resultado de la ejecución del código sin pulsar el botón de envío, en la ventana de salida (Serial output) observamos los mensajes procedentes de la pila de LoRa para establecer la conexión con el servidor de TTSCE y el resultado de los diferentes pasos de conexión.

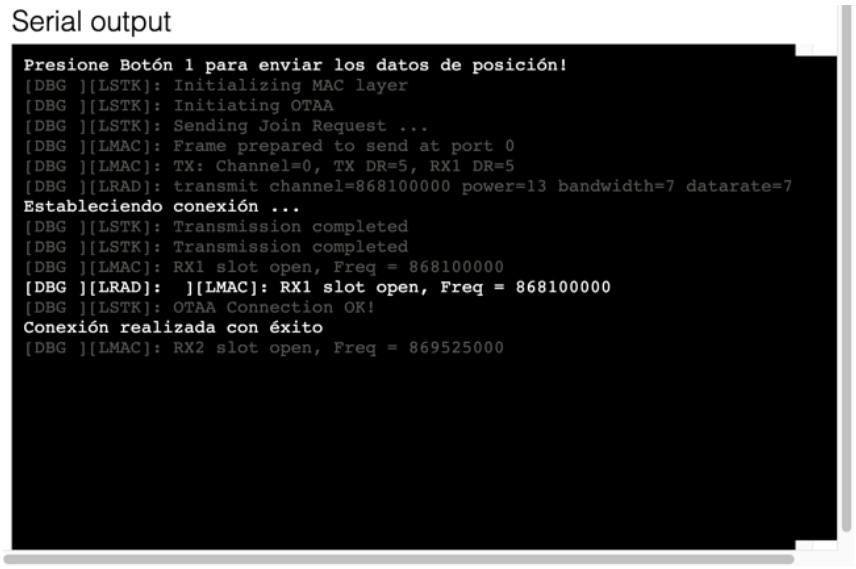


Figura 39 Establecimiento conexión con simulador Mbed

En la figura 40 tenemos los mensajes en la plataforma TTSCE, donde observamos que, una vez recibida la petición de conexión del dispositivo final a través del Gateway, posteriormente hay un intercambio de mensajes entre los elementos de la plataforma aceptando la petición de unión del usuario y el mensaje de repuesta hacia el simulador.

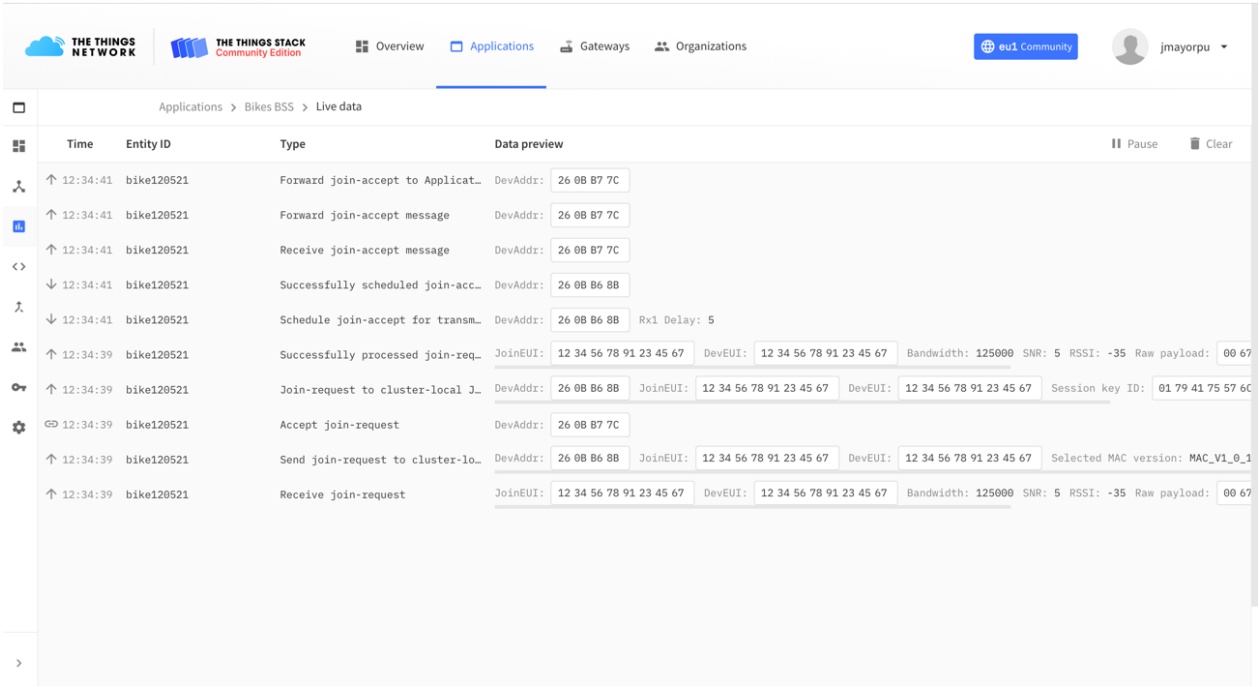


Figura 40 Mensajes en TTS

Cuando se pulsa el botón en el diagrama del microcontrolador, el código procede a crear el paquete a enviar, lo pone en la cola de transmisión y espera el paquete de repuesta que incluya la lista de los cruces y estaciones de estacionamiento cercanas.
En la figura 41 se observa en la pantalla de la salida el paquete a enviar, su tamaño, seguido del paquete recibido, su tamaño y la repuesta descodificada. También se incluye la distancia entre el cruce y la bicicleta que se ha calculado usando el algoritmo de Haversine [34].

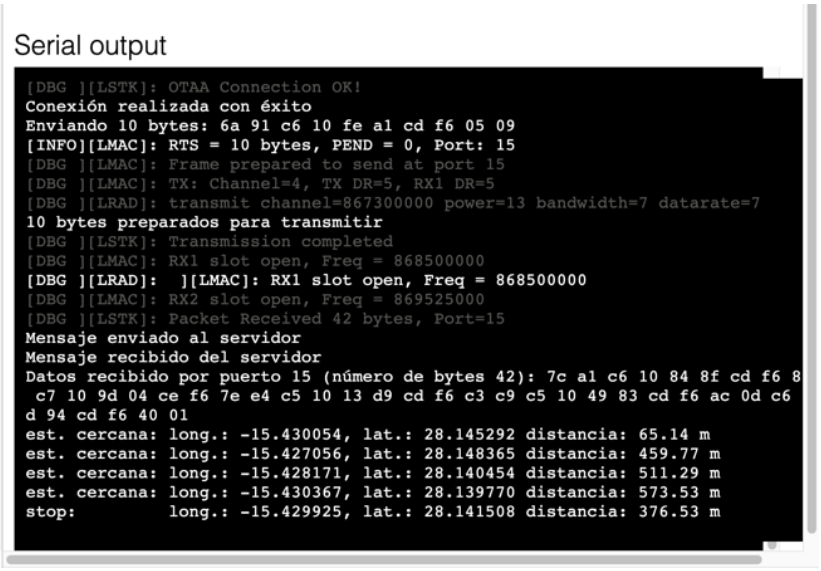


Figura 41 Envío y recepción de mensaje en el simulador Mbed

A continuación, en la figura 42 se observa el flujo de mensajes en dicha plataforma, en donde una vez recibido el mensaje desde el simulador se reenvía a la URL del servidor de aplicaciones, una vez recibida la repuesta el mensaje se reenvía al simulador.

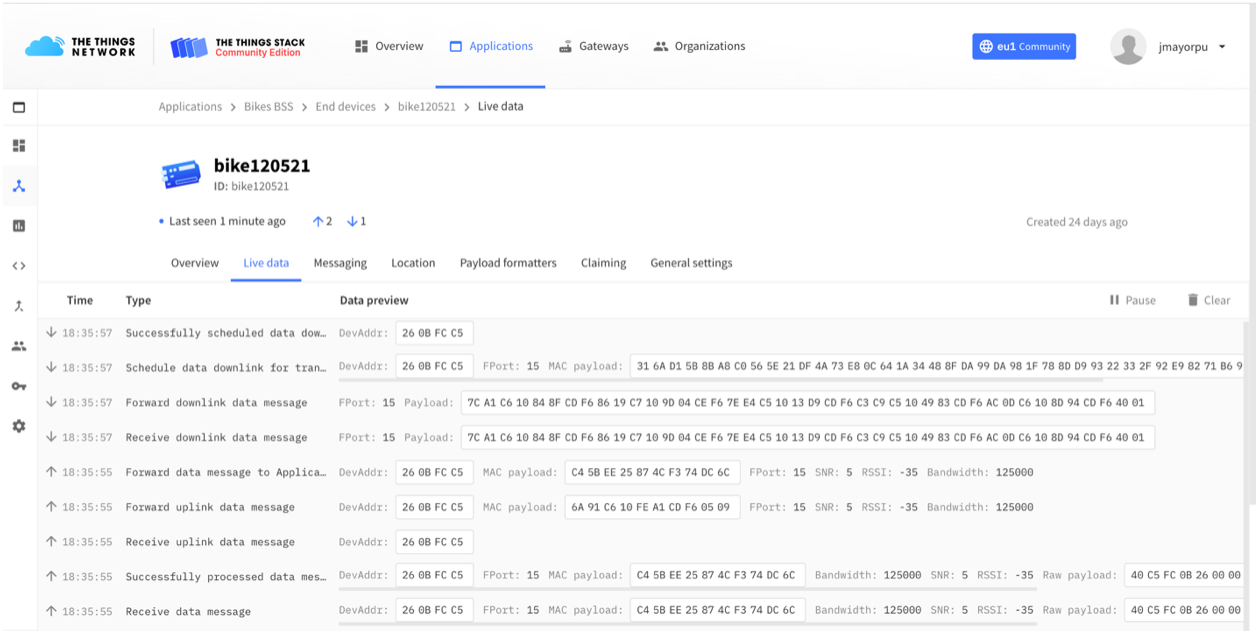


Figura 42 Flujo de mensajes plataforma TTN

6.1.2. Módulo Servidor de aplicaciones

En la figura anterior aparece el mensaje de repuesta que incluye la lista de cruces de la vía por la que se circula y las estaciones de estacionamiento próximas. Para crear esta repuesta se ha desarrollado dos aplicaciones PHP en el servidor de aplicaciones, la primera que se encuentra en el directorio "/bike/" del servidor web cuando recibe el mensaje POST procedente del servidor de red de TTSCE, guarda el contenido del mensaje en un fichero y hace una llamada a la aplicación de Python "bike.py", la cual se encarga de descodificar el mensaje recibido, hacer la consulta a la base de datos, crear el mensaje de Downlink y enviarlo usando HTTP POST al servidor TTSCE para que reenvíe el mensaje al dispositivo final que en nuestro caso es el simulador Mbed.

Por otro lado, en el directorio "/message/" de la web se encuentra la segunda aplicación PHP que almacena un log con los mensajes enviados por el dispositivo que no incluyan carga útil.

Para que el servidor de red de TTSCE sepa a que aplicación enviar cada tipo de mensaje, es necesario declaralo en la conexión Webhook creada anteriormente, para ello, se edita y como se pude ver en la figura 43, los mensajes con carga útil se reenvían a la URL http://bikebss.ddns.net/bike/ y el resto de los mensajes son enviados a la URL http://bikebss.ddns.net/message/

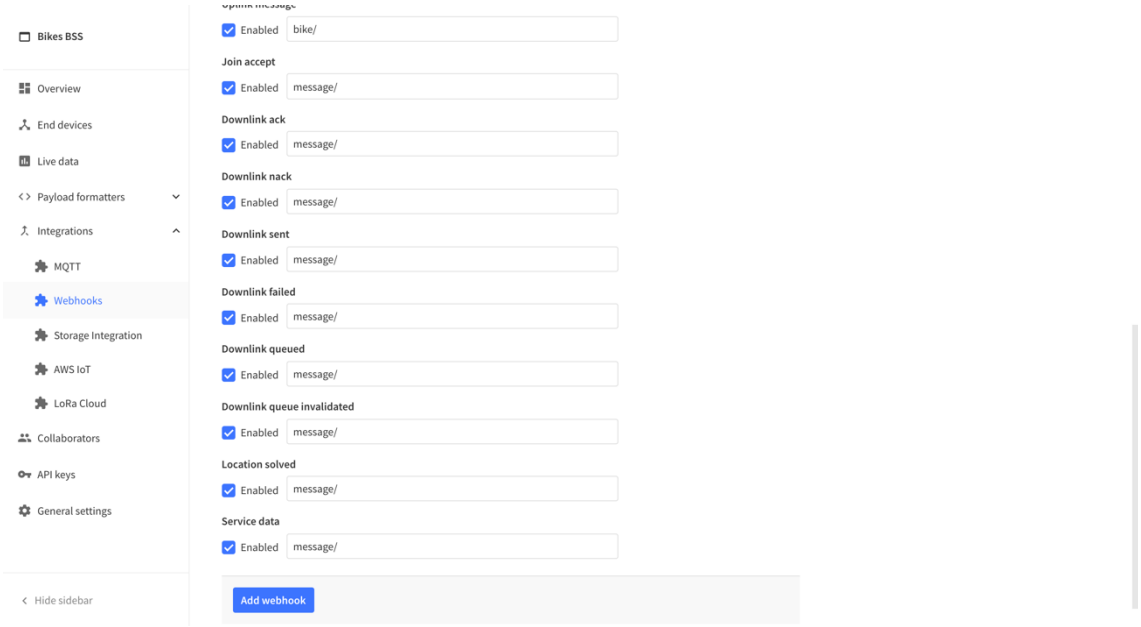


Figura 43 Configuración rutas HTTP POST

En el anexo III.a se incluye el código PHP que se encuentra en /bike/, donde como se ha comentado el mensaje recibido se guarda en el fichero "bike.txt" y se ejecuta la aplicación Python "bike.py". En la figura 44, se muestra un ejemplo del mensaje JSON recibido desde el servidor TTN, donde se puede observa que el campo "device_id" corresponde al dispositivo creado, el campo "application_id" corresponde a la aplicación creada, en el campo "frm_payload" es el mensaje de Uplink recibido y en la parte final del mensaje se incluye la información al interfaz radio incluido el ToA.

```
{
   "end_device_ids":{
      "device_id":"bike120521",
      "application_ids":{
         "application_id":"test-bici"
      },
      "dev_eui":"1234567891234567",
      "join_eui":"1234567891234567",
      "dev_addr":"260BFCC5"
   },
   "correlation_ids":[
      "as:up:01F51C05VZYESWZCP3WTJ4RGMY",
      "gs:conn:01F51AADP6BRP5HVWRGATXAFKF",
      "gs:up:host:01F51AADPE5BJXW77NX994WFZG",
      "gs:uplink:01F51C05NGP0QN1PXE82XRQE82",
      "ns:uplink:01F51C05NH4QYXDCS0WXR66GVE",
      "rpc:/ttn.lorawan.v3.GsNs/HandleUplink:01F51C05NHJQCH3Y8A73DYFG3B",
      "rpc:/ttn.lorawan.v3.NsAs/HandleUplink:01F51C05VZWC0BQ9B3DHXBYYRM"
   ],
   "received_at":"2021-05-06T17:35:55.776198268Z",
   "uplink_message":{
      "session_key_id":"AXlCv6opY2docBljSZx5aw==",
      "f_port":15,
      "frm_payload":"apHGEP6hzfYFCQ==",
      "rx_metadata":[
         {
            "gateway_ids":{
               "gateway_id":"eui-0242f1000097618a",
               "eui":"0242F1000097618A"
            },
            "time":"2021-05-06T17:35:55.536Z",
            "timestamp":1766725000,
            "rssi":-35,
            "channel_rssi":-35,
            "snr":5,

"uplink_token":"CiIKIAoUZXVpLTAyNDJmMTAwMDA5NzYxOGESCAJC8QAAl2GKEIiruMoGGgwI+
9HQhAYQrtuCjwIgwLaQyLUz",
            "channel_index":2
         }
      ],
      "settings":{
         "data_rate":{
            "lora":{
               "bandwidth":125000,
               "spreading_factor":7
            }
         },
         "data_rate_index":5,
         "coding_rate":"4/6",
         "frequency":"867300000",
         "timestamp":1766725000,
         "time":"2021-05-06T17:35:55.536Z"
      },
      "received_at":"2021-05-06T17:35:55.569696359Z",
      "consumed_airtime":"0.069888s"
   }
}
```


Figura 44 Mensaje Uplink en servidor de aplicaciones

Por otro lado, en la figura 45 se muestra un ejemplo de los mensajes procedente des simulador que no incluye carga útil de Uplink, como son el mensaje de petición registro del dispositivo y mensaje de Downlink.


```
{
    "end_device_ids": {
        "device_id": "bike120521",
        "application_ids": {
            "application_id": "test-bici"
        },
        "dev_eui": "1234567891234567",
        "join_eui": "1234567891234567",
        "dev_addr": "260BFCC5"
    },
    "correlation_ids": [
        "as:up:01F51BZC9R9N7DB0F459F1RFWY",
        "gs:conn:01F51AADP6BRP5HVWRGATXAFKF",
        "gs:up:host:01F51AADPE5BJXW77NX994WFZG",
        "gs:uplink:01F51BZAH4YDTCZA493YC1JZWK",
        "ns:uplink:01F51BZAH56RRQ91XZ6H49A281",

"rpc:\/ttn.lorawan.v3.GsNs\/HandleUplink:01F51BZAH5SM53FKPD0PRW4MYM",

"rpc:\/ttn.lorawan.v3.NsAs\/HandleUplink:01F51BZC9R3S2Q7EZNSSGEMADK"
    ],
    "received_at": "2021-05-06T17:35:29.593095504Z",
    "join_accept": {
        "session_key_id": "AXlCv6opY2docBljSZx5aw==",
        "received_at": "2021-05-06T17:35:27.781700932Z"
    }
}
{
    "end_device_ids": {
        "device_id": "bike120521",
        "application_ids": {
            "application_id": "test-bici"
        }
    },
    "correlation_ids": [
        "as:downlink:01F51C0752VNX156TTB69ZHB3T"
    ],
    "downlink_queued": {
        "f_port": 15,
        "frm_payload":
"fKHGEISPzfaGGccQnQTO9n7kxRAT2c32w8nFEEmDzfasDcYQjZTN9kAB",
        "priority": "NORMAL",
        "correlation_ids": [
            "as:downlink:01F51C0752VNX156TTB69ZHB3T"
        ]
    }
}
```


Figura 45 Mensajes Downlink y señalización

En el anexo IV corresponde con el código Python de la aplicación “bike.py”, donde a partir del mensaje de subida incluido en el campo “frm_payload”, se

descodifica en la latitud y longitud del dispositivo final. Con estos datos realizamos la consulta a la base datos buscando los cruces de la vía por la que se circula y las estaciones próximas, hasta un máximo de cinco registros los cuales se codifica y se envía usando HTTP POST al servidor de red. La aplicación guarda en el fichero “bikelog.log” los diferentes pasos realizados por la aplicación. Tal y como se muestra en la figura 46, donde se muestra el mensaje recibido, su descodificación es bytes y en latitud y longitud. Con estos datos, se realiza la búsqueda de la vía por la vía para conocer la posición de los cruces y estaciones de estacionamiento cercanas. Con estos datos, se crea el mensaje de Downlink, se codifica y se envía vía HTTP al servidor de red de red.

Figura 46 Log aplicación de búsqueda de cruces

6.1.3. Análisis simulación

En el capitulo 4 se calculo el tamaño de los paquetes de Uplink y Downlink teniendo en cuenta las cabeceras. En la figura 41, se observa el tamaño de los paquetes de Uplink (10 bytes) y de Downlink (48 bytes) que corresponde con los diseñado y en la salida del Gateway simulado de la figura 47 se puede observar el tamaño de los paquetes incluyendo la cabecera, 23 bytes para el Uplink y 55 para el Downlink, lo cuales coinciden con el calculado en el punto 4.2.

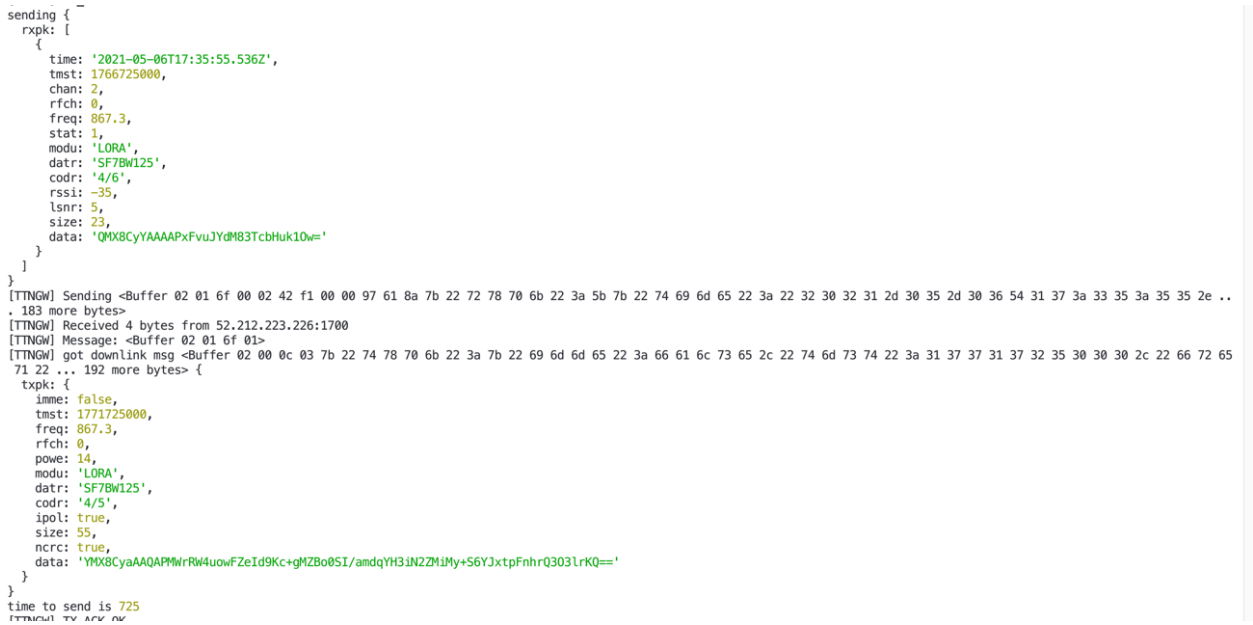


Figura 47 Log Gateway

El tamaño de los paquetes es importante, ya que, en las redes LoRaWAN está limitado el tiempo de uso del interfaz aire al 1% del tiempo. En el punto 4.2.4 se

ha realizado un cálculo de ToA dependiendo del Spreading Factor y ancho de banda usado. Dentro del flujo de mensajes de la figura 44, se puede observar que mensaje de Uplink incluye los parámetros del interfaz aire incluyendo el ToA medido por el servidor de red. Para comprobar la validez de los ToA calculados según el Spreading factor se modifica el parámetro “lorawan.set_datarate” en el código C++ del simulador, incluyendo los diferente SF disponibles.
En la tabla 8, se compara los valores calculados frente a los obtenidos en la simulación, hay que tener en cuenta la variación que se observa a la hora de seleccionar el Spreading Factor (SF), ya que, nos va a limitar el tiempo mínimo entre transmisión de los paquetes.

SF	BW(khz)	data_rate_index	ToA Teórico (ms)	ToA Simulación (ms)
7	125	5	61.7	69.9
8	125	4	113.1	127.5
9	125	3	205.8	230.4
10	125	2	370.7	411.6
11	125	1	823.3	921.6
12	125	0	1482.8	1646.6

Tabla 8 Comparación valores ToA

7. Conclusiones

En este trabajo se ha demostrado que es posible desarrollar un sistema BSS para las *Smart Cities,* que fomente la movilidad no motorizada, no solo para reducir la contaminación, sino para mejorar la seguridad en los trayectos premiando el buen comportamiento del usuario con una reducción del coste de alquiler.

El uso del entorno de simulación usado en este trabajo facilita el desarrollo de aplicaciones IoT en entornos de *Smart Cities*, permitiendo comprobar dichas aplicaciones un entorno de red real y poder probar el funcionamiento de la aplicación diseñada antes de pasar a compilarlo en el dispositivo IoT.

Esta simulación nos permite comprobar el principal parámetro de diseño de las redes ISM como es el tiempo de ocupación del interfaz radio. Con los datos obtenidos se puede planificar la red de Gateways necesaria para cubrir el servicio, optimizando el número de equipos a instalar. Teniendo en cuenta que dependería del SF que se selecciona, ya que, cuanto menor sea si valor menos área va a cubrir el Gateway. Otro factor en la planificación radio sería el número de bicicletas que forman parte del sistema. Este tipo de redes al ser escalables permiten ampliar la red Gateways sin mucho sobrecoste si el número de usuarios aumenta.

A la hora de realizar el diseño se ha verificado que es el Downlink quien limita el uso del interfaz aire, al ser el que mayor tiempo de uso tiene, por lo que, si se usa un SF menor, el área de cobertura es menor, el tiempo de uso del interfaz aire se reduce. Implicando que sea necesario un mayor número de Gateways para cubrir el área. Por otro lado, permite que el número de bicicletas que puede asumir el sistema sea mayor.

En relación con las líneas futuras, el trabajo se ha centrado en simular la comunicación entre los diferentes equipos de la red, quedando pendiente el diseño de los módulos inicio de alquiler y la plataforma de pago. Además, se ha observado la necesidad de un desarrollo más profundo de la aplicación de obtención de la posición de los cruces señalizado, ya que, en este caso se envía los datos de los cruces sin tener en cuenta la dirección que circula el usuario, siendo necesario desarrollar una aplicación que permita optimizar los mensajes de bajada, pudiendo reducir de este modo el ToA y, por tanto, optimizar el uso de la red.

8. Glosario

3GPP: 3rd Generation Partnership Project
ABP: Activation By Personalization
ADR: Adaptive Data Rate
API: Application Programming Interface
ARM: Advanced RISC Machine
BDS: Bike Distribution Service
BLE: Bluetooth Low Energy
BPAC: Bike and Parking Availability Check
BSON: Binary JSON
BSS: Bicycle Share System
BW: Band Width
CO2: Dióxido de carbono
CR: Coding Rate
CSS: Chirp Spread Spectrum
eDRX: Extended Discontinuous Reception
EPC: evolved packet core
ETSI: European Telecommunication Standards Institute
EUI: Extended Unique Identifier
FEC: Forward Error Correction
GPS: Global Positioning System
GMS: Global System for Mobile Communications
HTTP: Hypertext Transfer Protocol
IoT: Internet of Things
ISM: Industrial, Scientific and Medical
JSON: JavaScript Object Notation
LoRaWAN: Low Radio Wide Area Network
LPWAN: Low-Power Wide-Area Network
LTE: Long-Term Evolution
LTE-M: LTE-MTC (Machine Type Communication)
MAC: Media Access Control
M2M: Machine to Machine
MC: Multimodal connectivity
MCU: Microcontroller Unit
MQTT: Message Queuing Telemetry Transport
NB-IoT: Narrow Band IoT
OSM: Open Street Map
OTAA: Over The Air Activation
PHP: Hypertext Preprocessor
POI: Point of interest
PSM: Power Saving Mode
RS: Reservation Service
SIG: Sistemas de Información Geográfica
SNR: Signal Noise Relation
SF: Spreading Factor
TCP: Transmission Control Protocol
TIC: Tecnologías de Información y Comunicación

ToA: Time of Air
TP: Transmission Power
TTN: The Things Network
TTS: The Things Stack
TTSCE: The Things Stack Community Edition
UDP: User Datagram Protocol
UE: Unión Europea
UE: User Equipment
URL: Uniform Resource Locator
VM: Virtual Machine
WKT: Well Known Text
WGS84: World Geodetic System 84
WIFI: Wireless Fidelity
XML: eXtensible Markup Language

9. Bibliografía

[1] https://ec.europa.eu/transport/themes/urban/urban-mobility/urban-mobility-package_en . Visitado el 14 de mayo de 2021.

[2] https://civitas-initiative.eu/about-us/ .Visitado el 10 de marzo de 2021.

[3] https://ec.europa.eu/transport/road_safety/specialist/knowledge/pedestrians/promote_cycling_and_bicycle_helmets_or_not/promoting_cycling_changes_to_expect_en. Visitado el 23 de mayo de 2021.

[4] M. A. Razzaque and S. Clarke, "Smart management of next generation bike sharing systems using Internet of Things," 2015 IEEE First International Smart Cities Conference (ISC2), Guadalajara, Mexico, 2015, pp. 1-8, doi: 10.1109/ISC2.2015.7366219.

[5] http://indigoo.com/dox/itdp/12_MobileWireless/LPWAN.pdf. Visitado el 23 de marzo de 2021.

[6] K. K. Nair, A. M. Abu-Mahfouz and S. Lefophane, "Analysis of the Narrow Band Internet of Things (NB-IoT) Technology," 2019 Conference on Information Communications Technology and Society (ICTAS), Durban, South Africa, 2019, pp. 1-6, doi: 10.1109/ICTAS.2019.8703630.

[7] https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf. Visitado el 15 de marzo de 2021.

[8] J. de Carvalho Silva, J. J. P. C. Rodrigues, A. M. Alberti, P. Solic and A. L. L. Aquino, "LoRaWAN — A low power WAN protocol for Internet of Things: A review and opportunities," 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech), Split, Croatia, 2017, pp. 1-6.

[9] Y. Lalle, L. C. Fourati, M. Fourati and J. P. Barraca, "A Comparative Study of LoRaWAN, SigFox, and NB-IoT for Smart Water Grid," 2019 Global Information Infrastructure and Networking Symposium (GIIS), Paris, France, 2019, pp. 1-6, doi: 10.1109/GIIS48668.2019.9044961.

[10] A. Lavric and A. I. Petrariu, "LoRaWAN communication protocol: The new era of IoT," 2018 International Conference on Development and Application Systems (DAS), Suceava, Romania, 2018, pp. 74-77, doi: 10.1109/DAAS.2018.8396074.

[11] H. U. Rahman, M. Ahmad, H. Ahmad and M. A. Habib, "LoRaWAN: State of the Art, Challenges, Protocols and Research Issues," 2020 IEEE 23rd International Multitopic Conference (INMIC), Bahawalpur, Pakistan, 2020, pp. 1-6, doi: 10.1109/INMIC50486.2020.9318170.

[12] https://www.thethingsnetwork.org/docs/index.html. Visitado el 5 de mayo de 2021.

[13] https://es.wikipedia.org/wiki/Sistema_de_información_geográfica. Visitado el 12 de marzo de 2021.

[14] https://www.openstreetmap.org/. Visitado el 7 de mayo de 2021.

[15] https://docs.mongodb.com/manual/geospatial-queries/. Visitado el 29 de abril de 2021.

[16] https://ec.europa.eu/transport/themes/clean-transport-urban-transport/cycling_en. Visitado el 13 de mayo de 2021.

[17] Tan Yigitcanlar,Md., Kamruzzaman,Marcus, Foth,Jamile Sabatini-Marques, Eduardo da Costa, Giuseppe Ioppolo “Can cities become smart without being sustainable? A systematic review of the literature”. Sustainable Cities and Society.Publisher: Elsevier. February 2019.
[18] https://mobilityweek.eu/home/. Visitado el 14 de marzo de 2021.
[19] F. Chiariotti, C. Pielli, A. Cenedese, A. Zanella and M. Zorzi, "Bike sharing as a key smart city service: State of the art and future developments," 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 2018, pp. 1-6, doi: 10.1109/MOCAST.2018.8376628.
[20] T. Ngo and D. Kim, "A Smart TLVC-Based Traffic Light Scheduling for Preventing YLD-related Accidents in Smart City," 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 2018, pp. 1235-1239, doi: 10.1109/ETFA.2018.8502535.
[21] https://lora-alliance.org/resource_hub/rp2-102-lorawan-regional-parameters/. Visitado el 23 de abril de 2021.
[22] https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/. Visitado el 23 de abril de 2021.
[23] https://lora-developers.semtech.com/library/product-documents/. Visitado el 29 de abril de 2021.
[24] https://akirasan.net/enviar-coordenadas-gps-por-lorawan-the-things-network/. Visitado el 12 de abril de 2021.
[25] https://os.mbed.com/. Visitado el 14 de mayo de 2021.
[26] https://labs.mbed.com/. Visitado el 3 de mayo de 2021.
[27] https://www.virtualbox.org/. Visitado el 20 de marzo de 2021.
[28] https://github.com/armmbed/mbed-simulator/. Visitado el 6 de mayo de 2021.
[29] https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb. Visitado el 4 de mayo de 2021.
[30] http://sammerry.github.io/node-mongosm/. Visitado el 7 de mayo de 2021.
[31] https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/. Visitado el 4 de mayo de 2021.
[32] https://www.noip.com/support/knowledgebase/installing-the-linux-dynamic-update-client-on-ubuntu. Visitado el 7 de mayo de 2021.
[33] https://eu1.cloud.thethings.network. Visitado el 20 de mayo de 2021.
[34] https://es.wikipedia.org/wiki/F%C3%B3rmula_del_semiverseno. Visitado el 7 de mayo de 2021.
[35] https://ec.europa.eu/transport/themes/urban/programmes_projects_en. Visitado el de 13 mayo de 2021.
[36] Movilidad inteligente. Fiamma Pérez Prada ; Guillermo Velázquez Romera; Victoria Fernández Añez ; Javier Dorao Sánchez Centro de Investigación del Transporte (TRANS y T-UPM) Localización: Economía i ndustrial, ISSN 0422-2784, Nº 395, 2015 (Ejemplar dedicado a: Ciudades inteligentes), págs. 111-121
[37] https://www.3gpp.org/news-events/1785-nb_iot_complete. Visitado el de 17 mayo de 2021.

[38] K. Mekki, E. Bajic, F. Chaxel and F. Meyer, "Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT," 2018 EEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2018, pp. 197-202, doi: 10.1109/PERCOMW.2018.8480255.

[39] Habib Ur Rahman;Mudassar Ahmad;Haseeb Ahmad;Muhammad Asif Habib "LoRaWAN: State of the Art, Challenges, Protocols and Research Issues" 2020 IEEE 23rd International Multitopic Conference (INMIC) Year: 2020 | Conference Paper | Publisher: IEEE

[40] https://wiki.openstreetmap.org/wiki/Databases_and_data_access_APIs. Visitado el de 4 mayo de 2021.

[41] https://es.wikipedia.org/wiki/Las_Palmas_de_Gran_Canaria. Visitado el de 20 mayo de 2021.

[42] https://lora-alliance.org/resource_hub/online-simulation-of-lorawan-devices/. Visitado el 3 de mayo de 2021.

[43] https://en.wikipedia.org/wiki/Webhook. Visitado el de 26 mayo de 2021.

10. Anexos

I. Cálculo Time of Air (ToA) con Matlab.

A partir de las variables de entrada, se calcula el ToA, el número de paquetes por Hora y el tiempo en segundos que debe pasar entre envió de paquetes. Para ello, se usa las formulas descritas en el apartado 4.3.

```
PL = 55; % Carga útil
CR = 1; % Coding rate
BW = 125 % Band width
DE = 0; % Optimización desactivada
for f = 7:12
    SF= f % Spreading Factor
    if SF >= 11
        DE=1
    end
    Tsym = 2^SF/(BW*1000)*1000
    Tpre = (8 + 4.25)*Tsym
    npayload = 8 + max(ceil((8*PL-4*SF+28+16)/(4*(SF-
2*DE)))*(CR+4))
    Tpayload = npayload*Tsym

    ToA = Tpre + Tpayload   % Time of Air
    paqporhora = floor(36000/ToA) % Paquete por Hora
    cadseg = 3600/paqporhora % Tiempo entre paquetes
end
```

II. Código Mbed emulator

A continuación, se muestra el código del simulado Mbed escrito en C++. En la parte principal se incluye el procedimiento de conexión con los servidores de red de TTN, una vez aceptada la conexión se espera a que se pulse el botón 1 del simulador para llamar a la rutina de transmisión (*send_message*), la cual, codifica la latitud y longitud y lo transmite a través la simulación del interfaz radio. Posteriormente, cuando reciba la repuesta se llama a la rutina de recepción (*receive_message*), que descodifica el mensaje recibido, calcula la distancia entre POI y usuario (*haversine*) y los imprime junto con la latitud y longitud de los POI en la ventana de output.

```
#include "mbed.h"
#include "mbed_trace.h"
#include "mbed_events.h"
#include "LoRaWANInterface.h"
#include "Sht31.h"
#include "SX1276_LoRaRadio.h"
#include <cmath>

// Parámetros del dispositivo creado en TTS
static uint8_t DEV_EUI[] = { 0x12, 0x34, 0x56, 0x78, 0x91, 0x23, 0x45,
0x67 };
static uint8_t APP_EUI[] = { 0x12, 0x34, 0x56, 0x78, 0x91, 0x23, 0x45,
0x67 };
static uint8_t APP_KEY[] = { 0xAA, 0x0D, 0x70, 0xC3, 0xFE, 0xE8, 0x76,
0xBC, 0xEB, 0x9A, 0x3F, 0xB9, 0x83, 0x46, 0xF1, 0xA8 };

struct out1 {
    double long2;
    double lat2;
    double dist;
};
// Longitud y latitud a comprobar
static double float_longitude = -15.429581;
static double float_latitude = 28.144881;

// Puerto donde se transmite y recibe
#define MBED_CONF_LORA_APP_PORT     15

//  Periféricos (módulo de LoRa radio y botón)
SX1276_LoRaRadio radio(D11, D12, D13, D10, A0, D2, D3, D4, D5, D8, D9,
NC, NC, NC, NC, A4, NC, NC);
InterruptIn btn(BUTTON1);

static EventQueue ev_queue;

static LoRaWANInterface lorawan(radio);

static lorawan_app_callbacks_t callbacks;

static void lora_event_handler(lorawan_event_t event);

// Cálculo de distancia entre cruce y posición dispositivo usando
Haversine
static double haversine(double lat1, double lon1,
                        double lat2, double lon2)
```

```
    {
        // distancia entre latitudes y longitudes
        double dLat = (lat2 - lat1) *
                      M_PI / 180.0;
        double dLon = (lon2 - lon1) *
                      M_PI / 180.0;

        // Conversión a radianes
        lat1 = (lat1) * M_PI / 180.0;
        lat2 = (lat2) * M_PI / 180.0;

        // Cálculo distancia
        double a = pow(sin(dLat / 2), 2) +
                   pow(sin(dLon / 2), 2) *
                   cos(lat1) * cos(lat2);
        double rad = 6371;
        double c = 2 * asin(sqrt(a));
        return rad * c;
    }
// Envío de mensaje sobre LoRaWAN
static void send_message() {

    uint8_t tx_buffer[10] = { 0 };
    int32_t lat = float_latitude * 10000000;
    int32_t lon = float_longitude * 10000000;

    // Construcción del mensaje a enviar
    tx_buffer[0] = lat;
    tx_buffer[1] = lat >> 8;
    tx_buffer[2] = lat >> 16;
    tx_buffer[3] = lat >> 24;

    tx_buffer[4] = lon;
    tx_buffer[5] = lon >> 8;
    tx_buffer[6] = lon >> 16;
    tx_buffer[7] = lon >> 24;
    tx_buffer[8] = 5;
    tx_buffer[9] = 9;

    int packet_len = sizeof( tx_buffer);

    printf("Enviando %d bytes: ", packet_len);

    for (uint8_t i = 0; i < packet_len; i++) {
    printf("%02x ", tx_buffer[i]);
    }
    printf("\n");

    int16_t retcode = lorawan.send(MBED_CONF_LORA_APP_PORT, tx_buffer,
packet_len, MSG_UNCONFIRMED_FLAG);

    if (retcode < 0) {
        retcode == LORAWAN_STATUS_WOULD_BLOCK ? printf("send - duty
cycle violation\n")
                : printf("send() - Error code %d\n", retcode);
        return;
    }

    printf("%d bytes preparados para transmitir \n", retcode);
}
```

```
int main() {


    printf("Presione Botón 1 para enviar los datos de posición!\n");

    // Se habilita la salida del traceo
    mbed_trace_init();

    if (lorawan.initialize(&ev_queue) != LORAWAN_STATUS_OK) {
        printf("Fallo en la Inicialización de LoRa \n");
        return -1;
    }

    // Cuando se pulsa el botón se procede al envío del mensaje
    btn.fall(ev_queue.event(&send_message));


    callbacks.events = mbed::callback(lora_event_handler);
    lorawan.add_app_callbacks(&callbacks);

    // Se deshabilita adaptive data rating
    if (lorawan.disable_adaptive_datarate() != LORAWAN_STATUS_OK) {
        printf(" No se ha podido deshabilitar Adaptive Data Rate\n");
        return -1;
    }

    lorawan_connect_t connect_params;
    connect_params.connect_type = LORAWAN_CONNECTION_OTAA;
    connect_params.connection_u.otaa.dev_eui = DEV_EUI;
    connect_params.connection_u.otaa.app_eui = APP_EUI;
    connect_params.connection_u.otaa.app_key = APP_KEY;
    connect_params.connection_u.otaa.nb_trials = 3;

    lorawan_status_t retcode = lorawan.connect(connect_params);

    if (retcode == LORAWAN_STATUS_OK ||
        retcode == LORAWAN_STATUS_CONNECT_IN_PROGRESS) {
    } else {
        printf("Error de conexión, código = %d\n", retcode);
        return -1;
    }

    printf("Estableciendo conexión ...\r\n");

    ev_queue.dispatch_forever();

    return 0;
}

// Cuando se recibe el evento RX_DONE se llama a la función de
tratamiento del mensaje recibido
static void receive_message()
{
    uint8_t rx_buffer[150] = { 0 };
    int16_t retcode;
    retcode = lorawan.receive(MBED_CONF_LORA_APP_PORT, rx_buffer,
                              sizeof(rx_buffer),
MSG_CONFIRMED_FLAG|MSG_UNCONFIRMED_FLAG);

    if (retcode < 0) {
```

```
        printf("Código de error en la recepción %d\n", retcode);
        return;
    }

    printf("Datos recibido por puerto %d (número de bytes %d): ",
MBED_CONF_LORA_APP_PORT, retcode);

    for (uint8_t i = 0; i < retcode; i++) {
        printf("%02x ", rx_buffer[i]);
    }
    printf("\n");
    //Descodificación de datos recibidos
    int nb = sizeof(rx_buffer);
    int nstop = rx_buffer[retcode-1] & 0x0f;
    int nceda = (rx_buffer[retcode-1] & 0xf0) >>4;
    int nsema = rx_buffer[retcode-2] & 0x0f;
    int nesta = (rx_buffer[retcode-2] & 0xf0) >> 4;
    struct out1 stopr[nstop];
    struct out1 cedar[nceda];
    struct out1 semar[nsema];
    struct out1 estar[nesta];
    int j,k,l = 0;
    // presentación de los datos recibidos y cálculo de distancia
    for (int i = 0; i < (retcode-2)/8; i++ ){

        double latr = (rx_buffer[3+8*i] << 24 | rx_buffer[2+8*i] << 16
        | rx_buffer[1+8*i] << 8| rx_buffer[0+8*i]);
        double lonr = (rx_buffer[7+8*i] << 24 | rx_buffer[6+8*i] << 16
        | rx_buffer[5+8*i] << 8| rx_buffer[4+8*i]) ;
        double distr = haversine(float_latitude,
        float_longitude,latr/10000000,lonr/10000000);
        if ( (i < nesta) & (nesta > 0)) {
            estar[i].long2 = lonr/10000000;
            estar[i].lat2 = latr/10000000;
            estar[i].dist = distr*1000;
            printf("est. cercana: long.: %f, lat.: %f ",
            estar[i].long2,estar[i].lat2);
        }
        else if (( i < (nesta+nstop)) & (nstop >0)) {

            stopr[j].long2 = lonr/10000000;
            stopr[j].lat2 = latr/10000000;
            stopr[j].dist = distr*1000;
            printf("stop:         long.: %f, lat.: %f ",
            stopr[j].long2,stopr[j].lat2);
            j += 1;
        }
        else if (( i < (nesta+nstop+nsema)) & (nsema > 0)) {

            semar[k].long2 = lonr/10000000;
            semar[k].lat2 = latr/10000000;
            semar[k].dist = distr*1000;
            printf("semaforo:     long.: %f, lat.: %f ",
            semar[k].long2,semar[k].lat2);
            k += 1;
        }
        else if (( i < (nstop+nsema+nceda+nesta)) & (nceda > 0)) {

            cedar[l].long2 = lonr/10000000;
            cedar[l].lat2 = latr/10000000;
```

```
            cedar[l].dist = distr*1000;
            printf("ceda el paso: long.: %f, lat.: %f ",
            cedar[l].long2,cedar[l].lat2);
            l += 1;
        }
        printf ("distancia: %.2lf m \n",distr*1000);
    }
}

// Control de eventos
static void lora_event_handler(lorawan_event_t event) {
    switch (event) {
        case CONNECTED:
            printf("Conexión realizada con éxito\n");
          lorawan.set_datarate(5); // Spreading factor
            break;
        case DISCONNECTED:
            ev_queue.break_dispatch();
            printf("Desconexión realizada con éxito\n");
            break;
        case TX_DONE:
            printf("Mensaje enviado al servidor\n");
            break;
        case TX_TIMEOUT:
        case TX_ERROR:
        case TX_CRYPTO_ERROR:
        case TX_SCHEDULING_ERROR:
            printf("Error en transmisión - EventCode = %d\n", event);
            break;
        case RX_DONE:
            printf("Mensaje recibido del servidor\n");
            receive_message();
            break;
        case RX_TIMEOUT:
        case RX_ERROR:
            printf("Error en la recepción - código = %d\n", event);
            break;
        case JOIN_FAILURE:
            printf("OTAA falla - compruebe Keys\n");
            break;
        default:
            MBED_ASSERT("Evento desconocido ");
    }
}
```

III.A. Código PHP “/bike/”

Código escrito en PHP e instalado en el servidor de aplicaciones que guarda la información recibida desde el servidor de red de TTC con formato JSON. A continuación, ejecuta la aplicación “*bike.py*”, la cual analiza los datos útiles recibido desde el servidor de red de TTN.

```
<html>
 <head>
  <title>BSS LoRaWAN uplink</title>
 </head>
 <body>
    <?php

         $file = fopen("./bike.txt", "w");
         /* Escritura en fichero de mensaje de uplink recibido*/
         fwrite($file, file_get_contents('php://input'));
         /* Ejecución de consulta, creación de Downlink y envío de
         mensaje */
         $comando ="python3 bike.py ";
         exec($comando);
         fclose($file);

    ?>
 </body>
</html>
```

III.B. Código PHP “/message/”

Código escrito en PHP que almacena los mensajes sin carga útil de uplink, recibido desde el servidor de red de TTN.

```
<html>
 <head>
  <title>BSS LoRaWAN mensajes </title>
 </head>
 <body>
    <?php

         $file = fopen("./message.log", "a");
         $json = json_decode(file_get_contents('php://input'));
         /* Anexar mensaje recibido en log */
         fwrite($file,json_encode($json, JSON_PRETTY_PRINT));
         fclose($file);

    ?>
 </body>
</html>
```

IV. Código aplicación Python.

Aplicación escrita en Python que se ejecuta en el servidor de aplicaciones. Está aplicación extrae el mensaje con carga útil recibido a través del servidor de aplicaciones procedente del simulador. A continuación, lo descodifica y convierte en latitud y longitud y busca en la base de datos el código identificativo de la vía por que corresponde a la posición geográfica recibida. A continuación, busca la posición de los cruces señalizado de la vía y de los estacionamientos cercanos. Una vez identificado los 5 POI a transmitir, codifica el paquete y genera el mensaje a transmitir vía HTTP al servidor de red de TTN y lo transmite incluyendo el código de identificación del simulador y la clave de la aplicación en el servidor de aplicaciones.

```
from pymongo import MongoClient
import pprint
import json
from collections import defaultdict
from struct import *
import sys
import base64
from datetime import datetime
import requests
from requests.structures import CaseInsensitiveDict


ttn = open("bike.txt", "r")
f = open ("bikelog.log","a");
ttnr = json.loads(ttn.read());
f.write(str(datetime.now())+' ; mensaje recibido:
'+str(ttnr['uplink_message']['frm_payload'])+"\n");

downlink = defaultdict(list);
downlink = {"esta" : [], "stop" : [] , "sem" : [] , "ceda" : [] };
mult = 10000000;


# Descodificación mensaje recibido
cod = base64.b64decode(ttnr['uplink_message']['frm_payload']);
f.write(str(datetime.now())+' ; Bytes recibidos:'+ str(cod) + "\n");
decodif = unpack('ii',cod[0:-2]);

# Latitud y Longitud recibidas
lat = decodif[0]/mult;
long = decodif[1]/mult;
f.write(str(datetime.now())+' ; Latitud y Longitud :'+ str(lat) + ','
+ str(long) + "\n");
paquete = b'';


# Conexión a base de datos GIS
client = MongoClient();
db = client.osm;
vias = db.ways;
nodos =db.nodes;
# Busqueda de ID de vía por la que se circula
busqueda = vias.find_one({"loc": {"$near": [lat,long]}, "$and": [
{"ky" : "highway"}, {"ky" : "name"}]});
```

```
# Nombre de la vía
for l in busqueda['tg'] :
        if l[0] == 'name' :
                nvia = l[1];
                f.write(str(datetime.now())+' ; Via: ' + nvia + "\n");

# Busqueda de cruces
for n in busqueda["nd"] :
        cruce = nodos.find_one({"_id" : n}, {"tg" : 1, "_id" : 0});
        if cruce :
                for elcruce in cruce["tg"] :
                        if elcruce[0] == 'highway' :
                             cruce1 = elcruce[1];
                             if cruce1 == "stop" : # Stop
                                 stop = nodos.find_one({"_id" : n},
                                  {"loc" : 1, "_id" : 0});
                                  downlink["stop"].append(stop["loc"]);

                            if cruce1 == "traffic_signals" :#Semaforos
                                 sem = nodos.find_one({"_id" : n},
                                 {"loc" : 1, "_id" : 0});
                                  downlink["sem"].append(sem["loc"]);

                            if cruce1 == "give_way" : # Ceda el paso
                                 ceda = nodos.find_one({"_id" : n},
                                 {"loc" : 1, "_id" : 0});
                                  downlink["ceda"].append(ceda["loc"]);

st = len(downlink["stop"]);
se = len(downlink["sem"]);
ce = len(downlink["ceda"]);
est = 5 - (st + se + ce);

# Busqueda de estaciones de estacionamiento
cod = bytes([(est << 4) | se]) + bytes([(ce << 4) | st]) ;
for esta  in nodos.find({"loc": {"$near": [lat,long]},"ky" :
{"$exists" : "true" , "$in" : ["amenity"]}, "tg" :
["network","Sitycleta"]},{"loc" : 1, "_id" : 1}).limit(est) :
downlink["esta"].append(esta["loc"]);
f.write(str(datetime.now())+' ; Resultado busqueda: '+ str(Downlink) +
"\n");

# Creación de paquete Downlink
if downlink["esta"] :
        for n in downlink["esta"] :
                paquete = paquete +
pack('ii',int(n[0]*mult),int(n[1]*mult));
if downlink["stop"] :
        for n in downlink["stop"] :
                paquete = paquete +
pack('ii',int(n[0]*mult),int(n[1]*mult));
if downlink["sem"] :
        for n in downlink["sem"] :
                paquete = paquete +
pack('ii',int(n[0]*mult),int(n[1]*mult));
if downlink["ceda"] :
        for n in downlink["ceda"] :
                paquete = paquete +
pack('ii',int(n[0]*mult),int(n[1]*mult));
```

```
enviar = paquete +cod;

# Codificación del paquete a enviar
enviar_base64 = base64.b64encode(enviar);
f.write (str(datetime.now())+' ; Bytes a enviar: '+ str(enviar) +
"\n");
f.write (str(datetime.now())+' ; Mensaje codificado: '+
str(enviar_base64)[2:-1] + "\n");

headers = CaseInsensitiveDict();
# Cabecera HTTP POST
headers["Authorization"] = "Bearer
NNSXS.5RLXOPFCG5ZV7NRZABHRR5IJDHFZBV46H3RSYNY.WQHWYSXEXJGKLTB24RJS53ZY
YCSZGDU3AJNFUYEXTL32JMCQMHSA";

# URL HTTP POST
url =
"https://eu1.cloud.thethings.network/api/v3/as/applications/test-
bici/webhooks/bike-app-server/devices/"+
ttnr['end_device_ids']['device_id']+"/down/push";

# Contenido HTTP POST
datos = '{"downlinks":[{"frm_payload":"' + str(enviar_base64)[2:-1] +
'","f_port":' +
str(ttnr['uplink_message']['f_port'])+',"priority":"NORMAL"}]}';

f.write (str(datetime.now())+' ; enlace a enviar: '+ str(url) + "\n");
f.write (str(datetime.now())+' ; Json a enviar: '+ str(datos) + "\n");
client = requests.Session();

# Envío mensaje HTTP POST
result = client.post(url,data=datos,headers=headers);
f.write (str(datetime.now())+' ; resultado envío: '+
str(result.status_code) + ' ' + str(result.reason) + "\n");

f.close();
ttn.close();
```