Estudio y desarrollo de un equipo interfaz de red inalámbrica (WSN) con red pública basado en plataforma Raspberry Pi

Máster Universitario en Ingeniería de Sistemas Electrónicos
Trabajo Final de Máster

Autor:
Arnau Albert García
Directores:
Dr. Francisco José Ballester Merelo
Dr. Marcos Antonio Martínez Peiró

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

Resumen

Las redes de sensores inalámbricas han sido objeto de desarrollo de multitud de empresas y grupos de investigación. El ámbito de aplicación donde se puede usar este tipo de tecnología es multitudinario, ya sea en agricultura de precisión, domótica, industria, Smart cities…

El proyecto, o trabajo fin de máster, se ha realizado en el Instituto de investigación para Imagen Molecular (I3M) de la Universidad Politécnica de Valencia (UPV).

El objetivo del proyecto es el desarrollo de un equipo interfaz de red inalámbrica, para redes de sensores inalámbricos (WSN), con red pública basado en plataforma Raspberry Pi. Sin embargo, este es uno de los dispositivos que conforman el sistema completo.

El sistema está compuesto por un servidor central, un gateway o puerta de enlace (objeto de desarrollo) y nodos sensores. Los nodos tienen la capacidad de integrar distintos sensores, que medirán las variables físicas que se consideren de interés para obtener un seguimiento en tiempo real de las circunstancias para finalmente tratar esa información o actuar en consecuencia (humedad de la tierra, temperatura, velocidad del viento, etc). Estos nodos se comunicarán con el gateway mediante radiofrecuencia basada en el IEEE 802.15.4. El gateway mantendrá el roll de coordinador de la red. A su vez, es el encargado de establecer la comunicación con el servidor central y transmitir, interpretar y almacenar la información proporcionada por los sensores de los nodos. El usuario, mediante una aplicación web, será capaz de monitorizar, actuar y obtener históricos de las medidas así como cambiar parámetros y configurar la red.

El proyecto aborda satisfactoriamente el objetivo inicial. Los métodos de desarrollo que se han utilizado así como las herramientas necesarias y el banco de pruebas realizado están descritos y especificados en la memoria.

Resum

Les xarxes de sensors inalàmbrics han sigut objecte de desenvolupament a múltituds d'empreses i grups d'investigació. L'àmbit d'ús on es podem utilizar aquest tipus de tecnología és multitudinàri, com en l'agricultura de presició, domótica, industria, Smart cities…

El projecte, o treball final de màster, s'ha realitzat a l'Institut d'Investigació per a Imatge Mol·lecular (I3M) de la Universitat Politècnica de València (UPV).

L'objectiu del projecte és el desenvolupament d'un equip interfície de red inalàmbrica, per a reds de sensors inalàmbrics (WSN), amb red pública basat en plataforma Raspberry Pi. Però, aquest és un dels dispositius que conformen el sistema complet.

El sistema està compost per un servidor central, un gateway o porta d'enllaç (objecte de desenvolupament) i nodes sensors. Els nodes tenen la capacitat d'integrar diferents sensors, que mesuraran les variables físiques que es consideren d'interés per obtindre un seguiment en temps real de les circumstàncies per a finalment tractar eixa información o actuar en conseqüència (humitat del sòl, temperatura, velocitat del vent, ètc.). Aquests nodes es comunicaran amb el gateway mitjançant radiofreqüència basada amb l' IEEE 802.15.4. El gateway mantindrà el roll de coordinador de la red. Al mateix temps, és l'encarregat d'establir comunicació amb el servidor central i transmetre, interpretar i emmagatzenar la informació proporcionada per els sensors dels nodes. L'usuari, mitjanzant una aplicació web, podrà monitoritzar, actuar i obtindre històrics de les messures així com canviar paràmetres i configurar la red.

El projecte aborda satisfactòriament l'objectiu inicial. Els mètodes de desenvolupament que s'han utilitzat així com les ferramentes necessàries i el banc de proves realizat están descrits i especificats a la memoria.

Abstract

Lots of companies and research institutions are currently investigating the properties and functions of Wireless Sensor Networks. This technonology has many applications such as agriculture, automation in the home, industry and production and Smart cities to name a few.

My project was developed and researched in the Investigation Institute for Molecular Imaging (I3M) in the Universidad Politécnica de Valencia (UPV).

The objective of this project was to develop an interface for wireless networks. The wireless network chosen was a Wireless Sensor Network (WSN) with Public access based on the Raspberry Pi platform. Nevertheless the Raspberry Pi forms part of the complete system.

The system is made of a main server, a gateway or access point (research objective) and the sensor nodes. The nodes have the capacity to integrate varying sensors which will detect and measure the physical parameters of interest in real time. The information gathered (humidity, temperature, wind speed, etc...) is then processed so that the system con react accordingly. These nodes communicate with the gateway using radio frequencies in based on the IEEE standard 802.15.4. The gateway serves as a coordinator for the system. This role includes establishing a connection with the server, transmitting, interpreting and storing information received from the sensors. The user then can with the use of a web app monitor in real time, control certain parameters, configure the network and obtain the information received from the sensor nodes.

This project has successfully completed its objective. The methods of research and development that were used along with the tools and the history of tests described above are all specified in the memory bank.

ÍNDICE

Índice de ilustraciones

Índice de tablas

1. INTRODUCCIÓN

El presente proyecto tiene como objetivo el desarrollo de una puerta de enlace o gateway de bajo consumo que sea capaz de comunicarse con un servidor central así como con una red de nodos sensores.

La comunicación entre la puerta de enlace (gateway en adelante) y el servidor se realizará mediante comunicación GPRS o GSM, la cual dota al sistema de versatilidad y movilidad necesaria para que el dispositivo pueda ser instalado en cualquier lugar donde se disponga de cobertura. A su vez la comunicación con la red de nodos se realizará mediante radiofrecuencia (en adelante RF).

Se trata de una actualización del gateway ya existente y desarrollado por Balmart S.L. Principalmente la nueva versión tendrá la misma funcionalidad al existente, no obstante contará con una serie de ventajas tanto presentes como para líneas futuras de desarrollo con el mismo dispositivo.

En este documento se abordarán las distintas etapas durante el desarrollo del gateway, se realizará una explicación exhaustiva del sistema RfreeNet, las pruebas de verificación de funcionamiento, así como las herramientas utilizadas.

1.1. Entorno de trabajo y justificación del proyecto

El proyecto se ha desarrollado en el Instituto para Imagen Molecular (I3M) situado en el edificio 8B de la Universidad Politécnica de Valencia, concretamente en el cubo azul (entrada N).

Clipnode Solutions S.L. es la empresa que cuenta con los derechos de venta, distribución e instalación de estos sistemas de redes de sensores inalámbricos y telemando. Su principal campo de aplicación es la agricultura de precisión, monitorización ambiental y soluciones industriales tanto para monitorización como actuación.

El proyecto se ha desarrollado teniendo en cuenta la existencia de una versión anterior de gateway, con lo que ya estaban especificadas tanto las funcionalidades finales, como las tramas de comunicación a implementar.

El proyecto se plantea por la necesidad de actualizar el equipo. Los motivos por el cual el equipo ha de ser actualizado son varios:

- Obsolescencia de los componentes.
- Incompatibilidad con nuevas plataformas.
- Incapacidad de recuperar datos durante una pérdida de conexión.
- Imposibilidad de actualizar el firmware de manera remota.

Las ventajas que se obtienen al actualizar el equipo en la nueva plataforma son los siguientes:

- Mayor potencia de cómputo.
- Sistema operativo embebido en el nuevo dispositivo y todas las ventajas que esto conlleva.
- Coste relativamente bajo del nuevo hardware.
- Solucionar problemas durante pérdidas de conexión.
- Posibilidad de actualización del firmware de manera remota.
- Posibilita líneas futuras de trabajo entorno al sistema sin altos costes.

1.2. Arquitectura del sistema: red de sensores inalámbricos (WSN)

Una red de sensores inalámbricos o WSN por sus siglas en inglés (Wireless Sensor Network) es una red de computadores/microcontroladores/ordenadores equipados con uno o varios sensores y/o actuadores que colaboran en una tarea común. A estos dispositivos se les denomina nodos. Estos nodos se comunican entre sí o con un dispositivo central de manera inalámbrica utilizando como medio físico el aire. A este dispositivo central se le suele denominar coordinador.

En este caso, el coordinador central es el gateway que será el encargado de transmitir la información del servidor a los nodos (y viceversa) mediante GPRS y a los nodos mediante RF.

El sistema está formado por tres tipos de dispositivos principales:

Servidor web: encargado de la gestión del equipo de forma remota y del almacenamiento en la base de datos de las medidas adquiridas por el Gateway para su posterior análisis y monitorización.

Gateway: dispositivo programable encargado de la comunicación con los nodos mediante RF así como la comunicación con el servidor central mediante GPRS.

Nodos: son las unidades de sensorización que enviarán mediante RF, directa o indirectamente, los distintos datos sensados al Gateway. Indirectamente porque la ruta que seguirá la información que se desee enviar al Gateway dependerá de la distancia que haya entre estos dos dispositivos. Puede ser directamente de nodo a Gateway o indirectamente de nodo2 a nodo1 y a Gateway.

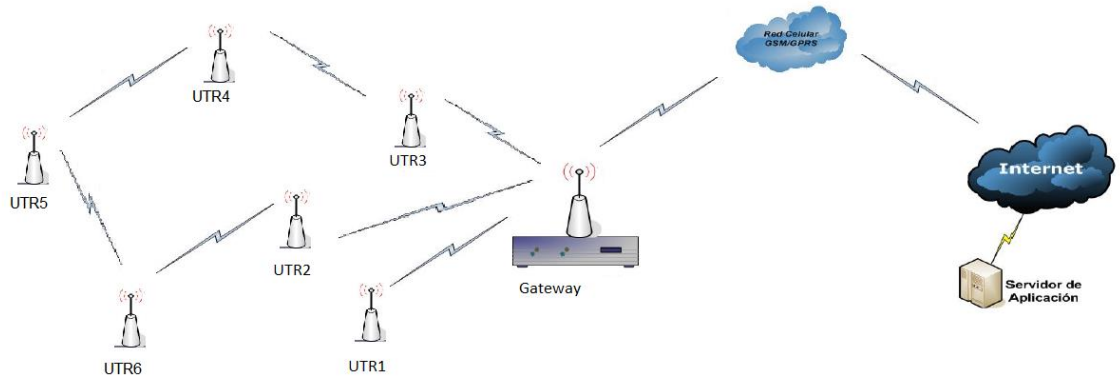


Figura 1. Arquitectura del sistema (WSN)

En la figura 1 se aprecia que los nodos UTR4, UTR5 y UTR6 no tienen comunicación directa con el Gateway, sino que mantienen comunicación con los nodos cercanos para que los datos sean transferidos hasta el Gateway utilizando a los nodos necesarios como pasarela. A su vez, si el nodo UTR4 dejará de funcionar por cualquier motivo el nodo UTR5 podría comunicarse con el Gateway a través del nodo UTR6, es decir, automáticamente detectaría sus vecinos cercanos y cuál es el conveniente para realizar la comunicación.

1.3. Objetivos del proyecto

El proyecto se centrará en el desarrollo del nuevo gateway siguiendo las especificaciones para conseguir todas las funcionalidades de las que disponía la versión anterior. Además se dotará al nuevo dispositivo de dos nuevas funcionalidades de las que carecía la versión anterior. Para ello debemos de asegurar que cumple con los siguientes puntos:

- Comunicaciones: será capaz de establecer comunicación tanto con el servidor como con los nodos.

- Recuperación ante desconexión: deberá de ser capaz de transmitir las tramas de los nodos que se han recibido durante una pérdida de conexión con el servidor en cuanto recupere dicha conexión.
- Lectura: el Gateway también actúa como adquisidor de datos mediante sensores.
- Base de datos: almacenará todas las tramas de comunicación en las que se vea implicado.
- Actualización remota: se dotará al sistema de la capacidad de actualizar su firmware de manera remota.

1.4. Metodología de trabajo

En este apartado se abordarán tanto la metodología de trabajo en el desarrollo del proyecto como la organización para su ejecución. Así pues se pueden diferenciar distintos puntos o acciones que marcarán hitos en el desarrollo del proyecto:

- **Comunicación TCP-IP:** en primer lugar se establecerá comunicación con el servidor mediante el protocolo TCP-IP.
- **Configuración:** el gateway almacenará unos valores que le serán proporcionados desde la web. Éste deberá almacenarlos y ser capaz de actualizarlos/sustituirlos siempre que así decida el usuario.
- **Comunicación RF:** la comunicación con los nodos se realizará mediante radiofrecuencia, aunque desde el punto de vista de nuestro dispositivo se tratará de una comunicación serie.
- **Configuración nodos:** el gateway dispondrá de unos archivos *json*, los cuales almacenarán todos los datos relativos a los distintos nodos que tenga dados de alta como la mac, la ip, el número de sensores activos, etc.
- **Enlace comunicación servidor/nodos:** se realizará la interconexión entre la comunicación RF con el servidor y viceversa.
- **Banco de pruebas y revisiones:** finalmente se tratará de estresar al máximo el sistema desde la web para comprobar que no se cuelga en ningún punto, ni por saturación de órdenes enviadas ni por desconocimiento de ninguna de ellas. También simularemos pérdidas de conexión al servidor para comprobar que envía los datos almacenados.

El desarrollo del proyecto se ha planificado alrededor de 4 tareas principales que a su vez se desglosarán en subtareas. Las tareas principales son las siguientes:

- Especificaciones de comunicación con el servidor.
- Comunicación con la red de nodos.
- Implementación en Raspberry Pi.
- Banco de pruebas.

A continuación se muestra el diagrama de Gantt de las tareas principales y las distintas subtareas:

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Especificaciones de comunicación con servidor**	**23 días**	**jue 15/10/15**	**lun 16/11/15**	
8		**Comunicación red de nodos**	**20 días**	**mar 17/11/15**	**lun 14/12/15**	
13		**Implementación en Raspberry Pi**	**20 días**	**mar 15/12/15**	**lun 01/02/16**	
21		**Banco de pruebas**	**19 días**	**mar 02/02/16**	**vie 26/02/16**	

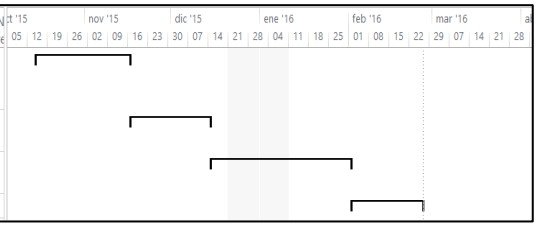


Tabla 1. Diagrama de Gantt, tareas principales

Tarea1: Especificaciones y comunicación con el servidor

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Especificaciones de comunicación con servidor**	**23 días**	**jue 15/10/15**	**lun 16/11/15**	
2		Implementación servidor de prueba	2 días	jue 15/10/15	vie 16/10/15	
3		Prueba de comunicación	1 día	lun 19/10/15	lun 19/10/15	2
4		Implementar tramas gateway-servidor	5 días	mar 20/10/15	lun 26/10/15	3
5		Prueba de transmisión y	3 días	mar 27/10/15	jue 29/10/15	4
6		Envío de tramas servidor central	5 días	vie 30/10/15	jue 05/11/15	5
7		Prueba de cambio de variables gateway desde servidor	7 días	vie 06/11/15	lun 16/11/15	6

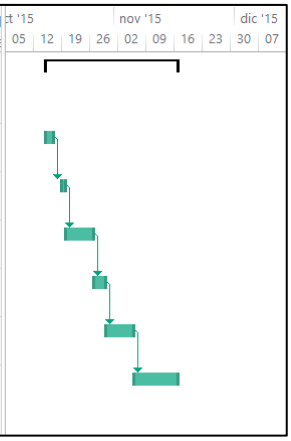


Tabla 2. Diagrama de Gantt, tarea 1

Tarea 2: Comunicación red de nodos

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Especificaciones de comunicación con servidor**	**23 días**	**jue 15/10/15**	**lun 16/11/15**	
8		**Comunicación red de nodos**	**20 días**	**mar 17/11/15**	**lun 14/12/15**	
9		Comunicación serie	2 días	mar 17/11/15	mié 18/11/15	7
10		Implementar tramas gateway-nodos	5 días	jue 19/11/15	mié 25/11/15	9
11		Prueba comunicación gateway-nodos	10 días	jue 26/11/15	mié 09/12/15	10
12		Revisión tramas gateway-nodos	3 días	jue 10/12/15	lun 14/12/15	11

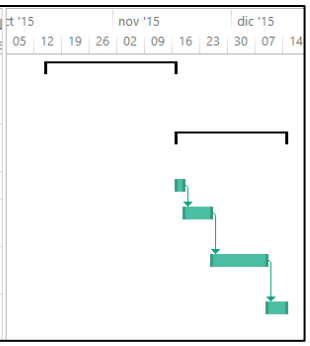


Tabla 3. Diagrama de Gantt, tarea 3

Tarea 3: Implementación en Raspberry Pi.

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Especificaciones de comunicación con servidor**	**23 días**	**jue 15/10/15**	**lun 16/11/15**	
8		**Comunicación red de nodos**	**20 días**	**mar 17/11/15**	**lun 14/12/15**	
13		**Implementación en Raspberry Pi**	**20 días**	**mar 15/12/15**	**lun 01/02/16**	
14		Bucle principal	3 días	mar 15/12/15	jue 17/12/15	12
15		Prueba tramas Servidor-->gateway-->	2 días	vie 18/12/15	lun 11/01/16	14
16		Prueba tramas nodos-->gateway-->S	2 días	mar 12/01/16	mié 13/01/16	15
17		Instalación minicom lectura serie nodos	1 día	jue 14/01/16	jue 14/01/16	16
18		Corrección de errores	5 días	vie 15/01/16	jue 21/01/16	17
19		Implementar json almacenamiento de variables	2 días	vie 22/01/16	lun 25/01/16	18
20		Lectura de sensores	5 días	mar 26/01/16	lun 01/02/16	19

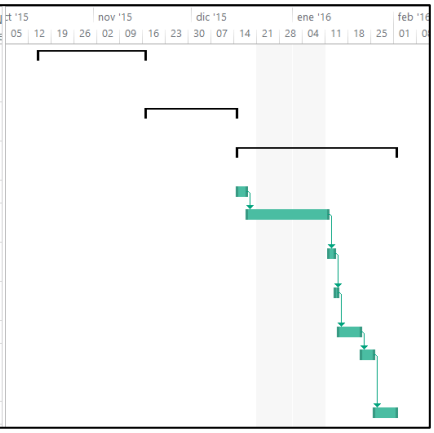


Tabla 4. Diagrama de Gantt, tarea 3

Tarea 4: Banco de pruebas.

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Especificaciones de comunicación con servidor**	**23 días**	**jue 15/10/15**	**lun 16/11/15**	
8		**Comunicación red de nodos**	**20 días**	**mar 17/11/15**	**lun 14/12/15**	
13		**Implementación en Raspberry Pi**	**20 días**	**mar 15/12/15**	**lun 01/02/16**	
21		**Banco de pruebas**	**19 días**	**mar 02/02/16**	**vie 26/02/16**	
22		Configuración sensore	4 días	mar 02/02/16	vie 05/02/16	20
23		Adición de nodos a la red	5 días	lun 08/02/16	vie 12/02/16	22
24		Prueba de estrés al gateway	3 días	lun 15/02/16	mié 17/02/16	23
25		Corrección de errores	4 días	jue 18/02/16	mar 23/02/16	24
26		Prueba de estrés al gateway	3 días	mié 24/02/16	vie 26/02/16	25

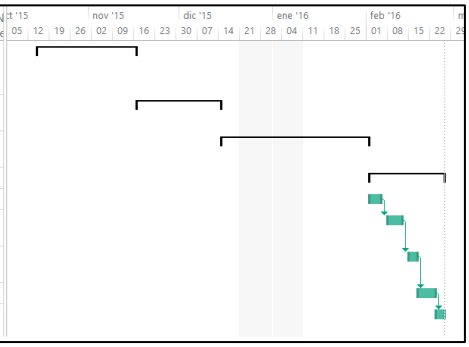


Tabla 5. Diagrama de Gantt, tarea 4

2. ESTADO DEL ARTE

En este apartado, el lector, obtendrá una visión global de los distintos dispositivos, que podríamos enmarcar en la misma familia que nuestro Gateway, existente en el mercado actual. También se mostrarán distintos tipos de sensores que se utilizan en este tipo de aplicaciones así como un análisis de las diferentes comunicaciones inalámbricas existentes, centrándonos en las que utiliza nuestro sistema.

2.1. Estaciones remotas autónomas de sensado

Una estación remota es aquel dispositivo o conjunto de dispositivos interconectados, que además de capturar, almacenar, mantener y gestionar de forma autónoma los datos de medición, de transmitir esos datos cuando se le solicite, de generar eventos y alarmas cuando así se requiera, unifica los formatos de todos esos datos permitiendo un tratamiento centralizado de toda la información.
Estos equipos han sido implementados para una gran diversidad de aplicaciones como control de regadíos, caracterización del suelo, seguridad, así como factores meteorológicos. Dentro de esta gran variedad de productos, nos centraremos en los que integran las cualidades de adquisición, procesado, almacenamiento y envío en un único dispositivo, ya que sus características son las que más se asemejan al proyecto que se desea realizar.

2.1.1. Lacroix Sofrel Cell-Box-Data

Lacroix Sofrel, es una empresa especializada en el diseño y fabricación de productos de televigilancia y de telegestión (control a distancia de instalaciones técnicas basándose en tecnologías informáticas, electrónicas y telecomunicaciones). El grupo Lacroix tiene sede en Italia, España y Francia.

Uno de sus productos de telegestión es el CellBox-Data. La CellBox-Data es una estación remota autónoma que comunica por GSM diseñada para la vigilancia de lugares desprovistos de fuente de energía y de comunicación. Está especialmente adaptada para leer a distancia los contadores generales y vigilar el buen

funcionamiento de la red de distribución de agua potable. También efectúa el registro de mediciones y transmite la información al Puesto Central.

Entre sus aplicaciones destacan:

- Sectorización de redes (análisis por tramos de redes de distribución de aguas)
- Telegestión de lugares aislados (vigilancia de lugares desprovistos de energía)
- Búsqueda de fugas

Entre las funciones principales el fabricante nombra las siguientes:

- Contadores
- Medidas
- Gestión de umbrales de alerta
- Comunicación GSM Data

Figura 2. Cell-Box de Lacroix

Las principales características técnicas del dispositivo son las siguientes:

- Presentación
  - Caja de plástico estanca IP67, sumergible
  - Dimensiones: A180 x L180 xP66 mm
  - Cables d conexión de 2 m integrados
- Entradas
  - 4 entradas digitales
  - 1 Entrada analógica opcional para captador de 4-20 mA
- Entorno
  - Temperaturas de utilización de -10 a 50 ^{o}C

- Alimentación
  - Por pila
  - Autonomía de 4 años
- Comunicación
  - Módem GSM integrado y alimentado con pila interna
  - Conexión RS232 para conexión de un PC

2.1.2. Hermes LC2 de Microcom

Microcom es una empresa Española que desarrolla su actividad principal en proporcionar soluciones de telecontrol y telemetría empleando redes de telefonía móvil GSM como medio de comunicación.
Dentro de su gama de productos de bajo coste cabe destacar el Hermes LC2, un dispositivo de telecontrol y datalogger GSM/GPRS. Este dispositivo se alimenta directamente a 220V e incorpora una batería LiPo interna que le permite funcionar durante varias horas sin alimentación externa.
Entre las características principales que el fabricante proporciona se encuentran las siguientes:

- Generales:

Alimentación	**220 VAC**
Consumo nominal	0.5 W
Consumo máximo	5 W
Procesador principal	ARM7-RISC
Procesador secundario	PIC
Memoria de programa	Flash de 256 KB
Memoria de datos	64 KB
Firmware	Multitarea protegido por Watchdog
GSM	Siemens MC55
Temperatura de operación	0 ºC a 50 ºC

Tabla 6. Especificaciones generales del Hermes LC2

- Histórico:

Profundidad de histórico	**>40000 registros**
Memoria de histórico	512 KB flash
Contadores en entradas digitales	contador totalizador de 32 bits y un contador parcial de 16 bits (caudalímetro configurable)

Tabla 7. Especificaciones del histórico del Hemes LC2

- Entradas / salidas:

Entradas digitales	**8**	**Activación por cortocircuito a masa**
Sondas de temperatura	4	Rango -55ºC a 85ºC. Precisión 0.5ºC (entre -10ºC y 85 ºC)
Salidas digitales	2	Salida por relé libre de potencial. Tensión máxima 250 VAC.
USB	1	Puerto de comunicación para configuración del equipo

Tabla 8. Especificaciones de Entradas/Salidas del Hermes LC2

Figura 3. Hermes LC2 de Microcom

En cuanto a aplicaciones, desde el sitio oficial de la compañía nombran las siguientes:

- Control de temperaturas en salas de servidores.
- Redes de suministro de agua.
- Redes de distribución eléctrica.
- Instalaciones de energía solar.
- Estaciones de bombeo.
- Instalaciones de frío industrial.
- Alarmas por temperatura.
- Control de cámaras frigoríficas.
- Supervisión en general de instalaciones industriales y transmisión de alarmas técnicas.

2.1.3. Remota Neptuno GPRS de ABB

ABB es una empresa internacional con sede en Zurich (Suiza), que opera en alrededor de 100 países y con más de 80.000 empleados. Desarrolla electrónica para equipos de monitorización y automatización así como robótica en el ámbito industrial.
El sistema Neptuno 4H es una solución para el control de regadíos mediante comunicación GPRS que se basa en estaciones remotas autónomas. Posee alta flexibilidad en cuanto a su alimentación ya que puede ser alimentado mediante baterías, paneles solares, red eléctrica de 220V, entre otras.

Figura 4. Neptuno GPRS de ABB

La unidad permite gestionar:

- Cuatro salidas digitales.
- Cuatro entradas digitales para contadores.
- Tres entradas analógicas.
- Dos entradas digitales para uso general.
- Una salida digital con relé de 0.5 A.

En cuanto a su hardware los principales componentes que caben destacar son:

- Microprocesador PIC18LF8720.
- Reloj en tiempo real.
- Memoria de hasta 2MB.
- Módulo de comunicaciones GPRS y/o radio, según aplicación.
- Puertos USART, I2C, ICSP.

2.2. Datalogger

Dado que el Gateway a desarrollar tiene funcionalidad de datalogger, se mostrará un ejemplo comercial registrador de datos.
Decagon Devices es una empresa que fabrica dispositivos de medión ambiental, calidad de aguas, entre otros. Fue fundada en 1983 con sede en Pullman, Washington, y cuenta con distribuidores en todo el mundo. En España Lab.Ferrer cuenta con su representación oficial.
El Em50 es una estación de almacenamiento de datos para monitorizar de forma continua los sensores que estén conectados, fabricados por Decagon. En cuanto a sus características técnicas cabe destacar:

- 5 canales de entrada con resolución A/D de 32 bits (Para sensores analógicos y digitales)
- Memoria de 1MB no volátil.
- Alimentación mediante pilas alcalinas (5AA) o de Litio.
- Dimensiones 12.7x20.3x5.1 cm
- Interfaz de cable serie o USB.

Figura 5. Datalogger EM50 de Decagon Devices

Cuenta con un software para monitorización proporcionado gratuitamente por Decagon para varias plataformas, Echo2O Utility. Proporciona una interfaz llamada Datatrac que muestra las medidas de los distintos sensores de manera gráfica.

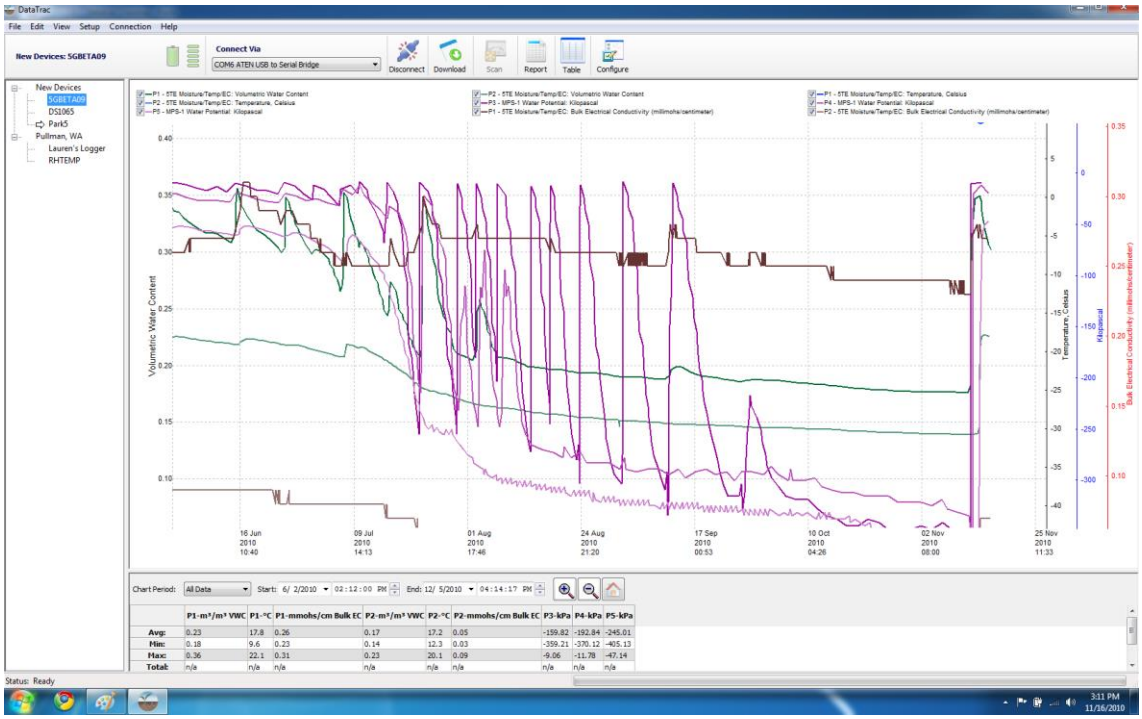


Figura 6. Ventana del datatrac de Decagon Devices

2.3. Sensores

Denominamos sensor a cualquier dispositivo diseñado para detectar magnitudes físicas o químicas y transformarlas en señales, normalmente eléctricas, las cuales se pueden cuantificar o manipular.
En el mercado actual se puede encontrar una gran cantidad de sensores distribuidos en función de sus aplicaciones y magnitudes a medir. Este apartado se ha clasificado en 2 grandes grupos: sensores analógicos y digitales. A su vez, se realizará una breve explicación de algunos de los sensores más utilizados en el ámbito de la agricultura de precisión clasificándolos dentro de dichos grupos.

2.3.1. Sensores analógicos

Un sensor analógico es aquel que emite una señal comprendida por un campo de valores instantáneos que varían en el tiempo y son proporcionales a los efectos que se están midiendo. Para su utilización en circuitos lógicos es necesario un conversor A/D, el cual codifica la entrada analógica como una combinación de salidas digitales, asignando un rango de valores de entrada analógicos un valor binario.

A continuación se mostrarán algunos de los sensores más utilizados para la monitorización ambiental y la agricultura de precisión:

- Sonda de humedad del suelo 10HS

La sonda 10HS es un sensor capacitivo del tipo FDR (Frecuency Domain Reflectrometry, Reflectometría en el dominio de la frecuencia) que miden la constante dieléctrica o permitividad del suelo para calcular su contenido de humedad. La fracción volumétrica del suelo ocupada por agua tiene enorme influencia en la permitividad dieléctrica del suelo ya que su valor dieléctrico e s mucho mayor que el de los otros constituyentes del suelo. Cuando la cantidad de agua del suelo varía miden la variación y la relacionan directamente con el cambio en contenido de agua. Entre las características técnicas el fabricante proporciona las siguientes:

- Precisión en suelos minerales:
  - ± 3% VWC
  - ± 2% VWC con calibración específica
- Voltaje : 3- 15 VDC a 12-15mA
- Frecuencia del oscilador: 70MHz
- Volumen de influencia: 1l
- Intervalo de VWC: 0 -57%
- Señal de salida: voltaje, correlacionado polinomialmente con el VWC

Figura 7. Sonda de humedad enterrada analógico de Decagon

2.3.2. Sensores digitales

Un sensor digital es aquel dispositivo que frente a un estímulo ofrece únicamente dos tipos de salidas: ON/OFF, verdad/falso, positivo/negativo, no existen valores intermedios. Los estados de un sensor digital son absolutos y únicos y se utilizan para verificar estados.

- Sonda de humedad del suelo 5TE

La sonda 5TE es un sensor capacitivo del tipo FDR (Frecuency Domain Reflectrometry, Reflectrometría en el dominio de la frecuencia) que miden la constante dieléctrica o permitividad del suelo para calcular su contenido de humedad. La fracción volumétrica del suelo ocupada por agua tiene una enorme influencia en la permitividad del suelo ya que su valor dieléctrico es mucho mayor que el de los otros constituyentes del suelo. Por este motivo, cuando la cantidad de agua del suelo varia, la sonda detecta y mide esta variación y la relacionan directamente con el cambio en el contenido de agua. Al contrario de lo que sucede en otros sensores, no son sensibles a la textura del suelo.
En cuanto a sus aplicaciones, el fabricante destaca:

- Estudios de salinidad y monitorización de la CE del suelo.
- Seguimiento de la disponibilidad de agua en el suelo en experimentos de campo.
- Estudios en la zona no saturada.
- Hidrología y aplicaciones industriales.

Especificaciones proporcionadas por el fabricante:

Contenido volumétrico de agua (VWC):

- ± 3% VWC en suelos minerales con una CE < 10 dS/m
- ± 1-2% VWC con calibración específica en cualquier medio poroso.

Conductividad eléctrica (CE):

- Intervalo: 0 a 23 dS/m (aparente)
- Precisión: ±10% de 0-7 dS/m
- Resolución:
  - 0.01 dS/m de 0-7 dS/m
  - 0.05dS/m de 7-23.1 dS/m

Temperatura (T):

- Intervalo: -40ºC a 50ºC
- Precisión: ±1ºC (0 – 50ºC)

- Resolución: 0.1ºC

Conexión:

- serial TTL, 3,6V o protocolo de comunicación SDI-12

Figura 8. Sonda de humedad enterrada Digital de Decagon

Se ha realizado una comparación entre dos sensores con una finalidad similar pero con sus dos vertientes: analógica y digital, con el objetivo de dar a conocer las posibles soluciones a las que el proyectista, experto o encargado de realizar el estudio sobre, en este caso un tipo de suelo, se puede encontrar en el mercado. Por lo general, la experiencia y las limitaciones del dispositivo adquisidor de las medidas de los sensores (si ofrecen o no entradas TTL o directamente analógicas) determinarán que sensor es conveniente en la aplicación.

2.4. Comunicación inalámbrica

Las comunicaciones inalámbricas son aquellas que gestionan la comunicación entre dispositivos (emisor/receptor) sin necesidad de un medio físico material, basándose en la propagación y modulación de las ondas electromagnéticas en el espacio. Esta tecnología permite cubrir largas distancias y dotan a los sistemas de movilidad dentro de la zona de cobertura. Para ello realizan la comunicación a través de ondas de radiofrecuencia.
Las principales ventajas residen en la no necesidad de encontrarse en un punto estático dentro de la zona de cobertura y el bajo coste, comparándolo con las instalaciones cableadas, al no utilizar medio físico material para establecer la

comunicación. Indirectamente se obtiene otras ventajas como el ahorro en mantenimiento de líneas, imposibilidad de daños o robos en el medio de transmisión y posibilidad de llegar a lugares remotos donde sería difícil, incluso peligroso, realizar una instalación de cable. Sin embargo, en contraposición, esta tecnología presenta una mayor aleatoriedad debido a las características del entorno en el que se encuentre además de necesidad de dotar al sistema de una mayor seguridad ante intrusiones externas.

La necesidad de estandarizar la gran diversidad de tecnologías inalámbricas existentes llevo al IEEE (Institute of Electrical and Electronics Engineers) a agruparlas, clasificarlas y englobarlas en función de su cobertura. El estándar 802 fue un proyecto creado en febrero de 1980. El fin era crear un estándar para que diferentes tipos de tecnología pudieran integrarse y trabajar conjuntamente. Define todos los aspectos en cuanto a cableado físico y transmisión de datos. Dentro de este estándar se definieron todos los aspectos de las comunicaciones inalámbricas.

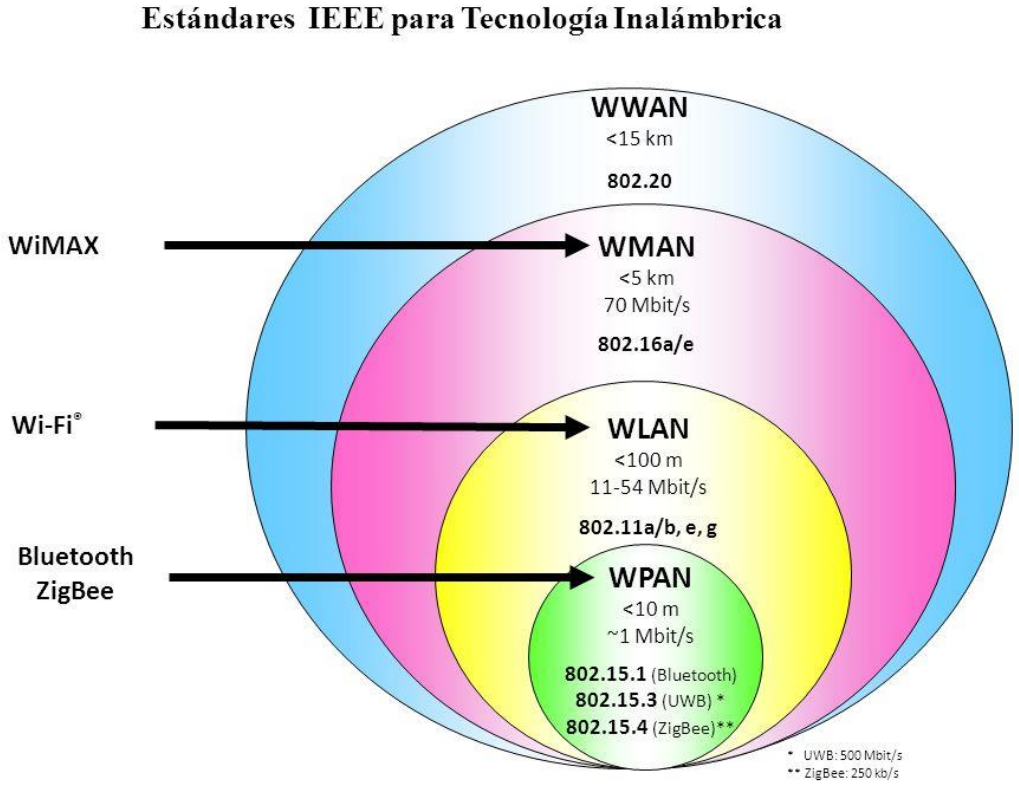


Figura 9. Posición estándares inalámbricos

- **Wireless Personal Area Network (WPAN)**

Se encuentran dentro de la norma 802.15. Se trata de redes de poco alcance, hasta una decena de metros, y suelen ser utilizados por distintos dispositivos como portátiles, móviles o impresoras entre otros. Están diseñadas para hacer un uso eficiente de los recursos y su infraestructura es prácticamente nula. Puede tener una capacidad en el rango de los 10 bps hasta los 10 Mbps.
Observamos diferentes tecnologías para las WPAN:

- **Bluetooth** (IEEE 802.15.1) Permite la transmisión de datos entre diferentes dispositivos mediante un enlace de radiofrecuencia en la banda ISM de los 2,4 GHz. Su ancho de banda va desde 1Mbit/s hasta 32Mbit/s (en su última versión)-
- **Zigbee** (IEEE 802.15.4) Permite conectar dispositivos con baja tasa de datos y poco consumo de energía, maximizando la vida útil de sus baterías. Opera en la banda de frecuencia de 2.4GHz, a una velocidad de transferencia de datos de 250Kbps, con un alcance de hasta 100 metros. Zigbee puede ser conexionado con distintas tipologías: estrella, árbol y malla.

- **Wireless Local Area Network (WLAN)**

Las redes inalámbricas de área local definidas en los estándares del IEEE 802.11, utilizan radiofrecuencia y técnicas de modulación de espectro ensanchado para transmitir y recibir datos. Se tratan de redes de trabajo que cubren un área equivalente a la red local de una empresa (100 metros aproximadamente), y que permiten conectar entre sí y a la red diferentes terminales que se encuentran dentro de la zona de cobertura.
La tecnología más conocida dentro de las WLAN es wifi.

- **Wifi** (Wireless fidelity): Dentro del estándar IEEE 802.11x. Trabajan en la banda de 2.4 GHZ. Existen cuatro tipos de conexiones cada una con una velocidad de entre 11Mbit/s hasta los 300Mbit/s. y un alcanze de hasta 100 metros.

- **Wireless Metropolitan Area Network (WMAN)**

Las redes WMAN, también conocidas como bucle local inalámbrico (WLL, Wireless Local Loop), dan cobertura a un área geográfica extensa. Basadas en el estándar IEEE 802.16, comprenden una ubicación determinada (ciudad, municipio), y su alcance es de entre 4 y 20 Km. Es una versión más grande de la WLAN y normalmente se basa en una tecnología similar a esta.
La principal tecnología es Wimax:

Wimax (Worldwide Interoperability for Microweve Access): Existen diferentes estándares referidos a esta tecnología que definen la banda de trabajo, así como la calidad del servicio y su interoperabilidad. Puede operar en la banda licenciada a 3.5 GHz o en banda libre a 5. GHz. Su velocidad de transmisión es de 40 Mbps para radios de 3 a 10km, y es capaz de ofrecer conexiones de banda ancha alcanzando distancias de hasta 50 kilómetros.

- **Wireless Wide Area Network (WWAN)**

Las redes inalámbricas de área amplia utilizan diversos dispositivos (como líneas telefónicas, antenas parabólicas y ondas de radio) para funcionar en áreas más amplias que las WLAN, aunque suelen tener un ancho de banda más reducido. Basadas en el IEEE802.20, cuyo objetivo es permitir el despliegue en todo el mundo de redes de banda ancha de móvil de acceso inalámbrico funcionando todo el tiempo e interoperables. Trabajan en bandas licenciadas por debajo de los 3.5GHz, optimizando para el transporte de datos Ip, a velocidades de entre 1Mbps y 16Mbps.

Tipo de red	WWAN	WLAN	WPAN
Estándar	GSM/GPRS/3G	IEEE 802.11x	IEEE 802.15 (bluetooth)
Velocidad	9.6/170/2000 Kb/s	11-54 MB/s	721 Kb/s
Frecuencia	0.9/1.8/2.1 GHz	2.4 y 5 GHz	2.4 GHz
Rango	35 Km	Hasta 100 metros	30 metros
Técnica de radio	varias	DSSS, OFDM	FHSS
Itinerancia	si	si	No
Equivalente a:	Conexión telefónica	LAN de media-alta velocidad	Cables de conexión

Tabla 9. Comparativa de distintos estándares

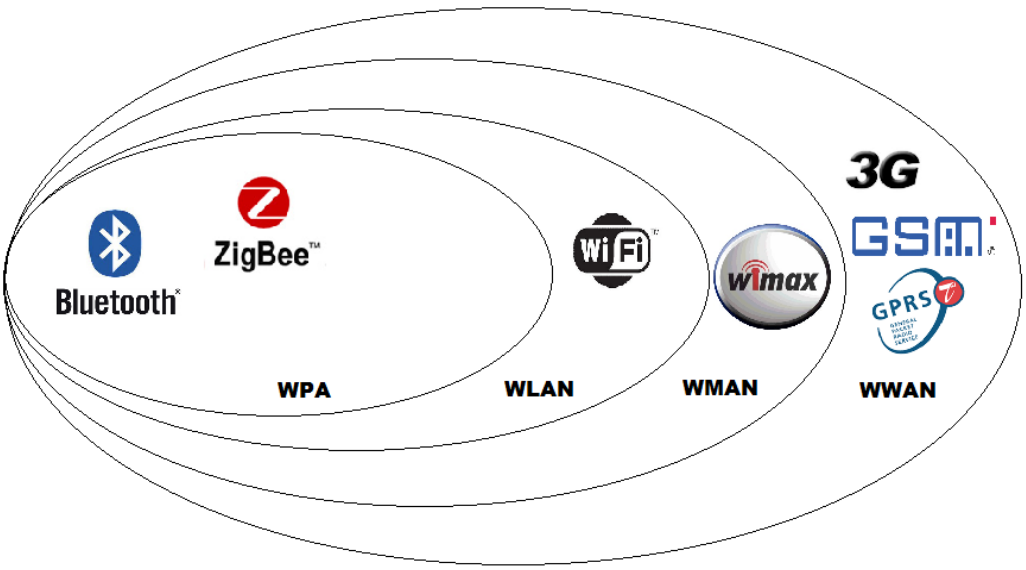


Figura 10. Esquema de distancias de redes

2.4.1. GSM (Global System for Mobile Communications)

Es un sistema estándar para la comunicación mediante teléfonos móviles que incorpora tecnología digital, haciendo un uso más eficiente del espectro que la telefonía analógica, utilizando técnicas de multiplexación en tiempo y frecuencia. Opera en las bandas de frecuencia de 900 MHz y de 1800 MHz en Europa y de 1900 MHz en Estado Unidos. Utilizan dos bandas de frecuencia cada una de 25 MHz:

- Desde la estación móvil a la estación base 890-915MHz.
- Desde la estación base a la estación móvil 935-960 MHz.

Se basa en la técnica de conmutación de circuitos, por lo que es necesario el establecimiento de la conexión mediante un canal dedicado. Su facturación depende del tiempo de ocupación del recurso, posibilita aplicaciones en tiempo real, y una sobrecarga del sistema da como resultado una señal de ocupado.
Ofrece servicios de voz de buena calidad, y de datos digitales de volumen bajo como SMS (Short Message Service), así como llamadas en espera e identificación de llamadas. Admite el movimiento desde la red de un operador a otra (Roaming). Además, la tecnología GSM permite utilizar una norma de cifrado que ofrece una protección razonable, lo que proporciona seguridad al usuario.

Al terminal usuario se le llama estación móvil, el cual está constituido por un dispositivo con un número único de identificación de 15 dígitos (IMEI:

International Equipment Identity) y una tarjeta SIM (Subscriber Identity Module) que permite identificar de manera unívoca l ususario.
Este terminal se conecta a la estación base mediante ondas de radiofrecuencia. El área cubierta por una estación base se denomina celda, en la que se comparten canales disponibles por todos los usuarios de la misma celda. Cuanto más usuarios hay dentro de una celda más riesgo hay de que se produzca saturación. Todas las estaciones bases de una red celular están conectadas a un controlador de estaciones base (BSC), que administran la distribución de los recursos, y estos a su vez están físicamente conectados al centro de conmutación móvil (MSC) que los conecta con la red de telefonía pública y con Internet.

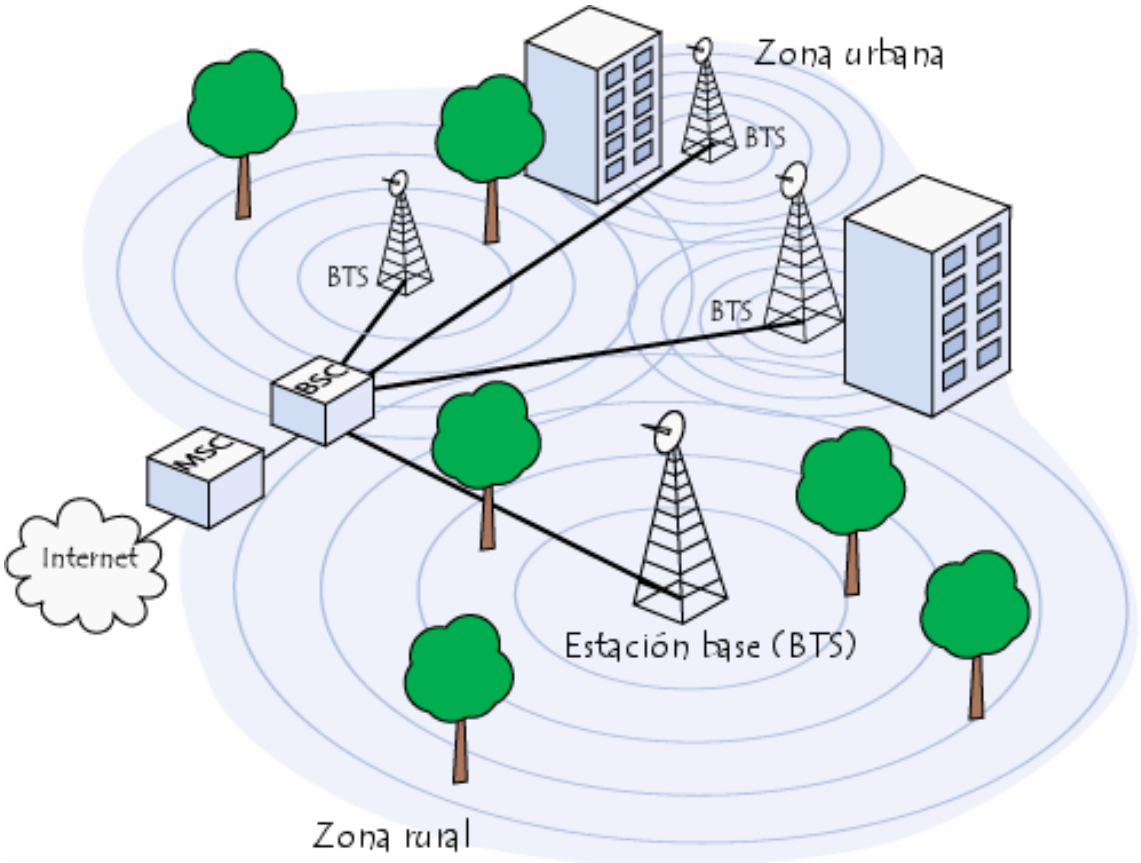


Figura 11. Distribución antenas GSM

Estación base (BTS): incluye transceptor y antenas. Cada BTS define una celda.
Controlador de Estaciones Base (BSC): Se encarga de cesiones, saltos de frecuencia. Actúa como concentrador de tráfico.
Centro de Conmutaciones Móvil (MSC): encargado de realizar las labores de conmutación, así como de proporcionar conexión con otras redes.

Registro de Ubicación Origen (HLR): Base de datos distribuida (única por red GSM) que contiene información de localización y características de los usuarios conectados a cada MSC.
Registro de Ubicación Visitante (VLR): Contiene toda la información sobre un usuario de otra red necesaria para que dicho usuario acceda a los servicios de red.
Registro de Identificación de Equipo (EIR): Proporciona seguridad a nivel de equipos válidos.
Centro de Autenticación (AuC): encargado de la autentificación de los usuarios.

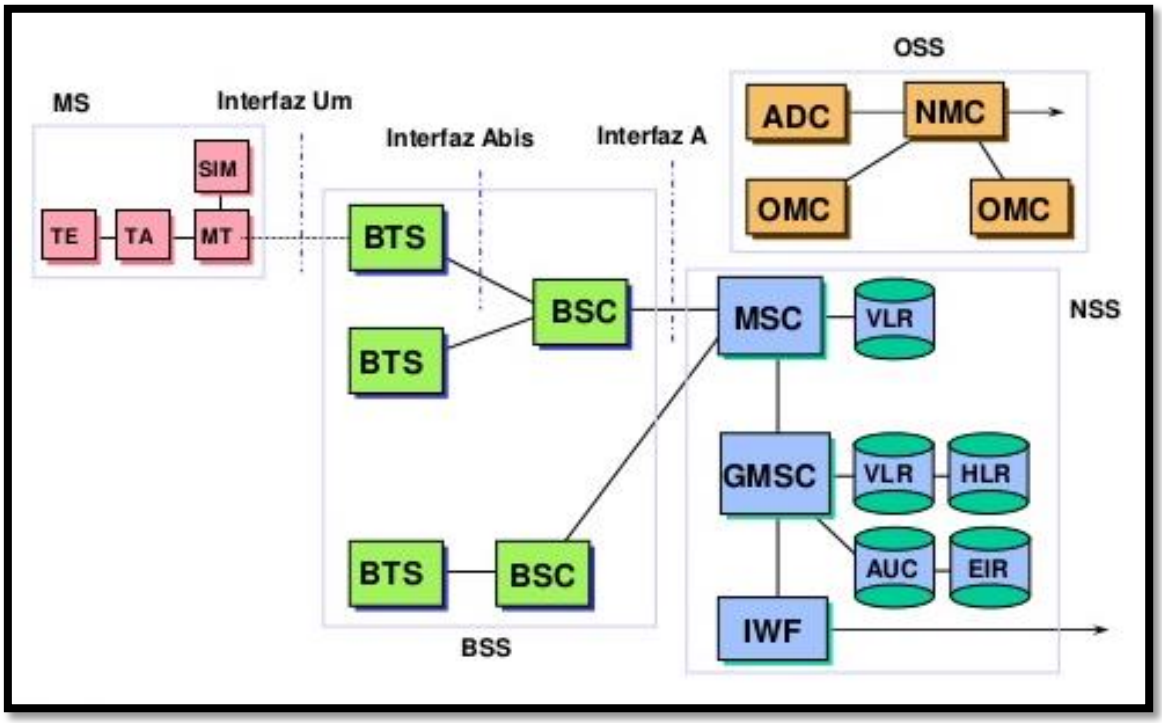


Figura 12. Diagrama de bloques GSM

2.4.2. GPRS (General Packet Radio Service)

Es una evolución del estándar GSM que permite a los usuarios móviles la transferencia de datos en forma de paquetes, haciendo uso de la red solo cuando es necesario. Provee acceso a la red de daros (especialmente Internet) a través del protocolo IP (Ipv4/Ipv6). Permite velocidades de transferencia de datos de entre 56 y 114 Kbps.
A continuación se muestran las características nuevas con respecto a GSM:

- Servicios punto a punto (PTP): capacidad de conectarse en modo cliente-servidor a un equipo en una red Ip.

- Servicio de punto a multipunto (PTMP): capacidad de enviar paquetes a un grupo de destinatarios.

Emplea la técnica de conmutación de paquetes mediante un canal compartido haciendo un uso más eficiente del espectro, y que no está orientado a conexión sino que ésta se realiza en el momento de utilización del canal. Su facturación depende del volumen de datos transmitidos en lugar de la duración de la conexión, permitiendo al usuario permanecer conectado en todo momento sin costo adicional. Solo permite aplicaciones en near real time, y una sobrecarga en el sistema produce una disminución de la velocidad.
Ofrece servicios basados en el envío de mensajes cortos, conexiones a Intranets, servicios generales de Internet, así como aqwuellos basados en la localización.
Integra el conecto de calidad del servicio (QoS), capacidad de adaptar el servicio a las necesidades de una aplicación, en función de la prioridad, demora y rendimiento. Para ello especifica 4 esquemas de codificación que definen la protección de los datos transmitidos contra interferencias para degradar la señal según la distancia entre terminales móviles y estaciones base.

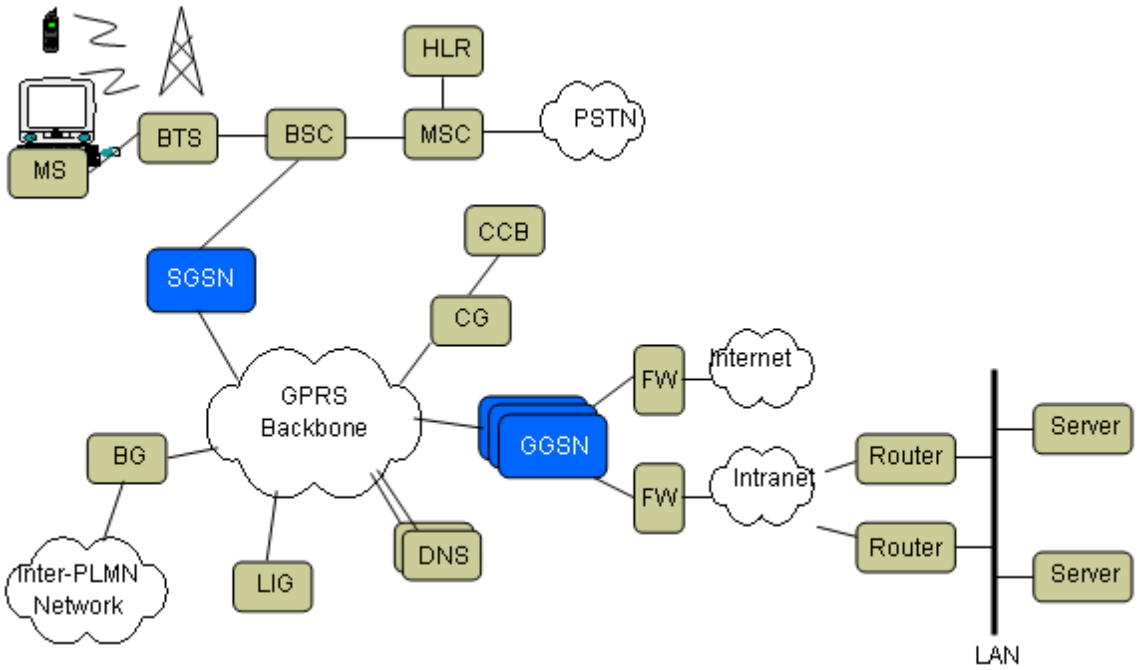


Figura 13. Diagrama de bloques GPRS

GPRS es una red superpuesta a GSM, es decir, comparten la misma red de acceso introduciendo dos nuevos nodos:

- SGSN (Serving GPRS Support Node): Es el elemento que gestiona todas las funciones de movilidad, autenticación y registro en la red de las estaciones móviles. Está conectado al BSC y es el punto de acceso a la red GPRS cuando un terminal solicita este servicio.

- GGSN (Gateway GPRS Support Node): Se conecta a redes externas como Internet. Es un dispositivo de encaminamiento hacia una subred, ya que hace que la infraestructura de la red GPRS sea transparente vista desde fuera. Cuando recibe datos dirigidos hacia un usuario específico, comprueba si la dirección está activa, y en caso afirmativo, envía los datos al SGSN.

La siguiente tabla muestra algunas de las diferencias entre GSM y GPRS:

Servicios GSM	Servicios GPRS
Establecimiento de conexión.	Sin establecimiento de conexión.
Duración media de la llamada 2 minutos	Conexión típica puede durar horas
Flujo continuo de datos en ambas direcciones.	Transmisión de datos a ráfagas. Enlaces ascendente y descendente independientes.
Todos los servicios son activados al acceder a la red.	El usuario puede activar servicios de forma independientes.
Tarificación en función del tiempo de conexión.	Tarificación basada en el volumen de datos transmitidos y/o recibidos.
Cada vez que se activa una llamada se requiere el acceso a la base de datos (HLR)	Cada paquete tratado como entidad independiente. No necesario acceder al HLR por cada paquete.
	Paquetes cortos (500 – 1500 octetos)

Tabla 10. Comparativa tecnologías GSM y GPRS

2.4.3. 3G

Se trata, como su nombre indica, de la tercera generación de transmisión de voz y datos a través de telefonía móvil. Proporcionan la posibilidad de transferir datos y voz (llamadas y videollamadas) y datos sin voz utilizados para descarga de programas, mensajería instantánea o correos electrónicos. Su orientación inicial seguía siendo la telefonía móvil aunque por las necesidades actuales, muchas compañías ofrecen un servicio 3G a modo de usb para proporcionar conexión a ordenadores portátiles, tablets y de más.
Ofrece ciertas ventajas con respecto a sus predecesoras como son la transmisión de voz con una calidad equiparable a las redes fijas y una mayor velocidad de conexión ante caídas de señal. A su vez apareció una desventaja como el llamado

efecto de 'respiración celular', según el cual, a medida que aumenta la carga de tráfico en un sector el sistema va disminuyendo la potencia de emisión, es decir, reduce la cobertura de la celda pudiendo llegar a generar zonas sin cobertura entre celdas adyacentes.
Esta tecnología proporciona una velocidad de conexión de hasta 2 Mbps en condiciones óptimas, claro que esto dependerá de cobertura y la velocidad del proveedor de telefonía.

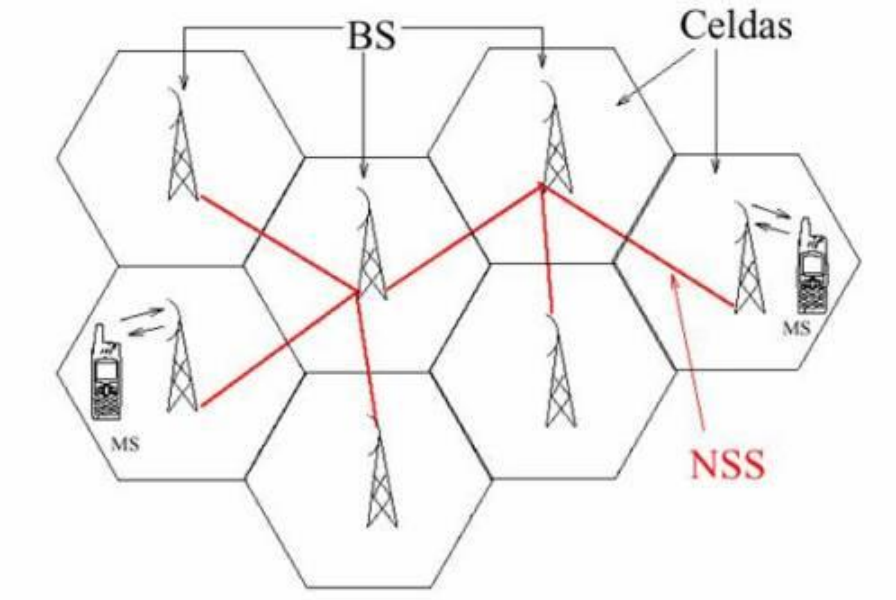


Figura 14. Distribución celda móvil

2.4.4. Radiofrecuencia, estándar IEEE 802.15.4

Se denomina radiofrecuencia a la porción menos energética del espectro electromagnético situada entre los 3KHz y 300GHz. Según su frecuencia, y por tanto su longitud de onda, se pueden clasificar en distintos grupos. D menor frecuencia a mayor encontramos:

- Frecuencias extremadamente bajas (3-30Hz)
- Super baja frecuencia (30-300Hz)
- Ultra baja frecuencia (300-3000Hz)
- Muy baja frecuencia (3-30kHz)
- Baja frecuencia (30-300kHz)
- Media frecuencia (300-3000 kHz)
- Alta frecuencia (3-30 MHz)
- Muy alta frecuencia (30-300MHz)
- Ultra alta frecuencia (300-3000MHz)
- Super alta frecuencia (3-30GHz)
- Frecuencia extremandamente alta (30-300GHz)

A partir de 1GHz las bandas están dentro del espectro de las microondas. A partir de los 300GHz la absorción de la radiación electromagnética por la atmósfera es tan alta que se vuelve opaca a ella.
El límite de potencia permitido para aplicaciones de RF depende de las regulaciones vigentes de cada país, dependiendo de la frecuencia de operación.
El IEEE (Instituto de Ingenieros Eléctricos y Electrónicos, Institute of Electric and Electronic Engineers) es una asociación a nivel mundial sin ánimo de lucro formada por profesionales ingenieros eléctricos, electrónicos, informáticos y telecomunicaciones entre otros. La asociación nació en 1884 en Nueva York (EEUU). El objetivo es promover la creatividad, desarrollo, integración y aplicar los avances tecnológicos en el campo de la electrónica y electricidad así como desarrollar estándares para su fácil integración dentro de los distintos ámbitos. Algunos de sus estándares más conocidos son IEEE 488 (BUS GPIB), IEEE 802 (Redes de ordenadores), IEEE 802.11 (Wifi) entre muchos otros.
El estándar en el que se basa el presente proyecto es el IEEE 802 para redes de computadores, claro que dentro de este estándar podemos encontrar hasta 23 grupos. La siguiente tabla muestro los distintos grupos dentro del estándar.

Nombre	Descripción
IEEE 802.1	Normalización de interfaz
IEEE 802.2	Control de enlace lógico LLC
IEEE 802.3	CSMA / CD (ETHERNET)
IEEE 802.4	Token bus LAN
IEEE 802.5	Token ring LAN (topología en anillo)
IEEE 802.6	Redes de área metropolitana (MAN)
IEEE 802.7	Grupo asesor Banda ancha
IEEE 802.8	Grupo asesor Fibras ópticas
IEEE 802.9	Servicios integrados de red de Área Local
IEEE 802.10	Seguridad en red
IEEE 802.11	Redes inalámbricas WLAN
IEEE 802.12	Acceso de prioridad por demanda
IEEE 802.13	Se ha evitado su uso por superstición
IEEE 802.14	Módems de cable
IEEE 802.15	WPAN
IEEE 802.15.4	**ZigBee**
IEEE 802.16	WIMAX
IEEE 802.17	Anillo de paquete elástico script
IEEE 802.18	Grupo Asesoría Técnica sobre Normativa Radio
IEEE 802.19	Grupo Asesoría técnica sobre Coexistencia
IEEE 802.20	Mobile Broadband Wireless Access
IEEE 802.21	Media Independent Handoff
IEEE 802.22	Wireless Regional Area Network
IEEE 802.23	Broadband ISDN systema

Tabla 11. Listado de grupos dentro del estándar IEEE 802

Dentro del estándar IEEE 802 y todos sus grupos debemos destacar el IEEE 802.15.4, el estándar de ZigBee.
ZigBee es una alianza de más de 100 empresas, la mayoría fabricantes de semiconductores, que tienen como objetivo respaldar el desarrollo e implantación de una tecnología inalámbrica de bajo coste. Algunas de las empresas más importantes dentro de esta alianza son Honeywell, Philips o Motorola, entre otras. Trabajan para desarrollar un sistema estándar de comunicaciones, vía radio y bidireccional, usado dentro de dispositivos en campos tan variados como domótica, control industrial, periféricos de PC, sensores médicos, etc. Está alianza se justica por el vacío que muestra el Bluetooth en algunos aspectos y que se intenta cubrir.
Los protocolos de ZigBee están definidos para su uso en aplicaciones embebidas con requerimientos muy bajos de transmisión de datos (Low Rate) y consumo energético (Low Power).
En cuanto a sus características técnicas, los protocolos ZigBee trabajan en una banda de frecuencias que incluye la 2.4 GHz (a nivel mundial), 902 a 928 MHz (en EEUU) y 866 MHz (en UE). La transferencia de datos de hasta 250 kbps puede ser transmitido en la banda de 2.4GHz (16 canales), hasta 40 kbps en 915 MHz (10 canales) y los 20 kbps en la de 868 MHz (1 canal). La distancia de transmisión puede variar desde los 10 metros hasta los 75, dependiendo de la potencia de transmisión del entorno.
En cuanto a la arquitectura de protocolos el modelo OSI (Open Systems Interconnection), las dos primeras capas, la física (PHY) y la de acceso al medio (MAC), son definidas por el estándar IEEE 802.15.4. Las capas superiores son las definidas por la alianza ZigBee. El grupo de trabajo de IEEE pasó el primer borrador de la capa física y la de acceso al medio en 2003.

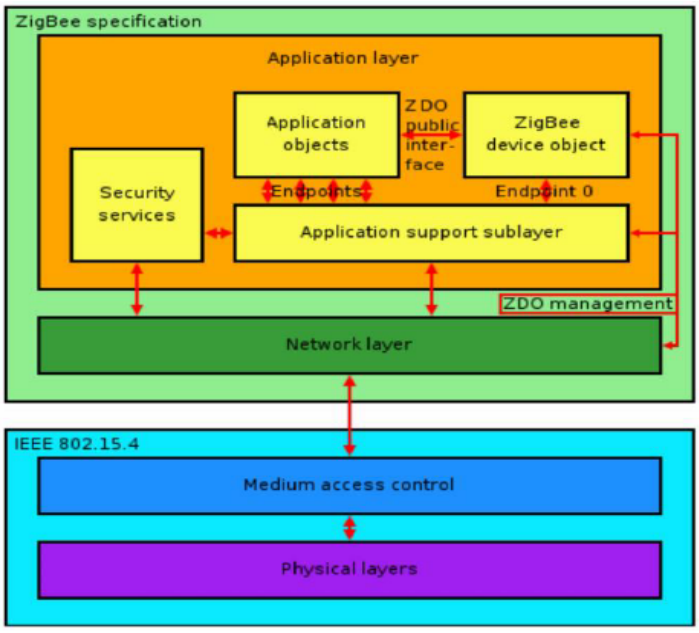


Figura 15. Diagrama de bloques IEEE 802.15.4 ZigBee

Los principales cometidos del nivel de red son permitir el correcto uso del subnivel MAC y ofrecer un interfaz adecuado para su uso por parte del nivel inmediatamente superior (el de aplicación).
La entidad de datos crea y gestiona las unidades de datos del nivel de red a partir del payload del nivel de aplicación y realiza un ruteo en base a la topología de la red en la que el dispositivo se encuentra. Las funciones de control del nivel controlan la configuración de nuevos dispositivos y el establecimiento de nuevas redes, puede decidir si un dispositivo colindante pertenece a la red e identifica nuevos routers y vecinos. El control puede detectar así mismo la presencia de receptores, lo que posibilita la comunicación directa y la sincronización a nivel MAC.
El nivel más alto es el de aplicación. Es la interfaz efectiva entre el nodo y sus usuarios. En él se ubican la mayor parte de los componentes definidos por la especificación.
El ZDO (ZigBee Device Object) se encarga de definir a cada dispositivo como coordinador o nodo de dispositivo. Identifica los dispositivos que se encuentran a un salto de red y los servicios que ofrecen. Una vez realizada esta tarea, se encuentra en condiciones de establecer enlaces seguros con dispositivos externos y responder peticiones de asociación.
El subnivel de soporte a la aplicación (Application support sublayer, APS) es el segundo componente básico del nivel. Trabaja como nexo de unión entre el nivel de red y el resto de componentes del nivel de aplicación. Mantiene actualizadas las tablas de asociaciones en forma de base de datos. Esta base de datos puede utilizarse para encontrar dispositivos adecuados en base a los servicios demandados y ofrecidos.
En cuanto a los paquetes de datos, ZigBee tiene una carga de hasta 127 bytes. La trama está enumerada para asegurar que los paquetes llegan.

5 bytes	1 byte	127 bytes
Sincronización	Cabecera del paquete	Campo de datos (Payload)

Tabla 12. Tama ZigBee

Los cuatro tipos de paquetes básicos son:

- ACK: es una realimentación desde el receptor al emisor para confirmar que el paquete se ha recibido sin errores.
- MAC: se utiliza para el control remoto y la configuración de dispositivos/nodos. Una red centralizada utiliza este tipo de paquetes para configurar la red a distancia.
- Baliza: es el encargado de 'despertar' los dispositivos, que escuchen y vuelvan a 'dormirse' si no reciben nada más. Además estos paquetes permiten la sincronización de los nodos sin utilizar una gran cantidad de energía de las baterías estando en todo momento encendidos.

En el apartado 4.1. Descripción de la red RfreeNet se muestran las peculiaridades de la red basada en este estándar.

3. Herramientas

En este apartado se mostrará las herramientas tanto software como hardware utilizadas durante el desarrollo del proyecto.

3.1. Software

Durante el desarrollo del proyecto se ha tenido que trabajar con distintos lenguajes de programación, proporcionando cada uno de estos una solución óptima para desempeñar cada una de las funciones objetivo del Gateway. En este apartado se realizará una definición de cada uno de estos y su finalidad dentro del proyecto.
También se describirá lo relativo a la aplicación web que permite interactuar con el sistema completo. Aunque no ha sido desarrollado dentro de los límites de este proyecto, es una pieza clave (en lo que a software se refiere) para entender cómo funciona, cómo se desenvuelve y que posibilidades ofrece el sistema completo.

3.1.1. Python

El código principal del firmware del Gateway se ha desarrollado con Python. ¿Por qué se ha elegido este lenguaje de programación? En este apartado vamos a ver los beneficios que se ha pensado tendría este lenguaje.
Este lenguaje se remonta a finales de los 80s principios de los 90s cuando Guido Van Rossum decidió empezar el proyecto como pasatiempo dándole continuidad al lenguaje ABC del que había formado parte del equipo de desarrollo en el CWI. Dicho lenguaje se enfocaba a ser fácil de usar y de aprender manteniendo potencia, aunque el hardware de la época hacía complicado su uso. Como nuevo lenguaje de programación que surgía, estaba influenciado directamente por ABC, ya que Van Rossum había trabajado en el desarrollo, también por C y por Icon. Cada uno de los anteriores dio solución a las desventajas que Guido observaba en cada uno. Al fin, la primera verisón de Python llega en 1994 y desde entonces se actualiza y modifica hasta la actualidad en la que se puede encontrar su última versión Python 3.5.1. La siguiente imagen muestra líneas cronológicas de las versiones que se han ido desarrollando desde su 'nacimiento' hasta la actualidad.

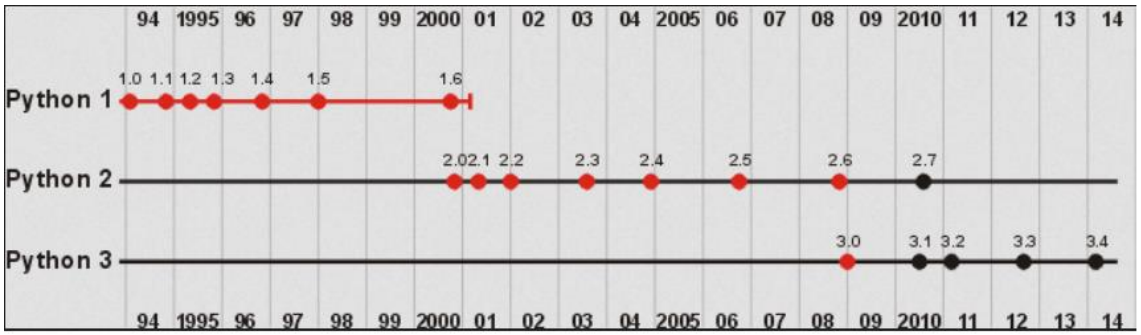


Figura 16. Cronología versiones de Python

Python es un lenguaje interpretado cuya filosofía se centra en una sintaxis 'sencilla' para obtener un código relativamente legible por humanos. Esto proporciona una ventaja para futuros desarrolladores que deseen modificar parte del código original, aumentar la funcionalidad del Gateway o conocer el funcionamiento simplemente leyendo el código directamente. Este lenguaje es multiparadigma, significa que soporta programación funcional, imperativa, orientada a objetos, entre otras. Hay infinidad de librerías creadas para facilitar el desarrollo de aplicaciones, por ejemplo, la librería socket permite establecer una comunicación TCP-IP con tan solo 4 líneas de código.
La *Python software foundatio* es una organización que se encarga de promover, proteger y seguir desarrollando nuevas versiones y actualizaciones en la comunidad de Python. Es su web oficial se puede descargar veriones de Python, obtener documentación, consultar el calendario de eventos, noticias relativas, etc. Cuenta con una comunidad activa de usuarios que ayudan a mejorar y divulgar este lenguaje.

Figura 17. Logotipo de Python

3.1.2. Archivos json

Json es la abreviatura en inglés de JavaScript Object Notatio. Es un formato para intercambiar datos básicamente. Describe los datos con una sintaxis dedicada que se utiliza para identificar y gestionar dichos datos. Nació como alternativa al XML. La mayor ventaja que podemos encontrar en este tipo de formato para gestionar y almacenar datos es que puede ser leído por cualquier lenguaje de programación. Esto añade flexibilidad y adaptabilidad a cualquier proyecto, ya que los datos almacenados en este formato podrán ser leídos por distintas plataformas sin ningún tipo de adaptación o reinterpretación.
Un json está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido un objeto, registro, estructura, diccionario, tabla hash, lista de calves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Figura 18. Logotipo JSON

En json, se presentan las siguientes formas:

Un **objeto**: es un conjunto desordenado de pares nombre/valor. Un objeto comienza con '{' y termina con '}'. Cada nombre es seguido por ':' y los pares nombre/valor están separados por ','.

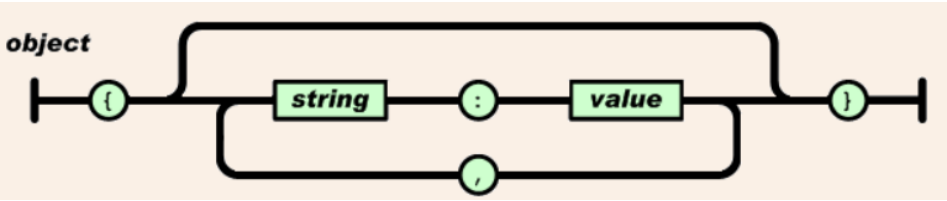


Figura 19. Diagrama representación objeto JSON

Un **array**: es una colección de valores. Un arreglo comienza con '[' y termina con ']'. Los valores se separan por ','.

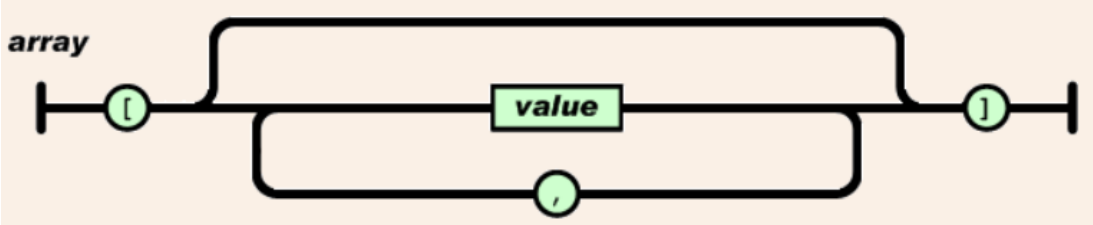


Figura 20. Diagrama representación array JSON

Un **valor:** puede ser una cadena de caracteres con comillas dobles, o un número, o *true o false o null,* o un objeto o un arreglo. Estas estructuras pueden anidarse.

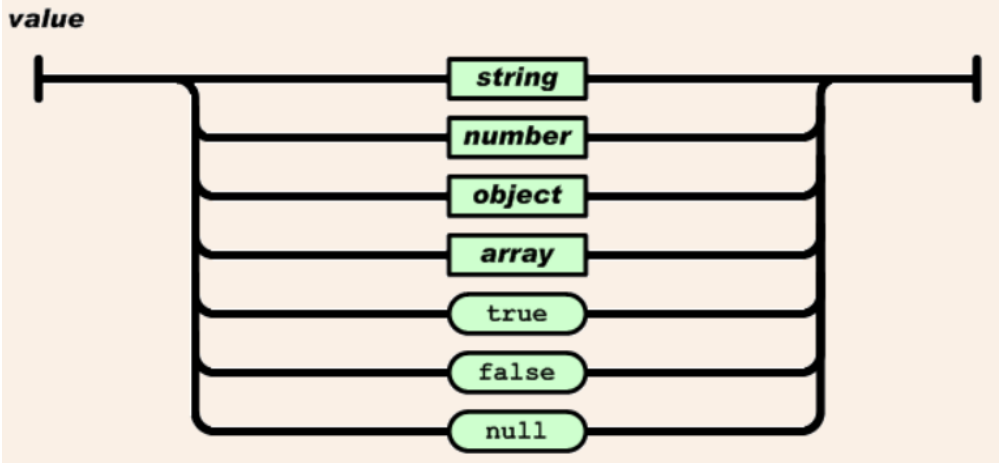


Figura 21. Diagrama representación valor JSON

Una **cadena:** es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape. Un carácter está representado por una cadena de caracteres de un único carácter. Una cadena de caracteres es parecida a una cadena de caracteres C o Java.

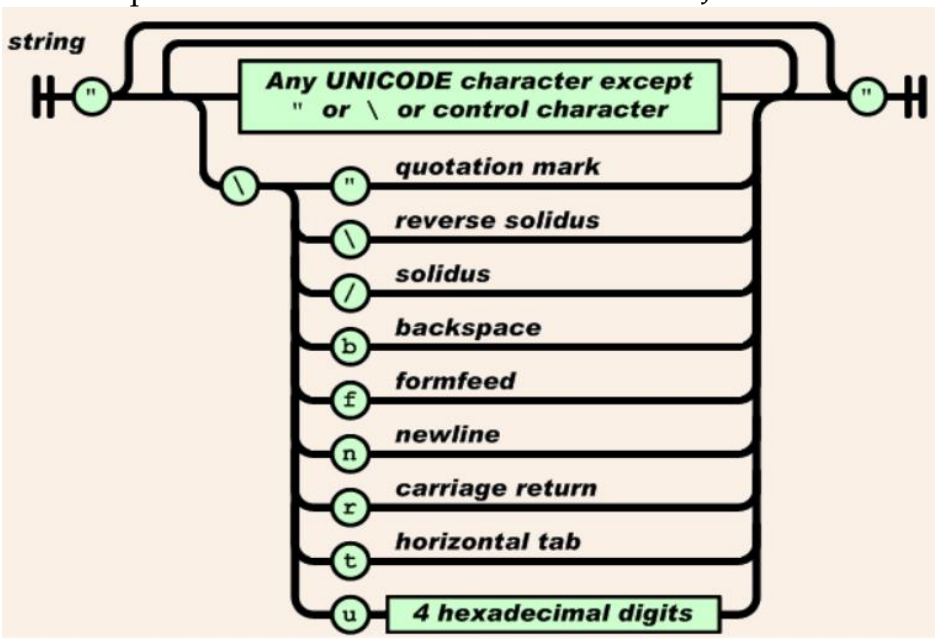


Figura 22. Diagrama representación string JSON

Un **número:** es similar a un número C o Java, excepto que no se usan formatos octales y hexadecimales.

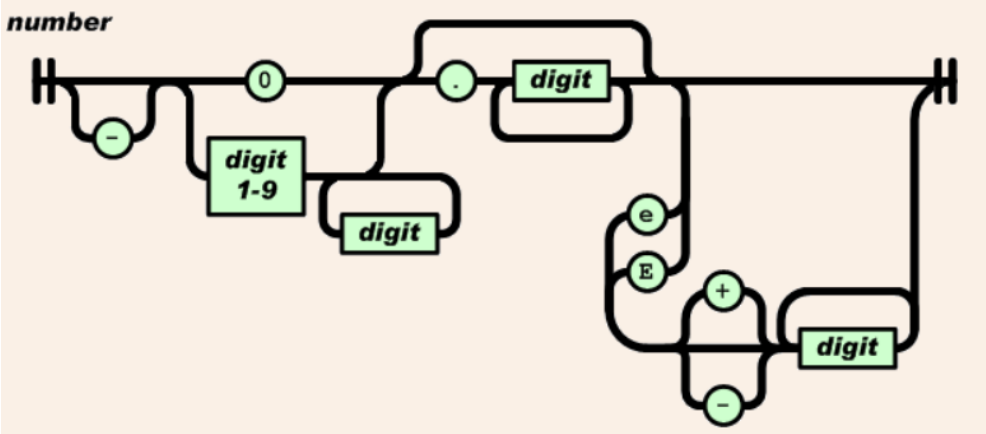


Figura 23. Diagrama representación number JSON

3.1.3. Minicom

Minicom es un programa de comunicaciones del puerto serie basado en texto. Se utiliza para interactuar con dispositivos RS-232 externos, tales como teléfonos móviles, routers y puertos de consola serie.
Durante el desarrollo del proyecto, Minicom ha sido utilizado para monitorizar la comunicación con los nodos, es decir, para ver qué recibía y enviaba el nodo. De esta forma se puede confirmar el correcto funcionamiento del nodo y en su defecto ver que posibles errores se estaban cometiendo.
Una vez instalado Minicom en el pc donde se monitorizará la comunicación, se debe configurar. En la imagen 24 se observa el panel de configuración y los distintos subpaneles que aparecerán.

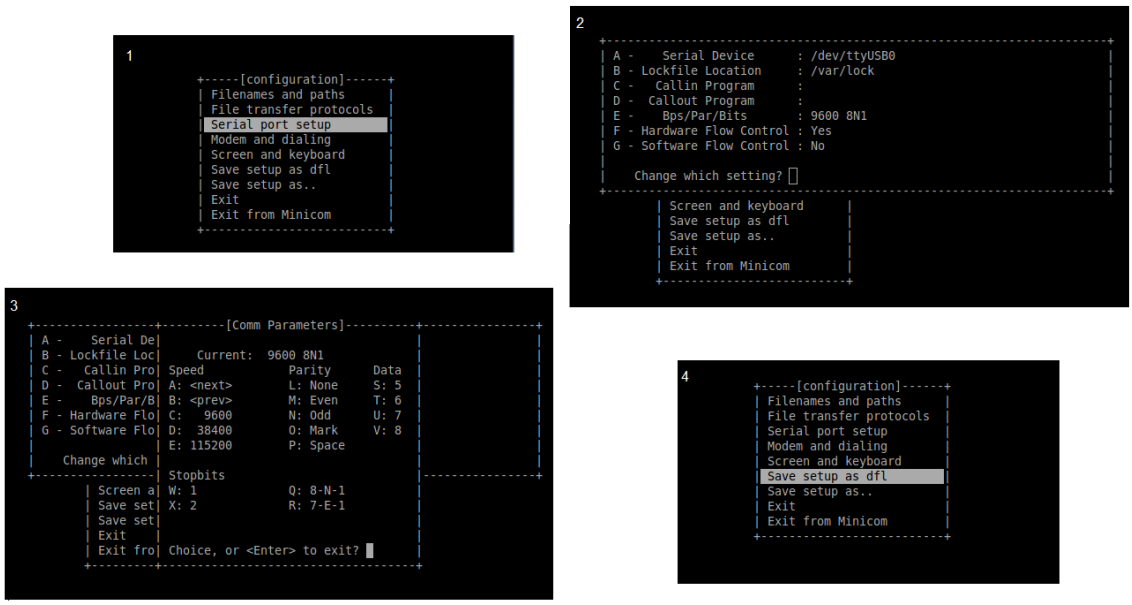


Figura 24. Capturas de pantalla configuración Minicom

Las principales opciones a configurar son:

- El puerto de comunicación.
- La velocidad de comunicación en baudios.
- El bit de paridad, si lo hubiese.

Una realizada la configuración, se guarda para acceder a esa misma configuración cada vez que sea necesario visualizar la comunicación.
En la siguiente imagen se muestra una captura de pantalla de la visualización en Minicom de la comunicación de un nodo cuando se le ha enviado un cambio de parámetros.

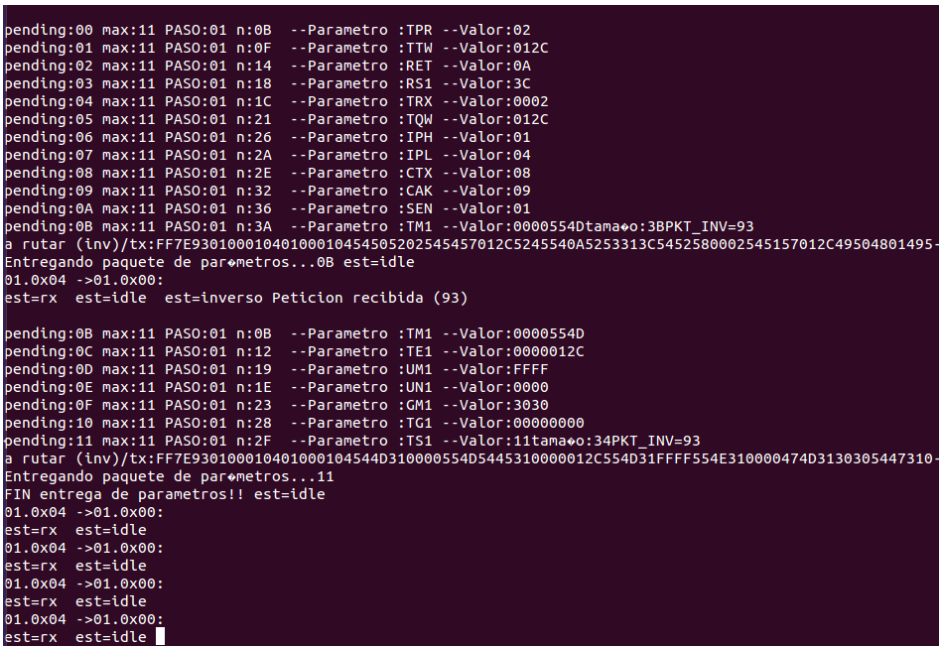


Figura 25. Captura de pantalla visualización comunicación con Minicom

3.1.4. Green Web Manager

La herramienta encargada de gestionar y actuar sobre el Gateway y, por tanto, sobre la red de nodos es Green Web Manager. En este apartado se mostrará al lector las principales características y los apartados que han sido utilizados durante el desarrollo del proyecto. En el anexo 8.3 se puede consultar el manual de usuario completo para completar la información se fuese necesario.
Green Web Manager es una aplicación web compatible con varios navegadores:

- Internet explorer 6.0 o superior
- Mozilla Firefox 2.0 o superior
- Google Chrome 2.0 o superior
- Safari 3.0 o superior
- Opera 9.0 o superior

La pantalla principal de Web Manager se divide en cuatro bloques: menú principal, menú secundario, barra de sesión y panel de aplicación.

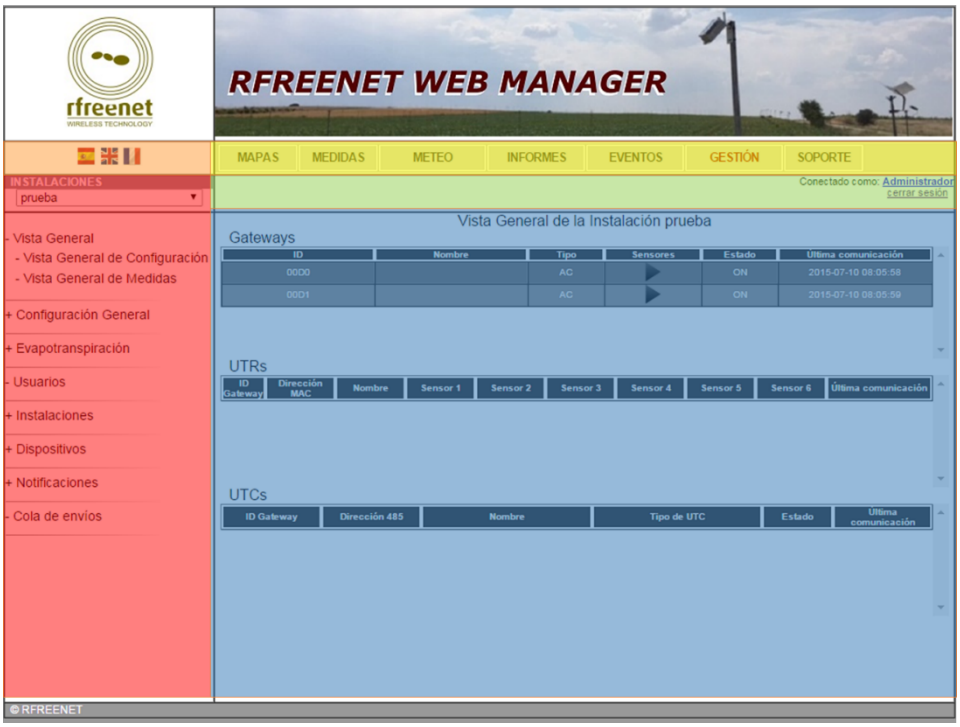


Figura 26. Partes de la interfaz Rfreenet Web Manager

- Remarcado en naranja, en la parte superior izquierda se encuentra el selector de idioma. Las opciones son: Español, Inglés y Francés.
- En amarillo, la barra de menú principal, desde la que se accede a todas las secciones de la aplicación: mapas, medidas, meteo, informes, eventos, gestión y soporte.
- En verde, bajo el menú principal, se encuentran los datos referentes al acceso de usuario.
- En rojo, parte izquierda de la ventana, se encuentra el menú secundario que incluye los accesos y controles propios de cada una de las selecciones de la aplicación.
- En azul, y ocupando la mayor parte de la ventana, aparece la información relativa a la sección activa en la aplicación.

Cuando se inicia sesión con un usuario, operador o visitante, se accede al modo cliente de la aplicación. El usuario tendrá acceso a la gestión de las instalaciones del cliente al que pertenezca. Esta gestión incluirá:

- Visualización de las instalaciones vía Google Mapas, con acceso a estado y últimas medidas de los sensores instalados en Gateways, UTRs y UTCs.

- Consulta de las últimas medidas recibidas de cada instalación generación de gráficas y exportación de datos ficheros descargables con las medidas recibidas.
- Consultas de las notificaciones enviadas por SMS y Email, así como el saldo disponible para SMS.
- Configuración de los dispositivos asociados a cada instalación, así como de otros parámetros de funcionamiento de la aplicación web.
- Consultas de soporte técnico.

A continuación se mostrará el funcionamiento y finalidad de cada una de las secciones del menú principal, anteriormente nombradas.

SECCIÓN MAPA

En esta sección el usuario tendrá acceso a un mapa con interfaz de Google Maps donde se situarán los UTRs, UTCs y gateways de la instalación seleccionada. Esta sección está constituida por tres partes: el selector de instalación (verde, imagen 27), los distintos dispositivos existentes en la instalación seleccionada (amarillo, imagen 27) y el mapa con la representación de la instalación mediante iconos.

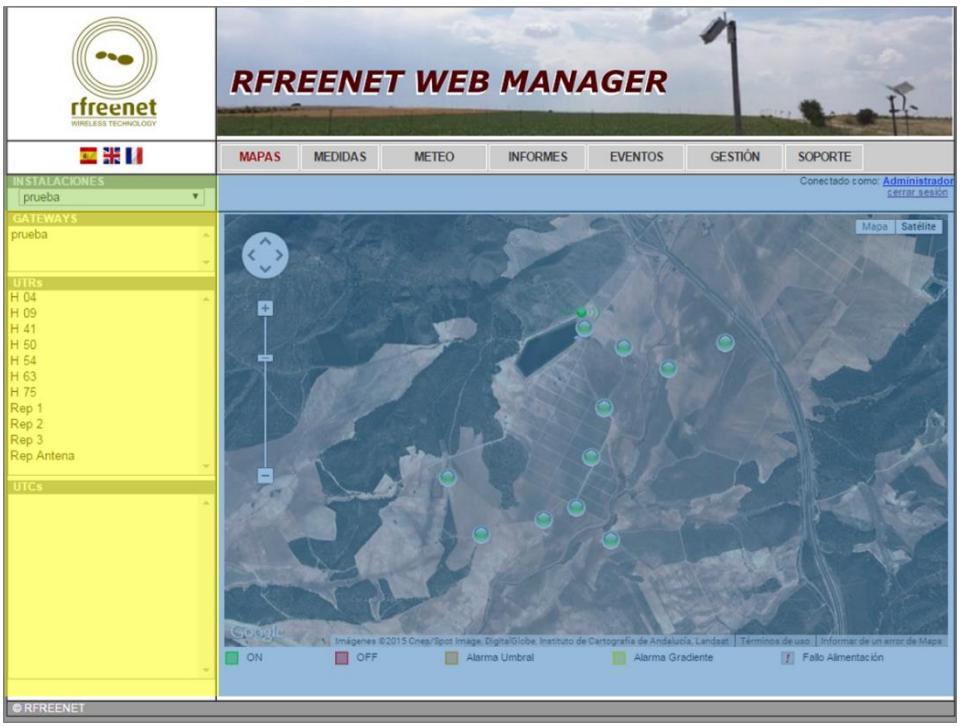


Figura 27. Sección mapa Web Manager

Los iconos que representan dispositivos utilizados en el proyecto se muestran a continuación:

Figura 28. Iconos Gateway y nodo

A la izquierda de la imagen 28 el icono del Gateway, a la derecha el icono del nodo. En la imagen están representados de color rojo, sin embargo, según el color que tengan en cada momento estarán representando distintos estados.
Se muestran los colores y los estados que representan cada uno de ellos en la siguiente imagen:

Figura 29. Leyenda estados de dispositivos

Si clicamos sobre alguno de los iconos en el mapa, se abrirá una ventana emergente con información sobre su estado y sus últimas medidas recibidas, véase la imagen 30.

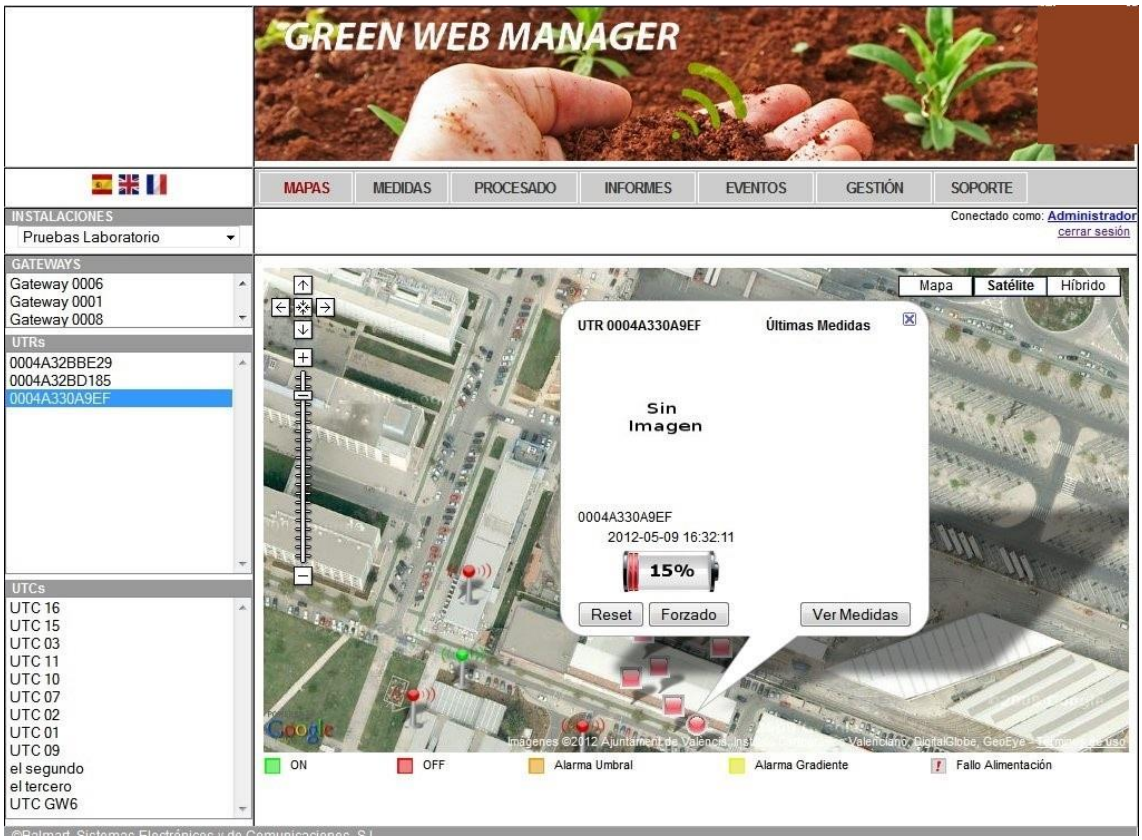


Figura 30. Despliegue de ventana al seleccionar un dispositivo

SECCIÓN MEDIDAS

En la sección medidas se puede realizar una consulta de las medidas recibidas de manera selectiva. En la imagen 31.

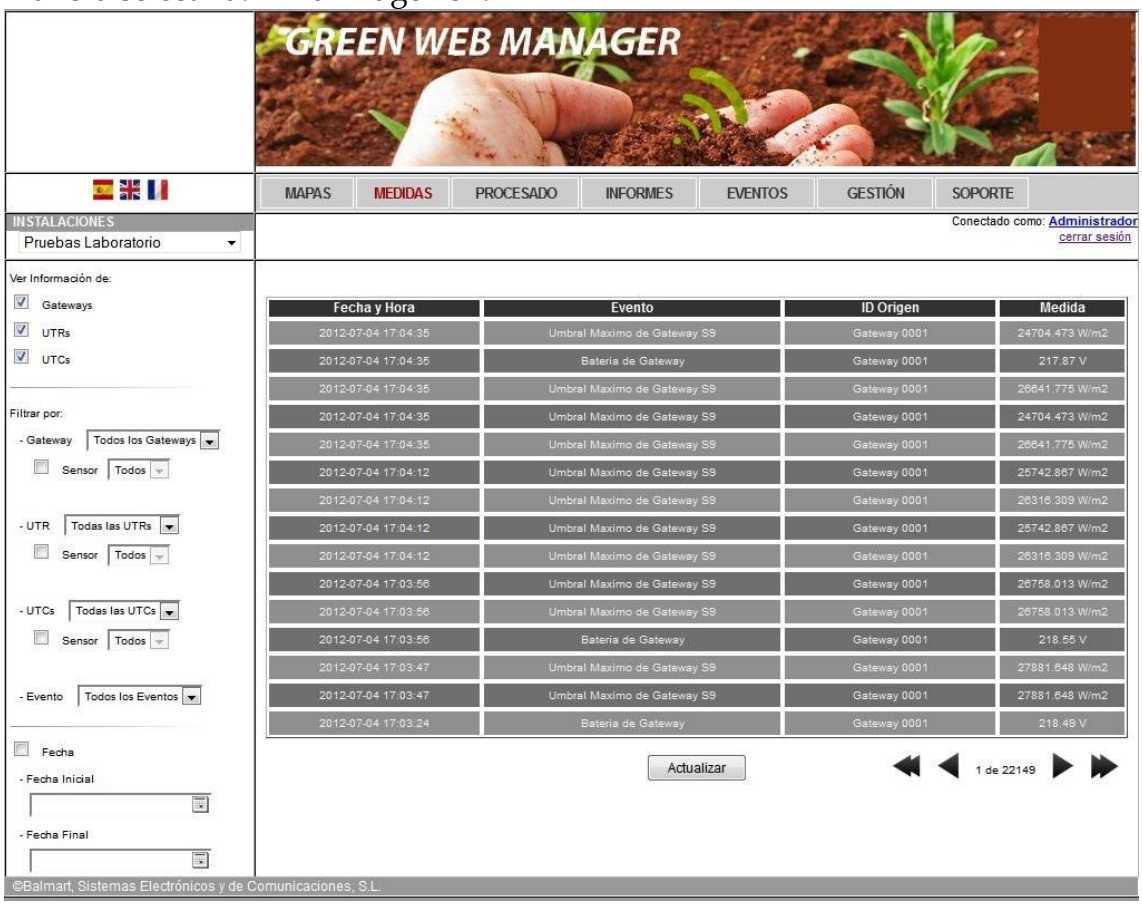


Figura 31. Sección medidas

Cabe destacar las flechas en la parte inferior derecha de la ventana, en la que de izquierda a derecha se obtiene: primera página de medidas (las más recientes), página anterior, página siguiente y última página (las más antiguas).

SECCIÓN PROCESADO

En esta sección se mostrarán los cálculos generados de Evapotranspiración. Véase la apariencia en la imagen 32.

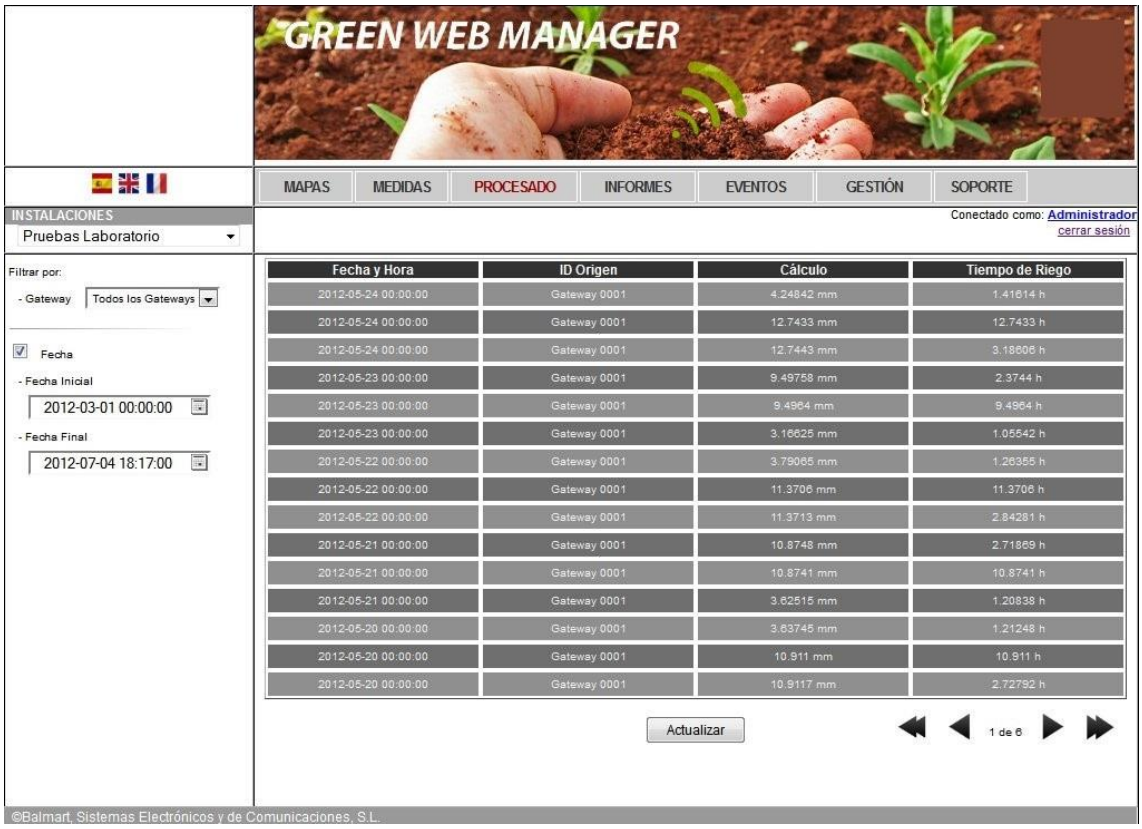


Figura 32. Sección procesado

SECCIÓN INFORMES

En esta sección podremos generar gráficas con las medidas almacenadas, así como exportar dichas medidas a ficheros descargables. El menú secundario está compuesto de controles de fechas, añadiendo la posibilidad de modificar el tipo de informe a realizar entre varias opciones, incluyendo gráficas y archivos CSV compatibles con hojas de cálculos. Además es posible guardar y eliminar plantillas, en estas se guardarán los sensores para cada gráfica, de que tipo es cada sensor y si es una gráfica o un CSV.
Las dos siguientes imágenes muestran la ventana para configurar dichas gráficas, así como un ejemplo de gráfica.

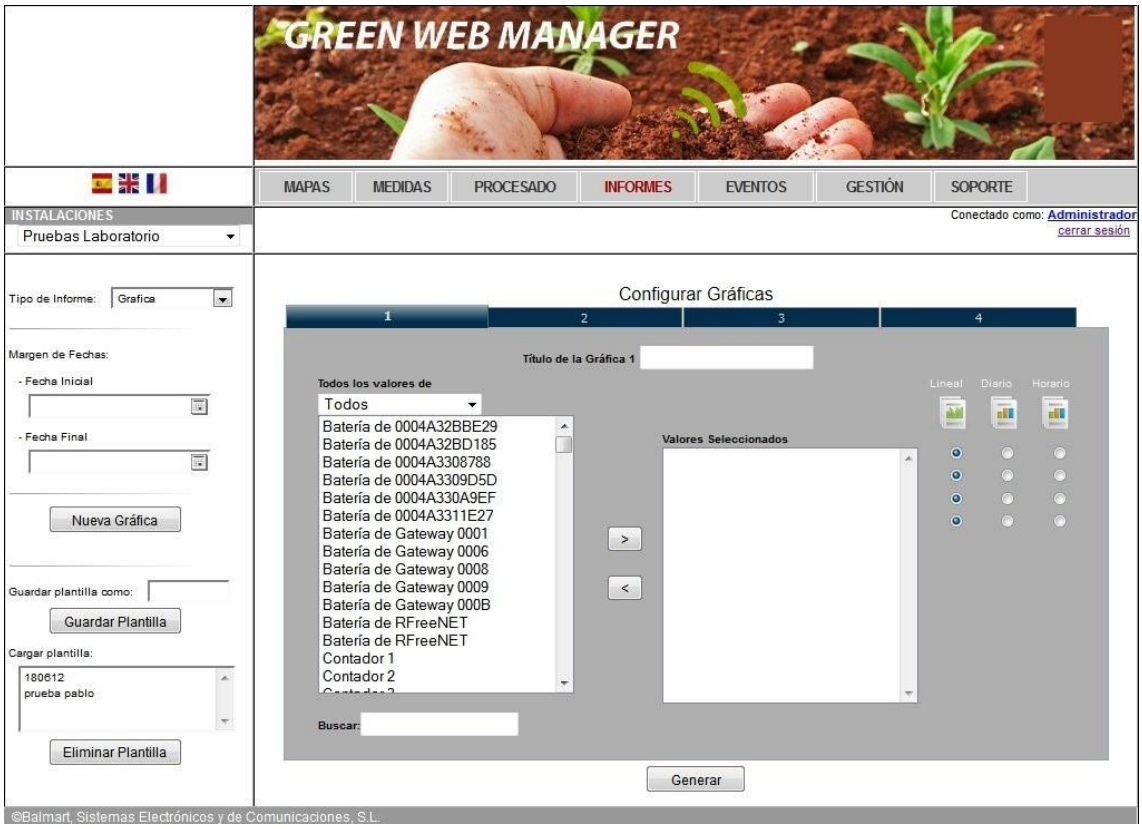


Figura 33. Sección informes

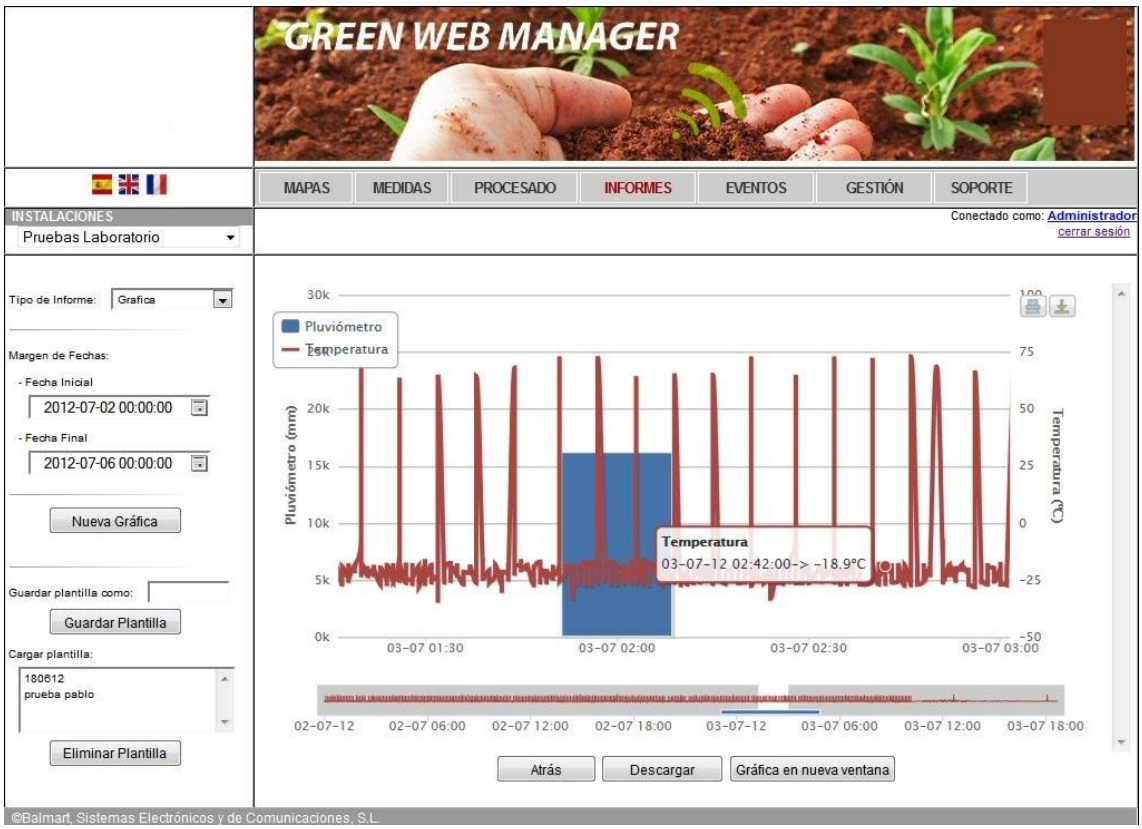


Figura 34. Sección informe con gráfica

SECCIÓN EVENTOS

Este apartado está dividido en dos secciones, seleccionables mediante un desplegable denominado Evento en el submenú. Las dos secciones que se pueden elegir son: notificaciones y diario.
En el apartado notificaciones es posible consultar las notificaciones generadas por el sistema, tanto vía SMS como Email, así como si ha sido posible enviarlas o no.

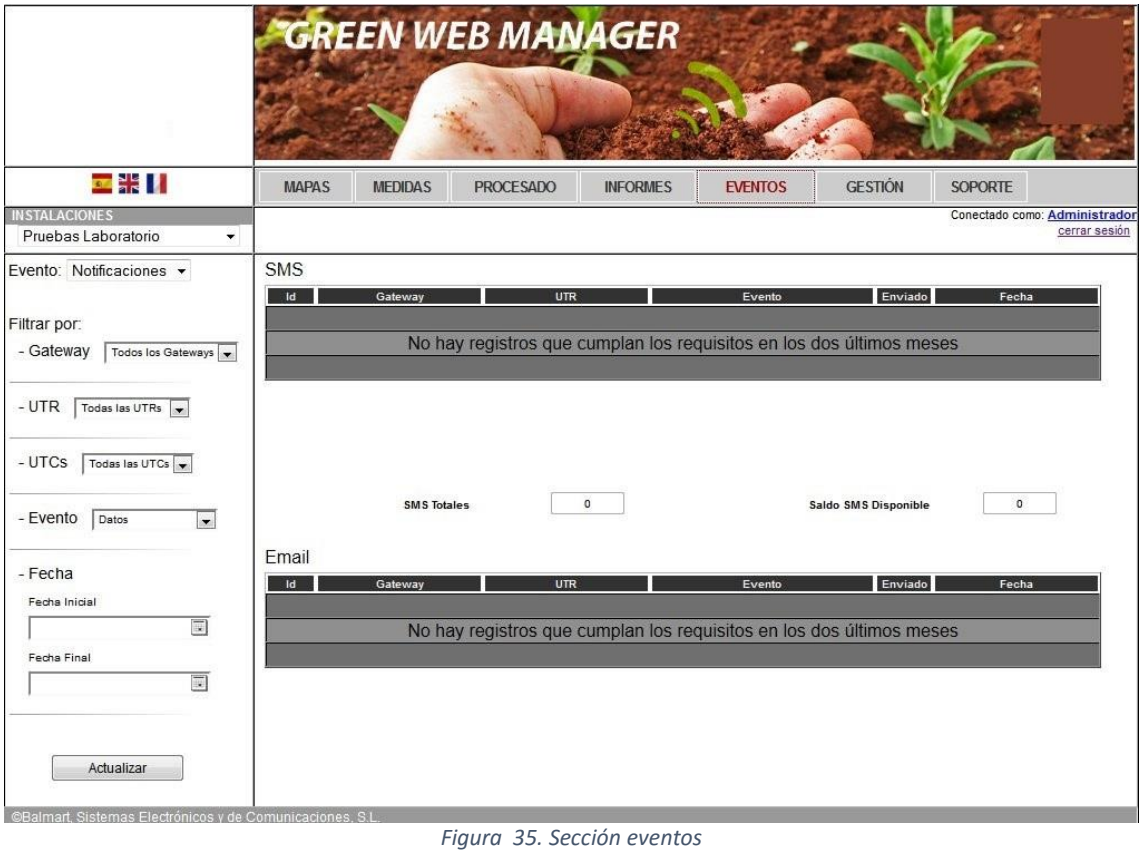


Figura 35. Sección eventos

El apartado diario utiliza anotaciones sobre la instalación. Se puede ver una lista con las últimas anotaciones y en el submenú existen uno controles para indicar el rango de fechas que se desea ver.

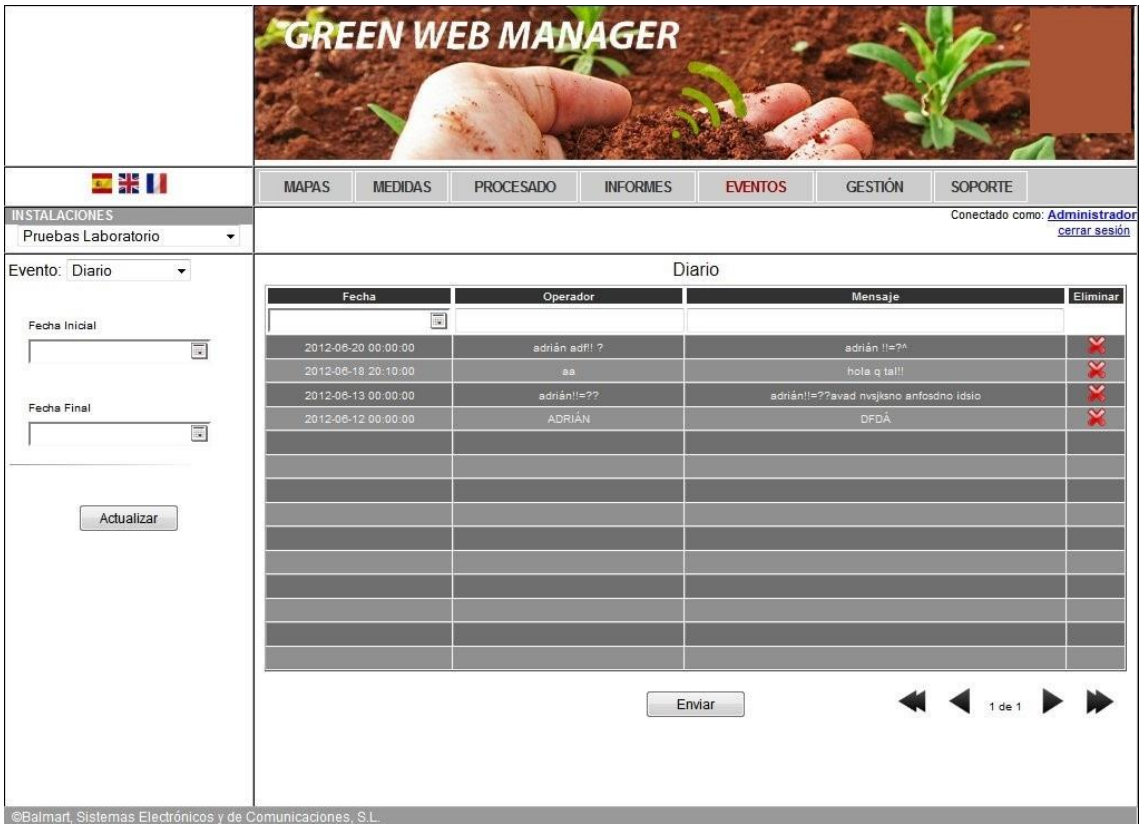


Figura 36. Sección eventos con anotaciones

SECCIÓN GESTIÓN

La sección gestión está constituida por todas las opciones que permiten al usuario personalizar y adaptar el sistema RfreeNet y la aplicación WebManager a sus necesidades. La información mostrada siempre irá referida a la instalación seleccionada.
Durante el desarrollo del proyecto se han utilizado únicamente las secciones configuración general, dispositivos y cola de envíos, en donde se encuentran la configuración de: tiempos para dispositivos caídos, tiempo de reenvíos de tramas, eliminar dispositivos de la instalación, adición de éstos, configuración y ver los envíos de tramas pendientes o ya enviados y cuando.

En la siguiente imagen se muestra la configuración general de los tiempos de los dispositivos caídos y el tiempo de reenvío de las tramas en el caso que no se hayan recibido correctamente.

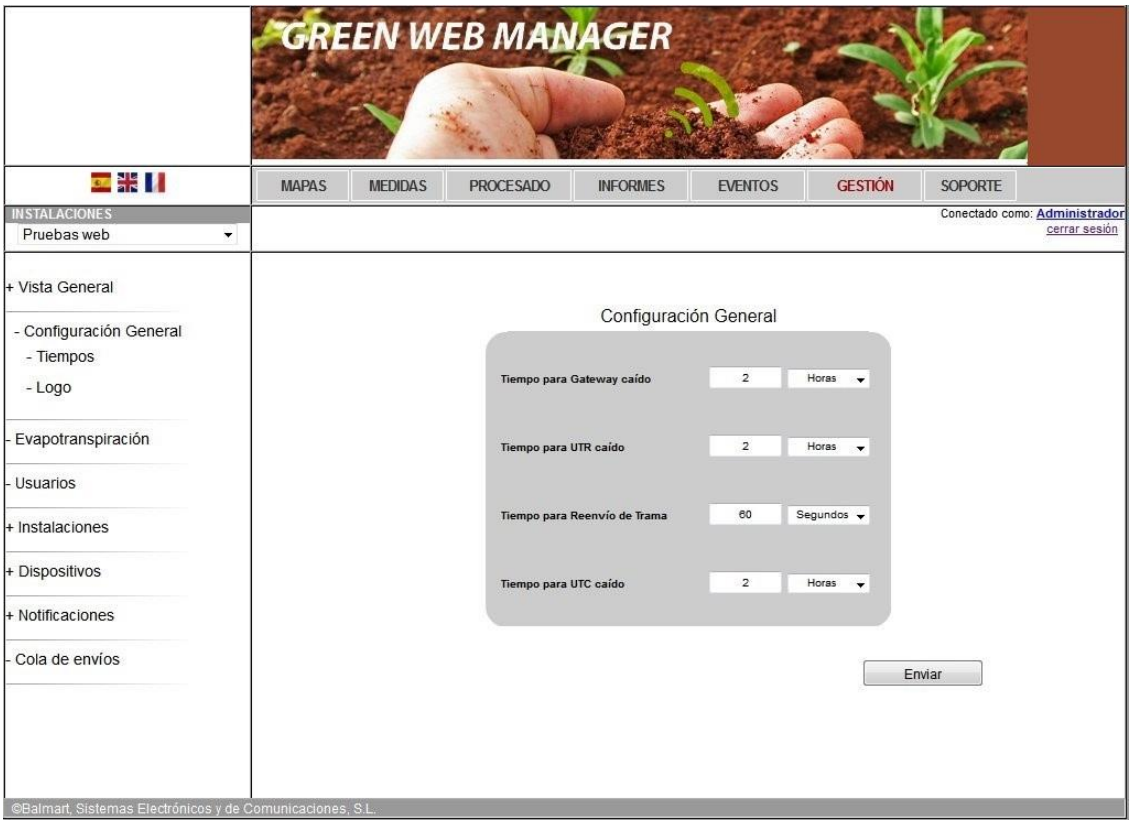


Figura 37. Sección gestión

En el apartado dispositivos, podemos eliminar cualquier dispositivo de la instalación así como situar nuevos y configurarlos. La siguiente imagen muestra el menú de configuración de los dispositivos.

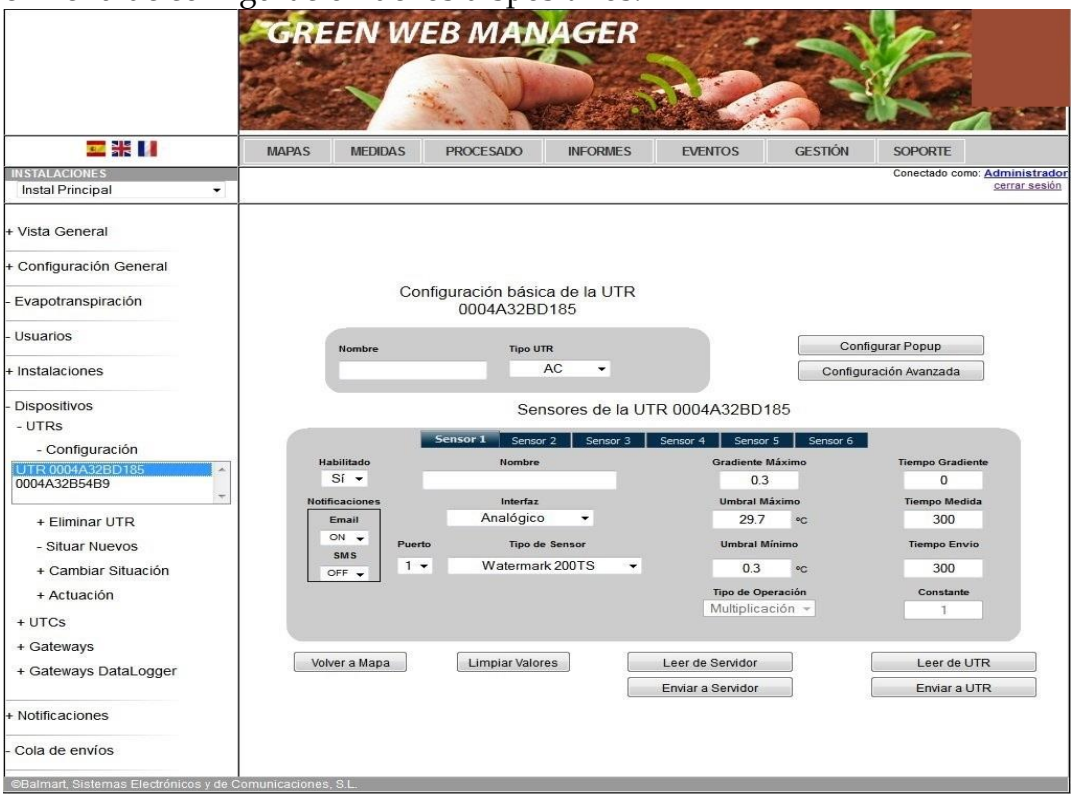


Figura 38. Sección gestión ventana de dispositivos

Finalmente en el apartado de cola de envíos, el usuario puede consultar los comandos enviados y en que momento. También situando el cursor encima del comando aparecerá una ventana superpuesta a la tabla que mostrará la trama enviada. Véase en la siguiente imagen.

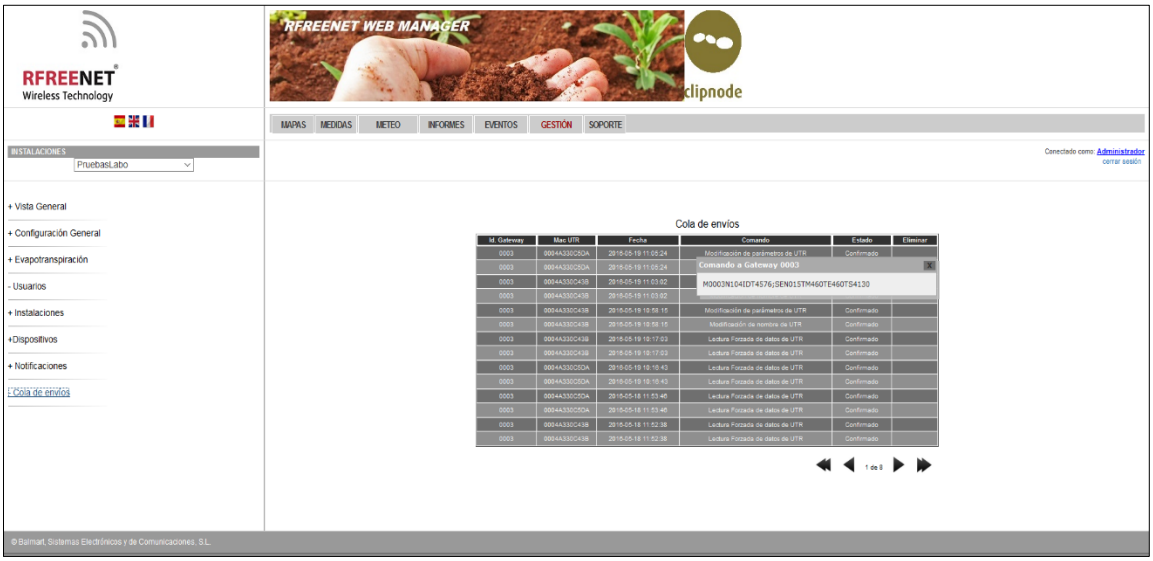


Figura 39. Sección gestión ventana de cola de envíos

SECCIÓN SOPORTE

Por último en la sección soporte, el usuario tiene la posibilidad de consultar el problema con el soporte técnico. Desde ella puede enviar un mensaje al servicio de soporte técnico (vía email), donde su duda será atendida.

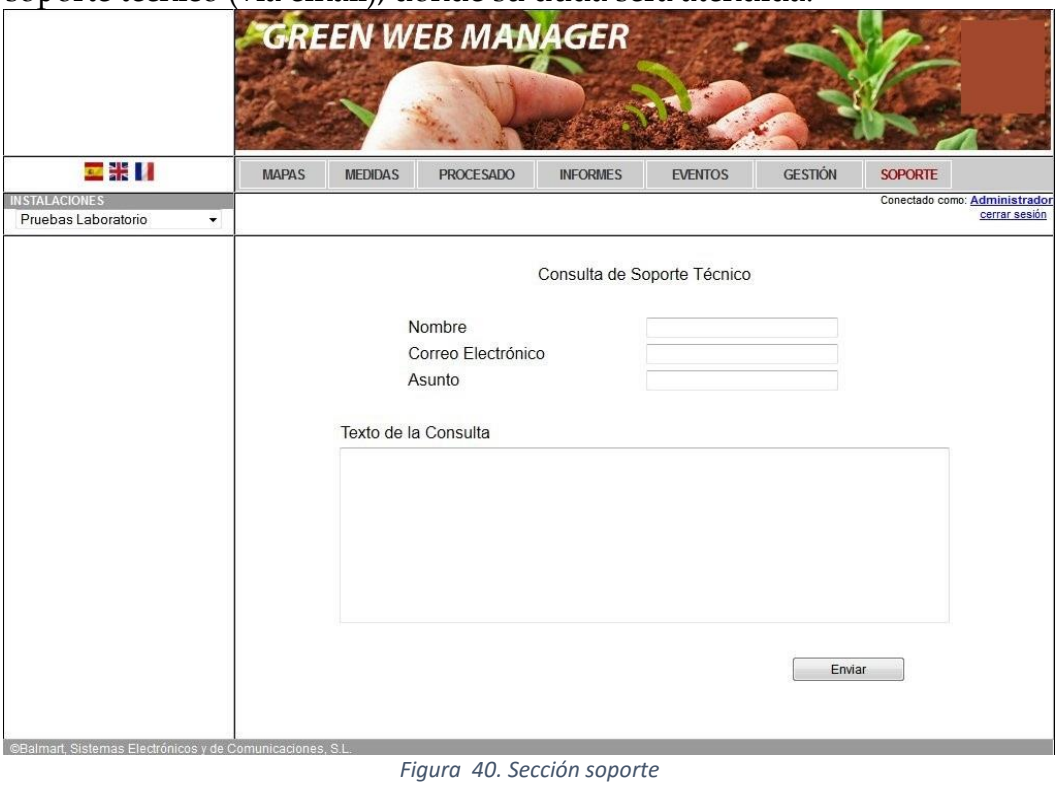


Figura 40. Sección soporte

3.1.5. FileZilla

FileZilla es un cliente FTP multiplataforma y de código abierto, bajo licencia pública general de GNU. Comenzó siendo un proyecto de clase de informática en enero de 201 de Tim Kosse y dos compañeros. Una de sus finalidades más utilizada es la de administrar sitios web, estableciendo una conexión cifrada que utiliza un protocolo SSH (Secure SHell).

Figura 41. Icono FileZilla

La siguiente imagen muestra una captura de pantalla de la ventana principal de FileZilla.

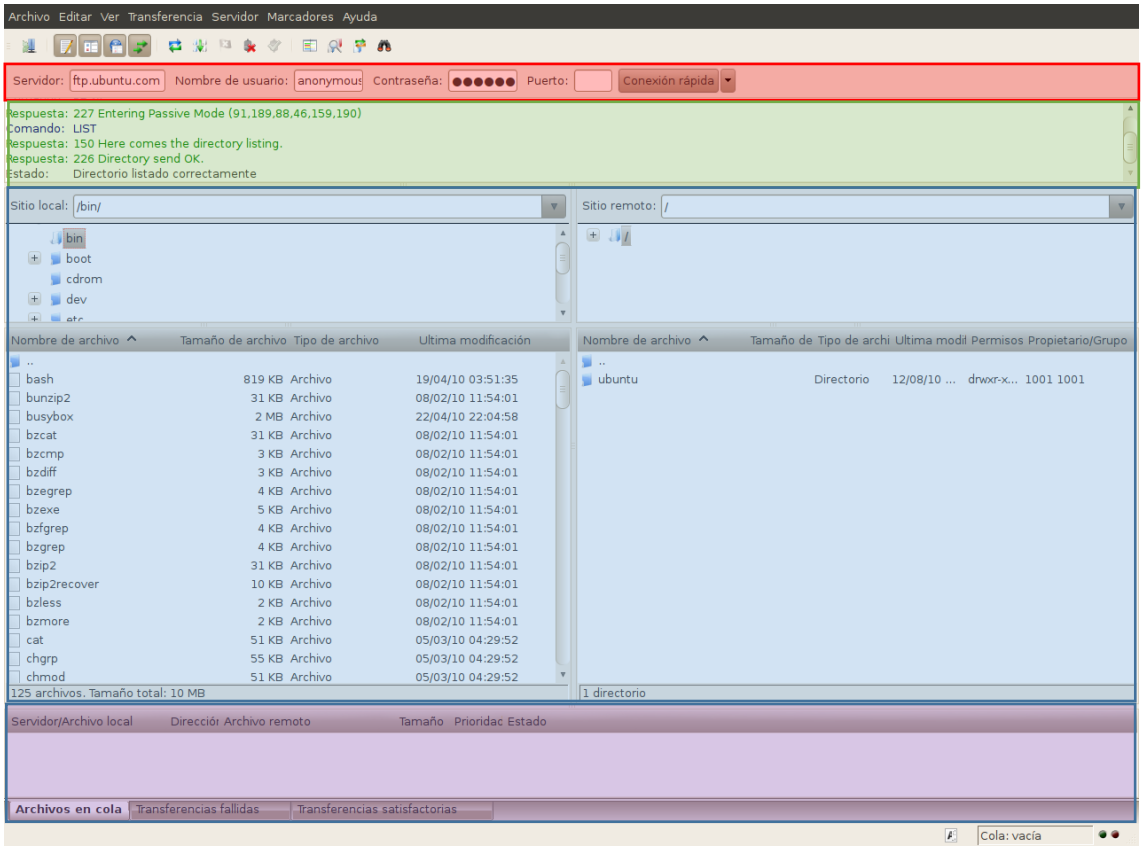


Figura 42. Ventana principal FileZilla

- La parte superior, remarcado en rojo, muestra los parámetros de conexión como son: el servidor al que conectarse, el nombre de usuario, la contraseña y el puerto. Para puerto FTP se utilizará el puerto 21.
- Bajo de la anterior y remarcado en verde, se muestra el registro de mensajes en forma de consola de comandos enviados por FileZilla y las respuestas del servidor remoto.
- En azul en la parte central de la ventana, se proporciona una interfaz gráfica para FTP. Los usuarios pueden navegar por carpetas y alterar sus contenidos tanto en el pc local como en el remoto. Esta interfaz se muestra en tipo de árbol de exploración. El intercambio de archivos es muy sencillo, puesto que solo se deben de arrastrar en cualquiera de las dos partes.
- La parte inferior y de morado muestra la cola de transferencia. Muestra en tiempo real el estado de cada transferencia activa o en cola.

3.2. Hardware

En este apartado se mostrarán todas las herramientas relativas a hardware que se han utilizado para desarrollar el proyecto, así como algunos dispositivos que formarán parte del sistema completo.

3.2.1. Raspberry pi

La raspberry pi es un sistema microprocesador de bajo coste y reducido tamaño que fue desarrollado por la Fundación Rapsberry Pi con el objetivo de estimular la enseñaza de ciencias de la computación, y campos relacionados, en las

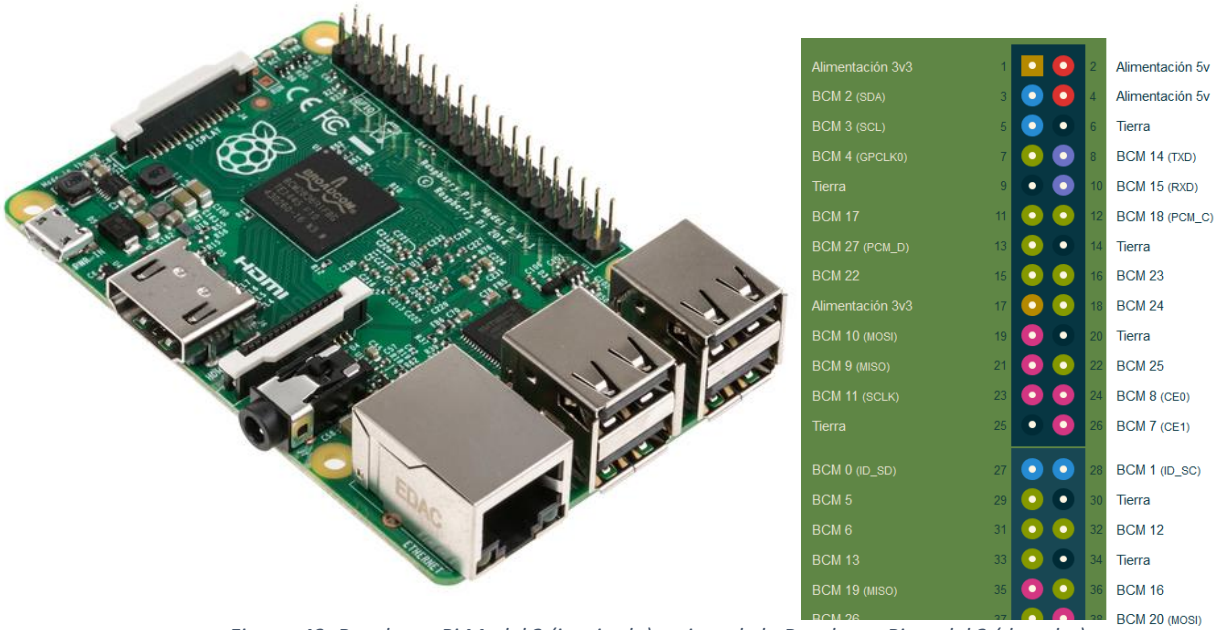


Figura 43. Raspberry Pi Model 2 (izquierda) y pines de la Raspberry Pi model 2 (derecha)

escuelas. Se trata de un producto con propiedad registrada pero de uso libre. Cualquiera puede fabricarlas o distribuirlas. De esa forma mantienen el control de la plataforma pero permitiendo su uso libre tanto a nivel educativo como particular.
El sistema operativo oficial que promueve la fundación es una versión adaptada de Debian, llamada Raspbian, aunque permite otros sistemas operativos incluidos una versión de Windows 10. En la web oficial de la fundación podemos descargar gratuitamente varios de los sistemas operativos con los que Raspberry Pi puede funcionar. La fundación cuenta con una comunidad activa que permite añadir adeptos a esta plataforma con ejemplos, proyectos relativamente fáciles de desarrollar, soporte de ayuda, foros de discusión e incluso blog, entre otras cosas.

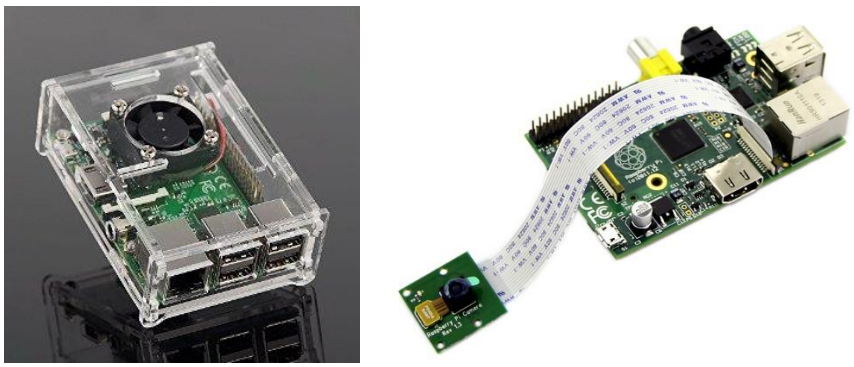


Figura 434. Funda con ventilador Raspberry Pi (izquierda) y pi camera (derecha)

Cuenta con varios periféricos adaptados directamente para la placa como la Pi camera, o tiras de cables para el puerto GPIO, además de cajas con ventilación integrada, pantallas táctiles, etc.
La versión que se ha utilizado para desarrollar el presente proyecto ha sido las Raspberry Pi modelo 2. En la siguiente tabla se ven las características técnicas más relevantes extraídas de su datasheet:

SoC	Broadcom BCM2836
CPU	ARM11 ARMv7 ARM cortex-A7 4 núcleos a 900 MHz
Overclocking	Lo permite dentro de unos límites
GPU	Broadcom VideoCore IV 250 MHz
RAM	1 GB LDDR2 SDRAM 450MHz
USB 2.0	4
Salidas de vídeo	HDMI 1.4 a 1920x1200 píxeles
Almacenamiento	Externa, micro SD
Ethernet	Sí, 10/100 Mbps
Tamaño	85.60x56.5 mm
Peso	45g
Consumo	5V, 900 mA (aunque depende de la carga de trabajo de los 4 cores)
Pines	40
Precio	35 $

Tabla 13. Especificaciones Raspberry Pi model 2

La siguiente imagen esquematiza un diagrama de bloques en el que se pueden ver sus elementos más relevantes:

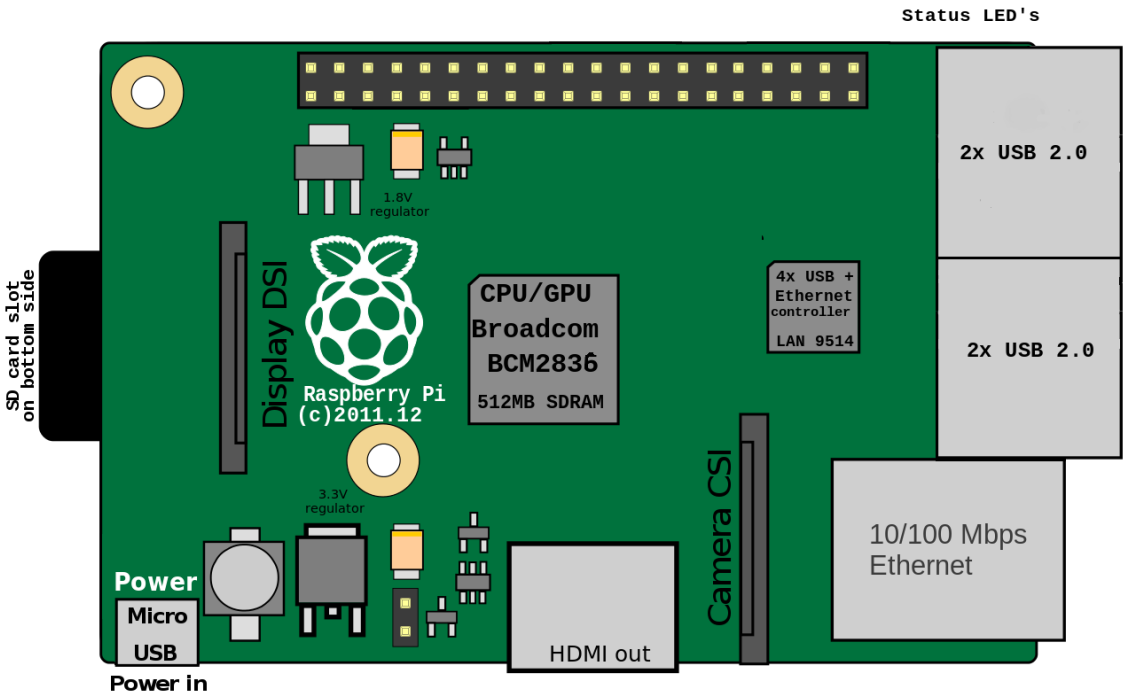


Figura 45. Diagrama bloques de Raspberry Pi model 2

3.2.2. Interfaz VPN (antena) red Zigbee

La antena RF será utilizada en este proyecto para la comunicación entre la red de nodos y Gateway, y entre los mismos nodos, que serán distribuidos en una zona siguiendo unas señas de distancia y visión directa de antenas.
Una antena es un dispositivo diseñado con el objetivo de emitir y recibir ondas electromagnéticas hacia el espacio libre. La parte de la antena que transforma la energía eléctrica en ondas electromagnéticas se llama transmisora, mientras que la parte de la antena que recibe ondas electromagnéticas y las convierte en señales eléctricas se le llama receptora. Las características de las antenas dependen de la relación entre sus dimensiones y la longitud de onda de las señales transmitidas o recibidas. Si las dimensiones de la antena son mucho menores que la longitud de onda se les denominan elementales, si tienen dimensiones del orden de media longitud de onda se les llaman resonantes, y si su tamaño es mucho mayor que la longitud de onda se trata de antenas directivas.

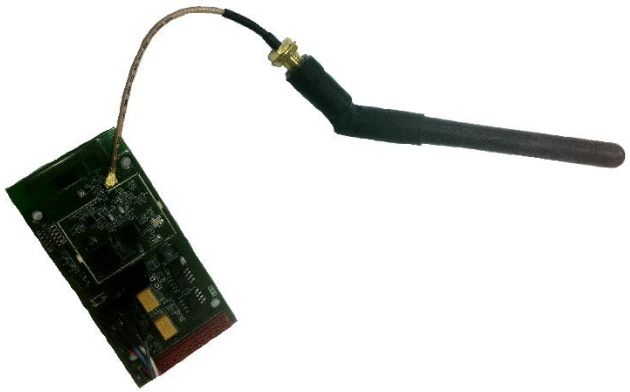


Figura 46. Antena RF

3.2.3. Módem GSM/GPRS

El módem USB es un dispositivo que permite conectar cualquier otro dispositivo, que tenga integrado una entrada USB, a Internet desde cualquier sitio, sin instalaciones y, por lo general, sin alta de línea ni consumo mínimo. La conexión a Internet por USB se realiza a través de una tarjeta SIM, que previamente el usuario habrá tenido que obtener de alguna de las compañías telefónicas existentes. Este dispositivo utiliza a la red de telefonía móvil. Utilizan conexión 3G si ésta está disponible en la zona de uso, aunque puede conectarse a GPRS o GSM si no hay buena cobertura.

Dependiendo de la aplicación final, puede resultar suficiente una conexión con una tasa no demasiada velocidad de datos. En el caso de estudio funciona perfectamente ya que simplemente enviará tramas de datos correspondientes a valores de lectura de sensores o respuestas de acciones entre otras, es decir, sin imágenes, audios ni, por supuesto, vídeos.

El funcionamiento de estos dispositivos es relativamente simple. Una vez obtenida la tarjeta SIM, y estando ésta totalmente operativa, la introducimos dentro del módem USB de manera que los conectores dorados de la tarjeta estén en contacto con los pines de entrada del modem. Una vez introducida se cierra la tapa para evitar cualquier tipo de movimiento de la SIM y quedando esta aprisionada dentro de la carcasa y en contacto con los pines anteriormente nombrados. Finalmente se conecta el USB al dispositivo e inmediatamente se obtiene conexión a Internet, siempre y cuando tengamos cobertura en el lugar de uso.

Dependiendo de la zona donde se realice la instalación que necesite conexión obtendremos una cobertura u otra, esto también depende de los operadores. Es interesante realizar una búsqueda de mapas de coberturas antes de contratar una SIM u otra ya que si la instalación se realiza en ciudades o alrededores no habrá ninguna complicación, aunque sí bien es cierto que en lugares remotos poco accesibles como montes, grandes extensiones agrarias, etc, no será tan fácil. Es conveniente realizar un estudio de cobertura en función de la zona donde se vaya

a utilizar. Existen webs que proporcionan mapas de coberturas según operadores. La imagen siguiente muestra un mapa con los distintos niveles de cobertura existentes en la isla de Mallorca.

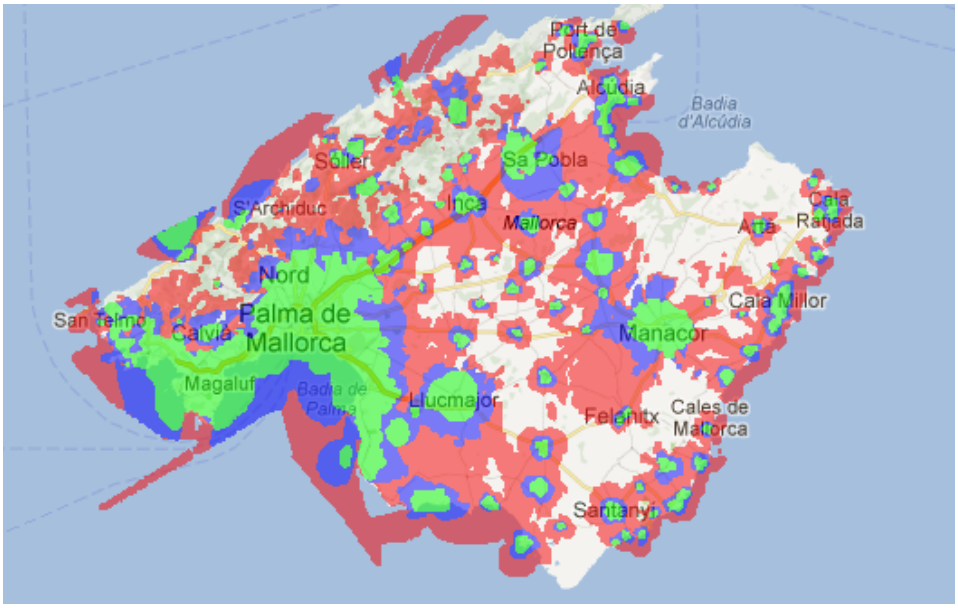


Figura 47. Mapa de coberturas de Mallorca

En cuanto a las especificaciones técnicas, la siguiente tabla muestra las más relevantes del modem USB E173 de la compañía Huawei:

Dimensions	Height: 84 mm Width: 27 mm Depth: 11.8 mm
Weight	< 30 g
Communication System	UMTS/HSUPAGSM/GPRS/EDGE/3G
Speed	HSDPA 7.2Mbps/HSUPA 5.76Mbps
microSD card slot	Yes
External Antenna Interface	Optional
Receive Diversity	Yes
Operation System	Windows XP, Windows Vista, Windows 7, Windows 8, Mac OS X 10.5, Mac OS X 10.6, Mac OS X10.7, Mac OS X 10.8

Tabla 14. Especificaciones modem usb E173 de Huawei

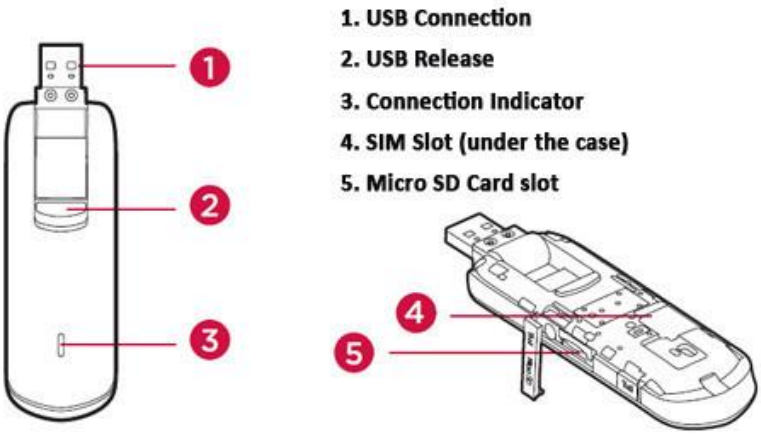


Figura 48. Esquema modem usb

3.2.4. Programador-Interfaz RS-232

Para visualizar por pantalla la comunicación del nodo se ha utilizado un conector RS-232 ya que el nodo cuenta con este puerto de comunicación. El puerto del nodo no está configurado como tal, sino que cuenta con los pines en abierto. Para ello se ha desarrollado un dispositivo interfaz que conecta estos pines con un conector RS-233 conectable al pc. El dispositivo cuenta con el encapsulado MAX3232C de Texas Instrument. El dispositivo consta de dos conductores de línea, dos receptores y un circuito doble de carga con protección de ±15kV de ESD. El dispositivo cumple con la norma TIA/EIA-232-F y proporciona una interfaz eléctrica entre una comunicación asíncrona y el conector de puerto serie. Opera con una tensión de alimentación de 3.3 a 5V, una tasa de datos de hasta 250Kbit/s

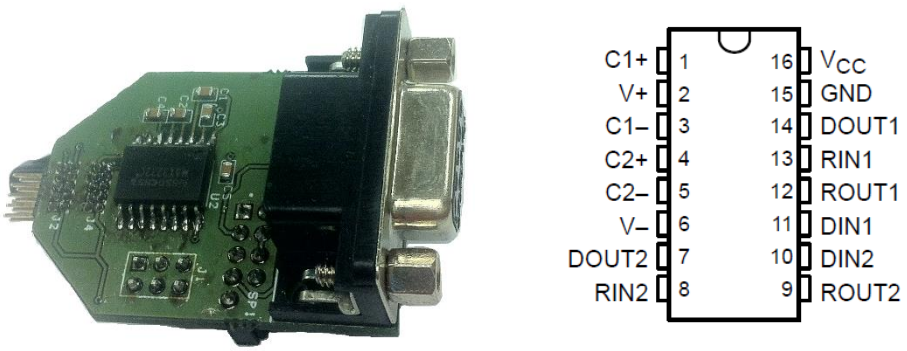


Figura 49. Conector RS-232 (izquierda) y esquema MAX3232C de Texas Instrument (derecha)

4. Desarrollo del Gateway

En el presente aparatado se mostrará al lector las diferentes fases en desarrollo del Gateway. Se explicará el funcionamiento completo del equipo para obtener una información básica que ayudará a comprender y a seguir los posteriores puntos de manera efectiva.

4.1. Descripción del sistema RfreeNet

El sistema RfreeNet es una solución completa de monitorización y control medioambiental desarrollada por la empresa Balmart S.L.. Está fabricado bajo los estándares de calidad ISO9001 e ISO14001. Es un sistema compatible con otras tecnologías software y hardware.
Está compuesto por una red de nodos sensores, un Gateway, un servidor central que procesa los datos y una herramienta web con la que poder gestionar dichas redes y actuar sobre ellas.
La red de nodos hace uso del estándar ZigBee (802.15.4) extendiéndolo a un protocolo propietario. El desarrollo de esta red surge por la necesidad de sensar distintos parámetros en lugares de difícil acceso o aisladas de redes de energía por la cual se debe utilizar protocolos orientados a bajo consumo, ampliando su autonomía.
Los nodos, serán distribuidos por la región objeto de monitorización o control ya sea en campo abierto (outdoor) o en edificios (indoor). La estructura propuesta presenta una topología basada en un único coordinador (gateway) y equipos terminales de red de medición. La distancia entre nodos puede llegar hasta 800 metros con visión directa entre antenas, abarcando grandes extensiones con un número reducido de dispositivos. Una de las características que hacen este sistema atractivo para distintos propósitos es la capacidad de autodescubrimiento y autoenrutamiento con actualización inmediata en la base de datos, es decir, si un nodo que conforma la red se avería o queda inhabilitado por cualquier motivo, los demás buscan una ruta alternativa hasta llegar al Gateway o coordinador de manera automática.

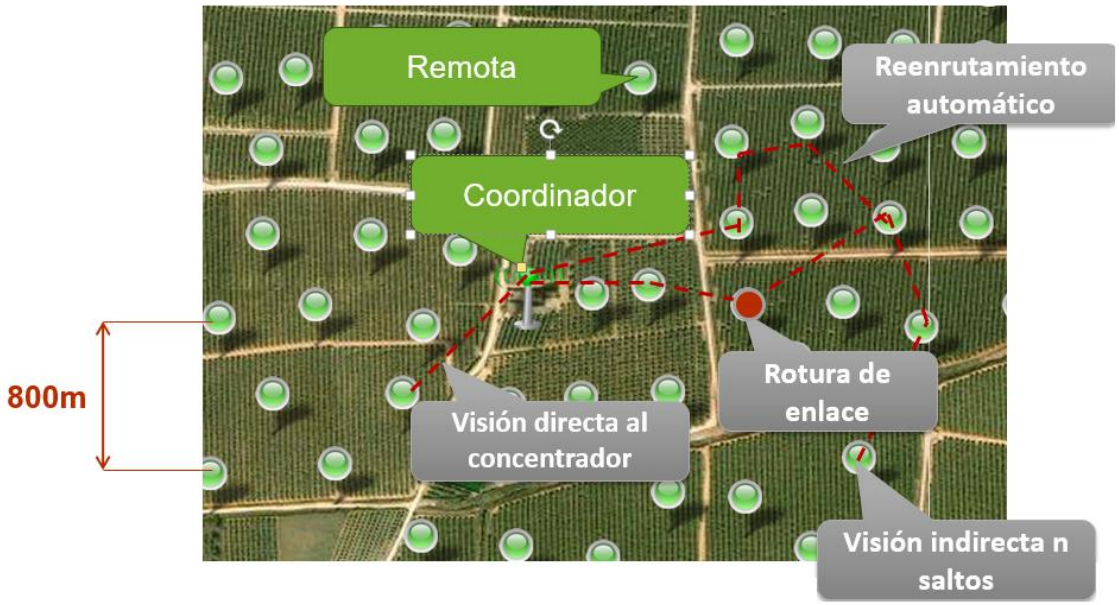


Figura 50. Mapa RfreeNet ejemplo de nodos y gateway

El dispositivo UTR (unidad de transmisión remota) o nodo presenta las siguientes características de funcionalidad:

- Basados en la ampliación del estándar 802.15.4 de largo alcance:
  - No presenta interferencias.
  - Permite comunicación bidireccional.
  - Alcance de hasta 100 metros en interior y hasta 800 metros en campo abierto.
  - Tasas de transferencia de datos hasta 250Kbps.
  - Permite crear redes de datos como internet, con identificadores de usuarios ilimitados.
  - Permite medir distancia del nodo vecino en función de la potencia de RF emitida.
  - Muy bajo consumo.
  - Frente a la tecnología wifi abarata de manera importante los costes si el entorno de gestión es dentro de los edificios o zonas colindantes.
- Aplicable para múltiples propósitos:
  - Información en tiempo real.
  - Gran variedad de sensores compatibles.
  - Tolerancia a fallos.
  - Localización exacta de nodos.
  - Adaptación a características de terreno y de cobertura radioeléctrica.
  - Comprobación de red y autorutado.
- Fácil instalación y bajo coste:
  - Instalación mínimamente invasiva.
  - Equipos de tamaño reducido.

Los nodos recibirán y retransmitirán tramas de coordinación del Gateway junto con tablas de enrutamiento a otros nodos de la red. También reciben parámetros de alarmas de gradiente y niveles de las distintas medidas para poder generar alarmas. Recibe también su ciclo de trabajo (dutty cycle) para parametrizar el consumo y la latencia de las alarmas.
Si salta una alarma, el nodo transmitirá la información de manera continua hasta que la reciba el nodo contiguo o en su caso el secundario.
Si el nodo no tiene información o alarma que transmitir estará 'dormido' la mayor parte del tiempo para ahorra energía y únicamente abrirá una ventana temporal de recepción de mensajes cada periodo T.

Figura 51. Periodo de recepción de mensajes

La latencia está marcada por la mitad del periodo y el número de saltos hasta llegar al Gateway, siendo esta relación fundamental para determinar el tiempo de vida de las baterías y el consumo del sistema.

$$L = \frac{N * T}{2}$$

El tiempo de vida de las baterías estará fijado por la siguiente expresión:

$$LifeTime = \frac{Q}{I_m} = \frac{Q}{I_r * \frac{R}{T} + I_s * \frac{(T-R)}{T} + I_t * A}$$

Donde:
I_m → intensidad media consumida por el nodo
I_r → intensidad consumida
T → periodo
R → tiempo en el que el sistema está despierto
I_s → corriente en modo 'dormido'
I_t → corriente consumida en transmisión
A → porcentaje de tiempo (peor caso) que un nodo está transmitiendo.

A continuación se muestra el diagrama de bloques de un nodo:

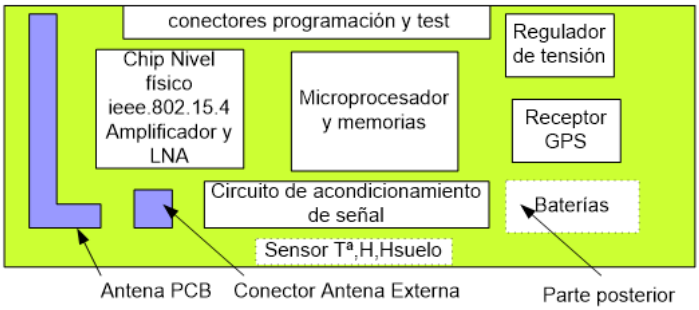


Figura 52. Diagrama de bloques de un nodo

El dispositivo Gateway realizará la tarea de coordinador y puerta de enlace entre la red de nodos y el servidor. Se encarga de recibir información de los nodos y retransmitirla al centro de control. Esta retransmisión puede realizarla por múltiples vías dependiendo del ámbito de aplicación: GPRS, Wifi, wired Ethernet, aunque por sus características, y su predilecto uso en exteriores se realizará normalmente mediante GPRS.
El Gateway recogerá la información de los nodos, generará sus propias tablas de rutado y las reenviará a los nodos. Periódicamente realizará pollings para comprobar el estado de la red y recogerá la información de parámetros de sensores. La frecuencia del polling será programable por el centro de control.
Estará alimentado por baterías recargadas por paneles solares o red eléctrica que mantendrán una autonomía de una semana en ausencia de fuente solar o red.
A continuación se muestra el diagrama de bloques del Gateway:

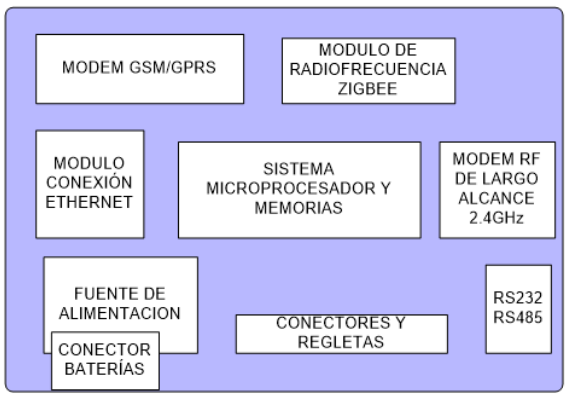


Figura 53. Diagrama de bloques de la versión anterior del gateway

4.2. Funcionamiento básico

Obtenida una breve explicación del sistema RfreeNet, el lector está en disposición de conocer el funcionamiento del nuevo Gateway, objeto del proyecto.
Al proporcionar alimentación al Gateway, y previa colocación del modem GSM y la antena RF, éste está en disposición de empezar el protocolo que tiene asignado al disponer de energía.
El firmware de control de las acciones y comunicación está formado principalmente por 3 scripts: bucle.py, funciones.py y analiza.py. Estos son los nombres de cada uno de los módulos que interactúan entre sí con el único objetivo de ejecutar las acciones pertinentes siempre que se requieran.
Según su función, estos tres scripts pueden ser clasificados como:
Bucle.py: mantendría el roll de bucle principal.
Analiza.py: el roll de este script sería el de parsear cada una de las tramas que lleguen al Gateway tanto por TCP como por RF.
Funciones.py: podría decirse que se trata del repositorio principal de todas las funciones que se ejecutan durante el funcionamiento del gateway.

En primer lugar se muestra el funcionamiento del bucle principal.

1. Envía trama de arranque y primer *alive*. Esta acción se realiza para alertar al servidor de que el Gateway, asignado previamente en la web su código de identificación, está conectado y en pleno funcionamiento. La trama *alive* sirve también para comparar la hora y fecha del sistema central, en este caso la del servidor, con la propia y modificarla en el caso de que no coincidan.
2. Una vez inicializado el Gateway y, con conexión al servidor, el socket TCP-IP mediante el cual obtendremos la comunicación con el servidor se mantendrá en modo escucha y esperando a recibir información del servidor. Este proceso tiene un timeout de 3 segundos. Si durante este tiempo se recibe alguna orden, el parseador interpretará la trama y realizará las acciones oportunas. Si agotado el tiempo de espera no se recibe ninguna trama, el programa pasará a escuchar el puerto serie, asociado en este caso a la comunicación con los nodos mediante RF.
3. El puerto serie, se mantendrá en escucha esperando alguna trama recibida desde los nodos. También consta de un timeout. Si durante este tiempo no se recibe ninguna trama el programa saltará a las líneas de escucha del socket. Por el contrario, si se reciben tramas el parseador analizará la trama y realizará las acciones predeterminadas.
4. Una vez cualquiera de los módulos, socket o puerto serie, reciben, parsean y ejecutan acciones la escucha pasará automáticamente a el módulo contrario del asociado a las acciones realizadas.

Si el Gateway quedara sin conexión al servidor, saltaría una excepción que cerraría el socket que estaba utilizando para intentar abrir otro. Durante este proceso el bucle principal continuaría ejecutándose de manera que la

comunicación con el puerto serie, la red de nodos, seguiría estando activa y el Gateway recopilando las tramas si las hubiese para enviarlas de nuevo una vez establecida la nueva comunicación con el servidor.

El siguiente flujograma muestra el funcionamiento principal del programa. Es decir, la inicialización y el bucle principal donde intervienen las escuchas del socket y el puerto serie y el parseador de cada una de las fuentes.

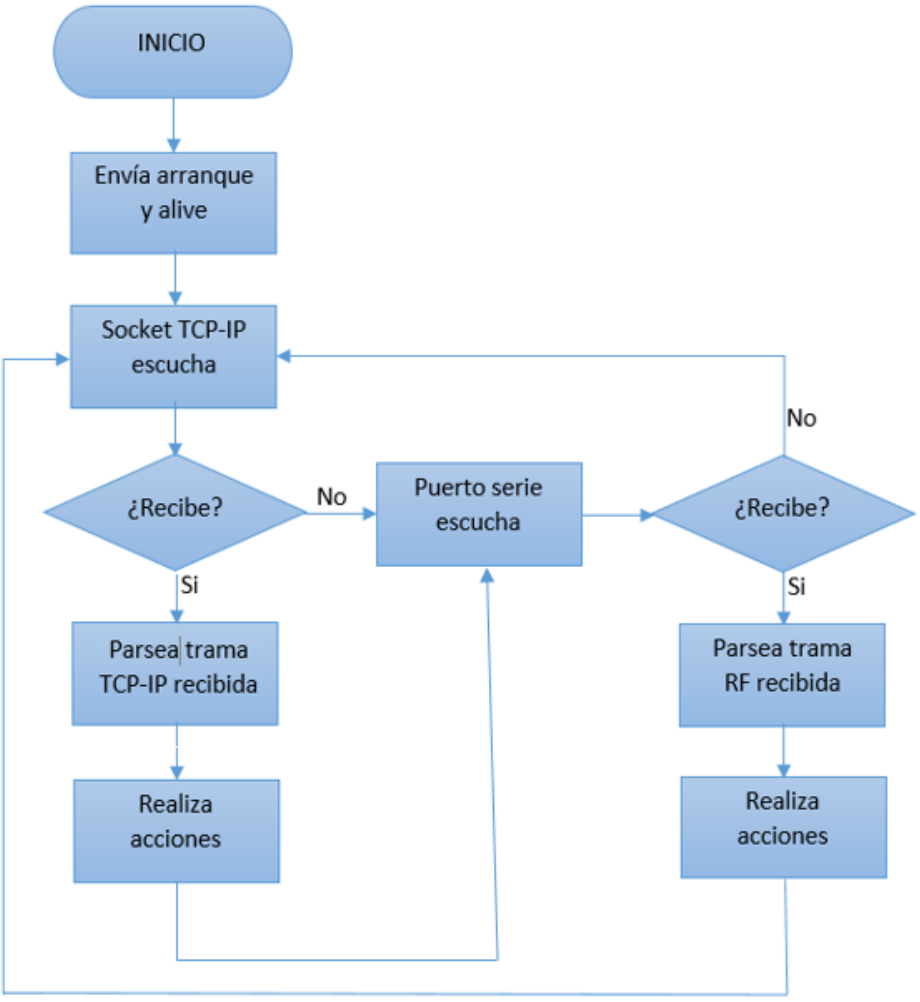


Figura 54. Diagrama de flujo bucle principal

Conocido el funcionamiento del bucle principal, podemos desglosarlo y comprobar de qué manera el parser clasifica y analiza las tramas. Antes de esto, vamos a obtener una visión general del conjunto de las tramas que intervienen.

Para ello vamos a diferenciar dos tipos de tramas: las tramas de comunicación servidor-Gateway/Gateway servidor, que contarán con una estructura similar todas ellas, y las tramas nodos-gateway/Gateway-nodos.
La comunicación inicial siempre es establecida por el Gateway mediante la trama de alive. Una vez enviada esta trama, el servidor puede realizar peticiones al Gateway. En los siguientes apartados se desglosarán las tramas de comunicación así como el funcionamiento del parser.

4.2.1. Tramas Gateway-servidor/servidor-Gateway

Las especificaciones proporcionadas por el I3M constaban con un punto en el que se encontraban todas las tramas a implementar y sus funciones. Este apartado servirá para ver la estructura de las tramas y así ver cómo han sido parseadas, en ningún caso se explicará la estructura particular de cada una de ellas, ya que esa información puede ser consultada en el anexo 8.1 índice de tramas.
Puntualizado esto, a continuación se muestra la estructura principal de estas tramas. La siguiente trama corresponde a una petición por parte del servidor lectura de parámetros.

Start		Terminal	Comando	Identificador	Checksum	**Stop**
0x02	'T'	C	'L'	ID_{SUS}	CS	**0x05**

Tabla 15. Estructura trama de comunicación servidor

Como se puede apreciar, las tramas comenzarán con un byte de **start** correspondiente a '0x02' y con un byte de **stop** correspondiente a '0x05'. Estos dos bytes son muy relevantes en cuanto a parseo se refiere ya que nos indicarán el principio y fin de cada trama.

El tercer dato **C** indica el tipo de terminal o consola con el que se está comunicando. Para este caso el código será 'X'.

El cuarto dato, en este caso de ejemplo **L,** indica el comando del que se trata. Este dato es muy útil en cuanto a parseo de la trama, ya que indicará que tipo de petición ha sido realizada.

El parámetro $\mathbf{ID_{SUS}}$ corresponde al identificador del Gateway. Este número identificador consta de 4 dígitos asignados desde la web al Gateway y almacenado en este. Sirve para diferenciar distintos gateways y no confundir

tramas que vayan dirigidas a uno u otro. También para que el servidor sepa que Gateway está respondiendo a alguna de sus peticiones.

El **checksum** corresponde a un byte que verifica que la trama ha sido recibida completa y no se ha corrompido. El servidor envía este byte y el Gateway lo comparará con el que el mismo obtenga analizando la trama recibida ¿Cómo lo hace? Tanto el servidor como el Gateway tienen asignada la función que corresponde a la obtención del checksum. En este caso la operación que se realiza es la siguiente: XOR de toda la trama exceptuando el byte de start, el checksum recibido y el stop. Así comparando los dos resultados verificamos que la trama recibida no ha sufrido pérdidas.

Esta es la estructura general de las tramas de comunicación entre Gateway-servidor y viceversa. No obstante, existen variaciones dependiendo de la información que se reclame en cada momento. Por ejemplo, algunas de las tramas como la de 'alive' evento de arranque, reset, o evento de batería incluyen información sobre la dirección ip, el puerto por el cual se comunica, el timestamp, entre otras, que se deberán estructurar y enviar en el formato predefinido para que el servidor interprete correctamente la información.

Siempre que el Gateway establezca comunicación o sea el primero en enviar una trama al servidor, este deberá contestar de manera que el Gateway obtenga una confirmación de que la trama ha llegado y que lo ha hecho de manera correcta. En este punto encontramos dos tipos de respuestas:

- La primera, contestará con una trama de estructura similar a la general, pero con las variaciones pertinentes a la trama recibida.
- La segunda, confirmará la correcta recepción con un ACK (del inglés acknowledgement, o acuse de recibo) con un valor de 0x06 y la incorrecta con un NACK (del inglés negative acknowledgement, o acuse de recibo negativo) con un valor de 0x0F

Llegados a este punto vamos a listar todas las tramas que intervienen en la comunicación entre Gateway-servidor y su utilidad:

- Tramas en las que el Gateway tiene la iniciativa de comunicación.
  - *Alive*: es la trama que indica al servidor que el Gateway está activo y para actualizar la fecha y hora del sistema comparándola con la del servidor.
  - Arranque: indica al servidor que el Gateway acaba de ser conectado.
  - Batería: proporciona el nivel de batería del Gateway.
  - Sensores: transmite las medidas de los sensores que el Gateway tenga conectados.

La respuesta del servidor a estas tramas será un ACK o NACK, excepto para la trama de *alive*.

- Tramas en las que el servidor solicita respuesta al Gateway:

- Lectura de parámetros: solicita la configuración de los parámetros del Gateway.
- Lectura de alias: solicita el alias almacenado en Gateway.
- Modificación de parámetros Gateway: envía los parámetros a modificar en el Gateway.
- Modificación de alias: modifica el alias almacenado en el Gateway.
- Forzado de datos: realiza una petición forzosa de los datos, sensados, al Gateway.
- Cambio de estados salidas Gateway: solicita al Gateway un cambio de estados a sus actuadores.
- Reset del Gateway: solicita al Gateway que se resetee.

La respuesta a estas tramas por parte del Gateway será una trama con los datos solicitados.

4.2.2. Tramas Gateway-nodos/nodos-gateway

Como en el anterior punto se indica, las tramas vienen definidas en las especificaciones de desarrollo, por lo tanto se va a ver la estrutura general de las tramas de comunicación entre Gateway y nodos mediante RF.

La estructura general de las tramas se muestra a continuación:

START	L	IP2	IP1	IP0	TIPO	DATOS	CS	STOP
0x02	0x04	Iputr[0]	Iputr[1]	Iputr[2]	'L'	- - -	CS	**0x05**

Tabla 16. Estructura trama comunicación RF

Igual que en las tramas entre Gateway y servidor, se aprecia el byte de **start** y **stop** correspondientes en este caso también a 0x02 y 0x05 respectivamente.

El dato **L** lo forma la suma de la longitud en bytes de los datos relevantes de la trama (IP2, IP1, IP0, TIPO, DATOS), entendido por relevantes los datos que dan información en cuanto a parámetros y no protocolarios en cuanto a comunicación, como start, stop o checksum.

Los apartados **IP2, IP1** e **IP0,** en realidad conforman un único dato, la IP asignada a ese nodo en concreto. Se representan mediante 3 dígitos ASCII. Cumplen la función de identificador de nodo, para saber en todo momento a que nodo va referida la trama enviada.

El **TIPO**, al igual que en las tramas entre Gateway y servidor, sirve para indicar que tipo de petición es la demandada.

El apartado **DATOS** incluirá la respuesta a la petición realizada. Cada una de las peticiones realizadas corresponderán a una acción distinta, por lo tanto este

apartado tendrá distintas estructuras dependiendo de la información a transmitir. Este campo puede quedar vacío si la petición no incluye dato alguno en la respuesta.

El **checksum,** como en el apartado anterior sirve para comprobar que la trama ha llegado completa. En este caso la operación que se realiza para obtenerlo es: IP+TIPO`DATOS, y ocupará 1 byte.

A continuación se listarán todas las tramas de comunicación entre el servidor-gateway-nodos. A tener en cuenta, todas las tramas solicitudes se pedirán desde el servidor hasta los nodos pasando por el Gateway, y viceversa en la respuesta de estos.

- Lectura de parámetros utr: solicita la configuración de un nodo concreto de la red.
- Lectura de alias utr: solicita el alias de un nodo concreto. Esta información está almacenada en el Gateway.
- Modificación de parámetros utr: solicita una modificación de los parámetros configurables de un nodo en concreto.
- Modificación de alias utr: solicita un cambio de alias de un nodo en concreto. Como anteriormente, dicha información se almacena en Gateway.
- Forzado de datos utr: fuerza el envío de las medidas de los sensores de todos los nodos.
- Forzado de estadísticas: fuerza el envío de la cobertura y el nivel de batería de todos los nodos de la red.
- Reset utr: solicita un reseteo del nodo seleccionado.
- Elimina utr: elimina un nodo de la red. Esta información la almacena el Gateway. El nodo en cuestión no es consciente aunque ninguna trama que envíe será direccionada por el Gateway hacia el servidor.
- Petición de ip utr: la iniciativa de comunicación de esta trama la tiene el nodo que reclama una dirección ip dentro de la red. Dicha dirección será asignada por el servidor.
- Actuación de salidas utr: solicita un cambio de estados en los actuadores de un nodo en concreto.
- Evento de datos utr: el nodo transmitirá la información de sensado de manera automática con una periodicidad previamente seleccionada por el usuario desde la web.
- Evento alarmas umbral/gradiente de utr: el nodo que haya obtenido unos valores de sensado que sobrepasen sus rangos de gradiente o umbral programados por el usuario, enviarán esta trama de alerta.
- Evento estadísticas utr: los nodos enviarán sus niveles de cobertura y batería con la periodicidad que se le haya programado.

4.3. Parser

En este apartado, y previamente obtenidos los conocimientos generales de cómo se conforman las tramas y su estructura general, se muestra el funcionamiento del parseador para discriminar las tramas y realizar las acciones que correspondan. El parseador constará de dos funciones principales, la primera analizará las tramas que correspondan a la comunicación TCP y la segunda a las tramas recibidas mediante el puerto serie. Puesto que la estructura de las tramas en ambos casos se asemeja, se realizará un análisis exhaustivo del funcionamiento de la primera siendo extrapolable prácticamente todo el contenido a la segunda función.

- Como se ha mencionado anteriormente, podemos recibir dos tipos de tramas o respuestas. Un ACK (o NACK) o una trama de características similares a la general. Entonces, la primera discriminación que tiene configurado el parseador es si se trata de respuesta ACK (o NACK) o trama general.
- Si se trata de una trama general, el siguiente objetivo será encontrar el byte de start y de stop. En el caso de no existencia de estos bytes, el programa llegará a su fin.
- Si se encuentran start y stop, obtenemos el resto de trama que llamaremos sección de interés. El primer dato de esta sección de interés lo llamaremos dato1. Es de mencionar que este dato1 puede ser de tres tipos:
    - 'i': corresponde a una petición de ip por parte de un nodo. Únicamente se encontrará 'i' en la posición dicha para este tipo de petición.
    - 'h': corresponde a un envío de *alive* donde se comparará la hora y la fecha del gateway con el del sistema central.
    - 'T': este será el encontrado en la mayoría de las tramas, a excepción de los dos casos anteriores. Por tanto continuaremos el análisis del funcionamiento del parseador suponiendo que el dato1 recibido corresponde a 'T'.
- En siguiente lugar se realizará una comparación del identificador que viene incluido en la trama con el identificador que el Gateway tienen almacenado.
- Comprobado el identificador del Gateway se realizará el cálculo del checksum para posteriormente compararlo con el valor que se ha recibido en la trama.
- Comprobado el checksum, el parseador discriminará las tramas según el comando al que correspondan. Encontramos 10 tipos de comandos:
    - L: lectura de parámetros.
    - P: lectura de alias.
    - N: modificación de alias.
    - M: modificación de parámetros.
    - R: reset.
    - A: forzado de datos.

- E: forzado de estadísticas.
- O: cambio de estado de salidas.
- T: lectura de estructura de red.
- K: estructura de red.

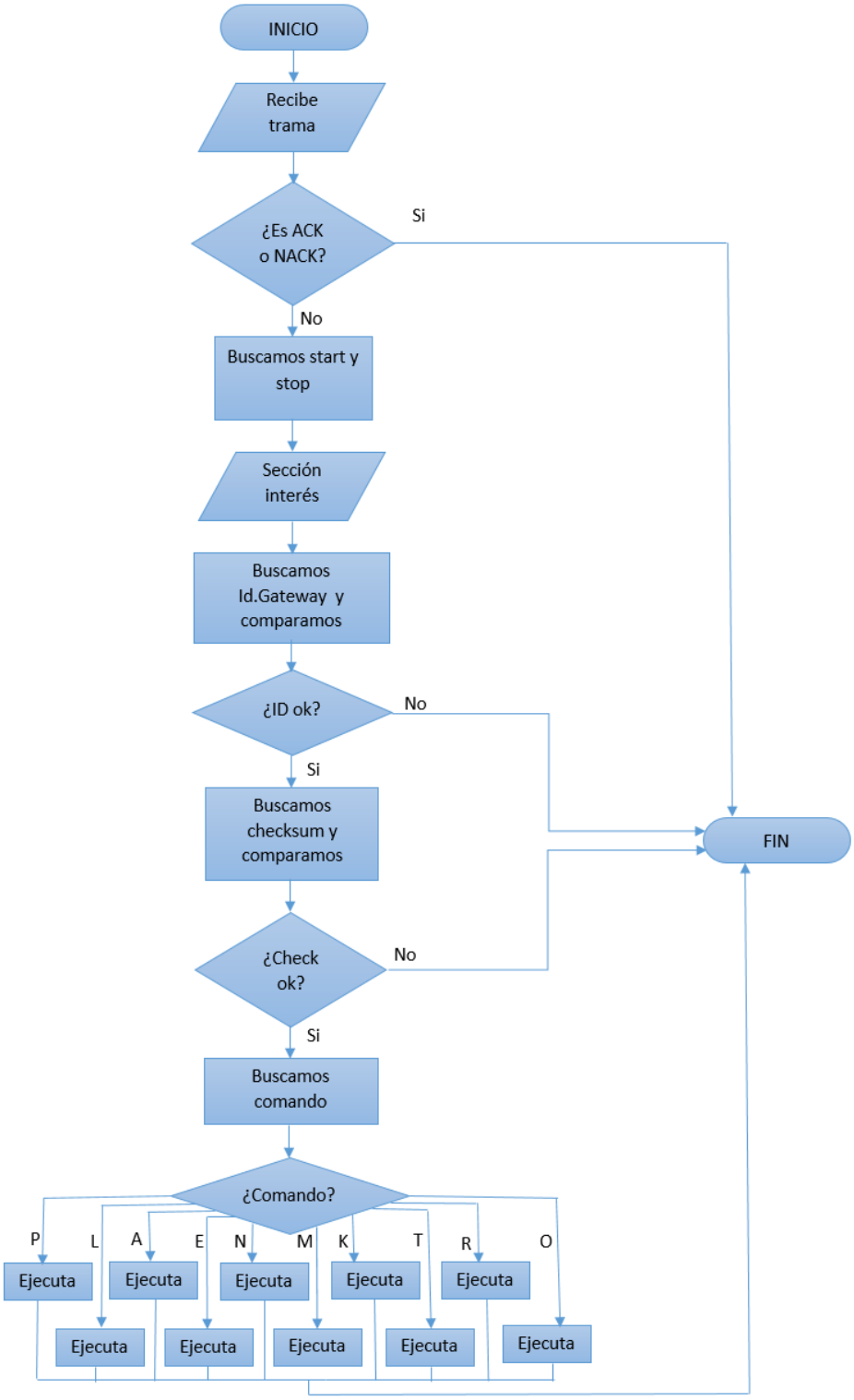


Figura 55. Diagrama de flujo parser

4.4. Funciones

El script funciones.py contienen todas las funciones que intervienen en el programa. Para poder analizarlas se han clasificado en tres bloques: las funciones de comunicación TCP, las funciones de comunicaciones serie y las funciones complementarias.
Las funciones de comunicación TCP y las funciones de comunicación serie, son similares en estructura. Se han agrupado en bloques diferentes por dos motivos. El primero, y el que marca la principal diferencia entre ellas, es porque cada grupo tiene un tipo de comunicación, y aunque como hemos dicho la estructura es similar, el receptor final es diferente. El segundo motivo es para obtener una lista visualmente sencilla y rápida mientras el código estaba en construcción, es decir, mientras se tuvieran que ir instanciando en los diferentes scripts facilitar la búsqueda. Véase la imagen 56.

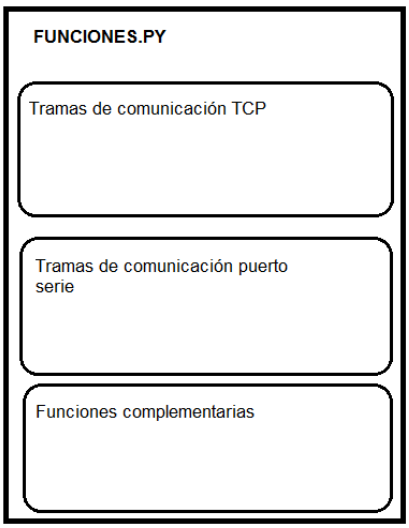
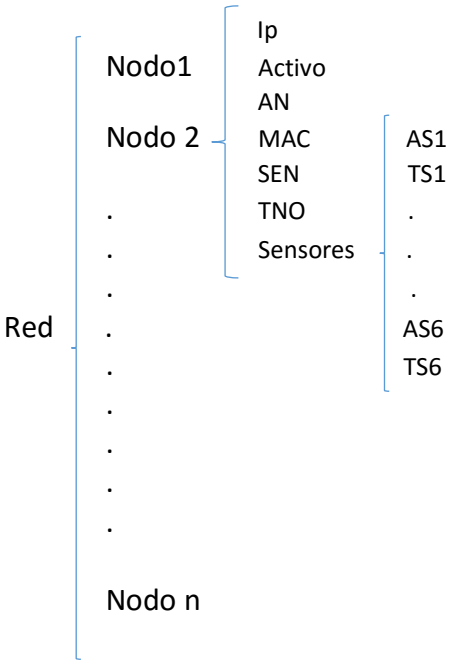


Figura 56. Esquema distribución funciones

Las funciones complementarias tienen multitud de finalidades: actualizar información de un archivo *json*, realizar conversiones sintácticas, realizar los checksums para cada caso, entre otras. En el anexo 8.2.1. funciones.py se puede consultar todas las funciones utilizadas en el programa.

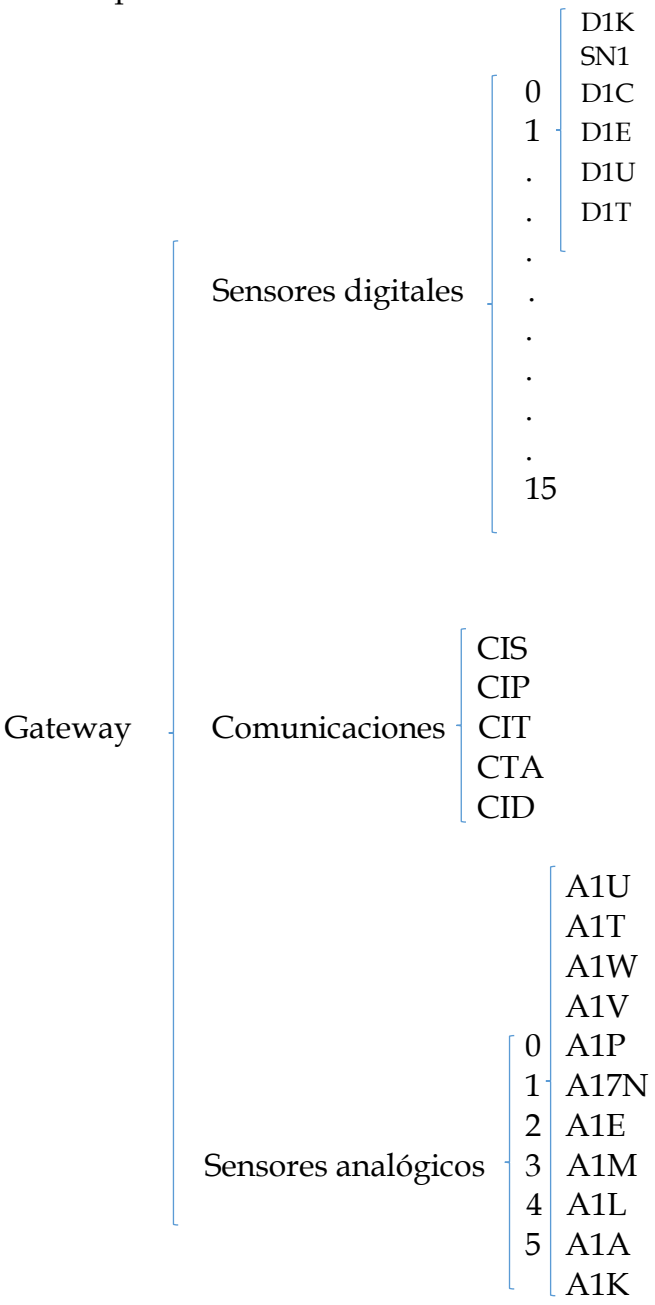
4.5. Archivos json

Existen dos archivos json que almacenarán distintos parámetros para poder realizar consultas y ser actualizados cuando así lo requiera el sistema.
El primer archivo almacenará la configuración de la red de nodos. El siguiente esquema muestra la estructura del archivo que los parámetros que almacena:



En donde:
IP: Es el número de identificador de nodo (3 caracteres).
Activo: 0 indica nodo inactivo, 1 nodo activo.
AN: nombre del nodo (hasta 20 caracteres).
MAC: la mac del nodo.
SEN: número de sensores activos.
TNO: Tipo de nodo.
Dentro del parámetro sensores encontramos hasta un máximo de 6 sensores por nodo, donde AS es el nombre del sensor y TS es el tipo.
El segundo archivo json almacenará los parámetros relativos al gateway. En el siguiente esquema podemos ver su estructura donde:
DxK: Tipo de sensor.
SNx: Numero de sensor.
DxC: Almacenar valor medido en Flash.
DxE: Envío de valor medido al Servidor.
DxU: Valor de umbral máximo.
DxT: Tiempo entre medidas.
CIS: IP del servidor principal.
CIP: Puerto TCP del servidor principal.
CIT: Puerto TCP del servidor secundario.
CTA: Tiempo de *alive* en minutos.

CID: IP del servidor secundario.
AxU: Valor de umbral máximo.
AxT: Tiempo entre medidas (en minutos).
AxW: Tiempo de *warming* (en segundos).
AxV: Almacenar vakor medido en Flash.
AxP: Alimentación del sensor.
AxN: Nº de medidas por debajo del umbral mínimo para disparo.
AxE: Envío del valor medido al Servidor.
AxM: Nº de medidas por encima del umbral máximo para disparo.
AxL: Valor de umbral mínimo.
AxA: Máscara de actuación de salidas.
AxK: Tipo de sensor.



4.6. Comunicaciones

Tras la lectura de este apartado se obtendrá una visión general de los protocolos de comunicación empleados y como se programan en Python. Se distribuirá en dos apartados: la comunicación con el servidor y la comunicación con los nodos.

4.6.1. Comunicación TCP-IP

El protocolo TCP (Transmission Control Protocol) es uno de los protocolos fundamentales de Internet. Fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. También tiene mecanismos para distinguir distintas aplicaciones dentro de una misma máquina con el concepto de puertos. Entre las características de este protocolo cabe destacar:

- Permite colocar datagramas nuevamente en orden cuando vienen del protocolo IP.
- Permite el monitoreo del flujo de los datos para evitar la saturación de red.
- Proporciona un servicio orientado a la conexión fiable i ordenado. Tres fases de conexión: establecimiento de conexión, transferencia de datos y cierre de la conexión (similar al teléfono).
- El flujo de datos se maneja como una secuencia de bytes (byte stream).
- Por motivos de flexibilidad, el protocolo no especifica la interfaz con la aplicación (sockets interface).

Como se ha mencionado anteriormente, el protocolo TCP se compone de tres fases o etapas:

1. ESTABLECIMIENTO DE CONEXIÓN

Normalmente una de las entidades, participantes en la comunicación, establece conexión abriendo un socket en un determinado puerto TCP y se queda a la escucha de nuevas conexiones. A este procedimiento se le denomina apertura pasiva y determina el lado del servidor de la conexión.

El lado cliente realiza una apertura activa de un puerto enviando un paquete SYN inicial al servidor. El computador servidor realiza un checkeo comprobando que el puerto al que el cliente se refiere este abierto y el servicio activo. En caso de no estarlo se envía un paquete con el bit RST activado lo que significa el rechazo de la conexión. En el caso de que se encuentre abierto el puerto el servidor contestará

al SYN enviándole un SYN/ACK. Finalmente el cliente deberá responder con un ACK para completar la conexión.

La siguiente imagen muestra de manera gráfica los tres pasos para la conexión.

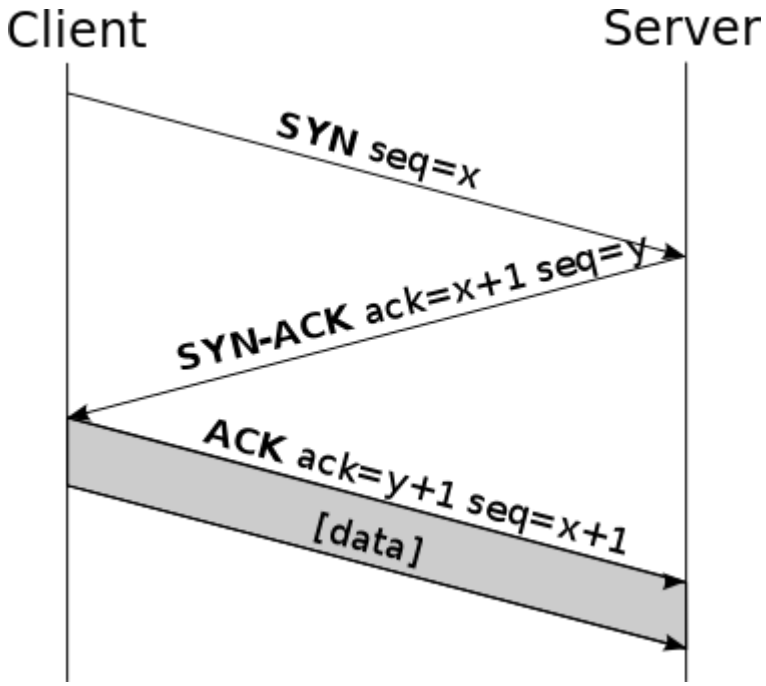


Figura 57. Protocolo de establecimiento de conexión TCP

2. TRANFERENCIA DE DATOS

Para ofrecer fiabilidad y robustez en las comunicaciones, el protocolo se dota de varios mecanismos como el uso de número de secuencia para ordenar los segmentos y detectar paquetes duplicados, checksums para detectar errores y temporizadores para detectar pérdidas y retrasos.

Los números de secuencia son intercambiados entre las dos entidades intervinientes en la comunicación. Estos números son usados para identificar los datos dentro del flujo de bytes y poder identificar los bytes de los datos de la aplicación. El emisor se refiere a su propio número de secuencia cuando habla del suyo y para referirse al del receptor utilizar número de asentamiento.

El checksum de 16 bits, consiste en el complemento a uno de la suma en complemento a uno del contenido de la cabecera y datos del segmento TCP, es calculado por el emisor, e incluido en la transmisión. El receptor TCP recalcula el checksum para comprobar que el segmento ha llegado intacto y sin errores.

3. FIN DE LA CONEXIÓN

La fase de finalización de la conexión utiliza un método de cuatro pasos llamado 'four-way handshake', terminando la conexión desde cada lado independientemente. En la siguiente imagen se observa los cuatro caminos para la desconexión.

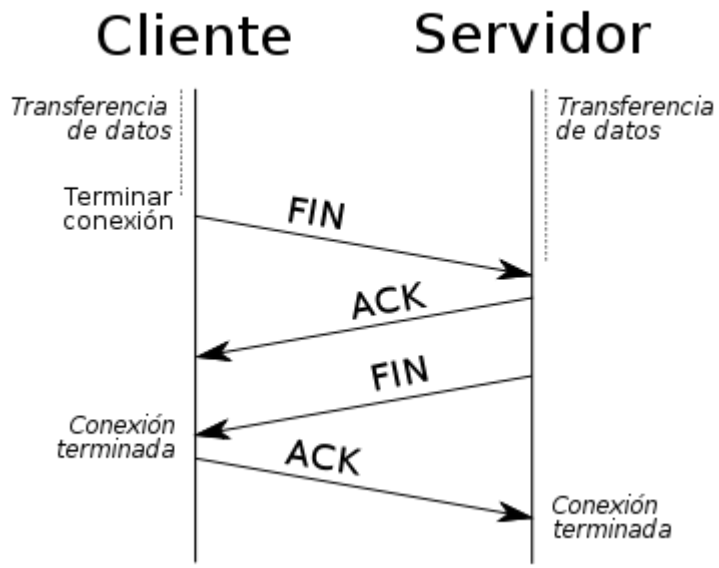


Figura 58. Protocolo fin de conexión TCP

Una vez obtenidos los conocimientos básicos sobre el protocolo TCP, se realizará una breve explicación de su implementación mediante un socket en Python.

Socket es un concepto abstracto por el cual dos programas pueden intercambiar cualquier flujo de datos. Es conocido también como interfaz de programación de aplicaciones (API). Un socket queda definido por la dirección IP de la máquina, el puerto en el que escucha y el protocolo que utiliza.

Los sockets se clasifican en sockets de flujo (`socket.SOCK_STREAM`) para protocolo TCP y socket de datagramas (`socket.SOCK_DGRAM`) para UDP.

Otra clasificación puede ser según su familia. Tenemos sockets UNIX (`socket.AF_UNIX`) que se crearon antes de la concepción de las redes y se basan en ficheros, sockets IPv (`socket.AF_INET`) y socket IPv6 (`socket.AF_INET6`).

Para crear el socket primero se debe indicar el protocolo a utilizar, en este caso TCP y la familia, en este caso IPv.

Para realizar la conexión se invoca la instrucción 'connect' relativo al objeto previamente creado 's' que define el socket y se añade la dirección IP y el puerto de conexión.

El siguiente texto muestra un ejemplo de la creación y configuración del socket TCP-IP.

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((‘158.42.105.71’, ‘6011’))
```

4.6.2. Comunicación serie (RF)

La comunicación con la red de nodos se realiza mediante RF. Pero desde el punto de vista del dispositivo la comunicación se realiza mediante el puerto serie de la Raspberry Pi.
El puerto serie es una interfaz de comunicaciones de datos digitales, frecuentemente utilizado por computadoras y periféricos donde la información es transmitida bit a bit.
Como se ha visto en apartados anteriores, la Raspberry Pi incluye una serie de pines de propósito general o GPIO (General Purpose Input/Output) entre los que se incluyen un puerto serie o UART (Universal Asynchronous Receiver-Transmitter).
Consultado el datasheet de la Raspberry Pi se observa que los pines del puerto UART son:

- Según su nomenclatura BCM (Boardcom SOC Channel) el pin transmisor TXD es el 14 y el receptor RXD es el 15.
- Según su nomenclatura BOARD el transmisor TXD es el pin 8 y el receptor RXD es el 10.

La antena cuenta con 4 conectores: alimentación (3.3V), masa (GND), transmisor (TX) y receptor (RX). En la siguiente imagen se observan las conexiones a la placa.

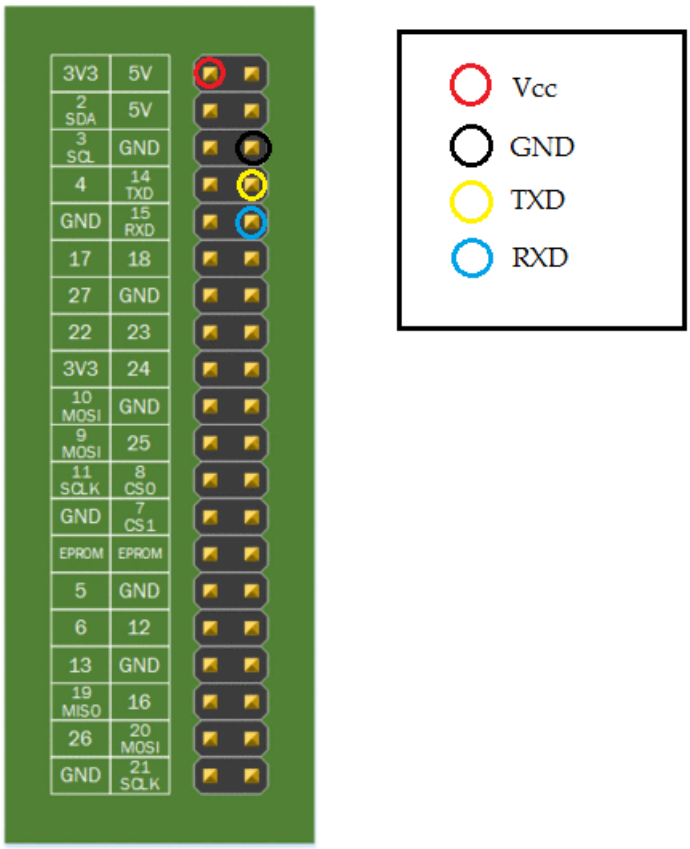


Figura 59. Conexionado antena a Raspberry Pi

Para configurar un puerto serie y realizar una comunicación correcta con otro dispositivo se debe crear el objeto (en este caso el objeto se llama antena) y se configura el puerto de comunicación (ttyAMA0) y la velocidad en baudios común entre los dos dispositivos para una correcta sincronización entre ellos (19200 baudios en este caso).
La siguiente línea de código muestra la configuración del puerto serie de la Raspberry Pi.

```
antena = serial.Serial('/dev/ttyAMA0',19200)
```

4.7. Mejoras respecto a versión antigua del Gateway

El objetivo del proyecto, además del desarrollo de todas las funcionalidades especificadas y que ya cumplía la versión anterior del Gateway, pretendía solucionar o mejorar en cuanto a funcionalidad dos aspectos que por la plataforma anterior de Gateway no podían ser resueltas.

4.7.1. Recuperación de datos ante pérdidas de conexión

La primera de estas dos mejoras era la recuperación de datos ante pérdidas de conexión, es decir, si en un momento dado el Gateway se quedará aislado con la red de nodos sin posibilidad de enviar datos hacia el servidor debía ser capaz de enviar todos los datos recibidos durante la desconexión con una tasa lo suficientemente alta para resolver el problema y no quedar retrasado largo tiempo. La solución propuesta ha sido: implementar en el bucle principal una estructura 'try' y 'except'. El programa mantendrá comunicación con el servidor hasta que se produzca la desconexión. En ese instante el socket abierto se cerrará y continuará la comunicación con la red de nodos por el puerto serie.
Las tramas recibidas de la red de nodos se almacenarán en un archivo que llamado 'log_serie.txt'.
A su vez cada iteración intentará abrir un nuevo socket para reestablecer la comunicación con el servidor. Cuando vuelva a tener conexión todos los datos almacenados en el fichero 'log_serie.txt' serán parseados y realizará las acciones asociadas a estos.

4.7.2. Actualización remota del firmware

La segunda mejora se centra en la posibilidad de actualizar el firmware de manera remota. Para ello deberemos instalar un cliente FTP (File Transfer Protocol) en la Raspberry Pi y automatizar el proceso para que consulte de manera periódica al servidor central si tiene nuevas versiones del firmware.
FTP es un protocolo de red para transferencia de archivos entre sistemas conectados a una red TCP basado en la arquitectura cliente-servidor. El servicio FTP es ofrecido por la capa de aplicación del modelo de capas TCP/IP.
La siguiente imagen muestra el diagrama de bloques del modelo de intercambio de archivos mediante un servidor FTP a un cliente y viceversa.

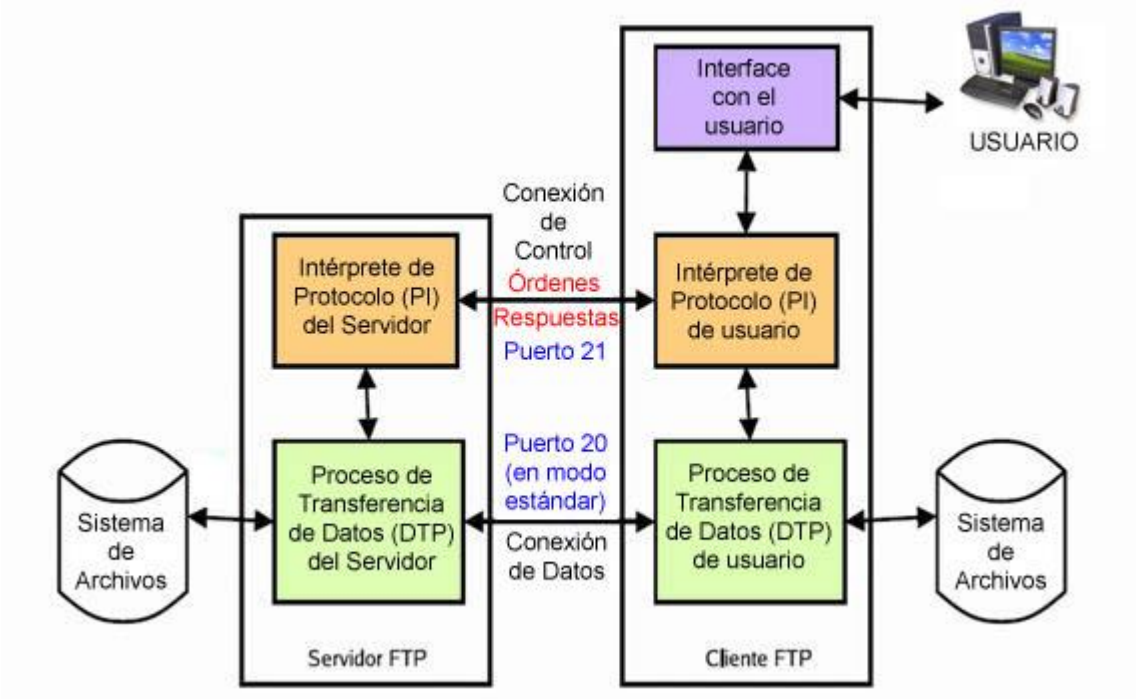


Figura 60. Modelo transferencia FTP

Existen múltiples softwares que realizan la tarea de conectarse con un servidor FTP e intercambiar archivos. En este proyecto, y como se ha explicado en puntos anteriores, se ha utilizado FileZilla para comprobar el buen funcionamiento de esta rutina.
En este caso el servidor FTP se encuentra en el servidor central, mientras que el rol de cliente FTP está incluido en la nueva función de la Raspberry Pi.
Se trata de una librería específica para Python llamada 'pysftp.py' que contiene todas las dependencias para realizar la conexión con el servidor.
El proceso que realizará periódicamente es el que se muestra en el diagrama de bloques:

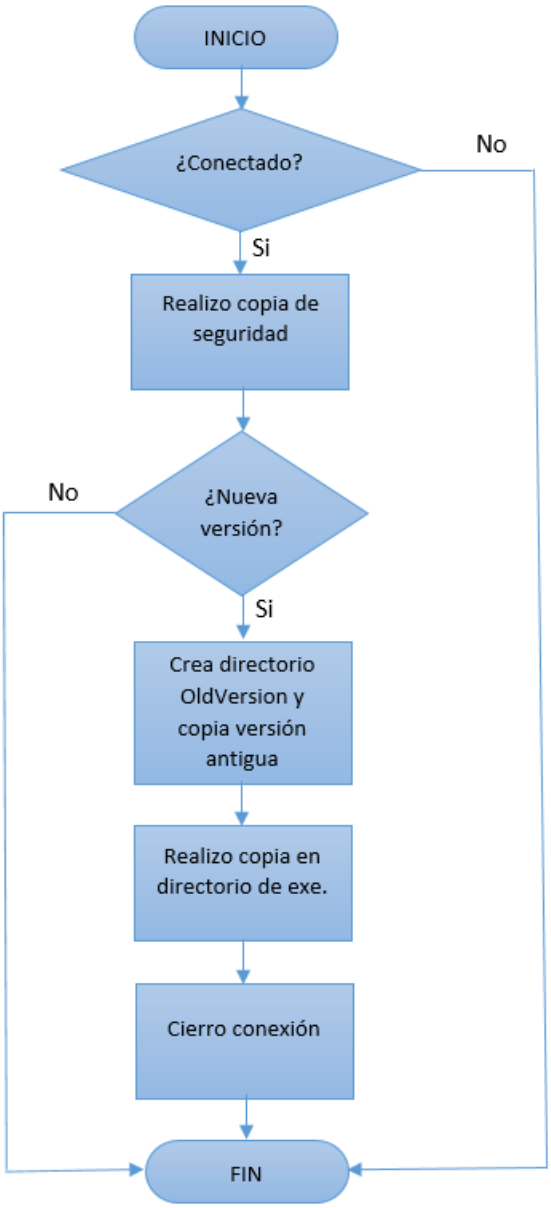


Figura 61. Flujograma actualización remota del firmware

El proceso que sigue el script para la actualización remota y automática es el siguiente:

- Primero realiza una conexión con el servidor central con la dirección, un nombre de usuario, una contraseña y el puerto.
- Después de realizar dicha conexión se realizará una copia de la versión con la que hasta el momento ha estado funcionando el Gateway. Esta copia se realiza por si hubiese algún error en la actualización del firmware tener la posibilidad de recuperar la última versión que funcionaba con seguridad.
- El tercer paso es descargar la versión del firmware nueva. No es necesario que descargue todos los archivos nuevos, sino que simplemente

descargará el módulo que se haya actualizado, por ejemplo, si se añade un nuevo parámetro al parser y al módulo funciones, pero el bucle principal sigue funcionando igual, únicamente deberá descargar estos dos módulos.

- Finalmente cierra la conexión con el servidor.

Se ha propuesto que esta actualización remota y automática, es decir, automáticamente realizará una consulta al servidor. La periodicidad de esta consulta puede ser elegida según la previsión de actualizaciones a corto-medio plazo.
La automatización de la consulta, la ejecución del script que realiza dicha consulta, se ha programado mediante la función que ofrece el sistema operativo llamada cron.
Cron es un proceso que se ejecuta en segundo plano, que está presente en todos los sistemas operativos de tipo Unix y que permite definir la ejecución automatizada de comandos en la terminal que previamente hemos definido con Crontab, pudiendo programar su ejecución con detalle de minuto, hora, día del mes, mes, día de la semana.
Crontab es el programa que permite el acceso a un archivo de texto para su lectura, modificación o sustitución. Este archivo de texto de Crontab permite actualizar el software de manera automática, liberar memoria, automatizar backups, entre otras funciones.
En el terminal de Linux ejecutando el comando >> crontab –l, muestra la lista actual de tareas programadas.
Para modificar es lista ejecutamos el comando >> crontab –e. En la siguiente captura de pantalla vemos nuestra lista de tareas programadas en las que únicamente está el script actualiza.py.

```
  GNU nano 2.2.6               Fichero: /tmp/crontab.lulr0f/crontab

# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

#@reboot python /home/pi/gateway/actualiza.py &
00 00 * * * python /home/pi/gateway/actualiza.py &
```

Figura 62. Captura de pantalla de crontab

Como hemos dicho anteriormente la periodicidad la elegiremos según la previsión que se tenga de nuevas versiones, sin embargo vemos como se programan.
En la siguiente imagen observamos el significado de cada uno de los 5 asteriscos.

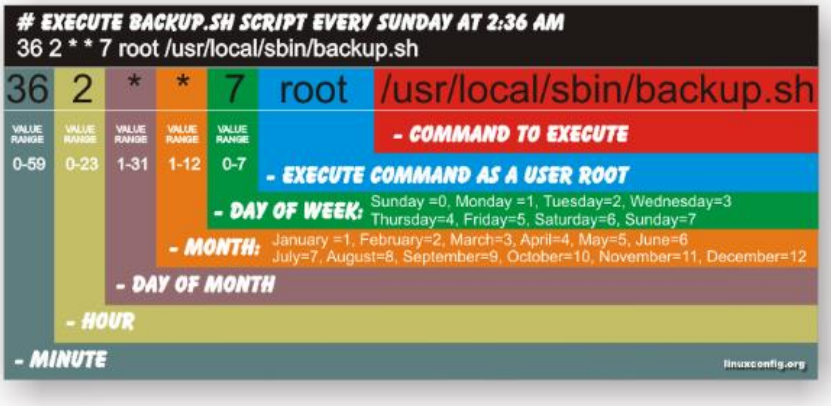


Figura 63. Cuadro ejemplo configuración crontab

De izquierda a derecha, los asteriscos representan:

1. Minutos de 0 a 59.
2. Horas: de 0 a 23.
3. Día del mes: de 1 a 31.
4. Mes: de 1 a 12.
5. Día de la semana: de 0 a 6, siendo 0 el domingo.

Si se deja un asterisco en cada lugar, significa "cada" minuto, hora, día de mes, mes o día de la semana.

También hay palabras reservadas para una periodicidad directa como:

- @reboot: se ejecuta una única vez al inicio.
- @yearly/@annually: ejecuta cada año.
- @monthly: ejecutar una vez al mes.
- @weekly: una vez a la semana.
- @daily: una vez al día.
- @hourly: cada hora.

En este caso realizará una consulta al servidor a las 00:00 h todos los días para comprobar si existe algún archivo nuevo y sustituirlo en el directorio.

4.7.3. Control de funcionamiento

Es posible que al realizar una actualización de firmware, haya habido algún tipo de error en que no haya podido ser detectado al realizar las pruebas pertinentes. Para ello se ha desarrollado un sistema de control de funcionamiento.
El objetivo de este control es asegurar que el bucle principal no se interrumpe en ningún momento, de ser así, el control cargará la última versión funcional y reseteará el dispositivo para aplicar los nuevos cambios.
El control se basa en un servidor TCP al que el bucle principal se conectará en cada iteración y enviará un mensaje: 'ok'. El servidor TCP tendrá un timeout suficientemente largo como para que el bucle realice una iteración, si durante ese periodo de tiempo no se ha recibido el mensaje de 'ok', cargará la versión y reseteará el dispositivo.
Finalmente el diagrama de flujo del bucle principal quedará de la siguiente forma:

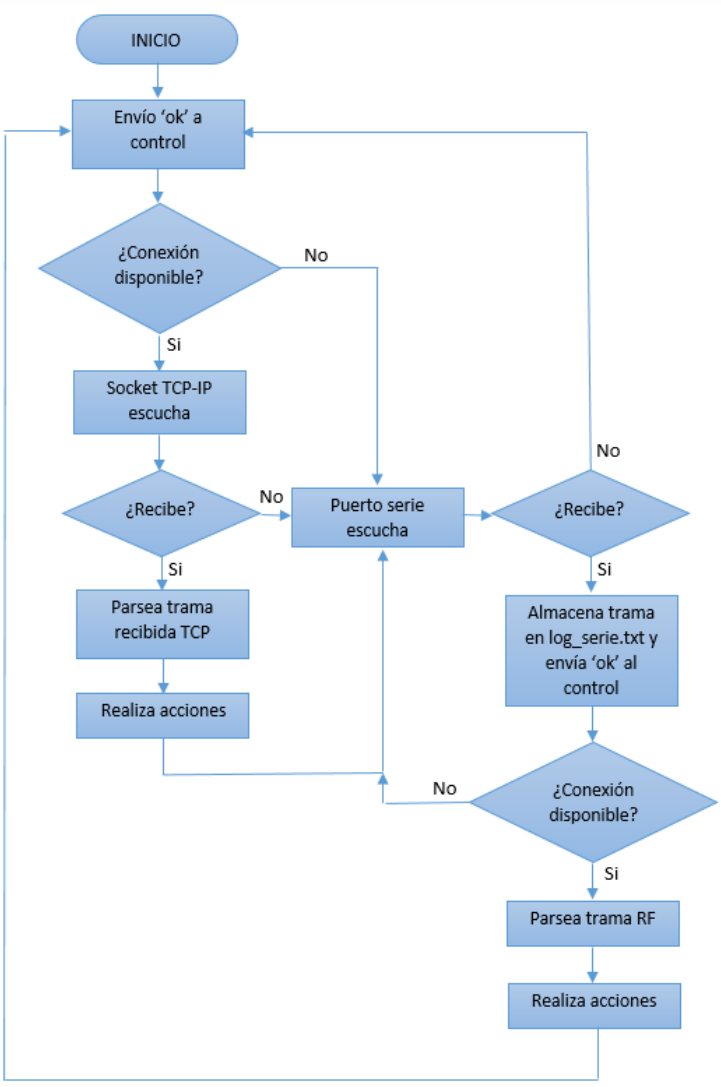


Figura 64. Flujograma final

5. Pruebas

En este apartado se explicará qué pruebas y cómo se han realizado al Gateway para testear su buen funcionamiento. Se ha dividido en dos apartados: el primero trata las pruebas realizadas en cuanto a funcionamiento básico se refiere; en el segundo las pruebas de funcionamiento de nuevas funcionalidades como la recuperación de datos después de una pérdida de conexión o la actualización del firmware de manera remota.

5.1. Pruebas de funcionamiento básico

Las pruebas de este apartado se han realizado, en su mayoría, durante el desarrollo del código para comprobar que el funcionamiento era el correcto y así poder continuar con la programación de las distintas funcionalidades de las que se debía dotar al Gateway.

PRUEBAS DE COMUNICACIÓN TCP

1. Siguiendo la planificación del proyecto empezamos con la comunicación con el servidor mediante TCP-IP. Antes de conectarse con el servidor real se creó un servidor propio y sencillo para testear que las tramas enviadas desde el Gateway eran correctas y el formato era el adecuado. También para comprobar la recepción del Gateway.
2. Comprobado el punto anterior realizamos una conexión al servidor y se envía una trama *alive* la cual el servidor debe responder. Así comprobamos que la comunicación servidor-gateway y Gateway-servidor era correcta.
3. Como última prueba en cuanto a la comunicación con el servidor realizamos desde la web Rfreenet modificaciones en las cuales solo intervengan Gateway y servidor, como cambio de nombre del servidor, cambio del timestamp, etc.

PRUEBAS DE COMUNICACIÓN PUERTO SERIE

1. Previamente a realizar las pruebas con los nodos, debíamos visualizar la comunicación por el puerto de uno de los nodos. Con ese objetivo se instaló Minicom en el pc para visualizar por puerto serie la situación del nodo. Así pues realizamos las pruebas de modificación de parámetros del nodo.
2. El segundo punto consistía en la recepción de las lecturas de los nodos. Se activaron y conectaron varios sensores para comprobar la recepción de esas medidas en el Gateway.

PRUEBAS INTERCONEXIÓN TCP-SERIE

1. Una vez conexionados los dos tipos de comunicación, la primera prueba era la lectura de los parámetros del nodo conectado.
2. Algunos de los puntos críticos son: la adición y la baja de nodos a la red. En cuanto a la adición se debía comprobar que no se repetían ip y que la asignación de direcciones se realizaba con fluidez, también que se añadía un nuevo nodo en el archivo json que almacena la configuración de los nodos de la red. Con respecto a la baja de nodos, el parámetro 'ACTIVO' que existe en el json debía ponerse a 0 para obviar, si las hubiese, tramas referenciadas al nodo que se dio de baja.

5.2. Prueba de nueva funcionalidad

En este aparatado se explicarán las pruebas que se han realizado para comprobar que las nuevas funcionalidades de las que se ha dotado al Gateway han resultado satisfactorias ante posibles eventos.

PRUEBA DE PÉRDIDA DE CONEXIÓN

El contexto por el cual se realiza esta prueba plantea la pérdida de conexión con el servidor durante un tiempo determinado. El objetivo es medir el tiempo total que tarda en enviar las tramas procedentes de los nodos al servidor y con qué throughput realiza estos envíos. Para asegurar el buen funcionamiento del Gateway se han planteado 3 casos:

Caso 1:

Configuramos 2 nodos, UTR1 y UTR2, con 3 sensores activos cada uno de ellos y un tiempo de medida y tiempo de envío de 60 segundos. Vamos a simular una pérdida de conexión de 5 minutos. Con estos datos obtenemos lo siguiente:

$$2nodos * 3sensores * 1minutos\ por\ envío * 5minutos = 30\ medidas$$

Durante el tiempo sin conexión deberíamos haber obtenido 30 medidas de los sensores.
Comprobamos la última medida almacenada en el servidor: 11:45:43.
Pasados los 5 minutos volvemos a proporcionar conexión al Gateway vemos en RfreeNet los 30 datos que debíamos haber obtenido en: 11:51:47.
La conclusión que obtenemos es que estas 30 medidas se envían de forma instantánea.

Las siguientes imágenes muestran las medidas desde la web RfreeNet que corroboran el buen funcionamiento del sistema ante una desconexión repentina de 5 minutos.

Caso 2:

Configuramos los mismos nodos con los mismos sensores activos y los mismos tiempos de medida y de envío, pero esta vez simularemos una pérdida de conexión de 15 minutos. Con los datos anteriores obtenemos la siguiente expresión:

$$2nodos * 3sensores * 1minutos\ por\ envío * 15minutos = 90\ medidas$$

La última medida almacenada en el servidor: 10:26:05.
Proporcionamos conexión pasados los 15 minutos que corresponde a: 10:41.
Obtenemos las medidas de los nodos en el momento: 10:41:46.
En las siguientes imágenes se muestran los resultados en la web RfreeNet para corroborar su buen funcionamiento.

Caso 3

En este caso configuramos dos nodos, pero esta vez 4 sensores por nodo. 60 segundos de tiempos de envío y medida y simularemos una pérdida de conexión de 15 minutos. Con los datos anteriores obtenemos:

$$2nodos * 4sensores * 1minutos\ por\ envío * 15minutos = 120\ medidas$$

Durante el tiempo sin conexión deberíamos haber obtenido 120 medidas de los sensores.
Comprobamos la última medida almacenada en el servidor: 11:46:20.
Pasados los 15 minutos volvemos a proporcionar conexión al Gateway vemos en Rfreenet los 120 datos que debíamos haber obtenido en: 12:01:00.
Las siguientes imágenes corroboran el funcionamiento del Gateway obteniendo unos resultados favorables a esta prueba en los 3 casos expuestos.

PRUEBA DE ACTUALIZACIÓN REMOTA DEL FIRMWARE

Esta prueba se ha realizado manteniendo el Gateway conectado 24 horas en el laboratorio e insertando en el servidor, en el directorio donde realizará la consulta de nuevos archivos de firmware, los mismos scripts que ya tiene programados. Para saber que ha realizado la copia correctamente, a los nuevos scripts colgados en el servidor se les ha añadido una cabecera comentada que muestra la fecha y un texto donde se lee: "Archivo copiado desde el servidor FTP".
Al día siguiente, se comprobaron los scripts del Gateway y correspondían a los que anteriormente se habían colgado en el servidor. A su vez, también se comprobó el nuevo directorio con la hora y fecha de su creación que contenía los scripts antiguos.
La prueba concluyó satisfactoriamente.

5.3. Pruebas de estrés

Este apartado ha consistido someter al equipo a un funcionamiento extremo, es decir, estudiar hasta donde es capaz de llegar el equipo sometido a un funcionamiento, en principio atípico, pero que ayudará a descubrir sus debilidades.
En primer lugar, se ha intentado saturar el gateway en cuanto a comunicación con la red de nodos y la comunicación con el servidor. La prueba ha consistido en enviar tantos comandos como ha sido posible en un breve espacio de tiempo, sin esperar la respuesta del gateway. Así mismo hemos activado todos los sensores de los nodos activos para que estos enviaran una medida de cada uno de ellos cada 30 segundos (un tiempo suficientemente pequeño). El resultado ha sido una breve latencia en las respuestas a las peticiones enviadas de la aplicación web, pero que el gateway ha resuelto satisfactoriamente.
Por último, se ha desconectado el gateway de la red durante 18 horas aproximadamente, es decir, hemos simulado una pérdida de conexión de casi un día. El resultado ha sido que el gateway ha sido capaz de almacenar más de 5500 medidas recibidas de los nodos y en el instante que ha recuperado la conexión las ha enviado de manera inmediata al servidor.
Una instalación real puede estar sometida a distintos factores por los cuales se pierda la conexión con el servidor, los motivos más frecuentes suelen ser:

- Agotamiento de saldo en la tarjeta SIM que proporciona el servicio de conexión.
- Vandalismo
- Zonas remotas donde con cierta periodicidad se pierda cobertura.

- Fallos de alimentación provocados por condiciones climatológicas adversas fuera de lo común.

A continuación se muestra una tabla comparativa donde se pueden ver distintos tiempos de desconexión, número de tramas totales que almacenaría, memoria que ocuparían estas tramas y tiempos de recuperación de datos. Para ello hemos extrapolado con los datos obtenidos en diferentes pruebas realizadas para una instalación media de 15 nodos con 6 sensores por nodo. El tamaño medio de las tramas, ya que no todas ocupan lo mismo, será de 25 bytes.

N. de nodos	2	2	2	15	15
N. sensores/nodos	3	4	3	6	6
Tiempo de envío	1 min	1 min	1 min	1 min	1 min
Tiempo de desconex.	5 min	15 min	18 h	24 h	72 h
Tramas almacenadas	30	120	6480	129600	388800
Memoria ocupada	750 B	3000 B	0.162 Mb	3.24 Mb	9.72 Mb
Tiempo de recup. *	Inst.	Inst.	0.2 s	0.2 s	0.5 s

Tabla 17. Comparativa de tiempos y memoria

* Los tiempos de recuperación son orientativos ya que estos dependerán de la cobertura en el lugar.

6. Conclusiones

Llegados a este punto estamos en condiciones de valorar el trabajo realizado y comprobar que cumple con las expectativas iniciales.

En primer lugar, el Gateway desarrollado está en condiciones para sustituir la versión antigua de éste. Tras realizar las pruebas de funcionamiento normal, se ha comprobado que la comunicación con el servidor y la red de nodos funcionan correctamente. Se ha realizado la simulando de una instalación en el laboratorio con dos nodos y con el modem USB en el dispositivo. No obstante, no se ha realizado una prueba en condiciones reales, es decir, en un campo, paraje o extensión al aire libre aunque estas condiciones no afectarían al funcionamiento del Gateway, por lo que se ha desestimado realizar ese ensayo. La experiencia del instalador, ya que se han realizado múltiples instalaciones con la versión antigua del Gateway, resolverá sin ninguna duda todos los problemas con respecto a la instalación.

En segundo lugar, se han abordado con éxito las nuevas condiciones para solucionar fallos o ampliar su funcionalidad. Con respecto a las posibles pérdidas de conexión se han simulado esas condiciones adversas en las que podría verse envuelto el Gateway y se han realizado ensayos que corroboran el buen funcionamiento ante las condiciones citadas obteniendo unos resultados, en cuanto a tasa de datos enviados, excelente. Por otra parte, es posible actualizar el firmware de manera remota. Esta nueva funcionalidad ofrecerá la posibilidad de realizar mejoras de manera rápida y cómoda pudiéndose testear en una instalación real sin necesidad de desplazarse hasta ella.

7. Referencias

[1] LabFerrer. Gestión del agua de riego. [En línea].
http://www.lab-ferrer.com/gestion-del-agua-de-riego.html.

[2] Microcom. Hermes LC2. [En línea].
http://www.microcom.es/hermes-lc2.php

[3] Lacroix-Sofrel. Gama Box. [En línea].
http://www.lacroix-sofrel.es/content/gama-box

[4] Wikipedia. Comunicación inalámbrica. [En línea].
https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_inal%C3%A1mbrica

[5] Json. Introducción a JSON. [En línea].
http://www.json.org/json-es.html

[6] Python. Documentation. [En línea].
https://www.python.org/doc/

[7] Rasberry Pi. Get to know the raspberry pi foundation. [En línea].
https://www.raspberrypi.org/blog/get-know-raspberry-pi-foundation/

[8] Modem GSM/GPRS. Huawei Device. [En líne].
http://huaweidevice.com.ar/mas-productos/item/74-huawei-e173/74-huawei-e173

[9] Clipnode. Clipnode productos. [En línea].
http://www.clipnode.com/productos/

[10] Manual Minicom. Tutorial básico Minicom. [En línea].
http://es.slideshare.net/Metaconta/manual-bsico-minicom-presentation

[11] Librosweb. Libro Python. [En línea].
http://librosweb.es/libro/python/capitulo_10/modulos_de_sistema.html#indice

[12] Raúl González Duque. Python para todos. [Versión más reciente].
http://mundogeek.net/tutorial-python/

8. Anexos

8.1. Descripción y formato de tramas de comunicación con servidor

Tramas gateway inicia comunicación

Trama de alive:

Envía Gateway:
Nombre función: **alive()**

0x02	'V'	ID_{SUS}	'I'	IP	'P'	Port	'T'	TimeStamp (14 bytes)	'D'	"000"	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* ["aaa.bbb.ccc.ddd"] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion ['0000'-'9999'] (4carác. ASCII DECIMALES). Para GW-LP este parámetro valdrá siempre "1234"

TimeStamp = Tiempo actual del Gateway, usando el formato: "HHMMSSDDMMAAAA" (14carác. ASCII)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Server:

0x02	'h'	ID_{SUS}	TimeStamp (14 bytes)	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

TimeStamp = Tiempo actual del Gateway, usando el formato: "HHMMSSDDMMAAAA" (14carác. ASCII)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Evento de arranque/reset gateway:

Envía Gateway:
Nombre función: **ResetArranque()**

0x02	'V'	ID_{SUS}	'I'	IP	'P'	Port	'T'	TimeStamp	'D'	"002"	"000"	versHWSW	CS	**0x05**

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* [“aaa.bbb.ccc.ddd”] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion [‘0000’-‘9999’] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre “1234”

TimeStamp = Tiempo actual del Gateway, usando el formato:
“HHMMSSDDMMAAAA” (14carác. ASCII)
versHWSW: versión del hardware y software (5carác. ASCII HEXADECIMALES)
Ejemplo: (versión HW: 10 / versión SW:1001)
VersionHWSW = ”18A89” (versión 101001)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Server:

- ACK = 0x06
- NACK = 0x0F

Evento de batería gateway:

Envía Gateway:
Nombre función: **bateria()**

0x02	‘V’	IDsus	‘I’,	IP	‘P’,	Port	‘T’,	TimeStamp	‘D’	“805”	“000”	Valor	CS	**0x05**

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* [“aaa.bbb.ccc.ddd”] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion [‘0000’-‘9999’] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre “1234”
Valor = Nivel de batería en *mV* (5carác. ASCII HEXADECIMALES)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde server:

- ACK = 0x06
- NACK = 0x0F

Evento de sensores gateway:

Envía Gateway:
Nombre función: **sensores()**

0x02	'V'	ID_{SUS}	'I'	IP	'P'	Port	'T'	TimeStamp	'D'	N.Evento	"000"	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* ["aaa.bbb.ccc.ddd"] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion ['0000'-'9999'] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre "1234"

N.Evento = Numero de evento (3carác. ASCII DECIMALES)

- ["600-622"] = Medida de sensor analógico y digital
- ["625-647"] = Umbral máximo superado sensor analógico y digital
- ["650-672"] = Umbral mínimo superado sensor analógico y digital

Los sensores analógicos/digitales se ordenan de esta forma:

Sensor AN0 = Sensor 0 Sensor DIG0 = Sensor 7

Sensor AN6 = Sensor 6 SensorDIG16 = Sensor 22

Valor = Valor medido por el sensor (5carác. ASCII HEXADECIMALES)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde server:

- ACK = 0x06
- NACK = 0x0F

Lectura de estructura de red:

Envía server:

0x02	'T'	C	'T'	ID_{SUS}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:
Nombre función: **estructura_red()**

0x02	‘T’,	C	‘T’,	ID_{SUS}	‘D’,	ID_{FR}	NUM_{FR}	NUM_{ND}	IP_1	MAC_1	..	IP_x	MAC_x	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

IDfr = Número de la trama [‘1’-‘9’] (1carác. ASCII DECIMALES)

NUMfr = Número total de tramas que se enviarán [‘1’-‘9’] (1carác. ASCII DECIMAL)

NUMnd = Número total de nodos que se envían en esta trama [‘01’-‘99’] (2carác. ASCII DECIMALES)

IPi = IP del nodo i [‘000’-‘999’] (3carác. ASCII DECIMALES)
MACi = MAC del nodo i [‘xxxxxxxxxxxx’] (12carác. ASCII HEXADECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (2carác. ASCII HEXADECIMALES)

Lectura de parámetros gateway:

Envía server:

0x02	‘T’	C	‘L’,	ID_{SUS}	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:
Nombre función: **lectura_parametros()**

0x02	‘T’	C	‘L’	ID_{SUS}	$Param_1$	$Valor_1$	‘;’	...	$Param_N$	$Valor_N$	‘;’	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)
$Param_i$ = Nombre del parámetro de configuración detallado en el punto anterior (3carác. ASCII)
$Valor_i$ = Valor del parámetro en cuestión (Ncarác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (2carác. ASCII HEXADECIMALES)

Lectura de alias gateway:

Envía server:

0x02	‘T’	C	‘P’,	ID_{SUS}	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:

Nombre función: **lectura_alias(nombre_GW)**

0x02	‘T’	C	‘P’	ID_{SUS}	“Alias” (máximo 20 caracteres)	‘;’	CS	0x05

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)
Alias = Alias del Gateway (20carác. ASCII como máximo)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (2carác. ASCII HEXADECIMALES)

Modificación de parámetros gateway:

Envía server:

0x02	‘T’	C	‘M’,	ID_{SUS}	“IDT”	R^1	‘;’	Prm_1	Vlr_1	‘;’	…	Prm_N	Vlr_N	‘;’	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

R^1 = Valor aleatorio de 4 dígitos, identificador de trama [‘0000’-‘9999’] (4carác. ASCII DECIMALES)
Prm_i = Nombre del parámetro de configuración detallado en el punto anterior (3carác. ASCII)
Vlr_i = Valor del parámetro en cuestión (Ncarác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:

Nombre función: **mod_parametros(parametros_valores,R_aleatorio)**

0x02	‘T’	C	‘M’,	ID_{SUS}	“IDT”	R^1	‘;’	Prm_1	Vlr_1	‘;’	…	Prm_N	Vlr_N	‘;’	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

R^1 = Valor aleatorio de 4 dígitos, identificador de trama [‘0000’-‘9999’] (4carác. ASCII DECIMALES)
Prm_i = Nombre del parámetro de configuración detallado en el punto anterior (3carác. ASCII)
Vlr_i = Valor del parámetro en cuestión (Ncarác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Modificación de alias gateway:

Envía server:

0x02	‘T’	C	‘N’	ID_{SUS}	‘G’	“Alias” (máx 20 caracteres)	‘;’	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)
Alias = Alias del Gateway (20carác. ASCII como máximo)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:
Nombre función: **modificacion_alias(nombre_mod)**

0x02	‘T’	C	‘N’	ID_{SUS}	‘G’	“Alias” (máx 20 caracteres)	‘;’	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)
Alias = Alias del Gateway (20carác. ASCII como máximo)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Forzado de datos gateway:

Envía server:

0x02	‘T’	C	‘A’	ID_{SUS}	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:

El gateway forzará, en ese mismo instante de tiempo, la medida de todos los sensores habilitados. E irá generando los eventos de las medidas correspondientes.

Forzado de estadísticas gateway:

Envía server:

0x02	'T'	C	'E,'	ID_{SUS}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:
Nombre función: **bateria()**

0x02	'V,'	ID_{SUS}	'I,'	IP	'P,'	Port	'T,'	TimeStamp	'D'	"805"	"000"	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* ["aaa.bbb.ccc.ddd"] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion ['0000'-'9999'] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre "1234"
Valor = Nivel de batería en *mV* (5carác. ASCII HEXADECIMALES)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Cambio estado salidas de gateway:

Envía server:

0x02	'T'	C	'O,'	ID_{SUS}	Salida	Valor	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

Salida = Número de salida sobre la que actuar ['1'-'5'] (1carác. ASCII DECIMAL)

Los salida relé/digitales disponibles en el Gateway se ordenan de esta forma:

Relé R1 = Salida 1 Salida DIG1 = Salida 4
Relé R2 = Salida 2 Salida DIG2 = Salida 5
Relé R3 = Salida 3

Valor = Estado deseado de la salida [‘0’-‘1’] (1carác. ASCII DECIMAL)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:
Nombre función: **cambio_estados(salida,valor)**
realmente es echo de la trama enviada por el servidor

0x02	‘T’	C	‘O’,	ID_{SUS}	Salida	Valor	CS	**0x05**

Echo de la trama recibida del Server

Reset del gateway:

Envía server:

0x02	‘T’	C	‘R’,	ID_{SUS}	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Gateway:
Nombre función: **Reset()**

0x02	‘T’	C	‘R’,	ID_{SUS}	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Tramas de comunicación con servidor de la red RfreeNet

Tramas de eventos de utr (medidas, umbrales y gradientes):

Envía Gateway:

0x02	'V'	ID_{SUS}	'I'	IP	'P'	Port	'T'	TimeStamp	'D'	N.Evento	IP_{UTR}	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* ["aaa.bbb.ccc.ddd"] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion ['0000'-'9999'] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre "1234"

N.Evento = Numero de evento (3carác. ASCII DECIMALES)

- ["500-505"] = Medida de sensor
- ["506-511"] = Umbral máximo de sensor superado
- ["512-517"] = Gradiente máximo de sensor superado

Los sensores se ordenan de esta forma:
Sensor AN0 = Sensor 0
Sensor AN6 = Sensor 6

IP_{UTR} = IP de la UTR. (3carác. ASCII DECIMALES)
Valor = Valor medido por el sensor (5carác. ASCII HEXADECIMALES)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Server:

- ACK = 0x06
- NACK = 0x0F

Tramas de evento de batería:

Envía gateway:

0x02	'V'	ID_{SUS}	'I'	IP	'P'	Port	'T'	TimeStamp	'D'	"801"	IP_{UTR}	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* ["aaa.bbb.ccc.ddd"] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion [‘0000’-‘9999’] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre “1234”

IP_{UTR} = IP de la UTR. (3carác. ASCII DECIMALES)
Valor = Valor medido de batería (5carác. ASCII HEXADECIMALES)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Server:

- ACK = 0x06
- NACK = 0x0F

Tramas de evento de cobertura utr:

Envía gateway:

0x02	‘V’	ID_{SUS}	‘I’	IP	‘P’	Port	‘T’	TimeStamp	‘D’	“800”	IP_{UTR}	Valor	CS	**0x05**

IDsus = Gateway ID [‘0000’-‘9999’] (4carác. ASCII DECIMALES)

IP= Dirección IP del Gateway en este *alive* [“aaa.bbb.ccc.ddd”] (15carác. ASCII como máximo)

Port = Puerto UTP de la conexion [‘0000’-‘9999’] (4carác. ASCII DECIMALES) . Para GW-LP este parámetro valdrá siempre “1234”

IP_{UTR} = IP de la UTR. (3carác. ASCII DECIMALES)
Valor = Valor medido de cobertura (5carác. ASCII HEXADECIMALES)
CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Responde Server:

- ACK = 0x06
- NACK = 0x0F

Tramas de lectura de parámetros utr:

Server → Gateway:

0x02	‘T’	C	‘L’	ID_{SUS}	‘N’	IP_{UTR}	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’]

IPutr = Nodo ID [‘000’-‘999’]

CS = Checksum XOR toda la trama (1byte)

Gateway → RF:
lectura_parametrosRF(nodo):

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**	**CS**	**STOP**
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'L'	- - -	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

RF → Gateway:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**					**CS**	**STOP**
0x02	L	IPutr[0]	IPutr[1]	IPutr[2]	'L'	$Param_1$	Val_1	. . .	$Param_n$	Val_n	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

$Param_i$ = Nombre parámetro (3 dig. ASCII)

Val_i = Valor parámetro (tamaño variable: 1-4bytes en ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Gateway → Server:
lectura_utr(nodo,datos):

0x02	'T'	C	'L'	ID_{SUS}	'D'	IP_{UTR}	$Param_1$	Val_1	';'	…	';'	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

$Param_i$ = Nombre parámetro (3 dig. ASCII)

Val_i = Valor parámetro (en ASCII, con los dígitos necesarios)

CS = Checksum XOR toda la trama (1byte)

Tramas de lectura de alias utr:

Servidor → Gateway:

0x02	'T'	C	'P'	ID_{SUS}	'N'	IP_{UTR}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

CS = Checksum XOR toda la trama (1byte)

Gateway → RF:
lectura_aliasRF(nodo):

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**	**CS**	**STOP**
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'L'	- - -	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

RF → Gateway:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**					**CS**	**STOP**
0x02	L	IPutr[0]	IPutr[1]	IPutr[2]	'L'	$Param_1$	Val_2	. . .	$Param_n$	Val_n	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

$Param_i$ = Nombre parámetro (3 dig. ASCII)

Val_i = Valor parámetro (tamaño variable: 1-5bytes en ASCII o en BINARIO!!!)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

LOS ALIAS DEL NODO SE ALMACENAN EN GATEWAY, (A EXCEPCION DE LOS SENSORES HABILITADOS ‘SEN’ Y DEL TIPO DE SENSOR ‘TSn’) , CON LO QUE REALIZARIÁ UNA LECTURA DE PARÁMETROS DEL NODO PARA RECIBIR LOS PARÁMETROS DE ‘SEN’ Y ‘TSn’ Y EL RESTO LO RECUPERARIA DE LA MEMORIA DEL GATEWAY

Gateway → Server:

0x02	‘T’	C	‘P’	ID_{SUS}	‘D’	IP_{UTR}	AN	;	TNO	SEN	TS1	AS1	;	TSn	ASn	;	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’]

IPutr = Nodo ID [‘000’-‘999’]

AN = Parámetro “AN” del nodo (Nombre Nodo) (max. 20carc. ASCII)

TNO = Parámetro “TNO” del nodo (Tipo de Nodo) (1dig. ASCII)

SEN = Parámetro “SEN” del nodo (Sensores habilitados) (3dig. ASCII)

TSn = Parámetro “TSn” del nodo (Tipo del sensor n) (2carác. ASCII HEXADECIMALES)

ASn = Parámetro “ASn” del nodo, n = [1, 6] (Nombre del Sensor n) (max. 20carc. ASCII)

CS = Checksum XOR toda la trama (1byte)

Modificación de parámetros utr:

Server → Gateway:

0x02	‘T’	C	‘M’	ID_{SUS}	“IDT”	R_1	‘N’	IP_{UTR}	$Param_1$	$Valor_1$	‘;’	…	‘;’	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’]

R^1 = Valor aleatorio de 4 dígitos [‘0000’-‘9999’] (4dig. ASCII)

AL RESPONDER A ESTA TRAMA DEBO UTILIZAR EL MISMO R1 QUE RECIBO AQUÍ!

IPutr = Nodo ID [‘000’-‘999’]

$Param_i$ = Nombre parámetro (3 dig. ASCII)

Val_i = Valor parámetro (en ASCII, con los dígitos necesarios)

CS = Checksum XOR toda la trama (1byte)

Gateway → RF:
mod_paramRF(nodo,datos,R_aleatorio):

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**						**CS**	**STOP**
0x02	L	IPutr[0]	IPutr[1]	IPutr[2]	'C'	R^1	$Param_1$	Val_1	...	$Param_n$	Val_n	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

R^1 = Valor aleatorio de 4 dígitos ['0000'-'9999'] (2bytes en BIN). El mismo que recibo en "Modem_RX"

$Param_i$ = Nombre parámetro (3 dig. ASCII)

Val_i = Valor parámetro (tamaño variable: 1-4bytes en BIN)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

RF → Gateway:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**		**CS**	**STOP**
0x02	L	IPutr[0]	IPutr[1]	IPutr[2]	'C'	R^1	NumParams	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

R^1 = Valor aleatorio de 4 dígitos ['0000'-'9999'] (2bytes en BIN). El mismo que le envio en "AntenaRF_TX"

NumParams = número de parámetros modificados (1byte en BIN)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Gateway → Server:
mod_param-utr(nodo,R_aleatorio)

0x02	'T'	C	'M'	ID_{SUS}	"IDT"	R^1	'N'	IP_{UTR}	';'	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID [‘0000’-‘9999’]

R[1] = Valor aleatorio de 4 dígitos [‘0000’-‘9999’] (4dig. ASCII)

EL MISMO VALOR QUE LE RECIBIDO EN LA TRAMA Modem_RX

IPutr = Nodo ID [‘000’-‘999’]

CS = Checksum XOR toda la trama (1byte)

Tramas de modificación de alias de utr:

Server → Gateway:

0x02	‘T’	C	‘N’	ID_{SUS}	‘D’	IP_{UTR}	AN	;	TNO	SEN	TS1	AS1	;	TSn	ASn	;	CS	**0x05**

C = Consola o Terminal [‘0’-‘9’]. Para GW-LP este parámetro valdrá siempre ‘X’

IDsus = Gateway ID [‘0000’-‘9999’]

IPutr = Nodo ID [‘000’-‘999’]

AN = Parámetro “AN” del nodo (Nombre Nodo) (max. 20carc. ASCII)

TNO = Parámetro “TNO” del nodo (Tipo de Nodo) (1dig. ASCII)

SEN = Parámetro “SEN” del nodo (Sensores habilitados) (3dig. ASCII)

TSn = Parámetro “TSn” del nodo, n =[1,6] (Tipo del sensor n) (3carac. ASCII)

ASn = Parámetro “ASn” del nodo , n =[1,6] (Nombre del sensor) (max. 20carc. ASCII)

CS = Checksum XOR toda la trama (1byte)

Gateway → RF:
mod_aliasRF(nodo,datos)

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**					**CS**	**STOP**
0x02	L	IPutr[0]	IPutr[1]	IPutr[2]	‘C’	$Param_1$	Val_2	. . .	$Param_n$	Val_n	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

$Param_i$ = Nombre parámetro de nodo a modificar (3 dig. ASCII)

Val_i = Valor parámetro (tamaño variable: 1-5bytes en ASCII o en BINARIO!!!)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

RF → Gateway:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**					**CS**	**STOP**
0x02	L	IPutr[0]	IPutr[1]	IPutr[2]	'C'	$Param_1$	Val_2	. . .	$Param_n$	Val_n	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

NumParam = Número de parámetros modificados (en ASCII o en binario ¿¿)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

LOS ALIAS DEL NODO SE ALMACENAN EN GATEWAY, (A EXCEPCION DE LOS SENSORES HABILITADOS 'SEN' Y DEL TIPO DE SENSOR 'TSn'), CON LO QUE REALIZARIÁ UNA ESCRITURA DE PARÁMETROS DEL NODO PARA ESCRIBIR LOS PARÁMETROS'SEN' Y 'TSn' (SI ÉSTOS HAN CAMBIADO) Y EL RESTO LO ESCRIBIRÁ EN LA MEMORIA DEL GATEWAY

Servidor → Gateway:

0x02	'T'	C	'N'	ID_{SUS}	'D'	IP_{UTR}	AN	;	TNO	SEN	TS1	AS1	;	TSn	ASn	;	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

AN = Parámetro "AN" del nodo (Nombre Nodo) (max. 20carc. ASCII)

TNO = Parámetro "TNO" del nodo (Tipo de Nodo) (1dig. ASCII)

SEN = Parámetro "SEN" del nodo (Sensores habilitados) (3dig. ASCII)

TSn = Parámetro "TSn" del nodo (Tipo del sensor n) (2carác. ASCII HEXADECIMALES)

ASn = Parámetro "ASn" del nodo, n = [1, 6] (Nombre del Sensor n) (max. 20carc. ASCII)

CS = Checksum XOR toda la trama (1byte)

Tramas de forzado de datos de sensores de utr:

Servidor → Gateway:

0x02	'T'	C	'A'	ID_{SUS}	'N'	IP_{UTR}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

CS = Checksum XOR toda la trama (1byte)

Gateway → RF:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**	**CS**	**STOP**
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'M'	- - -	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

AntenaRF RX:

No hay respuesta directa: el nodo generará tramas de datos (tramas 'D')

Tramas de forzado de estadísticas de utr:

Forzado de datos de batería y cobertura de utr/Servidor → Gateway:

0x02	'T'	C	'E'	ID_{SUS}	'N'	IP_{UTR}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

CS = Checksum XOR toda la trama (1byte)

Estadística y batería/ Gateway → RF :

START	L	IP2	IP1	IP0	TIPO	DATOS	CS	STOP
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'E'	- - -	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

AntenaRF RX:

No hay respuesta directa: el nodo generará tramas de estadísticas y batería (tramas 'W')

Tramas de reset de utr:

Reset de utr/server → Gateway:

0x02	'T'	C	'R'	ID_{SUS}	'N'	IP_{UTR}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

CS = Checksum XOR toda la trama (1byte)

Reset/ gateway → RF:

START	L	IP2	IP1	IP0	TIPO	DATOS	CS	STOP
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'X'	- - -	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Reset/RF → gateway:

START	L	IP2	IP1	IP0	TIPO	DATOS	CS	STOP
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'X'	- - -	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Reset de utr/gateway → Server:

0x02	'T'	C	'R'	ID_{SUS}	'N'	IP_{UTR}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

CS = Checksum XOR toda la trama (1byte)

Tramas de eliminación utr:

Eliminar utr del sistema/Server → Gateway:

0x02	'T'	C	'K'	ID_{SUS}	'N'	IP_{UTR}	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IDsus = Gateway ID ['0000'-'9999']

IPutr = Nodo ID ['000'-'999']

CS = Checksum XOR toda la trama (1byte)

AntenaRF_TX/ AntenaRF_RX:

NO GENERA TRAMA PARA LA ANTENA: ESTA ACCIÓN ES EXCLUSIVA PARA EL GATEWAY.

Borrar UTR del vector de estructuras de nodos. A partir del borrado, si llegan datos del nodo eliminado se descartan

Modem_TX:

NO HAY RESPUESTA

Tramas de petición de ip de utr:

Petición de ip/RF → gateway:

START	L	IP2	IP1	IP0	TIPO	DATOS	CS	STOP
0x02	0x0A	IPutr[0]	IPutr[1]	IPutr[2]	'Y'	MAC (6bytes)	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

IPutr[0], IPutr[1], IPutr[2] = IP actual de UTR (3dig. ASCII)

MAC = MAC de la UTR (6bytes)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Petición de ip de utr/ gateway → server:

0x02	'Y'	ID_{SUS}	'D'	'0'	IP_{UTR} OLD	MAC (12 bytes ASCII)	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999']

IPutr OLD = Nodo ID antiguo ['000'-'999']

MAC = MAC de la UTR (12dig. ASCII, en hexadecimal)

CS = Checksum XOR toda la trama (1byte)

Respuesta de petición de ip de utr/ server → Gateway:

0x02	'i'	ID_{SUS}	'D'	IP_{UTR} OLD	IP_{UTR} NEW	MAC (12 bytes ASCII)	CS	0x05

IDsus = Gateway ID ['0000'-'9999']

IPutr OLD = Nodo ID antiguo ['000'-'999']

IPutr NEW = Nodo ID nuevo ['000'-'999']

MAC = MAC de la UTR (12dig. ASCII, en hexadecimal)

CS = Checksum XOR toda la trama (1byte)

Respuesta de petición de ip de utr/ Gateway → RF:

START	L	IP2	IP1	IP0	TIPO	DATOS	CS	STOP
0x02	0x0C	IPutr[0]	IPutr[1]	IPutr[2]	'i'	IPH(1Byte) + IPL (1Byte) + MAC (6Bytes)	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

IPutr[0], IPutr[1], IPutr[2] = IP antigua de UTR (3dig. ASCII)

IPH + IPL + MAC = ByteH(IP_new)+ByteL(IP_new)+MAC(6bytes)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Tramas de actuación de salidas de utr:

Actuación salidas utr/ Server → gateway:

0x02	'T'	C	'O'	ID_{SUS}	'N'	IP_{UTR}	Salida	Valor	CS	**0x05**

C = Consola o Terminal ['0'-'9']. Para GW-LP este parámetro valdrá siempre 'X'

IPutr = IP del nodo ['000'-'999'] (3carác. ASCII DECIMALES)

IDsus = Gateway ID ['0000'-'9999'] (4carác. ASCII DECIMALES)

Salida = Número de salida sobre la que actuar ['1'-'6'] (1carác. ASCII DECIMAL)

Valor = Estado deseado de la salida ['0'-'1'] (1carác. ASCII DECIMAL)

CS = Checksum XOR toda la trama excepto byte de START [0x02] (1byte)

Actuación salidas/ Gateway → RF:

START	L	IP2	IP1	IP0	TIPO	DATOS		CS	STOP
0x02	0x04	IPutr[0]	IPutr[1]	IPutr[2]	'O'	Salida	Estado	CS	**0x05**

L = Longitud en bytes: IP+TIPO+DATOS (1byte)

IP2,IP1,IP0 = Nodo ID (3 dig. ASCII)

Salida = Número de salida sobre la que actuar [1-6] (1byte)

Estado = Estado deseado de la salida [0-1] (1byte)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

AntenaRF_RX:

EL NODO NO ENVIA RESPUESTA A ACTUACIÓN DE SALIDA

Actuación salidas utr/ Gateway → server:

0x02	'T'	C	'O,'	ID_{SUS}	'N'	IP_{UTR}	Salida	Valor	CS	**0x05**

Echo de la trama recibida del Server

Tramas de eventos de datos utr:

Datos/ RF → gateway:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**		**CS**	**STOP**
0x02	0x07	IPutr[0]	IPutr[1]	IPutr[2]	'D'	Sensor	Valor	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

IPutr[0], IPutr[1], IPutr[2] = IP actual de UTR (3dig. ASCII)

Sensor = Sensor medido (1byte, cod. jhonson) [1,2,4,8,16,32 : sensor0,sensor1,…,sensor5]

Valor = valor medido (2bytes)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Medidas, umbrales y gradientes de sensores de utr / Gateway → server:

0x02	'V'	ID_{SUS}	'I,'	IP	'P,'	Port	'T,'	TimeStamp	'D'	N.Evento	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999']

IP = IP del servidor ['111.111.111.1111']

Port = Puerto del servidor ['1234' : en el programa actual, siempre se envía este número de puerto]

TimeStamp = estructura de tiempo de sincronismo (14dig. ASCII)

N. Evento = Medida de sensor ['500-505']

Valor = IP del nodo (3dig. ASCII) + Valor medido por el sensor (5dig. ASCII, en hexadecimal)

CS = Checksum XOR toda la trama (1byte)

Modem_RX:

- ACK = 0x06
- NACK = 0x0F

Tramas de evento de alarmas umbral/gradiente de utr:

Estadísticay batería / RF → Gateway:

START	**L**	**IP2**	**IP1**	**IP0**	**TIPO**	**DATOS**				**CS**	**STOP**
0x02	0x0A	IPutr[0]	IPutr[1]	IPutr[2]	'W'	Código	Sensor	Valor	FILL	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

IPutr[0], IPutr[1], IPutr[2] = IP actual de UTR (3dig. ASCII)

Código = medida de gradiente ó de umbral (1byte)

- 0x00 = Umbral
- 0x01 = Gradiente

Sensor = Sensor medido (1byte) [0x00 – 0x05]

Valor = valor de gradiente/umbral medido (2bytes)

FILL= bytes de relleno (2bytes)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Medidas, umbrales y gradientes de sensores de utr / Gateway → server:

0x02	'V'	IDsUS	'I'	IP	'P'	Port	'T'	TimeStamp	'D'	N.Evento	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999']

IP = IP del servidor ['111.111.111.1111']

Port = Puerto del servidor ['1234' : en el programa actual, siempre se envía este número de puerto]

TimeStamp = estructura de tiempo de sincronismo (14dig. ASCII)

N. Evento =

- ["506-511"] = Umbral de sensor superado
- ["512-517"] = Gradiente de sensor superado

Valor = IP del nodo (3dig. ASCII) + Valor de gradiente/umbral medido por el sensor

(5dig. ASCII, en hexadecimal)

CS = Checksum XOR toda la trama (1byte)

Modem_RX:

- ACK = 0x06
- NACK = 0x0F

Tramas de evento de estadísticas de utr:

Estadística y batería / RF → Gateway:

START	L	IP2	IP1	IP0	TIPO	DATOS				CS	STOP
0x02	0x0A	IPutr[0]	IPutr[1]	IPutr[2]	'W'	0x02	FILL	Cobert.	Bat.	CS	**0x05**

L = Longitud de bytes de IP+TIPO+DATOS (1byte)

IPutr[0], IPutr[1], IPutr[2] = IP actual de UTR (3dig. ASCII)

FILL= bytes de relleno (2bytes)

Cobert. = Valor de cobertura (1byte)

Bat. = valor de batería (2bytes)

CS = Checksum: suma IP+TIPO+DATOS (1byte)

Evento de batería de utr / gateway → server:

0x02	'V'	IDsus	'I'	IP	'P'	Port	'T'	TimeStamp	'D'	"801"	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999']

IP = IP del servidor ['111.111.111.1111']

Port = Puerto del servidor ['1234']

TimeStamp = estructura de tiempo de sincronismo (14dig. ASCII)

Valor = IP del nodo (3dig. ASCII) + Valor batería medido (5dig. ASCII, en hexadecimal)

Modem_RX(1):

- ACK = 0x06
- NACK = 0x0F

Evento de cobertura de utr / Gateway → server:

0x02	'V'	ID_{SUS}	'I',	IP	'P',	Port	'T',	TimeStamp	'D'	"800"	Valor	CS	**0x05**

IDsus = Gateway ID ['0000'-'9999']

IP = IP del servidor ['111.111.111.1111']

Port = Puerto del servidor ['1234']

TimeStamp = estructura de tiempo de sincronismo (14dig. ASCII)

Valor = IP del nodo (3dig. ASCII) + Valor cobertura medido (5dig. ASCII, en hexadecimal)

Modem_RX(2):

- ACK = 0x06
- NACK = 0x0F

8.2. Código

8.2.1. Funciones.py

```
#Script que contiene las funciones para Gateway Low Power
import json
import RPi.GPIO as GPIO
import time
import datetime
import socket
global check
global check1
global check2
global nombre_GW
global identGW1
global identGW2
global identGW3
global identGW4
global timestampGW
global checksum_server1
global checksum_server2
global valor_bat
global nevento
global checkRF
global check1RF
global check2RF
global linea
global gate
#definimos start y stop
start ='\2'
stop = '\5'
#identificador GW
identGW1 = '0'
identGW2 = '0'
identGW3 = '0'
identGW4 = '3'
nombre_GW = 'TEST'
#Direccion, puerto y buffer
TCP_IP = '158.42.105.71'    #'www.test.balmart.es'
TCP_PORT = 6011
PORT = '6011'
BUFFER_SIZE=1024
#timestamp
def stamp():
        global timestampGW
        ts = time.time()
        timestampGW = datetime.datetime.fromtimestamp(ts).strftime('%H%M%S%d%m%Y')
        return timestampGW
#def checksum(trama):
def checksum(trama):
        global check
        global check1
        global check2
        check = 0
        pre_check = 0
        for i in trama:
                a=ord(i)
                pre_check = pre_check^a
        check = hex(pre_check)
        check = check.upper()
#---------------------------------------------------------
        tamanyo = len(check)
        check1 = check[tamanyo-2]
        print 'CHECKSUM1-->', check1
        check2 = check[tamanyo-1]
        print 'CHECKSUM2-->', check2
        if check1 == 'X':
                check1 = '0'
#---------------------------------------------------------
        '''check1 = check[2]
```

```
        check2 = check[3]'''
        return check,check1, check2

#--------------------------funciones GW (GW-->server)-------------------------------

#funcion trama Alive (periodica cada x tiempo) //alive()
def alive(s):
        global check1
        global check2
        global timestampGW
        datos = '000'
        stamp()
        trama_alive                                                                    =
['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D',datos ]
        alive_join="".join(trama_alive)
        checksum(alive_join)
        completa                    =                       [start,                       'V',
identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D',datos,check1,
check2,stop]
        envio = "".join(completa)
        s.send(envio)
        tam=len(envio)
        print 'TRAMA ENVIADA-->',envio

#Evento de arranque/reset GW
def ResetArranque(s):
        global check1
        global check2
        global timestampGW
        HWSW = '101001' #version hardware y software
        a=hex(int(HWSW))
        version =a[2:]
        stamp()
        trama_arranque=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D','002','000',version]
        join_arranque = "".join(trama_arranque)
        checksum(join_arranque)
        completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D','002','000',version,check1,check2,stop]
        envio = "".join(completa)
        s.send(envio)
        print 'RESET ARRANQUE ENVIADO -->', envio
#Evento de bateria GW
def bateria(s):
        global check1
        global check2
        global timestampGW
        global valor_bat
        valor_bat = '0005' #5 caracteres ascii hexadecimales
        stamp()
        trama_bat=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'
D','805','000',valor_bat]
        join_bat="".join(trama_bat)
        checksum(join_bat)
        completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D','805','000',valor_bat,check1,check2,stop]
        envio = "".join(completa)
        s.send(envio)

#Evento de sensores GW
def sensores(s):
        global check1
        global check2
        global timestampGW
        global nevento
        nevento='600'
        stamp()
        trama_sensores=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D',nevento,'000']
        join_sensores="".join(trama_sensores)
        checksum(join_sensores)
```

```
        completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D',nevento,'000',check1,check2,stop]
        envio="".join(completa)
        s.send(envio)

#lectura de parametros
def lectura_parametros(s):
        global check1
        global check2
        lee_fichero('parametros.txt')
        datos = str(linea)
        print 'LECTURA DE PARAMETROS----------------------------------------------------'
        print datos
        trama_parametros=['T','X','L',identGW1,identGW2,identGW3,identGW4,datos,';']
        join_parametros="".join(trama_parametros)
        checksum(join_parametros)
        completa=[start,'T','X','L',identGW1,identGW2,identGW3,identGW4,datos,';',check1,check2,
stop]
        envio="".join(completa)
        s.send(envio)

#Lectura de alias GW
def lectura_alias(s,nombre_GW):
        global check1
        global check2
        trama_lectura_nombre = ['T','X','P',identGW1,identGW2,identGW3,identGW4,nombre_GW,';']
        lectura_nombre_join = "".join(trama_lectura_nombre)
        checksum(lectura_nombre_join)
        completa=[start,'T','X','P',identGW1,identGW2,identGW3,identGW4,nombre_GW,';',check1,che
ck2,stop]
        envio="".join(completa)
        s.send(envio)

#modificacion de parametros de GW
def mod_parametros(s,parametros_valores,R_aleatorio):
        global check1
        global check2
        trama_mod_param=['T','X','M',identGW1,identGW2,identGW3,identGW4,'IDT',R_aleatorio,';',p
arametros_valores,';']
        join_mod_param = "".join(trama_mod_param)
        checksum(join_mod_param)
        completa=[start,'T','X','M',identGW1,identGW2,identGW3,identGW4,'IDT',R_aleatorio,';',pa
rametros_valores,';',check1,check2,stop]
        envio="".join(completa)
        s.send(envio)

#modificacion de alias GW
def modificacion_alias(s,nombre_mod):
        nombre_GW = nombre_mod
        trama_mod_nombre=['T','X','N',identGW1,identGW2,identGW3,identGW4,'G',nombre_GW,';']
        mod_nombre_join="".join(trama_mod_nombre)
        checksum(mod_nombre_join)
        completa=[start,'T','X','N',identGW1,identGW2,identGW3,identGW4,'G',nombre_GW,';',check1
,check2,stop]
        envio="".join(completa)
        s.send(envio)

#cambio estados salidas GW
def cambio_estados(s,salida,valor):
        trama_estados=['T','X','O',identGW1,identGW2,identGW3,identGW4,salida,valor]
        join_trama = "".join(trama_estados)
        checksum(join_trama)
        completo=[start,'T','X','O',identGW1,identGW2,identGW3,identGW4,salida,valor,check1,chec
k2,stop]
        envio = "".join(completo)
        s.send(envio)

#funcion cuando el GW recibe un Reset  // Reset()
def Reset(s):
        print '------------------------RESET--------------------------'
        trama_reset=['T','X','R',identGW1,identGW2,identGW3,identGW4]
        join_reset="".join(trama_reset)
```

```
        checksum(join_reset)
        completa=[start,'T','X','R',identGW1,identGW2,identGW3,identGW4,check1,check2,stop]
        envio="".join(completa)
        s.send(envio)
        #command= "/usr/bin/sudo /sbin/reboot -f"
        #import subprocess
        #process = subprocess.Popen(command.split(),stdout=subprocess.PIPE)
        #output=process.communicate()[0]
        #print output

#funcion  lectura de estructura de red
def estructura_red(s):
        num_trama='1' #['1'-'9'] (1 caracter ascii decimal)
        num_total_tramas='1'  # ['1'-'9'] (1 caracter ascii decimal)
        num_total_nodos='1' # ['01'-'99'] (2 caracteres ascii decimales)
        ip_nodo='102' #3 caracteres ascii decimales
        mac_nodo='0004a330c43b' # 12 caracteres ascii hexadecimales
        trama_lectura_red                                                                    =
['T','X','T',identGW1,identGW2,identGW3,identGW4,'D',num_trama,_num_total_tramas,num_total_nodos
,ip_nodo,mac_nodo]
        join_trama_lectura_red = "".join(trama_lectura_red)
        checksum(join_trama_lectura_red)
        completa                                                                             =
[start,'T','X','T',identGW1,identGW2,identGW3,identGW4,'D',num_trama,num_total_tramas,num_total_
nodos,ip_nodo,mac_nodo,check1,check2,stop]
        envio="".join(completa)
        s.send(envio)

#---------------funciones GW refreenet (GW-->server)-------------------------------------

#funcion trama de eventos de UTR (medidas, umbrales y gradientes)
def eventos_utr(s,nodo,sensor,valor_a,valor_b):
        stamp()
        num_evento_pre = int(sensor) + 500
        num_evento = str(num_evento_pre)
        print 'NUM_EVENTO--->', num_evento
        ip_utr = nodo
        print 'NODO--->', nodo
        af = str(valor_a)
        print 'VALOR SENSOR A', af
        bf = str(valor_b)
        print 'VALOR SENSOR B', bf
        ao = ord(af)
        bo = ord(bf)
        ah = hex(ao)
        bh = hex(bo)
        als = ah.lstrip('0x')
        print 'lstripa', als
        bls = bh.lstrip('0x')
        print 'lstripb', bls
        c_pre = als + bls
        c = c_pre.zfill(5)
        print 'C-->', c
        valor_sensor_pre = c.zfill(5)
        print 'VALOR_SENSOR -->', valor_sensor_pre
        valor_sensor = valor_sensor_pre
        trama_eventos_utr                                                                    =
['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D',num_evento,ip_u
tr,valor_sensor]
        join_trama_eventos_utr="".join(trama_eventos_utr)
        checksum(join_trama_eventos_utr)
        completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D',num_evento,ip_utr,valor_sensor,check1,check2,stop]
        print 'CHECK1-->', check1
        print 'CHECK2-->', check2
        envio="".join(completa)
        s.send(envio)
        print 'EVENTOS UTR -->', envio

#funcion evento de bateria de UTR
def bateria_utr(s):
        ip_utr='000'
```

```
        valor_bat='12221'
        trama=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D','
801',ip_utr,valor_bat]
        join_trama="".join(trama)
        checksum(join_trama)
        completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestam
pGW,'D','801',ip_utr,valor_bat,check1,check2,stop]
        envio="".join(completa)
        s.send(envio)

#funcion evento de cobertura de utr
def cobertura_utr(s):
        valor_cobertura='12345'
        trama=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D','
800',ip_utr,valor_cobertura]
        join_trama="".join(trama)
        checksum(join_trama)

completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D',
'800',ip_utr,valor_cobertura,check1,check2,stop]
        envio="".join(completa)
        s.send(envio)

#funcion lectura de parametros de UTR
def lectura_utr(s,nodo,datos):
        print '------------------------------------------------------------------------'
        print '                 ENTRO EN LECTURA_UTR'
        print 'datos que llegan a la funcion-->', datos
        lista_pos = []
        lista_var = []
        lista_ascii = []
        if 'TPR' in datos:
                pos_param9 = datos.find('TPR')
                lista_pos.append(pos_param9)
        if 'TTW' in datos:
                pos_param10 = datos.find('TTW')
                lista_pos.append(pos_param10)
        if 'RET' in datos:
                pos_param11 = datos.find('RET')
                lista_pos.append(pos_param11)
        if 'RS' in datos:
                pos_param12 = datos.find('RS1')
                lista_pos.append(pos_param12)
        if 'TRX' in datos:
                pos_param13 = datos.find('TRX')
                lista_pos.append(pos_param13)
        if 'TQW' in datos:
                pos_param14 = datos.find('TQW')
                lista_pos.append(pos_param14)
        if 'IPH' in datos:
                pos_param15 = datos.find('IPH')
                lista_pos.append(pos_param15)
        if 'IPL' in datos:
                pos_param16 = datos.find('IPL')
                lista_pos.append(pos_param16)
        if 'CTX' in datos:
                pos_param17 = datos.find('CTX')
                lista_pos.append(pos_param17)
        if 'CAK' in datos:
                pos_param18 = datos.find('CAK')
                lista_pos.append(pos_param18)
        if 'SEN' in datos:
                pos_param1 = datos.find('SE')
                lista_pos.append(pos_param1)
        if ('TM%d' %(1)) in datos or ('TM%d' %(2)) in datos or ('TM%d' %(3)) in datos or ('TM%d'
%(4)) in datos or ('TM%d' %(5)) in datos or ('TM%d' %(6)) in datos:
                pos_param2 = datos.find('TM')
                lista_pos.append(pos_param2)
        if ('TE%d' %(1)) in datos or ('TE%d' %(2)) in datos or ('TE%d' %(3)) in datos or ('TE%d'
%(4)) in datos or ('TE%d' %(5)) in datos or ('TE%d' %(6)) in datos:
                pos_param3 = datos.find('TE')
                lista_pos.append(pos_param3)
```

```
        if ('UM%d' %(1)) in datos or ('UM%d' %(2)) in datos or ('UM%d' %(3)) in datos or ('UM%d'
%(4)) in datos or ('UM%d' %(5)) in datos or ('UM%d' %(6)) in datos:
                pos_param4 = datos.find('UM')
                lista_pos.append(pos_param4)
        if ('UN%d' %(1)) in datos or ('UN%d' %(2)) in datos or ('UN%d' %(3)) in datos or ('UN%d'
%(4)) in datos or ('UN%d' %(5)) in datos or ('UN%d' %(6)) in datos:
                pos_param5 = datos.find('UN')
                lista_pos.append(pos_param5)
        if ('GM%d' %(1)) in datos or ('GM%d' %(2)) in datos or ('GM%d' %(3)) in datos or ('GM%d'
%(4)) in datos or ('GM%d' %(5)) in datos or ('GM%d' %(6)) in datos:
                pos_param6 = datos.find('GM')
                lista_pos.append(pos_param6)
        if ('TG%d' %(1)) in datos or ('TG%d' %(2)) in datos or ('TG%d' %(3)) in datos or ('TG%d'
%(4)) in datos or ('TG%d' %(5)) in datos or ('TG%d' %(6)) in datos:
                pos_param7 = datos.find('TG')
                lista_pos.append(pos_param7)
        if ('TS%d' %(1)) in datos or ('TS%d' %(2)) in datos or ('TS%d' %(3)) in datos or ('TS%d'
%(4)) in datos or ('TS%d' %(5)) in datos or ('TS%d' %(6)) in datos:
                pos_param8 = datos.find('TS')
                lista_pos.append(pos_param8)
        print 'LISTA DE POSICIONES -->', lista_pos
        for i in range(len(lista_pos)):
                print 'i -->', i
                if i != len(lista_pos)-1:
                        print 'ENTRO EN EL PRIMER IF'
                param_var = datos[lista_pos[i]:lista_pos[i+1]]
                parametro = param_var[0:3]
                        if 'TS' in parametro:
                                valor = param_var[3]
                                valor_3 = valor[3]
                                valor_4 = valor[4]
                                valor_c = valor_4 + valor_3
                                var_x = chr(int(var_c,16))
                                valor_join = valor_x
                        if 'CAK' in parametro:
                        valor = param_var[3]
                        for i in range(len(valor)):
                                        valor_ascii = str(ord(valor[i]))
                                lista_ascii.append(valor_ascii)
                                valor_join = "".join(lista_ascii)
                        if 'CTX' in parametro:
                                valor = param_var[3]
                                for i in range(len(valor)):
                                        valor_ascii = str(ord(valor[i]))
                                        lista_ascii.append(valor_ascii)
                                        valor_join = "".join(lista_ascii)
                        if 'SEN' in parametro:
                                valor = param_var[3]
                                for i in range(len(valor)):
                        valor_ascii = str(ord(valor[i]))
                                lista_ascii.append(valor_ascii)
                                valor_join = "".join(lista_ascii)
                        if 'TPR' in parametro:
                        valor = param_var[3]
                                for i in range(len(valor)):
                                valor_ascii = str(ord(valor[i]))
                                lista_ascii.append(valor_ascii)
                                valor_join = "".join(lista_ascii)
                        if 'TTW' in parametro:
                        valor = param_var[3:5]
                        b = hex(ord(valor[0]))
                        c = hex(ord(valor[1]))
                                b1 = b[2:]
                        b1 = b1.zfill(2)
                        c1 = c[2:]
                        c1 = c1.zfill(2)
                                d = '0x' + b1 + c1
                        valor_join = str(int(d,16))
                        if 'RET' in parametro:
                        valor = param_var[3]
                                for i in range(len(valor)):
                                valor_ascii = str(ord(valor[i]))
```

```
        lista_ascii.append(valor_ascii)
        valor_join = "".join(lista_ascii)
if 'RS' in parametro:
valor = param_var[3]
for i in range(len(valor)):
        valor_ascii = str(ord(valor[i]))
        lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'TRX' in parametro:
valor = param_var[3:5]
b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
b1 = b[2:]
b1 = b1.zfill(2)
c1 = c[2:]
        c1 = c1.zfill(2)
d = '0x' + b1 + c1
valor_join = str(int(d,16))
if 'TQW' in parametro:
valor = param_var[3:5]
b = hex(ord(valor[0]))
c = hex(ord(valor[1]))
        b1 = b[2:]
b1 = b1.zfill(2)
c1 = c[2:]
c1 = c1.zfill(2)
        d = '0x' + b1 + c1
valor_join = str(int(d,16))
if 'IPH' in parametro:
valor = param_var[3]
        for i in range(len(valor)):
        valor_ascii = str(ord(valor[i]))
        lista_ascii.append(valor_ascii)
        valor_join = "".join(lista_ascii)
if 'IPL' in parametro:
valor = param_var[3]
for i in range(len(valor)):
        valor_ascii = str(ord(valor[i]))
        lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'TM' in parametro:
valor = param_var[3:7]
        b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
        d = hex(ord(valor[2]))
        e = hex(ord(valor[3]))
        b1 = b[2:]
        b1 = b1.zfill(2)
        c1 = c[2:]
        c1 = c1.zfill(2)
        d1 = d[2:]
        d1 = d1.zfill(2)
        e1 = e[2:]
        e1 = e1.zfill(2)
        f = '0x' + b1 + c1 + d1 + e1
        valor_join = str(int(f,16))
if 'TE' in parametro:
valor = param_var[3:7]
b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
d = hex(ord(valor[2]))
e = hex(ord(valor[3]))
b1 = b[2:]
        b1 = b1.zfill(2)
c1 = c[2:]
c1 = c1.zfill(2)
d1 = d[2:]
        d1 = d1.zfill(2)
e1 = e[2:]
e1 = e1.zfill(2)
f = '0x' + b1 + c1 + d1 + e1
        valor_join = str(int(f,16))
```

```
                if 'UM' in parametro:
                valor = param_var[3:5]
                        b = hex(ord(valor[0]))
                        c = hex(ord(valor[1]))
                        b1 = b[2:]
                        b1 = b1.zfill(2)
                        c1 = c[2:]
                        c1 = c1.zfill(2)
                        d = '0x' + b1 + c1
                        valor_join = str(int(d,16))
                if 'UN' in parametro:
                        valor = param_var[3:5]
                b = hex(ord(valor[0]))
                c = hex(ord(valor[1]))
                b1 = b[2:]
                        b1 = b1.zfill(2)
                c1 = c[2:]
                c1 = c1.zfill(2)
                d = '0x' + b1 + c1
                        valor_join = str(int(d,16))
                if 'GM' in parametro:
                valor = param_var[3:5]
                b = hex(ord(valor[0]))
                        c = hex(ord(valor[1]))
                b1 = b[2:]
                b1 = b1.zfill(2)
                c1 = c[2:]
                        c1 = c1.zfill(2)
                d = '0x' + b1 + c1
                valor_join = str(int(d,16))
                if 'TG' in parametro:
                valor = param_var[3:7]
                b = hex(ord(valor[0]))
                c = hex(ord(valor[1]))
                        d = hex(ord(valor[2]))
                e = hex(ord(valor[3]))
                b1 = b[2:]
                b1 = b1.zfill(2)
                        c1 = c[2:]
                c1 = c1.zfill(2)
                d1 = d[2:]
                d1 = d1.zfill(2)
                        e1 = e[2:]
                e1 = e1.zfill(2)
                f = '0x' + b1 + c1 + d1 + e1
                valor_join = str(int(f,16))
                lista_var.append(';')
        lista_var.append(parametro)
                lista_var.append(';')
                lista_var.append(valor_join)
                lista_ascii = []
        else:
                print 'ENTRO EN EL ELSE'
                param_var = datos[lista_pos[i]:]
                parametro = param_var[0:3]
                if 'TS' in parametro:
                        valor = hex(ord(param_var[3]))
                        valor_3 = valor[2]
                        valor_4 = valor[3]
                        valor_c = '0x' + valor_4 + valor_3
                        valor_x = chr(int(valor_c,16))
                        valor_join = valor_x
                if 'CAK' in parametro:
                        valor = param_var[3]
                        for i in range(len(valor)):
                                valor_ascii = str(ord(valor[i]))
                                lista_ascii.append(valor_ascii)
                                valor_join = "".join(lista_ascii)
                if 'CTX' in parametro:
                        valor = param_var[3]
                        for i in range(len(valor)):
                                valor_ascii = str(ord(valor[i]))
```

```
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'SEN' in parametro:
        valor = param_var[3]
        for i in range(len(valor)):
                valor_ascii = str(ord(valor[i]))
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'TPR' in parametro:
        valor = param_var[3]
        for i in range(len(valor)):
                valor_ascii = str(ord(valor[i]))
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'TTW' in parametro:
        valor = param_var[3:5]
        b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
        b1 = b[2:]
        b1 = b1.zfill(2)
        c1 = c[2:]
        c1 = c1.zfill(2)
        d = '0x' + b1 + c1
        valor_join = str(int(d,16))
if 'RET' in parametro:
        valor = param_var[3]
        for i in range(len(valor)):
                valor_ascii = str(ord(valor[i]))
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'RS' in parametro:
        valor = param_var[3]
        for i in range(len(valor)):
                valor_ascii = str(ord(valor[i]))
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'TRX' in parametro:
        valor = param_var[3:5]
        b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
        b1 = b[2:]
        b1 = b1.zfill(2)
        c1 = c[2:]
        c1 = c1.zfill(2)
        d = '0x' + b1 + c1
        valor_join = str(int(d,16))
if 'TQW' in parametro:
        valor = param_var[3:5]
        b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
        b1 = b[2:]
        b1 = b1.zfill(2)
        c1 = c[2:]
        c1 = c1.zfill(2)
        d = '0x' + b1 + c1
        valor_join = str(int(d,16))
if 'IPH' in parametro:
        valor = param_var[3]
        for i in range(len(valor)):
                valor_ascii = str(ord(valor[i]))
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'IPL' in parametro:
        valor = param_var[3]
        for i in range(len(valor)):
                valor_ascii = str(ord(valor[i]))
                lista_ascii.append(valor_ascii)
                valor_join = "".join(lista_ascii)
if 'TM' in parametro:
        valor = param_var[3:7]
        b = hex(ord(valor[0]))
        c = hex(ord(valor[1]))
```

```
            d = hex(ord(valor[2]))
            e = hex(ord(valor[3]))
            b1 = b[2:]
            b1 = b1.zfill(2)
            c1 = c[2:]
            c1 = c1.zfill(2)
            d1 = d[2:]
            d1 = d1.zfill(2)
            e1 = e[2:]
            e1 = e1.zfill(2)
            f = '0x' + b1 + c1 + d1 + e1
            valor_join = str(int(f,16))
    if 'TE' in parametro:
            valor = param_var[3:7]
            b = hex(ord(valor[0]))
            c = hex(ord(valor[1]))
            d = hex(ord(valor[2]))
            e = hex(ord(valor[3]))
            b1 = b[2:]
            b1 = b1.zfill(2)
            c1 = c[2:]
            c1 = c1.zfill(2)
            d1 = d[2:]
            d1 = d1.zfill(2)
            e1 = e[2:]
            e1 = e1.zfill(2)
            f = '0x' + b1 + c1 + d1 + e1
            valor_join = str(int(f,16))
    if 'UM' in parametro:
            valor = param_var[3:5]
            b = hex(ord(valor[0]))
            c = hex(ord(valor[1]))
            b1 = b[2:]
            b1 = b1.zfill(2)
            c1 = c[2:]
            c1 = c1.zfill(2)
            d = '0x' + b1 + c1
            valor_join = str(int(d,16))
    if 'UN' in parametro:
            valor = param_var[3:5]
            b = hex(ord(valor[0]))
            c = hex(ord(valor[1]))
            b1 = b[2:]
            b1 = b1.zfill(2)
            c1 = c[2:]
            c1 = c1.zfill(2)
            d = '0x' + b1 + c1
            valor_join = str(int(d,16))
    if 'GM' in parametro:
            valor = param_var[3:5]
            b = hex(ord(valor[0]))
            c = hex(ord(valor[1]))
            b1 = b[2:]
            b1 = b1.zfill(2)
            c1 = c[2:]
            c1 = c1.zfill(2)
            d = '0x' + b1 + c1
            valor_join = str(int(d,16))
    if 'TG' in parametro:
            valor = param_var[3:7]
            b = hex(ord(valor[0]))
            c = hex(ord(valor[1]))
            d = hex(ord(valor[2]))
            e = hex(ord(valor[3]))
            b1 = b[2:]
            b1 = b1.zfill(2)
            c1 = c[2:]
            c1 = c1.zfill(2)
            d1 = d[2:]
            d1 = d1.zfill(2)
            e1 = e[2:]
            e1 = e1.zfill(2)
```

```
                                f = '0x' + b1 + c1 + d1 + e1
                                valor_join = str(int(f,16))
                        lista_var.append(';')
                        lista_var.append(parametro)
                        #lista_var.append(';')
                        lista_var.append(valor_join)
                        lista_ascii = []
        print 'LISTA DE VARIABLES-->', lista_var
        parametros = "".join(lista_var)
        trama=['T','X','L',identGW1,identGW2,identGW3,identGW4,'D',nodo,parametros]
        join_trama="".join(trama)
        checksum(join_trama)

completa=[start,'T','X','L',identGW1,identGW2,identGW3,identGW4,'D',nodo,parametros,check1,check
2,stop]
        envio="".join(completa)
        s.send(envio)
        print 'ENVIO -->', envio
        print '-------------------------------------------------------------------------------'
        print '                                    LECTURA UTR ENVIADA'
        print '-------------------------------------------------------------------------------'

#funcion modificacion de parametros de utr
def mod_param_utr(s,nodo,R_aleatorio):
        R1 = str(ord(R_aleatorio[0]))
        R2 = str(ord(R_aleatorio[1]))
        a = R1.zfill(2)
        b = R2.zfill(2)
        R_aleatorioS = a + b
        trama=['T','X','M',identGW1,identGW2,identGW3,identGW4,'N',nodo,'IDT',R_aleatorioS,';']
        join_trama="".join(trama)
        checksum(join_trama)
        completa=[start,'T','X','M',identGW1,identGW2,identGW3,identGW4,'N',nodo,'IDT',R_aleator
ioS,';',check1,check2,stop]
        envio="".join(completa)
        s.send(envio)
        print 'R ALEATORIO -->', R_aleatorioS
        print 'ENVIO -->', envio
        print '-------------------------------------------------------------------------'
        print '     CONFIRMACION AL SERVIDOR DE MODIFICACION DE PARAMETROS UTR'
        print '-------------------------------------------------------------------------'

#funcion lectura de alias de UTR
def alias_utr(s,nodo):
        print 'ENTRO EN ALIAS UTR'
        archivo = 'json.json'
        f = open(archivo)
        red = json.load(f)
        type(red)
        red.keys()
        f.close()
        for i in range(14):
                if i != 0:
                        result1 = red['red']['nodo%d' %(i)]['ACTIVO']
                        result2 = red['red']['nodo%d' %(i)]['IP']
                        if result1 == '1' and result2 == nodo:
                                AN = red['red']['nodo%d' %(i)]['AN']
                                SEN = red['red']['nodo%d' %(i)]['SEN']
                                TNO = red['red']['nodo%d' %(i)]['TNO']
                                TS1 = red['red']['nodo%d' %(i)]['sensores']['TS1']
                                if TS1 == 0:
                                        TS1 = ''
                                AS1 = red['red']['nodo%d' %(i)]['sensores']['AS1']
                                if AS1 == 0:
                                        AS1 = ''
                                TS2 = red['red']['nodo%d' %(i)]['sensores']['TS2']
                                if TS2 == 0:
                                        TS2 = ''
                                AS2 = red['red']['nodo%d' %(i)]['sensores']['AS2']
                                if AS2 == 0:
                                        AS2 = ''
                                TS3 = red['red']['nodo%d' %(i)]['sensores']['TS3']
```

```
                                if TS3 == 0:
                                        TS3 = ''
                                AS3 = red['red']['nodo%d' %(i)]['sensores']['AS3']
                                if AS3 == 0:
                                        AS3 = ''
                                TS4 = red['red']['nodo%d' %(i)]['sensores']['TS4']
                                if TS4 == 0:
                                        TS4 = ''
                                AS4 = red['red']['nodo%d' %(i)]['sensores']['AS4']
                                if AS4 == 0:
                                        AS4 = ''
                                TS5 = red['red']['nodo%d' %(i)]['sensores']['TS5']
                                if TS5 == 0:
                                        TS5 = ''
                                AS5 = red['red']['nodo%d' %(i)]['sensores']['AS5']
                                if AS5 == 0:
                                        AS5 = ''
                                TS6 = red['red']['nodo%d' %(i)]['sensores']['TS6']
                                if TS6 == 0:
                                        TS6 = ''
                                AS6 = red['red']['nodo%d' %(i)]['sensores']['AS6']
                                if AS6 == 0:
                                        AS6 = ''
                                break
        trama=['T','X','P',identGW1,identGW2,identGW3,identGW4,'D',nodo,AN,';',TNO,SEN,TS1,AS1,'
;',TS2,AS2,';',TS3,AS3,';',TS4,AS4,';',TS5,AS5,';',TS6,AS6,';']
        join_trama="".join(trama)
        checksum(join_trama)
        completa=[start,'T','X','P',identGW1,identGW2,identGW3,identGW4,'D',nodo,AN,';',TNO,SEN,
TS1,AS1,';',TS2,AS2,';',TS3,AS3,';',TS4,AS4,';',TS5,AS5,';',TS6,AS6,';',check1,check2,stop]
        envio="".join(completa)
        s.send(envio)
        print 'ENVIO DE ALIAS UTR -->', envio

#funcion modificacion de alias de UTR
def mod_alias_utr(s,sec_interes):
        print 'SEC_INTERES (COMPROBAR QUE EXISTE CHECKSUM) -->', sec_interes
        completa = [start,sec_interes,stop]
        envio = "".join(completa)
        s.send(envio)


#funcion reset UTR
def reset_utr(s,nodo):
        trama=['T','X','R',identGW1,identGW2,identGW3,identGW4,'N',nodo]
        join_trama="".join(trama)
        checksum(join_trama)
        completa=[start,'T','X','R',identGW1,identGW2,identGW3,identGW4,'N',nodo,check1,check2,s
top]
        envio="".join(completa)
        s.send(envio)
        print 'Reset utr de vuelta al servidor', envio

#funcion eliminar UTR
def eliminar_utr(s,ip_utr,mac_utr):
        trama=['T','X','K',identGW1,identGW2,identGW3,identGW4,'D','1','1','01',ip_utr,mac_utr]
        join_trama="".join(trama)
        checksum(join_trama)
        completa=[start,'T','X','K',identGW1,identGW2,identGW3,identGW4,'D','1','1','01',ip_utr,
mac_utr,check1,check2,stop]
        envio="".join(completa)
        s.send(envio)
        print '-------------------------------------------------------------------------'
        print '                           UTR ELIMINADO'
        print '-------------------------------------------------------------------------'
        eliminar_nodo_json(ip_utr)

#funcion peticion de IP de UTR
def peticion_ip_utr(s,nodo,mac):
        print 'EL PROGRAMA HA ENTRADO A LA FUNCION PETICION DE IP UTR'
        trama=['Y',identGW1,identGW2,identGW3,identGW4,'D',nodo,mac]
        join_trama="".join(trama)
```

```
        checksum(join_trama)
        completa=['\2','Y',identGW1,identGW2,identGW3,identGW4,'D',nodo,mac,check1,check2,'\5']
        envio="".join(completa)
        print 'ENVIAMOS---------->', envio
        s.send(envio)

#funcion cambio de estados salidas de nodo
def cambio_salidas_nodo(s,nodo,salida,valor):
        trama=['T','X','O',identGW1,identGW2,identGW3,identGW4,'N',nodo,salida,valor]
        join_trama="".join(trama)
        checksum(join_trama)
        completo=[start,'T','X','O',identGW1,identGW2,identGW3,identGW4,'N',nodo,salida,valor,ch
eck1,check2,stop]
        envio = "".join(completo)
        s.send(envio)
        print 'CAMBIO SALIDAS NODO', envio

#funcion alarma umbral/gradiente utr
def alarma_umbral_utr(s,nodo,codigo,sensor,valor1,valor2):
        if sensor == '\x00':
                sensor = 0
        if sensor == '\x01':
                sensor = 1
        if sensor == '\x02':
                sensor = 2
        if sensor == '\x03':
                sensor = 3
        if sensor == '\x04':
                sensor = 4
        if sensor == '\x05':
                sensor = 5
        if codigo == '\x00':
                n_evento1 = 506 + sensor
        if codigo == '\x01':
                n_evento1 = 512 + sensor
        n_evento = str(n_evento1)
        af = str(valor1)
        bf = str(valor2)
        ao = ord(af)
        bo = ord(bf)
        ah = hex(ao)
        bh = hex(bo)
        als = ah.lstrip('0x')
        bls = bh.lstrip('0x')
        az = als.zfill(2)
        bz = bls.zfill(2)
        c = az + bz
        grad_umbr = c.zfill(5)
        valor = nodo + grad_umbr
        stamp()

trama=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D',n_evento,
valor]
        join_trama="".join(trama)
        checksum(join_trama)

completa=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D',
n_evento,valor,check1,check2,stop]
        envio="".join(completa)
        s.send(envio)
        print ' enviado', envio

#funcion evento estadistica de utr
def estadistica_utr(s,nodo,cobertura,bateria1,bateria2):
        af = str(bateria1)
        bf = str(bateria2)
        ao = ord(af)
        bo = ord(bf)
        ah = hex(ao)
        bh = hex(bo)
        als = ah.lstrip('0x')
        bls = bh.lstrip('0x')
```

```
        az = als.zfill(2)
        bz = bls.zfill(2)
        c = az + bz
        bateria = c.zfill(5)
        print 'VALOR DE BATERIA -->', bateria
        valor1 = nodo + bateria
        print 'VALOR QUE SE ENVIA (IP+VALOR BAT)-->', valor1
        stamp()

trama=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D','801',val
or1]
        join_trama="".join(trama)
        checksum(join_trama)

completa1=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D'
,'801',valor1,check1,check2,stop]
        envio1="".join(completa1)
        #s.send(envio)
        print 'envio1 -->', envio1
        cob = str(cobertura)
        cob2 = ord(cob)
        cob3 = hex(cob2)
        cob4 = cob3.lstrip('0x')
        cob5 = cob4.zfill(5)
        print 'VALOR COB QUE SE OBTIENE DE TRAMA -->', cob
        print 'VALOR DE COBERTURA -->', cob5
        valor2 = nodo + cob5
        print 'VALOR QUE SE ENVIA (IP+VALOR COB)-->', valor2

trama=['V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D','800',val
or2]
        join_trama="".join(trama)
        checksum(join_trama)

completa2=[start,'V',identGW1,identGW2,identGW3,identGW4,'I',TCP_IP,'P',PORT,'T',timestampGW,'D'
,'800',valor2,check1,check2,stop]
        envio2="".join(completa2)
        envio_fin = envio1 + envio2
        envio_final = "".join(envio_fin)
        s.send(envio_final)
        print 'envio2 -->', envio2
        print 'envio_final -->', envio_final

#------------GATEWAY --> ANTENA RF-----------------------------------------------------
#funcion checksum RF
def checksumRF(checkeo):
        global check1RF
        global check2RF
        global checkRF
        checkRF = 0
        pre_checkRF = 0
        datos_check = 0
        ip_tipo = checkeo[0:4]
        datos = checkeo[4:]
        print 'IP_TIPO-->',ip_tipo
        print 'DATOS -->',datos
        tami = len(ip_tipo)
        print 'TAMI', tami
        tam = len(datos)
        print 'TAMAYO', tam
        for i in range(tami):
                a = ord(ip_tipo[i])
                pre_checkRF = pre_checkRF+a
                print 'pre_checkRF',pre_checkRF
                print 'ip_tipo 1', a
        if len(datos) > 1:
                for i in range(tam):
                        if i == 0:
                        print 'I',i
                                c = int(datos[i:i+2],16)
                        datos_check = datos_check + c
                        print 'datos_check',datos_check
```

```
                                print 'datos', c
                        else:
                                if i%2 == 0:
                                        print 'I',i
                                        c = int(datos[i:i+2],16)
                                        datos_check = datos_check + c
                                        print 'datos_check',datos_check
                                        print 'datos', c
        check1 = pre_checkRF + datos_check
        check = hex(check1)
        check = check.upper()
        print 'CHECK-->', check
        tamanyo = len(check)
        check1RF = check[tamanyo-2]
        print 'CHECKSUM1RF-->', check1RF
        check2RF = check[tamanyo-1]
        print 'CHECKSUM2RF-->', check2RF
        return checkRF,check1RF,check2RF
##-------------------------------------------------
#funcion lectura de parametros (GW --> antena RF)
def lectura_parametrosRF(antena,nodo):
        IP2 = nodo[0]
        IP1 = nodo[1]
        IP0 = nodo[2]
        tipo ='L'
        L = '\4'
        lect_param_RF = [IP2,IP1,IP0,tipo]
        join_lect_param_RF = "".join(lect_param_RF)
        checksumRF(join_lect_param_RF)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,IP2,IP1,IP0,tipo,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'LECTURA DE PARAMETROSRF -->', join_completo
        print '-------------------------------------------------------------------'
        print '                     Lectura de parametros RF'
        print '-------------------------------------------------------------------'

#funcion lectura de alias UTR (GW --> antena RF)
def lectura_aliasRF(antena,nodo):
        IP2 = nodo[0]
        IP1 = nodo[1]
        IP0 = nodo[2]
        tipo ='L'
        L = '\4'
        lect_alias_RF = [IP2,IP1,IP0,tipo]
        join_lect_alias_RF = "".join(lect_alias_RF)
        checksumRF(join_lect_alias_RF)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,IP2,IP1,IP0,tipo,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'LECTURA ALIAS RF-->', join_completo
        print '-------------------------------------------------------------------'
        print '                                lectura alias RF'
        print '-------------------------------------------------------------------'

#funcion modificacion de parametros utr (GW --> antena RF)
def mod_paramRF(antena,nodo,datosRF,R_aleatorio):
def mod_paramRF(antena,nodo,datosRF,R_aleatorio):
        print 'NODO -->', nodo
        print 'R aleatoria -->', R_aleatorio
        print 'DATOS RF -->', datosRF
        lista_var = []
        lista_final = []
        if 'SE' in datosRF:
                pos = datosRF.find('SE')
                lista_var.append(pos)
```

```
#--------------
        if 'TM1' in datosRF:
                pos1 = datosRF.find('TM1')
                lista_var.append(pos1)
        if 'TE1' in datosRF:
                pos2 = datosRF.find('TE1')
                lista_var.append(pos2)
        if 'UM1' in datosRF:
                pos3 = datosRF.find('UM1')
                lista_var.append(pos3)
        if 'UN1' in datosRF:
                pos4 = datosRF.find('UN1')
                lista_var.append(pos4)
        if 'GM1' in datosRF:
                pos5 = datosRF.find('GM1')
                lista_var.append(pos5)
        if 'TG1' in datosRF:
                pos6 = datosRF.find('TG1')
                lista_var.append(pos6)
        if 'TS1' in datosRF:
                pos7 = datosRF.find('TS1')
                lista_var.append(pos7)
#--------------
        if 'TM2' in datosRF:
                pos8 = datosRF.find('TM2')
                lista_var.append(pos8)
        if 'TE2' in datosRF:
                pos9 = datosRF.find('TE2')
                lista_var.append(pos9)
        if 'UM2' in datosRF:
                pos10 = datosRF.find('UM2')
                lista_var.append(pos10)
        if 'UN2' in datosRF:
                pos11 = datosRF.find('UN2')
                lista_var.append(pos11)
        if 'GM2' in datosRF:
                pos12 = datosRF.find('GM2')
                lista_var.append(pos12)
        if 'TG2' in datosRF:
                pos13 = datosRF.find('TG2')
                lista_var.append(pos13)
        if 'TS2' in datosRF:
                pos14 = datosRF.find('TS2')
                lista_var.append(pos14)
#-----------------
        if 'TM3' in datosRF:
                pos15 = datosRF.find('TM3')
                lista_var.append(pos15)
        if 'TE3' in datosRF:
                pos16 = datosRF.find('TE3')
                lista_var.append(pos16)
        if 'UM3' in datosRF:
                pos17 = datosRF.find('UM3')
                lista_var.append(pos17)
        if 'UN3' in datosRF:
                pos18 = datosRF.find('UN3')
                lista_var.append(pos18)
        if 'GM3' in datosRF:
                pos19 = datosRF.find('GM3')
                lista_var.append(pos19)
        if 'TG3' in datosRF:
                pos20 = datosRF.find('TG3')
                lista_var.append(pos20)
        if 'TS3' in datosRF:
                pos21 = datosRF.find('TS3')
                lista_var.append(pos21)
#--------------------------------
        if 'TM4' in datosRF:
                pos22 = datosRF.find('TM4')
                lista_var.append(pos22)
        if 'TE4' in datosRF:
                pos23 = datosRF.find('TE4')
```

```
                lista_var.append(pos23)
        if 'UM4' in datosRF:
                pos24 = datosRF.find('UM4')
                lista_var.append(pos24)
        if 'UN4' in datosRF:
                pos25 = datosRF.find('UN4')
                lista_var.append(pos25)
        if 'GM4' in datosRF:
                pos26 = datosRF.find('GM4')
                lista_var.append(pos26)
        if 'TG4' in datosRF:
                pos27 = datosRF.find('TG4')
                lista_var.append(pos27)
        if 'TS4' in datosRF:
                pos28 = datosRF.find('TS4')
                lista_var.append(pos28)
#----------------------
        if 'TM5' in datosRF:
                pos29 = datosRF.find('TM5')
                lista_var.append(pos29)
        if 'TE5' in datosRF:
                pos30 = datosRF.find('TE5')
                lista_var.append(pos30)
        if 'UM5' in datosRF:
                pos31 = datosRF.find('UM5')
                lista_var.append(pos31)
        if 'UN5' in datosRF:
                pos32 = datosRF.find('UN5')
                lista_var.append(pos32)
        if 'GM5' in datosRF:
                pos33 = datosRF.find('GM5')
                lista_var.append(pos33)
        if 'TG5' in datosRF:
                pos34 = datosRF.find('TG5')
                lista_var.append(pos34)
        if 'TS5' in datosRF:
                pos35 = datosRF.find('TS5')
                lista_var.append(pos35)
#---------------------------
        if 'TM6' in datosRF:
                pos36 = datosRF.find('TM6')
                lista_var.append(pos36)
        if 'TE6' in datosRF:
                pos37 = datosRF.find('TE6')
                lista_var.append(pos37)
        if 'UM6' in datosRF:
                pos38 = datosRF.find('UM6')
                lista_var.append(pos38)
        if 'UN6' in datosRF:
                pos39 = datosRF.find('UN6')
                lista_var.append(pos39)
        if 'GM6' in datosRF:
                pos40 = datosRF.find('GM6')
                lista_var.append(pos40)
        if 'TG6' in datosRF:
                pos41 = datosRF.find('TG6')
                lista_var.append(pos41)
        if 'TS6' in datosRF:
                pos42 = datosRF.find('TS6')
                lista_var.append(pos42)

#-------------------------------

        print 'VARIABLE 0 DE LA LISTA_VAR -->', lista_var[0]
        print 'LISTA DE VARIABLES ENCONTRADAS -->', lista_var
        for i in range(len(lista_var)):
                print 'I -->', i
                print 'TAMANYO  LISTA VAR -->', len(lista_var)
                if i == len(lista_var)-1:
                        var = datosRF[lista_var[i]:]
                        if 'TS' in var:
                                var = datosRF[lista_var[i]:lista_var[i]+5]
```

```
                        print 'ENTRO EN EL IF (TS IN VAR1)'
                        var_3 = var[3]
                        var_4 = var[4]
                        var_c = var_4+var_3
                        var_x = chr(int(var_c,16))
                        var_final = var[0:3] + var_x
                        print 'VAR_FINAL -->', var_final
                        lista_final.append(var_final)
                        print 'lista acumulados -->', lista_final
                if 'SE' in var:
                        lista = []
                        a = var[3:]
                        b = int(a)
                        for i in range(1):
                                c = b&255
                                d = chr(c)
                                b = b>>8
                                lista.append(d)
                        lista.reverse()
                        j = "".join(lista)
                        f = var[0:3] + j
                        lista_final.append(f)
                        print 'lista acumulados -->', lista_final
#-------------------------------------------------------------------------------
                if 'UM' in var:
                        lista = []
                        a = var[3:]
                        b = int(a)
                        for i in range(2):
                                c = b&255
                                d = chr(c)
                                b = b>>8
                                lista.append(d)
                        lista.reverse()
                        j = "".join(lista)
                        f = var[0:3] + j
                        lista_final.append(f)
                if 'UN' in var:
                        lista = []
                        a = var[3:]
                        b = int(a)
                        for i in range(2):
                                c = b&255
                                d = chr(c)
                                b = b>>8
                                lista.append(d)
                        lista.reverse()
                        j = "".join(lista)
                        f = var[0:3] + j
                        lista_final.append(f)
                        print 'lista acumulados -->', lista_final
                if 'TM' in var:
                        lista = []
                        a = var[3:]
                        b = int(a)
                        for i in range(4):
                                c = b&255
                                d = chr(c)
                                b = b>>8
                                lista.append(d)
                        lista.reverse()
                        j = "".join(lista)
                        f = var[0:3] + j
                        lista_final.append(f)
                        print 'lista acumulados -->', lista_final
                if 'TE' in var:
                        lista = []
                        a = var[3:]
                        b = int(a)
                        for i in range(4):
                                c = b&255
                                d = chr(c)
```

```
                                        b = b>>8
                                        lista.append(d)
                                lista.reverse()
                                j = "".join(lista)
                                f = var[0:3] + j
                                lista_final.append(f)
                                print 'lista acumulados -->', lista_final
                        if 'GM' in var:
                                lista = []
                                a = var[3:]
                                b = int(a)
                                for i in range(2):
                                        c = b&255
                                        d = chr(c)
                                        b = b>>8
                                        lista.append(d)
                                lista.reverse()
                                j = "".join(lista)
                                f = var[0:3] + j
                                lista_final.append(f)
                                print 'lista acumulados -->', lista_final
                        if 'TG' in var:
                                lista = []
                                a = var[3:]
                                b = int(a)
                                for i in range(4):
                                        c = b&255
                                        d = chr(c)
                                        b = b>>8
                                        lista.append(d)
                                lista.reverse()
                                j = "".join(lista)
                                f = var[0:3] + j
                                lista_final.append(f)
                                print 'lista acumulados -->', lista_final
#-------------------------------------------------------------------------------
                        print 'lista final -->', lista_final
                else:
                        var = datosRF[lista_var[i]:lista_var[i+1]]
                        if 'TS' in var:
                                print 'ENTRO EN EL IF (TS IN VAR2)'
                                var_3 = var[3]
                                var_4 = var[4]
                                var_c = var_4 + var_3
                                var_x = chr(int(var_c,16))
                                var_final = var[0:3] + var_x
                                print 'VAR_FINAL -->', var_final
                                lista_final.append(var_final)
                                print 'lista acumulados -->', lista_final
                        if 'SE' in var:
                                lista = []
                                a = var[3:6]
                                b = int(a)
                                for i in range(1):
                                        c = b&255
                                        d = chr(c)
                                        b = b>>8
                                        lista.append(d)
                                lista.reverse()
                                j = "".join(lista)
                                f = var[0:3] + j
                                lista_final.append(f)
                                print 'lista acumulados -->', lista_final
#-------------------------------------------------------------------------------
                        if 'TM' in var:
                                lista = []
                                a = var[3:]
                                b = int(a)
                                for i in range(4):
                                        c = b&255
                                        d = chr(c)
                                        b = b>>8
```

```
                lista.append(d)
        lista.reverse()
        j = "".join(lista)
        f = var[0:3] + j
        lista_final.append(f)
        print 'lista acumulados -->', lista_final
if 'TE' in var:
        lista = []
        a = var[3:]
        b = int(a)
        for i in range(4):
                c = b&255
                d = chr(c)
                b = b>>8
                lista.append(d)
        lista.reverse()
        j = "".join(lista)
        f = var[0:3] + j
        lista_final.append(f)
        print 'lista acumulados -->', lista_final
if 'UM' in var:
        lista = []
        a = var[3:]
        b = int(a)
        for i in range(2):
                c = b&255
                d = chr(c)
                b = b>>8
                lista.append(d)
        lista.reverse()
        j = "".join(lista)
        f = var[0:3] + j
        lista_final.append(f)
        print 'lista acumulados -->', lista_final
if 'UN' in var:
        lista = []
        a = var[3:]
        b = int(a)
        for i in range(2):
                c = b&255
                d = chr(c)
                b = b>>8
                lista.append(d)
        lista.reverse()
        j = "".join(lista)
        f = var[0:3] + j
        lista_final.append(f)
        print 'lista acumulados -->', lista_final
if 'GM' in var:
        lista = []
        a = var[3:]
        b = int(a)
        for i in range(2):
                c = b&255
                d = chr(c)
                b = b>>8
                lista.append(d)
        lista.reverse()
        j = "".join(lista)
        f = var[0:3] + j
        lista_final.append(f)
        print 'lista acumulados -->', lista_final
if 'TG' in var:
        lista = []
        a = var[3:]
        b = int(a)
        for i in range(4):
                c = b&255
                d = chr(c)
                b = b>>8
                lista.append(d)
        lista.reverse()
```

```
                                j = "".join(lista)
                                f = var[0:3] + j
                                lista_final.append(f)
                                print 'lista acumulados -->', lista_final
#-------------------------------------------------------------------------------
                        print 'lista acumulados -->', lista_final
        lista_join = "".join(lista_final)
        print 'LISTA FINAL -->', lista_final
        print 'LISTA FINAL JOIN -->', lista_join
        tipo ='C'
        R_aleatorioRF1 = chr(int(hex(int(R_aleatorio[0:2])),16))
        R_aleatorioRF2 = chr(int(hex(int(R_aleatorio[2:])),16))
        R_aleatorioRF = R_aleatorioRF1 + R_aleatorioRF2
        print 'R ALEATORIO -->', R_aleatorio
        print 'HEX R ALEATORIO -->', R_aleatorioRF
        datosRF_sin = [R_aleatorioRF,lista_join]
        datosRF_join = "".join(datosRF_sin)
        print 'R aleatoria + lista -->', datosRF_join
        tam_L=[nodo,tipo,datosRF_join]
        join_L="".join(tam_L)
        L = chr(int(str(len(join_L))))
        mod_param_RF = [nodo,tipo]
        join_mod_param_RF = "".join(mod_param_RF)
        checkeo = join_mod_param_RF + datosRF_join
        print 'CHECKEO -->', checkeo
        checksumRF2(checkeo)
        print 'CHECKSUM RF -->', checkRF
        check_conv = check1RF + check2RF
        print 'CHECK_CONV -->', check_conv
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,nodo,tipo,datosRF_join,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'ENVIO A RF -->', completo


#funcion modificacion de alias UTR (GW --> antena RF)
def mod_aliasRF(s,nodo,datosRF):
        global envio_completo
        tipo ='C'
        alias_mod_aliasRF(datosRF)
        dat = envio_completo
        tam_L=[nodo,tipo,dat]
        join_L="".join(tam_L)
        L = chr(int(str(len(join_L))))
        mod_alias_RF = [nodo,tipo,dat]
        join_mod_alias_RF = "".join(mod_alias_RF)
        print 'INFORMACION CON EL QUE SE CALCULA EL CHECKSUMRF -->', join_mod_alias_RF
        checksumRF2(join_mod_alias_RF)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,nodo,tipo,dat,new_check,stop]
        join_completo = "".join(completo)
        #s.send(join_completo)
        print 'TRAMA ENVIADA A RF-->', join_completo
        print '-------------------------------------------------------------------------'
        print '                                modificacion de alias RF'
        print '-------------------------------------------------------------------------'

#funcion forzado de datos UTR (GW --> antena RF)
def forzado_datosRF(antena,nodo):
        tipo ='M'
        tam_L=[nodo,tipo]
        join_L="".join(tam_L)
        L = chr(int(str(len(join_L))))
        forzado_datos_RF = [nodo,tipo]
        join_forzado_datos_RF = "".join(forzado_datos_RF)
        checksumRF(join_forzado_datos_RF)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
```

```
        new_check = completa
        completo = [start,L,nodo,tipo,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'JOIN_COMPLETO -->', join_completo
        print '------------------------------------------------------------------------'
        print '                                forzado de datos RF'
        print '------------------------------------------------------------------------'

#funcion forzado de estadisticas UTR  (GW --> antena RF)
def forzado_estRF(antena,nodo):
        tipo ='E'
        tam_L=[nodo,tipo]
        join_L="".join(tam_L)
        L = chr(int(str(len(join_L))))
        forzado_est_RF = [nodo,tipo]
        join_forzado_est_RF = "".join(forzado_est_RF)
        checksumRF(join_forzado_est_RF)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,nodo,tipo,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'FORZADO DE ESTADISTICAS RF -->', join_completo
        print '------------------------------------------------------------------------'
        print '                                forzado de estadisticas RF'
        print '------------------------------------------------------------------------'

#funcion reset UTR (GW --> antena RF)
def resetRF(antena,nodo):
        global completa
        L = '\4'
        tipo = 'X'
        reset = [nodo,tipo]
        join_reset = "".join(reset)
        checksumRF(join_reset)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,nodo,tipo,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'RESET RF -->', join_completo
        print '------------------------------------------------------------------------'
        print '                                     RESET RF'
        print '------------------------------------------------------------------------'

#funcion peticion de IP UTR (GW --> antena RF)
def peticion_IP(antena,nodo,dats):
        global completa
        global lista
        print 'ENVIAMOS NUEVA IP A NODO'
        IP2 = nodo[0]
        print 'IP2-->', IP2
        IP1 = nodo[1]
        print 'IP1-->', IP1
        IP0 = nodo[2]
        print 'IP0-->', IP0
        L = '\x0C'
        tipo = 'i'
        datosRF = dats
        print 'datosRF (los que se les pasa a la funcion datos)', datosRF
        datos_to_hex(datosRF)
        com = completa
        datos_to_hex(IP2)
        IP2b = completa
        datos_to_hex(IP1)
        IP1b = completa
        datos_to_hex(IP0)
        IP0b = completa
        print 'IP2------>', IP2b
```

```
        print 'IP1------>', IP1b
        print 'IP0------>', IP0b
        print 'completa (lo que devuelve la funcion datos)', com
        peticion_IP = [IP2,IP1,IP0,tipo,datosRF]
        #peticion_IP = [IP2,IP1,IP0,tipo,com]
        join_peticion_IP ="".join(peticion_IP)
        print 'LO QUE SE LE PASA AL CHECKSUM------------>', join_peticion_IP
        checksumRF(join_peticion_IP)
        check_conv = check1RF+check2RF
        datos_to_hex(check_conv)
        new_check = completa
        #completo=[start,L,IP2b,IP1b,IP0b,tipo,com,new_check,stop]
        completo = [start,L,IP2,IP1,IP0,tipo,com,new_check,stop]
        join_completo = "".join(completo)
        print 'New_check-->', new_check
        print 'trama que se envia al nodo-->',join_completo
        antena.write(join_completo)
        print 'PETICION DE IP ENVIADA -->', join_completo
        print  '-------------------------------------------------------------------------------------
------'
        print '----------------------------SE HA ENVIADO IP NUEVA A NODO-----------------------
------'
        print  '-------------------------------------------------------------------------------------
------'

#funcion respuesta de IP a nodo (GW --> antena RF)
def respuest_IP(antena,nodo,nodon,mac):
        IP2n = nodon[0]
        IP1n = nodon[1]
        IP2n = nodon[2]
        IPHn = ['0',IP2n]
        joinIPHn = "".join(IPHn)
        IPLn = [IP1n,IP0n]
        joinIPLn = "".join(IPLn)
        IP2 = nodo[0]
        IP1 = nodo[1]
        IP0 = nodo[2]
        L = [IP2,IP1,IP0,'i',joinIPHn,joinIPLn,mac]
        joinL = "".join(L)
        longL = len(joinL)
        trama = [longL,IP2,IP1,IP0,'i',joinIPHn,joinIPLn,mac]
        checksumRF(joinL)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,longL,IP2,IP1,IP0,'i',IPHn,IPLn,mac,new_check,stop]
        envio = "".join(completo)
        antena.write(envio)
        print 'RESPUESTA IP -->', envio
        print '-------------------------------------------------------------------------'
        print '                                respuesta IP'
        print '-------------------------------------------------------------------------'

#funcion actuacion salidas UTR (GW --> antena RF)
def actuacion_salidasRF(antena,nodo,salida,valor):
        L = '\4'
        tipo = 'O'
        actuacion_salidas_RF = [nodo,tipo,salida,valor]
        join_actuacion_salidas_RF = "".join(actuacion_salidas_RF)
        checksumRF(join_actuacion_salidas_RF)
        check_conv = check1RF + check2RF
        datos_to_hex(check_conv)
        new_check = completa
        completo = [start,L,nodo,tipo,salida,valor,new_check,stop]
        join_completo = "".join(completo)
        antena.write(join_completo)
        print 'ACTUACION SALIDAS RF', join_completo
        print '-------------------------------------------------------------------------'
        print '                                        actuacion salidas RF'
        print '-------------------------------------------------------------------------'

#------------------------- FUNCIONES VARIAS --------------------------------
```

```
#funcion crea un fichero
def crea_fichero(nombre):
        arch = open(nombre,'w')
        arch.close

#funcion graba datos en un fichero existente con timestamp (recordar poner retorno de carro para
cada linea)
def graba_fichero(nombre, trama):
        arch = open(nombre, 'a')
        stamp()
        arch.write('TIMESTAMP: ')
        arch.write(timestampGW)
        arch.write('\n')
        arch.write(trama)
        arch.write('\n')

#funcion graba parametros en un .txt para ser consultado por otras funciones
def graba_fichero_parametros(nombre,trama):
        arch = open(nombre,'r')
        lineas = arch.readlines()
        arch.close()
        arch = open(nombre,'w')
        arch.write(trama)
        #arch.write('\n')

#funcion acceder a una linea de un txt obviando el timestamp
def lee_fichero(nombre):
        global linea
        arch = open(nombre, 'r')
        linea = arch.readlines()
        print linea
        arch.close()
        return linea

#funcion que comprueba que existe un archivo
def checkfile(nombre):
        global existe
        import os.path
        if os.path.exists(nombre):
                existe = 1
                return existe
        else:
                existe = 0
                return existe

#funcion analiza las tramas que modifican parametros y sensores y actualiza el archivo json
def actualiza_json(trama):
        f = open('variables_json2.json')
        gw = json.load(f)
        type(gw)
        gw.keys()
        lista_comas = []
        pos = -1
        try:
                while True:
                        pos = trama.index(';', pos+1)
                        lista_comas.append(pos)
        except ValueError:
                print 'NUMERO DE COMAS ENCONTRADAS-->', len(lista_comas)
                print 'POSICION DE LAS COMAS-->', lista_comas
        for i in range(len(lista_comas)-1):
                orden = trama[lista_comas[i]+1:lista_comas[i+1]]
                nombre = orden[0:3]
                valor = orden[3:]
                print 'NOMBRE-->', nombre
                print 'ORDEN-->', orden
                print 'VALOR-->', valor
                if 'D' in orden[0]:
                        clase = 'sensores digitales'
                        comp_num = orden[1]
                        numero_sensor = comp_num
                        if comp_num == 'A':
```

```
                numero_sensor = '10'
        if comp_num == 'B':
                numero_sensor = '11'
        if comp_num == 'C':
                numero_sensor = '12'
        if comp_num == 'D':
                numero_sensor = '13'
        if comp_num == 'E':
                numero_sensor = '14'
        if comp_num == 'F':
                numero_sensor = '15'
if 'C' in orden[0]:
        clase = 'comunicaciones'
        nombre = orden[0:3]
if 'A' in orden[0]:
        clase = 'sensores analogicos'
        numero_sensor = orden[1]
if 'SN' in orden:
        val = -1
        if 'cont' in orden:
                val = orden.index('cont', val+1)
                nombre = orden[0:val]
                print 'NOMBRE EXCEPCIONAL-->', nombre
                valor = orden[val:]
                print 'VALOR EXCEPCIONAL-->', valor
                tamanyo_nombre = len(nombre)

                if tamanyo_nombre <= 3:
                        clase = 'sensores digitales'
                        numero_sensor = nombre[2]
                else:
                        nombre2 = nombre[2:]
                        nombre_int = int(nombre2)

                        if nombre_int <=14:
                                clase = 'sensores digitales'
                                numero_sensor = nombre2
                        else:
                                clase = 'sensores analogicos'
                                numero_sensor = str(nombre_int-16)
        else:
                val = orden.index('d', val+1)
                nombre = orden[0:val]
                print 'NOMBRE EXCEPCIONAL-->', nombre
                valor = orden[val:]
                print 'VALOR EXCEPCIONAL-->', valor

                if len(nombre) <= 3:
                        clase = 'sensores digitales'
                        nombre_sensor = nombre[2]
                else:
                        nombre2 = nombre[2:]
                        nombre_int = int(nombre2)
                        numero_sensor = nombre2
                        if nombre_int <=14:
                                clase = 'sensores digitales'
                                numero_sensor = nombre2
                        else:
                                clase = 'sensores analogicos'
                                numero_sensor = str(nombre_int-16)
f.close()
if clase == 'comunicaciones':
        nombre = orden[0:3]
        gw['gateway'][clase][nombre] = valor
        f = open('variables_json2.json', 'w')
        json.dump(gw,f)
        f.close
        print 'ORDEN-->',orden
        print 'NOMBRE-->', nombre
        print 'VALOR-->', valor
else:
        gw['gateway'][clase][numero_sensor][nombre] = valor
```

```
                        f = open('variables_json2.json', 'w')
                        json.dump(gw,f)
                        f.close()
                        print 'ORDEN-->',orden
                        print 'NOMBRE-->', nombre
                        print 'NUMERO DE SENSOR-->', numero_sensor
                        print 'VALOR-->', valor

#funcion consulta en json (valores de comunicaciones) y devuelve este valor
def consulta_json(archivo_json,dato):
        global gate
        archivo_json_str = str(archivo_json)
        f = open(archivo_json_str)
        gw = json.load(f)
        type(gw)
        gw.keys()
        f.close()
        dato_str = str(dato)
        gate = gw['gateway']['comunicaciones'][dato]
        return(gate)

#funcion que comprueba si un .txt es vacio o continiene informacion print 'file is empty' si este
esta vacio
def comprueba(fpath):
        with open(fpath) as my_file:
                my_file.seek(0)
                first_char = my_file.read(1)
                if not first_char:
                        print "file is empty"
                else:
                        my_file.seek(0)
                return True if os.path.isfile(fpath) and os.path.getsize(fpath) > 0 else False

#funcion para mod_paramRF (convierte datos en formato correcto para envio)
#la estructura de datos que se le pasa es como : datos = ';PARvalor;PARvalor;'
def parametros_mod_paramRF(datos):
        global envio_completo
        pos = -1
        lista = []
        param = []
        val = []
        completo = []
        try:
                while True:
                        pos = datos.index(';',pos+1)
                        lista.append(pos)
                        print 'lista-->', lista
        except ValueError:
                print 'ok'
        for i in range(len(lista)-1):
                if i == 0:
                        if (lista[i+1] - lista[i]) < 2:
                                print 'NO HAY PARAMETRO', i
                        else:
                                parametro = datos[lista[i]+1:lista[i]+4]
                        param.append(parametro)
                                if len(datos[lista[i]+3:lista[i+1]]) < 2:
                                        print 'NO HAY VALOR', i
                                else:
                                valor = datos[lista[i]+4:lista[i+1]]
                                valor_bin = bin(int(valor))
                        val.append(valor_bin)

                else:
                        if (lista[i+1] - lista[i]) < 2:
                                print 'NO HAY PARAMETRO', i
                        else:
                                parametro = datos[lista[i]+1:lista[i]+4]
                        param.append(parametro)
                                if len(datos[lista[i]+3:lista[i+1]]) < 2:
                                        print 'HO HAY VALOR', i
                                else:
```

```
                                valor = datos[lista[i]+4:lista[i+1]]
                            valor_bin = bin(int(valor))
                            val.append(valor_bin)
        print 'PARAM-->',param
        print 'VAL-->', val

        for i in range(len(val)):
                envio = param[i] + val[i]
                completo.append(envio)

        print 'COMPLETO-->', completo
        print 'LISTA-->',lista
        envio_completo = "".join(completo)
        print 'ENVIO COMPLETO -->', envio_completo
        return(envio_completo)

#funcion convierte un string en formato hex para envio del
def datos_to_hex(mac):
        global completa
        global lista
        datos = mac
        lista=[]
        tam = len(datos)
        for i in range(tam):
                if i == 0:
                        a = hex(int(datos[i:i+2],16))
                        b = chr(int(a,16))
                        lista.append(b)
                        print 'LISTA -->', lista
                else :
                        if i%2 == 0:
                                a = hex(int(datos[i:i+2],16))
                                b = chr(int(a,16))
                                lista.append(b)
                                print 'LISTA-->', lista
        completa = "".join(lista)
        print 'COMPLETO-->', completa
        return lista, completa

#funcion que configura el GPIO como salida y que pin (en modo BCM) ejemp: salida_GPIO(17,'alto')
#Cuando se deja de utilizar hacer un GPIO.cleanup() para limpiar valores del GPIO
def salida_GPIO(numero, nivel):
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(numero,GPIO.OUT)
        if nivel == 'alto':
                GPIO.output(numero, GPIO.HIGH)
        if nivel == 'bajo':
                GPIO.output(numero,GPIO.LOW)

def alias_mod_aliasRF(datos):
        pos = -1
        global envio_completo
        lista = []
        param = []
        val = []
        completo = []
        try:
                while True:
                        pos = datos.index(';', pos+1)
                        lista.append(pos)
                print 'lista-->', lista
        except ValueError:
                print 'ok'
        for i in range(len(lista)-1):
                if i == 0:
                        if (lista[i+1] - lista[i]) < 2:
                                print 'NOO hay parametro',i
                        else:
                                parametro = datos[lista[i]+1:lista[i]+4]
                                param.append(parametro)
                                if len(datos[lista[i]+3:lista[i+1]]) < 2:
                                        print 'NO HAY VALOR', i
```

```
                                else:
                                        valor = datos[lista[i]+4:lista[i+1]]
                                        val.append(valor)
                                        print 'PARAMA 0 -->', param
                                        print 'VAL 0 -->', val

                        else:
                                if (lista[i+1] - lista[i]) < 2:
                                        print 'NOOO HAY PARAMETRO', i
                                else:
                                        parametro = datos[lista[i]+1:lista[i]+4]
                                        param.append(parametro)
                                        if len(datos[lista[i]+3:lista[i+1]]) < 2:
                                                print 'NO HAY VALOR', i
                                        else:
                                                valor = datos[lista[i]+4:lista[i+1]]
                                                val.append(valor)
                print 'PARAM-->',param
                print 'VAL-->', val
                for i in range(len(val)):
                        envio = param[i] + val[i]
                        completo.append(envio)

                print 'COMPLETO-->', completo
                print 'LISTA-->',lista
                print 'tamanyo lista-->', len(lista)
                envio_completo = "".join(completo)
                print 'ENVIO COMPLETO -->',envio_completo
                return(envio_completo)

#checksumRF 2
def checksumRF2(checkeo):
        global check1RF
        global check2RF
        global checkRF
        checkRF = 0
        pre_checkRF = 0
        datos_check = 0
        ip_tipo = checkeo[0:4]
        datos = checkeo[4:]
        print 'IP_TIPO-->',ip_tipo
        print 'DATOS -->',datos
        tami = len(ip_tipo)
        print 'TAMI', tami
        tam = len(datos)
        print 'TAMAYO', tam
        for i in range(tami):
                a = ord(ip_tipo[i])
                pre_checkRF = pre_checkRF+a
                print 'pre_checkRF',pre_checkRF
                print 'ip_tipo 1', a
        if tam > 1:
                for i in range(tam):
                        b = ord(datos[i])
                        datos_check = datos_check + b
                        print 'datos check-->', datos_check
                        print 'B-->',b
        check1 = pre_checkRF + datos_check
        check = hex(check1)
        #check = check.upper()
        print 'CHECK-->', check
        tamanyo = len(check)
        check1RF = check[tamanyo-2]
        print 'CHECKSUM1RF-->', check1RF
        check2RF = check[tamanyo-1]
        print 'CHECKSUM2RF-->', check2RF
        return checkRF,check1RF,check2RF

#Funciones para tratamiento de json de la red de nodos
def consulta_red(archivo_json,dato,dato2):
        global result
        archivo_json_str = str(archivo_json)
```

```
        f = open(archivo_json_str)
        red = json.load(f)
        type(red)
        red.keys()
        f.close()
        dato_str = str(dato)
        if dato2 == 0:
                result = red['red']['nodo1'][dato]
        else:
                result = red['red']['nodo1'][dato][dato2]
        return(result)

#Coloca un 0 en el campo ACTIVO del json de info_nodos para eliminarlo
def eliminar_nodo_json(nodo):
        archivo_json_str = 'json.json'
        f = open(archivo_json_str)
        red = json.load(f)
        type(red)
        red.keys()
        f.close()
        for i in range(10):
                if i != 0:
                        result = red['red']['nodo%d' %(i)]['IP']
                        if result == nodo:
                                red['red']['nodo%d' %(i)]['ACTIVO'] = '0'
                                f = open(archivo_json_str,'w')
                                json.dump(red,f)
                                f.close()
                                break

#funcion que introduce nuevo nodo en json con los parametros de nodo y mac
def new_utr(nodo,mac):
        archivo_json_str = 'json.json'
        f = open(archivo_json_str)
        red = json.load(f)
        type(red)
        red.keys()
        f.close()
        for i in range(10):
                if i == 0:
                        print 'ES 0'
                else:
                        result = red['red']['nodo%d' %(i)]
                        print 'RESULT -->', result
                        resultado_activo = red['red']['nodo%d' %(i)]['ACTIVO']
                        if resultado_activo == '0':
                                print 'ACTIVAMOS EL NODO%d' %(i+1)
                                print result
                                mac = str(mac)
                                nodo = str(nodo)
                                red['red']['nodo%d' %(i)]['mac'] = mac
                                red['red']['nodo%d' %(i)]['IP'] = nodo
                                red['red']['nodo%d' %(i)]['ACTIVO'] = '1'
                                f = open(archivo_json_str,'w')
                                json.dump(red,f)
                                f.close()
                                break

#funcion que actualiza el json de nodos
def actualiza_alias_nodos(lista_TS,lista_AS,nodo,AN,SEN,TNO):
        archivo = 'json.json'
        f = open(archivo)
        red = json.load(f)
        type(red)
        red.keys()
        f.close()
        for i in range(15):
                if i != 0:
                        result1 = red['red']['nodo%d' %(i)]['ACTIVO']
                        result2 = red['red']['nodo%d' %(i)]['IP']
                        if result1 == '1' and result2 == nodo:
                                print 'NODO -->', result2
```

```
print 'ACTIVO -->', result1
red['red']['nodo%d' %(i)]['AN'] = AN
red['red']['nodo%d' %(i)]['SEN'] = SEN
red['red']['nodo%d' %(i)]['TNO'] = TNO
red['red']['nodo%d' %(i)]['sensores']['TS1'] = lista_TS[0]
red['red']['nodo%d' %(i)]['sensores']['AS1'] = lista_AS[0]
red['red']['nodo%d' %(i)]['sensores']['TS2'] = lista_TS[1]
red['red']['nodo%d' %(i)]['sensores']['AS2'] = lista_AS[1]
red['red']['nodo%d' %(i)]['sensores']['TS3'] = lista_TS[2]
red['red']['nodo%d' %(i)]['sensores']['AS3'] = lista_AS[2]
red['red']['nodo%d' %(i)]['sensores']['TS4'] = lista_TS[3]
red['red']['nodo%d' %(i)]['sensores']['AS4'] = lista_AS[3]
red['red']['nodo%d' %(i)]['sensores']['TS5'] = lista_TS[4]
red['red']['nodo%d' %(i)]['sensores']['AS5'] = lista_AS[4]
red['red']['nodo%d' %(i)]['sensores']['TS6'] = lista_TS[5]
red['red']['nodo%d' %(i)]['sensores']['AS6'] = lista_AS[5]
f = open(archivo,'w')
json.dump(red,f)
f.close()
break
```

8.2.2. Analiza.py

```
#Script que contiene las funciones para Gateway Low Power
import time
import json
import datetime
import socket
from funciones import alias_utr
from funciones import lectura_parametrosRF
from funciones import actualiza_alias_nodos
from funciones import mod_alias_utr
from funciones import mod_param_utr
from funciones import eliminar_utr
from funciones import new_utr
from funciones import eliminar_utr
from funciones import mod_aliasRF
from funciones import resetRF
from funciones import eventos_utr
from funciones import modificacion_alias
from funciones import estadistica_utr
from funciones import alarma_umbral_utr
from funciones import cambio_salidas_nodo
from funciones import actuacion_salidasRF
from funciones import reset_utr
from funciones import forzado_estRF
from funciones import actualiza_json
from funciones import mod_paramRF
from funciones import lectura_alias
from funciones import estructura_red
from funciones import cambio_estados
from funciones import bateria
from funciones import lectura_parametros
from funciones import Reset
from funciones import mod_parametros
from funciones import stamp
from funciones import checksum
from funciones import alive
from funciones import lee_fichero
from funciones import lectura_utr
from funciones import peticion_ip_utr
from funciones import peticion_IP
from funciones import graba_fichero_parametros
from funciones import checkfile
from funciones import crea_fichero
from funciones import forzado_datosRF
global nodo
global datos
global linea
global check
global check1
global check2
global nombre_GW
global identGW1
global identGW2
global identGW3
global identGW4
global timestampGW
global checksum_server1
global checksum_server2
global valor_bat
global nevento
global checkRF
global check1RF
global check2RF
#definimos start y stop
start ='\2'
stop = '\5'

#identificador GW
identGW1 = '0'
```

```
identGW2 = '0'
identGW3 = '0'
identGW4 = '3'
nombre_GW = 'TEST'

#Direccion, puerto y buffer
TCP_IP = '158.42.105.71'    #'www.test.balmart.es'
TCP_PORT = 6011
PORT = '6011'
BUFFER_SIZE=1024

#definimos lista de comandos (envio del GW al servidor)
#s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #constructor del socket familia IPv prot.
flujo
#s.connect((TCP_IP,TCP_PORT))

#timestamp
def stamp():
        global timestampGW
        ts = time.time()
        timestampGW = datetime.datetime.fromtimestamp(ts).strftime('%H%M%S%d%m%Y')
        return timestampGW

#def checksum(trama):
def checksum(trama):
        global check
        global check1
        global check2
        check = 0
        pre_check = 0
        for i in trama:
                a=ord(i)
                pre_check = pre_check^a
        check = hex(pre_check)
        print 'check -->', check
        if len(check) < 4:
                check = check.upper()
                check1 = 0
                print 'check1 -->', check1
                check2 = check[2]
                print 'check2 -->', check2
        else:
                check = check.upper()
                print 'check.upper -->', check
                check1 = check[2]
                print 'check1 -->', check1
                check2 = check[3]
                print 'check2 -->', check2
        return check,check1, check2

#funcion que comprueba que existe un archivo
def checkfile(nombre):
        global existe
        import os.path
        if os.path.exists(nombre):
                existe = 1
                return existe
        else:
                existe = 0
                return existe

#funcion analizador de tramas  // Analiza(trama)
def Analiza(s,antena,trama):
        global check
        global timestampGW
        global checksum_server
        global lista_start
        global lista_stop
        lista_start=[]
        lista_stop=[]
        pos_inicial_start=-1
        pos_inicial_stop=-1
```

```
print 'ANALIZAMOS LA SIGUIENTE TRAMA-->>', trama
if  '\x06' in trama:
        print 'TODO OK!!!!!!!!!!'
if '\x0f' in trama:
        print 'MAAAAAAAAAAL ??????'
try:
        while True:
                pos_inicial_start=trama.index('\2',pos_inicial_start+1)
                lista_start.append(pos_inicial_start)
except ValueError:
        print 'numero de start encontrados-->',len(lista_start)
try:
        while True:
                pos_inicial_stop=trama.index('\5', pos_inicial_stop+1)
                lista_stop.append(pos_inicial_stop)
except ValueError:
        print 'numero de stop encontrados-->', len(lista_stop)

for i in range(len(lista_start)):
        trama_ind = trama[lista_start[i]:lista_stop[i]+1]
        print 'TRAMA_IND-->',trama_ind
        print 'TAMANYO TRAMA_IND', len(trama_ind)
        pos_start = trama_ind.find('\2')
        pos_stop = trama_ind.find('\5')
        if pos_start == -1 or pos_stop ==-1:
                print 'TRAMA INCOMPLETA'
        else:
                sec_interes = trama_ind[pos_start+1:pos_stop]
                print 'SECCION DE INTERES-->', sec_interes
                if sec_interes[0] == 'i' or sec_interes[0] == 'h':
                        if sec_interes[0] == 'i':
                                idGW1 = sec_interes[1]
                                idGW2 = sec_interes[2]
                                idGW3 = sec_interes[3]
                                idGW4 = sec_interes[4]
                                nodo_old = sec_interes[6:9]
                                nodo_new = sec_interes[9:12]
                                mac = sec_interes[12:24]
                                print 'DATOS SOBRE IP RECIBIDOS DE SERVER'
                                print 'MAC (desde server)-->', mac
                                print 'NODO OLD -->', nodo_old
                                print 'NODO NEW -->', nodo_new
                                IPnewH_pre = hex(int(nodo_new[0]))
                                IPnewH_pre_f = IPnewH_pre.lstrip('0x')
                                IPnewH = IPnewH_pre_f.zfill(2)
                                IPnewL_pre = hex(int(nodo_new[1:]))
                                IPnewL_pre_f = IPnewL_pre.lstrip('0x')
                                IPnewL = IPnewL_pre_f.zfill(2)
                                mac_com = []
                                mac1 = hex(int(mac[0:2],16))
                                mac1m = mac1[2:]
                                mac1mz = mac1m.zfill(2)
                                mac_com.append(mac1mz)
                                mac2 = hex(int(mac[2:4],16))
                                mac2m = mac2[2:]
                                mac2mz = mac2m.zfill(2)
                                mac_com.append(mac2mz)
                                mac3 = hex(int(mac[4:6],16))
                                mac3m = mac3[2:]
                                mac3mz = mac3m.zfill(2)
                                mac_com.append(mac3mz)
                                mac4 = hex(int(mac[6:8],16))
                                mac4m = mac4[2:]
                                mac4mz = mac4m.zfill(2)
                                mac_com.append(mac4mz)
                                mac5 = hex(int(mac[8:10],16))
                                mac5m = mac5[2:]
                                mac5mz = mac5m.zfill(2)
                                mac_com.append(mac5mz)
                                mac6 = hex(int(mac[10:12],16))
                                mac6m = mac6[2:]
```

```
                                        mac6mz = mac6m.zfill(2)
                                        mac_com.append(mac6mz)
                                        mac_final = "".join(mac_com)
                                        datos = []
                                        datos.append(IPnewH)
                                        datos.append(IPnewL)
                                        datos.append(mac_final)
                                        print 'MAAAAAAAAAAAAAAAC--->>', mac_final
                                        completa = "".join(datos)
                                        print 'NODO_OLD----------->>>', nodo_old
                                        print 'COMPLETA------------>><', completa
                                        print 'RECIBIDA IP DEL SERVER'
                                        peticion_IP(antena,nodo_old,completa)
                                        new_utr(nodo_new,mac)

                        if sec_interes[0] == 'h':
                                global timestampGW
                                stamp()
                                        print 'RESPUESTA SERVIDOR ALIVE-->', sec_interes
                                timestamp_server = sec_interes[5:19]
                                print 'TIMESTAMP_SERVER-->', timestamp_server
                                print 'TIMESTAMP GATEWAY-->', timestampGW
                                if timestamp_server != timestampGW:
                                                timestampGW = timestamp_server
                                        print '-----------TIMESTAMP CAMBIADO----------'
                                        print 'NUEVO TIMESTAMPGW-->', timestampGW
                                else:
                                        print 'TIMESTAMP CORRECTO'
                        else:
                                comando = sec_interes[2]
                                print 'COMANDO LEIDO-->',comando
                                idGW1 = sec_interes[3]
                                idGW2 = sec_interes[4]
                                idGW3 = sec_interes[5]
                                idGW4 = sec_interes[6]
                                print 'IDENTIFICADOR DEL GW-->', idGW1,idGW2,idGW3,idGW4

                                if idGW1 != identGW1 and idGW2 != identGW2 and idGW3 != identGW3
and idGW4 != identGW4:
                                        print 'IDENTIFICADOR ERRONEO'

                                else:
                                        print 'IDENTIFICADOR OK'
                                        global checksum_server1
                                        global checksum_server2
                                        tam_total=len(sec_interes)
                                        checksum_server1 = sec_interes[tam_total-2:tam_total-1]
                                        checksum_server2 = sec_interes[tam_total-1:tam_total]
                                        print     'CHECKSUM     SERVER-->',     checksum_server1,
checksum_server2
                                        checkeo = sec_interes[0:tam_total-2]
                                        checksum(checkeo)
                                        print 'CHECKSUM ANALIZADO-->', check1,check2

                                        if  checksum_server1  !=  check1  and  checksum_server2  !=
check2:
                                                print '-----------ERROR CHECKSUM----------'
                                        else:
                                                print 'CHECKSUM OK!'
                                                if comando == 'L':
                                                        pos_N = sec_interes.find('N')
                                                        if pos_N >=0:
                                                                nodo                            =
sec_interes[pos_N+1:pos_N+4]

        lectura_parametrosRF(antena,nodo)
                                                        else:
                                                        lectura_parametros(s)

                                                if comando == 'P':
                                                        pos_N = sec_interes.find('N')
                                                        if pos_N >=0:
```

```
                                                        nodo                            =
sec_interes[pos_N+1:pos_N+4]
                                                        alias_utr(s,nodo)
                                                else:
                                                        lectura_alias(s,nombre_GW)

                                        if comando == 'N':
                                                lista_TS = []
                                                lista_AS = []
                                                pos_G=sec_interes.find('G')
                                                if pos_G >= 0:
nombre_mod=sec_interes[pos_G+1:tam_total-3]
                                                print 'NUEVO NOMBRE-->', nombre_mod
                                                modificacion_alias(s,nombre_mod)
                                                global nombre_GW
                                                nombre_GW = nombre_mod
                                                else:
                                                        lista_comas = []
                                                        sec_interes = str(sec_interes)
                                                        pos_comas = -1
                                                        pos_D = sec_interes.find('D')
                                                        nodo                            =
sec_interes[pos_D+1:pos_D+4]
                                                        print 'nodo-->', nodo
                                                        try:
                                                                while True:
                                                                        pos_comas       =
sec_interes.index(';',pos_comas+1)
        lista_comas.append(pos_comas)
                                                        except ValueError:
                                                                print 'numero de comas',
len(lista_comas)
                                                                print  'lista  de  comas
encontradas', lista_comas
                                                        AN                              =
sec_interes[pos_D+4:lista_comas[0]]
                                                        print 'AN-->', AN
                                                        TNO                             =
sec_interes[lista_comas[0]+1]
                                                        print 'TNO-->', TNO
                                                        SEN                             =
sec_interes[lista_comas[0]+2:lista_comas[0]+5]
                                                        print 'SEN-->', SEN
                                                        print      'LISTA      COMAS-->',
lista_comas
                                                        for i in range(len(lista_comas)-
1):
                                                                if  i == 0:
                                                                        if
lista_comas[i+1] - lista_comas[i] > 2:
                                                                                TS      =
sec_interes[lista_comas[i]+5:lista_comas[i]+8]
                                                                                AS      =
sec_interes[lista_comas[i]+8:lista_comas[i+1]]
        lista_TS.append(TS)
        lista_AS.append(AS)
                                                                        else:
                                                                                Ts = 0
                                                                                AS = 0

        lista_TS.append(TS)
        lista_AS.append(AS)
                                                                else:
                                                                        if
lista_comas[i+1] - lista_comas[i] > 2:
```

```
                                                                        TS      =
sec_interes[lista_comas[i]+1:lista_comas[i]+4]
                                                                        AS      =
sec_interes[lista_comas[i]+4:lista_comas[i+1]]

        lista_TS.append(TS)

        lista_AS.append(AS)
                                                                else:
                                                                        TS = 0
                                                                        AS = 0

        lista_TS.append(TS)

        lista_AS.append(AS)
                                                        print 'LISTA TS -->', lista_TS
                                                        print 'LISTA AS -->', lista_AS

        actualiza_alias_nodos(lista_TS,lista_AS,nodo,AN,SEN,TNO)
                                                        mod_alias_utr(s,sec_interes)

                                                if comando == 'M':
                                                        global R_aleatorio
                                                        pos_R = sec_interes.find('IDT')
                                                        R_aleatorio                     =
sec_interes[pos_R+3:pos_R+7]
                                                        print 'NUMERO ALEATORIO -->', R_aleatorio
                                                        pos_N = sec_interes.find('N')
                                                        if pos_N >= 0 and pos_N <= 14:
                                                                nodo                    =
sec_interes[pos_N+1:pos_N+4]
                                                                datosRF                 =
sec_interes[pos_N+12:len(sec_interes)-2]
                                                                print 'DATOS -->', datosRF

        mod_paramRF(antena,nodo,datosRF,R_aleatorio)
                                                        else:
                                                                trama_mod               =
sec_interes[pos_R+7:]
                                                                lista_comas=[]
                                                                pos = -1
                                                                try:

                                                                while True:
                                                                        pos             =
trama_mod.index(';', pos+1)

lista_comas.append(pos)
                                                                except ValueError:
                                                                        print 'NUMERO DE
COMAS ENCONTRADAS-->', len(lista_comas)
                                                                print 'POSICION DE LAS
COMAS-->', lista_comas
                                                                trama_mod_param         =
trama_mod[1:lista_comas[len(lista_comas)-1]]
                                                                print   'TRAMA   PARA   FUNCION
CONTESTA SERVIDOR', trama_mod_param
                                                                actualiza_json(trama_mod)
                                                                print 'TRAMA PARA MOD_PARAMETROS-
->',trama_mod

        mod_parametros(s,trama_mod_param,R_aleatorio)
                                                                checkfile('parametros.txt')
                                                                if existe == 0:

        crea_fichero('parametros.txt')

        graba_fichero_parametros('parametros.txt',trama_mod_param)

                                                if comando == 'R':
                                                        pos_N = sec_interes.find('N')
                                                        if pos_N >= 0:
```

```
                                                        nodo                            =
sec_interes[pos_N+1:pos_N+4]
                                                        resetRF(antena,nodo)
                                                else:
                                                Reset(s)

                                        if comando == 'A':
                                                pos_N = sec_interes.find('N')
                                                if pos_N >= 0:
                                                        nodo                            =
sec_interes[pos_N+1:pos_N+4]
                                                        forzado_datosRF(antena,nodo)
                                                        print '-------- forzado de datos
RF pedido-------'

                                                else:
                                                lectura_parametros(s)
                                                print   '..........lectura   forzada   de
datos pedida..........'

                                        if comando == 'E':
                                                pos_N = sec_interes.find('N')
                                                if pos_N >= 0:
                                                        nodo                            =
sec_interes[pos_N+1:pos_N+4]
                                                        forzado_estRF(antena,nodo)
                                                else:
                                                bateria(s)


                                        if comando == 'O':
                                                pos_N = sec_interes.find('N')
                                                if pos_N >= 0:
                                                        nodo                            =
sec_interes[pos_N+1:pos_N+4]
                                                        salida = sec_interes[pos_N+5]
                                                        valor = sec_interes[pos_N+6]

        actuacion_salidasRF(antena,nodo,salida,valor)

        cambio_salidas_nodo(s,nodo,salida,valor)

                                                else:
                                                        salida = sec_interes[8]
                                                        valor = sec_interes[9]
                                                cambio_estados(s,salida,valor)

                                        if comando == 'T':
                                                estructura_red(s)

                                        if comando == 'K':
                                                pos_01 = sec_interes.find('01')
                                                nodo = sec_interes[pos_01+2:pos_01+5]
                                                mac = sec_interes[pos_01+5:pos_01+17]
                                                eliminar_utr(s,nodo,mac)


def analizaRF(s,antena,trama):
        trama_join = "".join(trama)
        trama = trama_join
        global lista_start_RF
        global lista_stop_RF
        pos_inicial_stop_RF=-1
        pos_inicial_start_RF=-1
        lista_start_RF=[]
        lista_stop_RF=[]
        lista_def = []
        lista_def_def = []
        lista_def_def_def = []
        flag = 0
        lista_stop_2 = []
        lista_start_2 = []
```

```
        lista_dupla = []
        pos_inicial_dupla = -1
        lista_start_3 = []
        lista_stop_3 = []
        lista_pos_stop_RF = []
        pos_stop_RF = -1
        print 'TRAMA RF QUE SE PARSEA--------->>', trama

        try:
                while True:
                        pos_inicial_start_RF = trama.index('\2',pos_inicial_start_RF+1)
                        lista_start_RF.append(pos_inicial_start_RF)
        except ValueError:
                print 'numero de start RF encontrados-->', len(lista_start_RF)

        try:
                while True:
                        pos_inicial_stop_RF=trama.index('\5',pos_inicial_stop_RF+1)
                        lista_stop_RF.append(pos_inicial_stop_RF)
        except ValueError:
                print 'numero de stop RF encontrados-->', len(lista_stop_RF)

#---------------NUEVO PARSEADOR DE TRAMAS RF-----------------------------------------
        try:
                while True:
                        pos_inicial_dupla = trama.index('\5\2', pos_inicial_dupla +1)
                        lista_dupla.append(pos_inicial_dupla)
        except ValueError:
                print 'numero de duplas encontradas-->', len(lista_dupla)


        print 'lista_start_RF-->', lista_start_RF
        print 'lista_stop_RF-->', lista_stop_RF
        print 'lista dupla -->', lista_dupla
        if len(lista_dupla) != 0:
                for i in range(len(lista_dupla)):
                                lista_start_2.append(lista_dupla[i]+1)
                                lista_stop_2.append(lista_dupla[i])

                if (lista_stop_RF[len(lista_stop_RF)-1]) != (lista_dupla[len(lista_dupla)-1]):
                        lista_stop_2.append(lista_stop_RF[len(lista_stop_RF)-1])

                if (lista_start_RF[0]) != (lista_dupla[0]+1):
                        lista_start_2.insert(0,lista_start_RF[0])

                print 'lista_start_2 -->', lista_start_2
                print 'lista_stop_2 -->', lista_stop_2
        else:
                lista_start_2.append(lista_start_RF[0])
                lista_stop_2.append(lista_stop_RF[len(lista_stop_RF)-1])

        for i in range(len(lista_start_2)):
                if lista_start_2[0] > lista_stop_2[0]:
                        lista_stop_2[0] = 0
        for i in range(len(lista_stop_2)):
                if lista_stop_2[i] != 0:
                        lista_stop_3.append(lista_stop_2[i])

        for i in range(len(lista_stop_2)):
                if lista_stop_2[len(lista_stop_2)-1] < lista_start_2[len(lista_start_2)-1]:
                        lista_start_2[len(lista_start_2)-1] = 0
        for i in range(len(lista_start_2)):
                if i == 0:
                        lista_start_3.append(lista_start_2[i])
                else:
                        if lista_start_2[i] != 0:
                                lista_start_3.append(lista_start_2[i])
        print 'lista_start_3 -->', lista_start_3
        print 'lista_stop_3 -->', lista_stop_3
#-------------------------------------------------------------------------------------------
        if '\2' in trama and '\5' in trama:
                print 'ENTRO EN EL PRIMER IF'
```

```
                for i in range(len(lista_start_3)):
                        trama_ind_RF = trama[lista_start_3[i]:lista_stop_3[i]+1]
                        print 'trama_ind_rf -->', trama_ind_RF
                        print 'tamanyo trama_ind_rf -->', len(trama_ind_RF)
                        pos_start_RF = trama_ind_RF.find('\x02')
                        try:
                                while True:
                                        pos_stop_RF = trama_ind_RF.index('\x05',pos_stop_RF+1)
                                        lista_pos_stop_RF.append(pos_stop_RF)
                        except ValueError:
                                print 'pos_stop_RF-->', lista_pos_stop_RF
                        pos_stop_RF = lista_pos_stop_RF[len(lista_pos_stop_RF)-1]
#---------------------------
                        if ('\2' in trama) == False or ('\5' in trama) == False :
                        print 'TRAMA RF INCOMPLETA'
                        if len(trama) > 7:
                                print '------------------------posicion start-->', pos_start_RF
                                print '------------------------posicion stop -->', pos_stop_RF
                                if (pos_stop_RF - pos_start_RF) > 6:
                                sec_interes_RF = trama_ind_RF[pos_start_RF+1:pos_stop_RF]
                                print 'SECCION DE INTERES RF-->', sec_interes_RF
                                print 'TAMANYO L-->', sec_interes_RF[0]
                                nodo = sec_interes_RF[1:4]
                                print 'NODO', nodo
                                L = sec_interes_RF[0]
                                print 'L-->', ord(L)
                                tipo_RF = sec_interes_RF[4]

                                        if tipo_RF == 'W':
                                                print 'ENTRO EN TIPO RF W -->', sec_interes_RF
                                                print    'tamanyo    de    sec_interes_RF    -->',
len(sec_interes_RF)
                                                nodo = sec_interes_RF[1:4]
                                                datos = sec_interes_RF[5:len(sec_interes_RF)-1]
                                                if '\2' in datos:
                                                        print 'DETECTO EL 0X02 Y LEO COBERTURA Y
BATERIA'
                                                        print 'DATOS -->',datos
                                                        print 'TAMANYO DATOS -->', len(datos)
                                                        cobertura = datos[3]
                                                        bateria1 = datos[4]
                                                        bateria2 = datos[5]

        estadistica_utr(s,nodo,cobertura,bateria1,bateria2)
                                                        print 'tipo --> W'
                                                        print 'Evento de estadisticas y bateria
por RF'
                                                else:
                                                        codigo = datos[0]
                                                        sensor = datos[1]
                                                        valor1 = datos[2]
                                                        valor2 = datos[3]

        alarma_umbral_utr(s,nodo,codigo,sensor,valor1,valor2)
                                                        print 'tipo --->  W '
                                                        print 'Evento de alarma/umbral'

                                        if tipo_RF == 'P':
                                                print 'tipo_RF--> P'

                                        if tipo_RF == 'X':
                                                nodo = sec_interes_RF[1:4]
                                                reset_utr(s,nodo)
                                                print 'tipo --->  X'
                                                print 'REset de utr'

                                        if tipo_RF == 'L':
                                                print 'TIPO RF--> L'
                                                nodo = sec_interes_RF[1:4]
                                                print 'COMPROBACION NODO-->', nodo
                                                datos = sec_interes_RF[5:len(sec_interes_RF)-1]
                                                print 'DATOS DESDE UTR-->', datos
```

```
                                        lectura_utr(s,nodo,datos)


                                if tipo_RF == 'C':
                                        print 'TIPO RF RECIBIDO C'
                                        pos_R = sec_interes_RF.find('C')
                                        longitud_datos                                  =
len(sec_interes_RF[pos_R+1:len(sec_interes_RF)-1])
                                        nodo = sec_interes_RF[1:4]
                                        if longitud_datos > 2:
                                                R_aleatorio                             =
sec_interes_RF[pos_R+1:pos_R+3]
                                                NumParams                               =
sec_interes_RF[pos_R+3:len(sec_interes_RF)-2]
                                                print   'RECIBIDO   DATO   RF   C   -->',
sec_interes_RF
                                                print 'IFIFIFIIFIFIFFIIFIFIFIFIFIFIFI'
                                                mod_param_utr(s,nodo,R_aleatorio)
                                        else:
                                                print 'ELSEELSEELSEELSEELSEESLE'
                                                R_aleatorio                             =
sec_interes_RF[pos_R+1:pos_R+3]
                                                mod_param_utr(s,nodo,R_aleatorio)

                                if tipo_RF == 'O':
                                        print 'tipo_RF--> O'

                                if tipo_RF == 'D':
                                        print 'tipo RF --> D'
                                        print 'EVENTO DE DATOS'
                                        nodo = sec_interes_RF[1:4]
                                        print 'NODO--->', nodo
                                        pos_D = sec_interes_RF.find('D')
                                        sensor = sec_interes_RF[pos_D+1]
                                        if sensor == '\x01':
                                                sensor ='0'
                                        if sensor == '\x02':
                                                sensor = '1'
                                        if sensor == '\x04':
                                                sensor = '2'
                                        if sensor == '\x08':
                                                sensor = '3'
                                        if sensor == '\x10':
                                                sensor = '4'
                                        if sensor == '\x20':
                                                sensor = '5'
                                        print 'SENSOR --->', sensor
                                        valor_a = sec_interes_RF[pos_D+2]
                                        valor_b = sec_interes_RF[pos_D+3]
                                        print 'VALOR A --->',valor_a
                                        print 'VALOR B --->', valor_b
                                        eventos_utr(s,nodo,sensor,valor_a,valor_b)

                                if tipo_RF == 'Y':
                                        print  'TRAMA  PETICION  DE  IP  NODO  RECIBIDA  Y
PARSEADA'
                                        pos_mac = sec_interes_RF.find('Y')
                                        mac = sec_interes_RF[pos_mac+1:pos_mac+7]
                                        mac_server = []
                                        mac_con = []
                                        mac_server.append(hex(ord(mac[0])))

                                        print 'MAC_SERVER[0]-->', mac_server[0]
                                        mac_server.append(hex(ord(mac[1])))
                                        print 'MAC_SERVER[1]-->', mac_server[1]
                                        mac_server.append(hex(ord(mac[2])))
                                        print 'MAC_SERVER[2]-->', mac_server[2]
                                        mac_server.append(hex(ord(mac[3])))
                                        print 'MAC_SERVER[3]-->', mac_server[3]
                                        mac_server.append(hex(ord(mac[4])))
                                        print 'MAC_SERVER[4]-->', mac_server[4]
                                        mac_server.append(hex(ord(mac[5])))
```

```
                                        print 'MAC_SERVER[5]-->', mac_server[5]
                                        mac_server_final = []
                                        mac_server_sin = []

                                        for i in range(len(mac_server)):
                                                a = mac_server[i].lstrip("0x")
                                                mac_server_sin.append(a)
                                                print   'SERVER   MAC   SIN   0x   -->',
mac_server_sin[i]

                                                b = mac_server_sin[i].zfill(2)
                                                mac_con.append(b)
                                                print 'SERVER MAC ANYADO 0-->',mac_con
                                        mac_completa = "".join(mac_con)
                                        print 'MAC --> ', mac
                                        print 'MAC ENVIADA A SERVIDOR -->',mac_completa
                                        print 'PETICION DE IP EJECUTADA'
                                        print '----------------------------------------
------------------'
                                        print '\n'
                                        peticion_ip_utr(s,nodo,mac_completa)

                                else:
                                        print 'Trama erronea'

                        else:
                                sec_interes_RF = trama_ind_RF[pos_start_RF+1:pos_stop_RF]
                                tipo_RF = sec_interes_RF[4]
                        if tipo_RF == 'L':
                                print 'TIPO RF-->', tipo_RF
                                        nodo = sec_interes_RF[1:4]
                                        print 'COMPROBACION NODO-->', nodo
                                        datos = sec_interes_RF[5:len(sec_interes_RF)-1]
                                        print 'DATOS DESDE UTR-->', datos
                                        lectura_utr(s,nodo,datos)
```

8.2.3. Bucle.py

```
#version copia desde Oldversion para gateway, comprobando el funcionamiento de v.py

import json
from analizar import analizaRF
import time
import serial
import os.path
from analizar import Analiza
from funciones import alive
from funciones import ResetArranque
from funciones import bateria
from funciones import sensores
from funciones import lectura_parametros
from funciones import lectura_alias
from funciones import mod_parametros
from funciones import modificacion_alias
from funciones import cambio_estados
from funciones import Reset
from funciones import estructura_red
import datetime
import socket
global check
global check1
global check2
global nombre_GW
global identGW1
global identGW2
global identGW3
global identGW4
global timestampGW
global checksum_server1
global checksum_server2
global valor_bat
global nevento
global existe


#funcion consulta en json (valores de comunicaciones) y devuelve este valor
def consulta_json(archivo_json,dato):
        global gate
        archivo_json_str = str(archivo_json)
        f = open(archivo_json_str)
        gw = json.load(f)
        type(gw)
        gw.keys()
        f.close()
        dato_str = str(dato)
        gate = gw['gateway']['comunicaciones'][dato]
        return(gate)


#Direccion, puerto y buffer
#TCP_IP = '158.42.105.71'    #'www.test.balmart.es'
#TCP_PORT = 6011
#PORT = '6011'
#BUFFER_SIZE=1024
BUFFER_SIZE = 1000000
TCP_IP = consulta_json('variables_json2.json','CIS')
TCP_IP = gate
TCP_PORT = consulta_json('variables_json2.json','CIP')
TCP_PORT = int(gate)
PORT = gate

#definimos start y stop
start ='\2'
stop = '\5'

#identificador GW
```

```
identGW1 = '0'
identGW2 = '0'
identGW3 = '0'
identGW4 = '3'
nombre_GW = 'TEST'

#funcion que comprueba que existe un archivo
def checkfile(nombre):
        global existe
        import os.path
        if os.path.exists(nombre):
                existe = 1
                return existe
        else:
                existe = 0
                return existe

#creamos timestamp
def stamp():
        global timestampGW
        ts = time.time()
        timestampGW = datetime.datetime.fromtimestamp(ts).strftime('%H%M%S%d%m%Y')
        return timestampGW

#def checksum(trama):
def checksum(trama):
        global check
        global check1
        global check2
        check = 0
        pre_check = 0
        for i in trama:
                a=ord(i)
                pre_check = pre_check^a
        check = hex(pre_check)
        check = check.upper()
        check1 = check[2]
        check2 = check[3]
        return check,check1, check2

#------------GATEWAY --> ANTENA RF-----------------------------------------------------
#funcion crea un fichero
def crea_fichero(nombre):
        arch = open(nombre,'w')
        arch.close

#funcion graba datos en un fichero existente con timestamp (recordar poner retorno de carro para
cada l$
def graba_fichero(nombre, trama):
        arch = open(nombre, 'a')
        stamp()
        arch.write('TIMESTAMP: ')
        arch.write(timestampGW)
        arch.write('\n')
        arch.write(trama)
        arch.write('\n')

#graba fichero de RF para parsear desde ahi
def graba2(nombre, trama):
        arch = open(nombre,'a')
        arch.write(trama)

#borra el fichero log de comunicacion serie
def borrar():
        os.remove('/home/pi/gateway/log_serie.txt')
#lectura de datos desde log_serie para parser
def leertxt():
        global trama_serie
        arch = open('log_serie.txt', 'r')
        trama_serie = arch.readlines()
        arch.close()
        return(trama_serie)
```

```
#funcion acceder a una linea de un txt obviando el timestamp
def lee_fichero(nombre):
        global linea
        arch = open(nombre, 'r')
        linea = arch.readlines()
        print linea
        arch.close()
        return linea

#funcion que comprueba que existe un archivo
def checkfile(nombre):
        global existe
        import os.path
        if os.path.exists(nombre):
                existe = 1
                return existe
        else:
                existe = 0
                return existe

#construimos socket TCP y activamos comunicacion serie

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #constructor del socket familia IPv prot.
flujo
s.settimeout(3)
s.connect((TCP_IP,TCP_PORT))
#-----------------------------------------------------
antena = serial.Serial('/dev/ttyAMA0',19200, timeout = 2) #establece comunicacion (puerto,
baudios)

#comprobamos que server.txt existe, si no creamos el archivo
checkfile('server.txt')
if existe == 0:
        crea_fichero('server.txt')
#alive(s)
#bucle principal
ResetArranque(s)
s.close()

while True:

#---------------PRUEBA CONECTO TCP AQUI
#-------------------------------------------------------------------------------------
        try:

                monitor = socket.socket()
                monitor.connect(('192.174.157.109',9998))



                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #constructor del socket
familia IPv prot. flujo
                s.settimeout(3)
                s.connect((TCP_IP,TCP_PORT))



        alive(s)
        print 'ESPERANDO SERVER'
        data = s.recv(BUFFER_SIZE) #datos recibidos
        print 'DATOS RECIBIDOS'
        if len(data) > 1:
                print 'DATO RECIBIDO-->', data
                trama = data
                tam_data = len(data)
                print 'tamanyo recibido-->', tam_data
                Analiza(s,antena,trama)
        print 'PARSEADO SERVIDOR'
        graba_fichero('server.txt',data)
```

```
#recibe server
#-----------------------------------------------
#recibe RF
        n = 0
        listaa = []
        lista = []
        lista_ascii = []
        listafinal = []
        print 'ESPERANDO RF'
        recibido = antena.read(100)
        print 'TAMANYO RECIBIDO-->', len(recibido)
        if len(recibido) !=  0:
                recibido_rf = recibido
                for n in range (len(recibido)):
                        a = recibido[n]
                        b = hex(ord(a))
                        c = b.encode("ascii")
                        lista.append(c)
                        listaa.append(b)
                        d = int(c,16)
                        e = chr(d)
                        lista_ascii.append(e)
                completo = "".join(lista_ascii)
                        graba2('log_serie.txt',completo)
                graba_fichero('RF.txt',completo)
                print 'RECIBIDO -->',recibido_rf
                print 'ASCII-->', lista_ascii
                print 'ENCODE', lista
                print 'HEX', listaa
                print 'COMPLETO-->', completo
                        leertxt()
                analizaRF(s,antena,trama_serie)
                        borrar()
                monitor.send('ok')
        except:
                        s.close()
                        print 'Error de conexion'
                        n = 0
                listaa = []
                lista = []
                lista_ascii = []
                listafinal = []
                print 'ESPERANDO RF'
                recibido = antena.read(100)
                print 'TAMANYO RECIBIDO-->', len(recibido)
                if len(recibido) !=  0:
                        recibido_rf = recibido
                        for n in range (len(recibido)):
                                a = recibido[n]
                                b = hex(ord(a))
                                c = b.encode("ascii")
                                lista.append(c)
                                listaa.append(b)
                                d = int(c,16)
                                e = chr(d)
                                lista_ascii.append(e)
                        completo = "".join(lista_ascii)
                        graba2('log_serie.txt',completo)
                        graba_fichero('RF.txt',completo)
                        print 'RECIBIDO -->',recibido_rf
                        print 'ASCII-->', lista_ascii
                        print 'ENCODE', lista
                        print 'HEX', listaa
                        print 'COMPLETO-->', completo
                        monitor.send('ok')
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------


print "cierre"
s.close()
```

8.2.4. V.py

```
import datetime
import time
import os
import shutil
import socket
#socket TCP servidor para control de procesos
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('192.174.157.109', 9998))
s.settimeout(15)
s.listen(1)
sc, addr = s.accept()


while True:


        try:
                print 'Dentro del try:'
                sc, addr = s.accept()
                print 'salgo del try'

        except:
                sc.close()
                print 'Entro en el except'
                print 'fallo de version'
                time.sleep(10)
                #realiza copia en directorio de ejecución
                shutil.copy2('/home/pi/Oldversion/bucle22.py', '/home/pi/gateway/bucle22.py')
                #resetea las raspberry para aplicar el cambio
                #command= "/usr/bin/sudo /sbin/reboot -f"
        #import subprocess
        #process = subprocess.Popen(command.split(),stdout=subprocess.PIPE)
        #output=process.communicate()[0]


print 'exit'
s.close()
```

8.2.5. Nodos.json


```
{"red": {"nodo15": {"IP": "000", "ACTIVO": "0", "AN": "nombre", "mac": "0004a330c43b", "SEN":
"000", "TNO": "0", "sensores": {"AS3": "max20", "AS2": "max20", "AS1": "max20caracteres", "AS6":
"max206", "AS5": "max20", "AS4": "max20", "TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2",
"TS3": "a3", "TS1": "a1"}}, "nodo14": {"IP": "000", "ACTIVO": "0", "AN": "nombre", "mac":
"0004a330c43b", "SEN": "000", "TNO": "0", "sensores": {"AS3": "max20", "AS2": "max20", "AS1":
"max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20", "TS6": "a6", "TS4": "a4",
"TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo13": {"IP": "000", "ACTIVO": "0", "AN":
"nombre", "mac": "0004a330c43b", "SEN": "000", "TNO": "0", "sensores": {"AS3": "max20", "AS2":
"max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20", "TS6": "a6",
"TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo12": {"IP": "000",
"ACTIVO": "0", "AN": "nombre", "mac": "0004a330c43b", "SEN": "000", "TNO": "0", "sensores": {"AS3":
"max20", "AS2": "max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20",
"TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo11": {"IP":
"000", "ACTIVO": "0", "AN": "nombre", "mac": "0004a330c43b", "SEN": "000", "TNO": "0", "sensores":
{"AS3": "max20", "AS2": "max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4":
"max20", "TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo10":
{"IP": "000", "ACTIVO": "0", "AN": "nombre", "mac": "0004a330c43b", "SEN": "000", "TNO": "0",
"sensores": {"AS3": "max20", "AS2": "max20", "AS1": "max20caracteres", "AS6": "max206", "AS5":
"max20", "AS4": "max20", "TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1":
"a1"}}, "nodo9": {"IP": "102", "ACTIVO": "0", "AN": "nombre", "mac": "0004A330C43B", "SEN": "000",
"TNO": "0", "sensores": {"AS3": "max20", "AS2": "max20", "AS1": "max20caracteres", "AS6": "max206",
"AS5": "max20", "AS4": "max20", "TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3",
"TS1": "a1"}}, "nodo8": {"IP": "102", "ACTIVO": "0", "AN": "nombre", "mac": "0004A330C43B", "SEN":
"000", "TNO": "0", "sensores": {"AS3": "max20", "AS2": "max20", "AS1": "max20caracteres", "AS6":
"max206", "AS5": "max20", "AS4": "max20", "TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2",
"TS3": "a3", "TS1": "a1"}}, "nodo7": {"IP": "102", "ACTIVO": "0", "AN": "nombre", "mac":
"0004A330C43B", "SEN": "000", "TNO": "0", "sensores": {"AS3": "max20", "AS2": "max20", "AS1":
"max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20", "TS6": "a6", "TS4": "a4",
"TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo6": {"IP": "102", "ACTIVO": "0", "AN":
"nombre", "mac": "0004A330C43B", "SEN": "000", "TNO": "0", "sensores": {"AS3": "max20", "AS2":
"max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20", "TS6": "a6",
"TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo5": {"IP": "102", "ACTIVO":
"0", "AN": "nombre", "mac": "0004A330C43B", "SEN": "000", "TNO": "0", "sensores": {"AS3": "max20",
"AS2": "max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20", "TS6":
"a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo4": {"IP": "102",
"ACTIVO": "0", "AN": "nombre", "mac": "0004A330C43B", "SEN": "000", "TNO": "0", "sensores": {"AS3":
"max20", "AS2": "max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4": "max20",
"TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo3": {"IP":
"102", "ACTIVO": "0", "AN": "nombre", "mac": "0004A330C43B", "SEN": "000", "TNO": "0", "sensores":
{"AS3": "max20", "AS2": "max20", "AS1": "max20caracteres", "AS6": "max206", "AS5": "max20", "AS4":
"max20", "TS6": "a6", "TS4": "a4", "TS5": "a5", "TS2": "a2", "TS3": "a3", "TS1": "a1"}}, "nodo2":
{"IP": "104", "ACTIVO": "1", "AN": "UTR2", "mac": "0004A330C5DA", "SEN": "001", "TNO": "0",
"sensores": {"AS3": 0, "AS2": 0, "AS1": "sensor1", "AS6": 0, "AS5": 0, "AS4": 0, "TS6": 0, "TS4":
0, "TS5": 0, "TS2": 0, "TS3": 0, "TS1": "110"}}, "nodo1": {"IP": "102", "ACTIVO": "1", "AN":
"UTR1", "mac": "0004A330C43B", "SEN": "001", "TNO": "0", "sensores": {"AS3": 0, "AS2": 0, "AS1":
"sensor temperatura", "AS6": 0, "AS5": 0, "AS4": 0, "TS6": 0, "TS4": 0, "TS5": 0, "TS2": 0, "TS3":
0, "TS1": "610"}}}}
```

8.2.6. Parámetros nodos.json


```
{"gateway": {"sensores digitales": {"11": {"DBK": "0", "SN11": "cont1", "DBC": "0", "DBE": "0",
"DBU": "00000", "DBT": "0"}, "10": {"DAE": "0", "DAC": "0", "SN10": "cont1", "DAK": "0", "DAT":
"0", "DAU": "00000"}, "13": {"SN13": "cont1", "DDU": "01160", "DDT": "2", "DDK": "34", "DDC": "0",
"DDE": "1"}, "12": {"DCK": "13", "SN12": "cont1", "DCE": "1", "DCC": "0", "DCT": "1", "DCU":
"00050"}, "15": {"DFU": "00070", "DFT": "1", "SN15": "cont1", "DFK": "0", "DFE": "1", "DFC": "0"},
"14": {"DET": "1", "SN14": "cont1", "DEU": "00060", "DEK": "0", "DEC": "0", "DEE": "1"}, "1":
{"D1T": "1", "D1U": "65531", "D1C": "0", "D1E": "1", "SN1": "cont1", "D1K": "8"}, "0": {"D0U":
"00020", "D0T": "1", "D0E": "1", "D0C": "0", "SN0": "cont1", "D0K": "13", "SN8": "d2"}, "3":
{"D3T": "0", "D3U": "00000", "SN3": "cont1", "D3E": "0", "D3C": "0", "D3K": "0"}, "2": {"D2U":
"00000", "D2T": "0", "D2K": "0", "SN2": "cont1", "D2C": "0", "D2E": "0"}, "5": {"D5K": "0", "D5C":
"0", "D5E": "0", "SN5": "cont1", "D5T": "0", "D5U": "00000"}, "4": {"D4K": "0", "D4C": "0", "D4E":
"0", "SN4": "cont1", "D4U": "00000", "D4T": "0"}, "7": {"D7T": "0", "SN7": "cont1", "D7U": "00000",
"D7C": "0", "D7E": "0", "D7K": "0"}, "6": {"D6K": "0", "D6E": "0", "D6C": "0", "SN6": "cont1",
"SN7": "d1", "D6U": "00000", "D6T": "0"}, "9": {"D9T": "0", "D9U": "00000", "D9K": "0", "D9E":
"0", "D9C": "0", "SN9": "cont1"}, "8": {"D8U": "00000", "D8T": "0", "D8K": "0", "D8E": "0", "D8C":
"0", "SN8": "cont2"}}, "comunicaciones": {"CIS": "158.42.105.71", "CIP": "6011", "CIT": "6011",
"CTA": "2", "CID": "158.42.105.71"}, "sensores analogicos": {"1": {"A1U": "64540", "A1T": "2",
"A1W": "60", "A1V": "0", "A1P": "4", "SN17": "cont1", "A1E": "1", "A1M": "90", "A1L": "00000",
"A1N": "1", "A1K": "36"}, "0": {"A0P": "4", "A0V": "0", "A0W": "60", "A0T": "2", "A0U": "65535",
"A0K": "35", "A0N": "1", "SN16": "cont1", "A0L": "00000", "A0M": "1", "A0E": "1"}, "3": {"SN19":
"d3", "A3P": "0", "A3W": "10", "A3V": "0", "A3U": "05000", "A3T": "1", "A3E": "1", "A3K": "0",
"A3N": "3", "A3M": "3", "A3L": "40536"}, "2": {"A2P": "0", "SN18": "cont1", "A2T": "1", "A2U":
"65535", "A2V": "0", "A2W": "10", "A2E": "1", "A2K": "0", "A2L": "00000", "A2M": "1", "A2N": "1"},
"5": {"A5E": "1", "A5K": "0", "A5M": "3", "A5L": "00000", "A5N": "3", "SN21": "cont1", "A5P": "0",
"A5U": "00060", "A5T": "1", "A5W": "10", "A5V": "0"}, "4": {"A4E": "1", "SN20": "d4", "A4N": "3",
"A4L": "00000", "A4M": "3", "A4K": "0", "A4V": "0", "A4W": "10", "A4T": "1", "A4U": "00060", "A4P":
"0"}, "6": {"A6T": "1", "A6U": "00060", "A6V": "0", "A6W": "10", "A6P": "0", "A6L": "00000", "A6M":
"3", "A6N": "3", "SN22": "cont1", "A6K": "0", "A6E": "1"}}}}
```