FREEZE SENSE:

Sensor IoT para monitorizar la cadena de frío en el transporte y almacenamiento de alimentos

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Irene Cerro de Paz
Virginia Galisteo Fernández
Alejandro Martín Seijas
Carlos Membrilla Cobo

Director:
Christian Tenllado Van der Reijden

Codirector:
Luis Piñuel Moreno

Madrid, Septiembre de 2016

Agradecimientos

Quisiéramos agradecer a Luis Piñuel Moreno y Christian Tenllado Van der Reijden, los directores de este proyecto, el apoyo brindado.

Asimismo, queríamos agradecer su dedicación y tiempo a Ricardo Badía Melis, sin su ayuda no habría sido posible la realización de las pruebas necesarias para este proyecto.

Resumen

Mantener y asegurar la cadena del frío en el transporte de alimentos perecederos es uno de los aspectos más importantes que deben tener en cuenta las empresas de logística y cadenas de venta al consumidor. Con el control de la cadena del frío se puede asegurar tanto unos mínimos de calidad como de seguridad del producto en cuestión.

Para asegurar el cumplimiento de la cadena de frío en el transporte de alimentos (o medicamentos) existen actualmente multitud de sistemas o dispositivos en el mercado que pueden cumplir perfectamente ese papel. Algunos de ellos son sistemas que únicamente informan al operario en destino si se ha producido una ruptura de la cadena del frío sirviendo como control de calidad previo a la venta de los alimentos, pero por el contrario otros sistemas sí que realizan un control exhaustivo de la cadena de frío en tiempo real dando una mayor capacidad de reacción a la empresa logística o cadena de venta para subsanar cuanto antes esa ruptura en la cadena del frío.

Es este tipo de sistemas en el que se va a basar este proyecto. Por ello con la ayuda de la arquitectura IoT se mejorarán las principales ventajas que tienen este tipo de sistemas (funcionalidad) y disminuirán o incluso eliminarán las desventajas que tienen este tipo de sistemas, principalmente coste (objetivo más importante del proyecto) y dificultad de instalación.

Palabras clave

Arquitectura IoT, cadena de frío, logística, tiempo real.

Abstract

Keep and ensure the cold chain transport of perishable food is one of the most important aspects to be considered by logistics companies and sales chain. Controlling the cold chain it can be ensured both a minimum quality and safety of the product.

To ensure the compliance of the cold chain in food transport (or medicine), currently there are many systems or devices that can fulfill this role. Some of them are systems which only inform the operator at the arrival if there has been a breakdown in the cold chain serving as a previous sale quality control of the food, but there are others systems that perform an exhaustive cold chain control in real time giving greater foresight to the logistics company or sales chain as soon as possible to remedy this break in the cold chain.

This project will be bases in this type of system. So, using the IoT architecture we will improve the main advantages that this type of system has (functionality) and decrease ore even eliminate their disadvantages, mainly the cost (most important objective of the project) and the difficulty of installation.

Key Words

IoT architecture, cold chain, logistics, real time.

Los abajo firmantes autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: "Freeze Sense", realizado durante el curso académico 2015-2016 bajo la dirección de Luis Piñuel Moreno y Christian Tenllado Van der Reijden en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Irene Cerro de Paz

Virginia Galisteo Fernández

Alejandro Martín Seijas

Carlos Membrilla Cobo

ÍNDICE

ÍNDICE DE IMÁGENES

ÍNDICE DE TABLAS

Lista de Acrónimos

3G	*Abreviación de tercera generación de transmisión de voz y datos a través de telefonía móvil mediante UMTS*
ACK	*Acknowledgement*
AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
CoAP	*Constrained Application Protocol*
CPU	*Central Processing Unit*
CVS	*Concurrent Versions System*
DC	*Direct Current*
DC-HSPA	*Direct Current-High Speed Downlink Packet Access*
DMZ	*Demilitarized Zone*
DTLS	*Datagram Transport Layer Security*
EDGE	*Enhanced Data Rates for GSM Evolution*
FDD	*Feature-driven development*
GPIO	*General Purpose Input/Output*
GPRS	*General Packet Radio Service*
GSM	*Global System for Mobile communications*
HTTP	*Hypertext Transfer Protocol*
I²C	*Inter-Integrated Circuit*
ID	*Identification*
IEEE	*Institute of electrical and Electronics Engineers*
IoT	*Internet of Things*
LE	*Low Energy*
LTE	*Long Term Evolution*
MQ	*Message Queing*
MQTT	*Message Queue Telemetry Transport*
OS	*Operating System*
PPI	*Periferical Parallel Interface*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
RSSI	*Received Signal Strength Indicator*
SDK	*Software Development Kit*
SMS	*Short Message Service*
SNMP	*Simple Network Management Protocol*
SPI	*Serial Peripheral Interface*
SQL	*Structured Query Language*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TKIP	*Temporal Key Integrity Protocol*
UDP	*User Datagram Protocol*
UMTS	*Universal Mobile Telecommunications System*
USART	*Universal Synchronous/Asynchronous Receiver/Transmitter*
USB	*Universal Serial Bus*
VPN	*Virtual Private Network*
WAN	*Wide Area Network*
WEP	*Wired Equivalent Privacy*

Wi-Fi	*Wireless Fidelity*
WPA	*Wi-Fi Protected Access*
WPA2	*Wi-Fi Protected Access 2*

1. INTRODUCCIÓN

Mantener la cadena del frío en el transporte de objetos perecederos tales como alimentos, es una de las obligaciones más importantes que deben cumplir tanto las empresas de logística como las empresas que se dedican a la venta de estos productos directamente al consumidor.

Se define la cadena del frío como el conjunto de eslabones o pasos necesarios en el proceso de refrigeración (-1ºC hasta 4ºC) o congelación (entre -18ºC y -25ºC) para que los alimentos perecederos lleguen al consumidor con una garantía de calidad y seguridad [1]. Este conjunto de eslabones abarca desde la fase de producción, pasando por la etapa de transporte y su posible almacenamiento para finalizar con la etapa de venta al consumidor. De nada sirve asegurar la cadena del frío en una de esas etapas si no se cumple en el resto de ellas.

Como se ha comentado en el párrafo anterior, cumplir con la cadena de frío en los alimentos aporta un valor tanto de calidad como de seguridad. La aplicación del frío sobre los alimentos actúa inhibiendo total (productos congelados) o parcialmente (productos refrigerados) las reacciones metabólicas que se producen en los productos. Con ello se consigue retrasar la degradación del alimento y de sus propiedades en cuanto al olor y sabor. Además, en el caso de que la cadena de frío se rompa en alguna de sus fases se produce un aumento de la actividad microbiana en estos productos, aumentando la posibilidad de que el consumidor final pueda contraer algún tipo de enfermedad grave al ingerir este tipo de alimentos (Salmonella, E.Coli, etc).

Tal y como se ha comentado anteriormente, la cadena del frío debe cumplirse en todas las etapas de producción, transporte y venta de los productos y, por ello los diferentes actores que intervienen deben hacer especial hincapié en el control de la temperatura de cada una de estas fases. A modo de ejemplo, utilizaremos un producto lácteo elaborado (Queso, Yogur, etc) como producto final para mostrar los diferentes actores y fases que intervienen en la cadena del frío:

- **Proveedor de materia prima:** como es bien sabido, la principal materia prima de este tipo de productos elaborados es la leche. Por ello, en el transporte de esta materia prima desde las granjas de recolección hasta las fábricas de procesamiento se debe controlar la temperatura y de esta manera confirmar la calidad y seguridad de la principal materia prima antes de la elaboración del producto final.

- **Fábrica de procesamiento:** durante el procesamiento y elaboración de productos lácteos se debe mantener controlada en todo momento la temperatura de todos los elementos que intervienen. En esta fase se debe hacer especial hincapié en el almacenamiento de los productos antes de que sean enviados a través de las diferentes empresas de logística existentes. Una ruptura de la cadena del frío en esta fase puede conllevar grandes pérdidas económicas para la empresa productora y por consiguiente la pérdida de prestigio como marca empresarial.
- **Transporte del producto final:** en esta etapa existe una mayor probabilidad de la ruptura de la cadena del frío, ya que pueden intervenir diferentes empresas de

logística para un mismo producto final dependiendo del lugar donde vaya a ser enviado. Además, no solo influye el lugar donde va a ser enviado el producto sino también los lugares por donde va a ser transportado. De esta manera es más sencillo asegurar la cadena del frío en transportes que vayan por zonas de bajas temperaturas y por el contrario es necesario hacer un seguimiento más cercano a los transportes que transcurran por lugares de altas temperaturas. Es aquí donde se centrará el principal objetivo de nuestro proyecto, control y seguimiento de la temperatura en el transporte de productos perecederos.

- **Comercialización: en esta etapa hay que hacer especial hincapié en dos momentos clave:**

    - Paso de camión de transporte a las instalaciones de venta (o incluso a otro camión de transporte): en este punto existe también una mayor probabilidad de la ruptura de la cadena del frío y por tanto es muy importante para la cadena de venta asegurar que el traspaso desde el camión de transporte hasta las instalaciones de venta (o a otro camión diferente) se haga lo más rápido posible.

    - Realizar controles periódicos de la temperatura a la que se almacena el producto final en los expositores de venta: De nada serviría realizar un control exhaustivo de la cadena del frío en todas las fases descritas anteriormente si en la última fase, la cadena de venta no pudiera ser capaz de asegurar la temperatura adecuada de sus productos refrigerados o congelados.

Una vez introducido el concepto de la cadena del frío y haber mostrado las características y problemáticas de este concepto, se explicará en las siguientes secciones los diferentes sistemas de control de la cadena del frío que actualmente existen en el mercado. Una vez mostrados todos estos tipos y haber realizado un análisis de los pros y contras de cada uno de ellos se mostrará al lector los principales objetivos y requisitos de los que consta este proyecto, las fases de desarrollo por las que ha ido pasando para terminar mostrando la estructura del resto del documento.

1.1. Sistemas De Seguimiento De La Cadena De Frío Actuales

Como se ha podido comprobar en la sección anterior, la cadena de frío es un concepto muy importante tanto para las cadenas de venta de comestibles como para las empresas de logística. Por ello, para controlar de alguna manera el problema de la cadena del frío, existen actualmente diferentes tipos de sistemas los cuales se diferencian principalmente por su precio, funcionalidad y tamaño. Estos sistemas de control de la cadena del frío se pueden clasificar en 3 grandes grupos:

- Data Loggers
- Etiquetas TTI
- Sistemas de control de la cadena del frío en tiempo real

Data Loggers

Estos sistemas únicamente almacenan la temperatura periódicamente en su memoria interna. Una vez el producto llega al destino, el operario descarga los datos en su PC o móvil mediante la conexión USB o Bluetooth y comprueba el registro de temperaturas durante el transporte.

- **Ventajas**: este tipo de dispositivos ofrecen una vida útil de batería bastante buena (500 días aprox). Al tener un tamaño bastante reducido (depende del modelo escogido) se puede acoplar prácticamente a nivel de caja que vaya a ser transportada en las cámaras frigoríficas, aunque su elevado coste limite esta posibilidad en gran medida. Por ello, suelen instalarse únicamente 1 o 2 dispositivos por cámara frigorífica, dependiendo del tamaño de la misma. Además de todas estas ventajas, este tipo de dispositivos suelen ser bastante sencillos de instalar.

- **Desventajas**: la principal desventaja que tienen este tipo de dispositivos es que únicamente son capaces de registrar las temperaturas periódicamente las cuales podrán ser verificadas exclusivamente al final del trayecto. Por ello, no podremos controlar la temperatura en tiempo real y de esta manera evitar una posible ruptura de la cadena del frío. Por esta desventaja mostrada y por su elevado coste se descartó su aplicación a nuestro proyecto.

- **Ejemplos**: testo 174 T [2], el cual ofrece hasta 500 horas de autonomía con un tamaño bastante reducido. Este modelo se puede llegar a encontrar por 70 euros en el mercado, imposibilitando de esta manera (como se citó anteriormente) el acoplamiento de varios dispositivos en una misma cámara frigorífica debido a su elevado coste.

Etiquetas TTI

Estas etiquetas están compuestas por reactivos químicos sensibles a los cambios de temperatura. De esa manera, en el momento en que la cadena del frío se rompe, comienza la reacción química en la etiqueta cambiando el color de la misma. Por ello, con un simple vistazo el operario puede comprobar si el producto ha sufrido una ruptura de la cadena del frío durante el transporte.

- **Ventajas:** este tipo de dispositivos suelen ser de pequeño tamaño (3-4 cm) pudiendo acoplarse de mejor manera que los data loggers. Por ello se puede instalar tanto a nivel de palé, caja o incluso a nivel de producto (en contados casos).

- **Desventajas:** este tipo de dispositivos nos valdrán únicamente para comprobar al final del trayecto si se ha sufrido una ruptura de la cadena del frío, pero sin indicar en qué momento o durante cuánto tiempo se produjo. Aunque el coste de estas etiquetas sea reducido, son de un único uso, obligando a la empresa logística a renovar constantemente estos dispositivos.

- **Ejemplos:** Etiqueta WarmMark. estas etiquetas se pueden llegar a encontrar por 2€ en el mercado. Como se ha comentado anteriormente, son etiquetas de un único uso sin ofrecernos la posibilidad de saber en qué momento o durante cuánto tiempo se rompió la cadena del frío. [3]

Sistemas de control de la cadena del frío en tiempo real

Estos sistemas permiten controlar en tiempo real la temperatura del transporte de alimentos. De esta manera la empresa logística o la cadena de venta puede comprobar en cualquier momento el estado de los productos y por tanto reaccionar con más rapidez frente a los posibles cambios o rupturas de la cadena del frío. Este tipo de dispositivos normalmente se componen de uno o varios sensores de temperatura que se colocan en diferentes puntos de la cámara frigorífica y se conectan vía Wi-Fi o Bluetooth directamente con un *router* o dispositivo móvil el cual se encargará de enviar en tiempo real las mediciones a un servidor. Posteriormente estas mediciones normalmente pueden ser visualizadas ya sea directamente desde el dispositivo móvil o desde una interfaz web.

- **Ventajas:** este tipo de sistemas son bastante completos ofreciendo una amplia gama de sensores de medición y multitud de servicios adicionales (herramientas de visualización, aplicaciones móviles, etc) que facilitan en gran medida tanto a los operarios como a las empresas cliente.

- **Desventajas:** normalmente este tipo de sistemas suelen ser bastante caros por la cantidad de servicios y la calidad del dato que recogen (muy precisos). Por ello se limitan a empresas con gran volumen de negocio o para transportes que requieren de un control muy exhaustivo de la temperatura (alimentos, medicamentos, etc). Además, la instalación de este tipo de sistemas suele ser costosa tanto en tiempo como en dificultad (ver siguiente ejemplo).

- **Ejemplos:** Astrata es una empresa que enfoca parte de su negocio al control de la cadena del frío en transportes a nivel mundial. Ofrecen un servicio totalmente completo, tanto a nivel de control de temperatura en el interior de las cámaras frigoríficas, como a nivel de visualización en diferentes dispositivos. Este sistema en concreto ronda los 2000 euros además de requerir una instalación específica, ya que los sensores de temperatura están directamente conectados a la batería del medio de transporte (Tráiler, furgoneta, etc) a través del CAN-BUS. [4]

1.2. Objetivos Del Proyecto

Este proyecto se basará principalmente en el último sistema citado en el punto anterior (Sistemas de control de la cadena del frío en tiempo real). Para ello se fijarán una serie de objetivos que se centrarán en mejorar las ventajas de este tipo de sistema (funcionalidad y visibilidad) y en minimizar o eliminar por completo las desventajas que tienen. (instalación y coste).
Pero antes de fijar estos objetivos es importante conocer los aspectos más importantes que tendrá en cuenta una empresa (logística o cadena de supermercados) a la hora de seleccionar alguno de los sistemas citados anteriormente:

- **Coste del producto transportado**: la empresa logística asumirá más riesgos, es decir, hará un control menos exhaustivo de la cadena del frío en productos que tengan un precio de coste más bajo. Al tener un precio de coste menor, la empresa logística y la cadena de venta podrán asumir un mayor volumen de pérdida por deterioro del producto, producido en este caso por la ruptura de la cadena del frío.

- **Sensibilidad al cambio de temperatura**: las empresas tomarán más medidas de control de la temperatura en productos que tengan una mayor sensibilidad al cambio de temperatura. Por ello utilizarán sistemas más avanzados para el control de la cadena del frío y por lo tanto más caros. Este punto iría muy ligado al anterior ya que normalmente los productos con mayor sensibilidad al cambio de temperatura suelen ser más caros en cuanto a costes de producción y mantenimiento. Un buen ejemplo de esto sería el transporte de ciertos tipos de medicamentos (vacunas).

- **Volumen de negocio de la empresa de logística o transporte**: el control de la cadena de frío depende en gran medida del tamaño de la empresa y por consiguiente del volumen de negocio que acapara. A mayor volumen de negocio, mayor presupuesto tiene disponible para llevar a cabo un control más exhaustivo de la cadena del frío.

Teniendo en cuenta todos estos aspectos, se muestra a continuación la lista de principales objetivos (ordenados de mayor a menor prioridad) que se definieron en un principio antes de comenzar el desarrollo del proyecto.

- Maximizar la relación entre coste y funcionalidad
- Posibilidad de visualizar los datos en tiempo real y en cualquier dispositivo móvil
- Conexión Wi-Fi con los dispositivos.
- Minimizar el tamaño del dispositivo lo máximo posible
- Fácilmente intercambiable entre cámaras frigoríficas
- Facilidad de instalación

Como se ha podido comprobar en el punto anterior existen multitud de productos en el mercado que abarcan todo tipo de precios y funcionalidades, a mayor funcionalidad mayor coste. Por ello el principal objetivo de nuestro proyecto se basa en obtener el menor coste posible sin necesidad de disminuir las funcionalidades del sistema. Este ahorro en el coste se aplicará a todos los ámbitos del prototipo, tanto en el microprocesador como el sensor de temperatura, en el *router* y en la batería. Cabe destacar que este ajuste en el coste del prototipo también es extremadamente necesario ya que se pretende colocar un dispositivo por cada uno de los palés que tenga el camión.

Por ello, en caso de que no se realizara un gran ajuste en cuanto al coste del prototipo, el precio con el que se debería poner en venta el producto final aumentaría exponencialmente haciéndolo inviable para las empresas de logística que trabajan con márgenes de beneficio muy ajustados.

Aunque se centren la mayoría de los esfuerzos en abaratar todo lo posible el coste del prototipo, el futuro cliente debe tener al alcance una serie de facilidades a la hora de usarlo. Una de las más importantes es tener la posibilidad de visualizar (en tiempo real) y configurar ciertos parámetros del dispositivo sin necesidad de utilizar ningún elemento específico para la lectura de los mismos. De esta manera, la empresa logística o incluso el conductor pueden comprobar la temperatura del transporte tanto desde una aplicación para dispositivos móviles como desde una interfaz web respectivamente.

Para que esta visualización de datos y configuración de parámetros pueda ser posible y además de manera inalámbrica se eligió el protocolo Wi-Fi. Esto nos valdría como prueba de concepto para un futuro lanzamiento real al mercado.

Por otro lado, el prototipo también debe de facilitar las tareas a los operarios que se encarguen de las operaciones de carga y descarga del camión. Esto es realmente importante para la viabilidad de nuestro proyecto ya que nuestro dispositivo irá acoplado a la parte inferior del palé. Al tener un tamaño pequeño facilita al operario las labores de carga y descarga del propio palé además de minimizar de esta manera la posibilidad de caída o pérdida del dispositivo en estas labores.

Además, este pequeño tamaño debe ir acompañado de la capacidad de reconocer el cambio de camión o cámara frigorífica. Esto es realmente importante ya que los palés que lleva un camión en un transporte pueden reutilizarse posteriormente en otro camión diferente y por ello el dispositivo debe ser capaz por sí mismo de reconocer este cambio y ser capaz de emitir los datos de la misma manera.

Aunque con todo lo citado anteriormente se cubra prácticamente todas las características necesarias a nivel de usuario o funcionalidad, también se debe tener en cuenta la facilidad de instalación de este prototipo en las cámaras frigoríficas. Como se ha mostrado en el punto 1.1, existen actualmente sistemas realmente avanzados y muy completos en cuanto a funcionalidad que requieren de una instalación específica en las cámaras frigoríficas. Uno de estos sistemas es Astrata, que ofrece un sistema completo de tracking de la cadena de frío que requiere de una pequeña obra en la cámara frigorífica para poder suministrar electricidad desde la batería del propio camión al sensor de temperatura (conexión a través del CAN-BUS).

1.3. Plan De Trabajo

Para llevar a cabo la realización del proyecto de manera eficiente y desarrollarlo en el tiempo estimado de 1 año académico, se dividió en una consecución de fases incrementales. Al final de cada una de estas fases y como requisito indispensable para avanzar a la siguiente fase, se realizaron reuniones con los tutores del proyecto.
Estas reuniones sirvieron como control de estado del proyecto, en las que se revisaban las tareas completadas en la última fase realizada y se indicaban los puntos a completar en las siguientes.

A continuación, se muestran las principales fases de las que consta el proyecto, incluyendo las características de cada una de ellas y el espacio temporal que ocuparon. Todas estas fases pueden verse detalladamente en el diagrama de Gantt que se muestra en la siguiente página.

1.3.1. Primera fase: Investigación (Octubre)

En esta primera fase, se enfocó el esfuerzo en la investigación del sensor de temperatura y el chip necesarios, para la realización del proyecto. Tras explorar los diferentes modelos y haber analizado cuál de ellos se adecuaba de la mejor manera se procedió a la selección inicial (no definitiva) tanto del chip como del termómetro. Al ser una fase inicial, se fijaron reuniones quincenales con los tutores del proyecto para facilitar de la mejor manera el inicio del proyecto.

1.3.2. Segunda Fase: desarrollo de código básico inicial (Noviembre-Diciembre)

Una vez investigados tanto el sensor de temperatura como el chip seleccionado se desarrolló un código inicial que únicamente medía la temperatura del lugar y lo enviaba a una herramienta web de visualización de datos (ThingSpeak) que se explicará más adelante con detalle. Además, en esta fase se realizó una prueba inicial con el código mencionado anteriormente en la que se comprobó el funcionamiento del dispositivo con una pila externa (ver Sección 3.2 Baterías). Al igual que en la anterior fase, se mantuvieron reuniones quincenales ya que se corresponde con una etapa muy inicial del proyecto.

1.3.3. Tercera fase: Reparto y desarrollo del proyecto (Enero-Abril)

Previamente a las vacaciones de Navidad se realizó una reunión en la que se dividieron cada una de las actividades a realizar por cada uno de los integrantes del grupo. De esta manera, el proyecto quedó dividido en dos partes funcionales totalmente separadas:

- Desarrollo del código funcional del nodo: en el que se incluye conexión selectiva dependiendo de la intensidad de la señal Wi-Fi, la medición de la temperatura, su almacenamiento, y su envío secuencial al servidor.

- Desarrollo de las comunicaciones entre el nodo y el servidor y desarrollo de aplicaciones móviles (una de ellas para configuración de parámetros del nodo y otra para la visualización de datos). Además, al finalizar esta parte, se implementó un bridge (o enlace) entre el servidor y la web de visualización de datos ThingSpeak para que de esta manera el futuro cliente tuviera varias opciones de visualización de los datos (desde dispositivo móvil o entorno web)

1.3.4. Cuarta fase: Realización de pruebas (Mayo-Junio)

Esta fase fue la última con respecto al desarrollo del prototipo. En ella se realizaron pruebas en las cámaras frigoríficas de la facultad de Agrónomos de la Universidad Politécnica de Madrid para comprobar el funcionamiento y la conectividad entre el nodo y el servidor bajo condiciones extremas de frío e intensidad de señal WiFi (ver sección 4, Experimentación).

Una vez realizadas estas pruebas en las cámaras frigoríficas, se llevaron a cabo las pruebas de consumo del dispositivo, las cuales se pueden comprobar en el punto (ver sección 4, Pruebas de consumo).

1.3.5. Quinta fase: Realización de la memoria (Mayo-Septiembre)

Esta última fase se dedicó principalmente a la realización de la memoria final del proyecto.

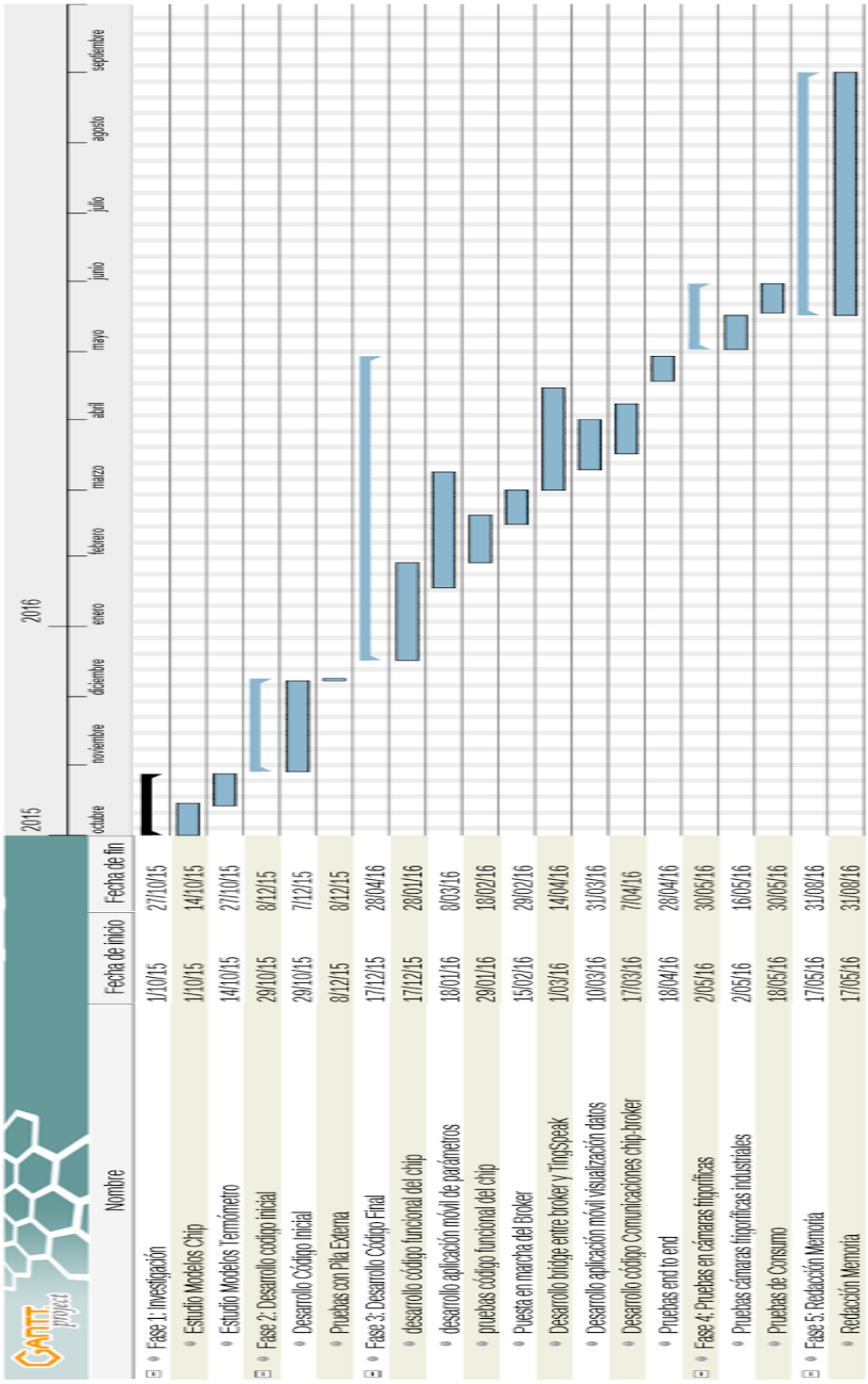

Figura 1.1 Diagrama de Gantt

2. SISTEMA IOT

El cambio de la tecnología analógica, mecánica, y electrónica, a la tecnología digital forma una pieza clave en la revolución digital que supone cambios radicales provocados por la computación y la tecnología de la comunicación.
La tendencia a la evolución de la tecnología nos abre un amplio abanico de posibilidades hacia la comunicación que transforma no sólo la industria sino todos los aspectos de la vida humana: la economía, la educación, la sanidad, el arte y la cultura. Esto supone que el mundo físico se va a convertir en un tipo de sistema de información, a través de sensores y los propios objetos físicos que estarán vinculados a través de cables o redes inalámbricas utilizando los protocolos de Internet.
Estos nos permiten medir distintos parámetros (temperatura, luz, humedad, presión, aceleración, distancia, etc).

A medida que fue transcurriendo el tiempo, los microcontroladores adquirieron mayor importancia en el ámbito de la computación y de la electrónica. Esto se debió a los grandes resultados que se obtuvieron al integrarlos en los distintos circuitos electrónicos junto con sensores. Poco a poco fueron evolucionando dando lugar a redes locales, formadas por un conjunto de sensores que colaboran en una tarea común y poseen ciertas capacidades sensitivas y de comunicación inalámbrica. Tienen unas características propias y otras adaptadas de las redes inalámbricas. Se puede apreciar el rápido desarrollo que las tecnologías de redes inalámbricas han experimentado en los últimos años.
El más interesante es el de las redes de sensores inalámbricos, ya que tienen múltiples aplicaciones en distintos sectores como el de la seguridad, industria farmacéutica, transporte e incluso agricultura.

Estas redes de sensores se basan en dispositivos de bajo coste y consumo también conocidos como nodos, capaces de obtener información de su entorno, procesarla y enviarla a un nodo recolector.
Una red de sensores inalámbricos está formada por numerosos dispositivos distribuidos, que son unidades autónomas formados por un microcontrolador, una fuente de energía, una memoria externa, un transceptor, es decir, un dispositivo que cuenta con transmisor y receptor y, por último, un sensor. Debido a las limitaciones de la vida de la batería, los nodos se diseñan teniendo en cuenta la conservación de ésta y, por tanto, pasan mucho tiempo 'dormidos', un estado en el cual el consumo de batería es mínimo.

El mundo digital ha evolucionado hasta tal punto que somos capaces de dotar de inteligencia a objetos que no la tenían. Un ejemplo sencillo sería una bombilla: si estuviera conectada a Internet, se podría encender de forma remota desde cualquier lugar, indicar cuál es el consumo eléctrico que ha tenido durante un período de tiempo determinado, si está fundida, cuánta energía le queda, etc. [5]

De esta manera, la conexión a internet de los objetos y el ser accesibles en todo el mundo suponen un cambio diferencial sobre las redes de sensores que conocíamos. Además, debido a la cantidad de datos que nos proporcionan estos nuevos sensores surge la necesidad de analizar y procesar toda esta información.

Es en este punto de la evolución donde tienen lugar los sistemas IoT, que son pieza clave para el procesamiento de datos y en los que cabe destacar el papel tan importante que tienen las redes mencionadas ya que son la vía principal por la que los "objetos inteligentes" se comunican entre ellos y hacia Internet.

2.1. Sistemas IoT

En la actualidad, la sociedad está cada vez más cerca del 'todo conectado' y con esto, más cerca de los sistemas IoT.

Con IoT nos referimos a la interconexión digital de objetos cotidianos con internet. [6] El objetivo de IoT es conseguir dispositivos más inteligentes e independientes a través de la comunicación entre unos y otros.

Dentro de esta tecnología, los sensores representan un papel básico en una gran cantidad de sistemas integrados IoT.

Ante los problemas que puedan surgir, por ejemplo, en el contexto de logística, el uso de IoT ofrece alternativas para solucionarlos. La principal, es la posibilidad de conectar todos los objetos de manera simple y económica a Internet, obteniendo la posibilidad de operarlos y monitorearlos desde cualquier dispositivo en cualquier parte del mundo con una simple conexión a la red, ya que toda la información y la lógica asociada radica en Internet.

Existen múltiples industrias que ya adoptaron este tipo de soluciones, como las de transporte de larga distancia (punto a punto), las cuales consiguieron optimizar las herramientas de seguridad, gestión y servicios [7] Recientemente otras industrias también han incluido el concepto de IoT, como, por ejemplo:

- **Transporte de mercancías**: para la optimización y control de entregas, para mejorar la gestión con sensores en contenedores, etc.

- **Correos:** para proporcionar información al usuario final del estado de la entrega.

- **Transporte de pasajeros**: para dar a conocer estimación de tiempos de llegada, sistemas de información en estaciones, etc.

Actualmente IoT supone una red global de información y comunicación en la que todos los objetos que nos rodean se encuentran identificados y conectados permanentemente a Internet. Tal y como se cuenta al inicio de este capítulo, con IoT, ahora las redes de sensores generan información accesible en todo el mundo con internet y suponen la pieza clave para la comunicación entre objetos y hacia internet.

2.2. Arquitectura De Sistemas IoT

A continuación, se explicará la arquitectura genérica de un sistema IoT basándonos en el siguiente esquema:

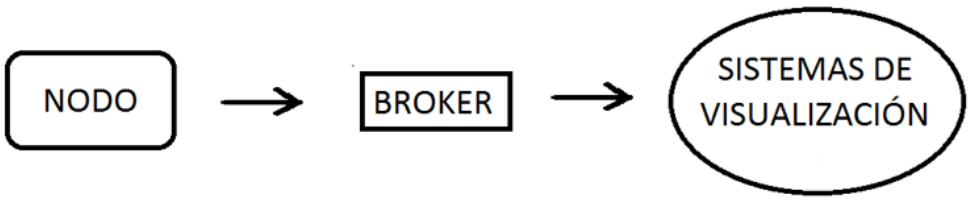

Figura 2.1 Esquema arquitectura IoT

La figura 2.1 representa un esquema general de los sistemas IoT, su arquitectura y funcionalidad del propio sistema. El esquema recoge los tres elementos principales de un sistema IoT, empezando por el nodo que recogerá la información y será enviada a un *broker* a través de un dispositivo que proporcione conectividad a nivel de red, como por ejemplo un *router*. Una vez en el *broker*, se enviarán los datos recibidos en el *broker* a los sistemas de visualización de manera que el usuario tenga acceso de forma más sencilla a los datos recogidos.

A partir del esquema visto, en las siguientes secciones se detallarán las partes que componen un sistema IoT.

2.2.1. Redes de sensores e IoT

Tal y como se detalla en la figura al principio de este apartado, el primer elemento que forma parte de la arquitectura IoT es el nodo.

Los nodos se comportan como unidades individuales que pueden recoger información, comunicarse y por supuesto, cuentan con la capacidad de conectarse a internet.

Esta parte de la arquitectura que constituyen los nodos lo podrá formar desde uno a varios nodos que interconectados entre sí inalámbricamente mediante algún protocolo formen una red de sensores. Cada nodo de la red recogerá información y se comunicará con el *broker*.

Cabe destacar los nodos capaces de leer de los sensores que son conocidos como nodos de sensor. Estos nodos son capaces de realizar algún procesamiento, reuniendo información sensible y comunicándose con otros nodos conectados en la red. Están formados por un controlador, una fuente de alimentación, un sensor y una memoria externa.

El conjunto de nodos como red nos permite monitorear múltiples variables (temperatura, energía, distancia, entre otras) y cubrir un área grande como puede llegar a ser un almacén o espacios similares donde podamos tomar la información.
Los nodos tendrán la capacidad para comunicarse con los diferentes dispositivos que están en la misma red.

En la siguiente sección, se tratará el tema de la comunicación con los nodos y las características necesarias para que sea posible el intercambio de información.

2.2.2. Comunicaciones con los nodos

Tal y como decíamos, la tecnología IoT se aplica para recoger información sobre casi cualquier cosa que queramos controlar y son las redes que hemos mencionado las que ofrecen la vía principal para que la comunicación entre objetos y hacia internet sea posible.

Ante la posibilidad de comunicación entre los dispositivos surge la necesidad de un *gateway* o puerta de enlace que haga posible la interconexión entre la red de sensores y una red. De este modo, el *gateway* actuará de interfaz de conexión entre dispositivos y también posibilita compartir recursos entre ellos.
Por lo tanto, para que IoT funcione se requiere una interfaz común donde los sistemas y los distintos dispositivos que forman IoT puedan conectarse. Para facilitar éstas comunicaciones se han desarrollado y utilizado protocolos ya existentes con los que la información se encapsula y así, facilitar su envío y recepción.

En primer lugar, podría considerarse Http como principal protocolo de comunicación, ya que es uno de los más utilizados a nivel global. Además, Http está considerado como un protocolo robusto y que puede llegar a ofrecer un alto grado de seguridad el cual puede ser útil en ciertas aplicaciones basadas en IoT.
HTTP, basado en TCP, es utilizado como protocolo de comunicación estándar entre cliente y servidor, es decir, basado en el esquema petición/respuesta. El cliente envía un mensaje de petición y el servidor contesta con un mensaje de respuesta.

A su vez, las limitaciones que presenta Http en su uso como protocolo de comunicación para la tecnología IoT hace que hayan surgido otros protocolos más adecuados para su uso en este ámbito debido a que IoT está pensado para dispositivos livianos y, por esta razón, los protocolos idóneos para usar deben ser ligeros y sencillos. Es en este punto donde nacen los protocolos CoAp y MQTT.

CoAp, basado en UDP, se diseña para trasladar el modelo HTTP manteniendo su arquitectura, pero incluyendo los requisitos mencionados anteriormente como la simplicidad, que son muy importantes para IoT. [8]

Por otra parte, MQTT, basado en TCP, se presenta como un protocolo de mensajería tipo publicación/suscripción muy ligero y pensado para dispositivos con una autonomía muy reducida ya que la mayoría de ellos llevan una batería asociada Cabe destacar que el uso del protocolo HTTP hace consumir a los dispositivos del orden de 5 veces más energía que utilizando el protocolo MQTT. Cada mensaje enviado por el cliente se envía a una dirección en concreto, denominada *topic*. Para visualizar los datos enviados a este *topic*, es necesario que el cliente se suscriba a ese canal y de esta manera poder

visualizarlos sin problema, teniendo la posibilidad de suscribirse a varios de estos *topic* y asi poder recibir mayor cantidad de información [9].

La principal diferencia existente entre estos dos últimos protocolos idóneos para dispositivos IoT (MQTT,CoAP), es que mientras MQTT está dirigido a proyectos/sistemas donde es necesaria una comunicación de n a n dispositivos, CoAP está enfocado a modelos de cliente/servidor donde el cliente notifica al servidor el estado del mismo. Por ello MQTT necesita un componente adicional que decida qué hacer y a dónde reenviar los mensajes recibidos por los clientes. Este componente adicional se denomina *broker* y es el encargado de redirigir los mensajes a través de los diferentes *topics* y de esta manera permitir la comunicación de n a n dispositivos [10].

MQTT y CoAp son protocolos mucho más ligeros, con menos bytes en el encabezado lo que les hace muy válidos para sistemas con una capacidad limitada en cuanto a batería y uso de memoria, es decir, se crean básicamente para el entorno IoT.

2.2.3. Visualización de la información

Anteriormente mencionamos que IoT nos permitía publicar datos legibles por aplicaciones. Estas aplicaciones serán una pieza clave para que el usuario final pueda comprobar los resultados de las mediciones de los dispositivos IoT.

Por tanto, estas aplicaciones y servicios de visualización tendrán su papel en el procesamiento de los datos y nos proporcionarán funcionalidades que resultarán de total utilidad a la hora de recoger, almacenar y visualizar la información.

3. SISTEMA FREEZE SENSE

Una vez explicado en el capítulo 2 en qué consiste la arquitectura IoT se mostrará a continuación como se ha adaptado esta arquitectura al sistema Freeze Sense.

En la Figura 3.1 se muestra de manera gráfica la arquitectura y la funcionalidad separadas en tres fases.

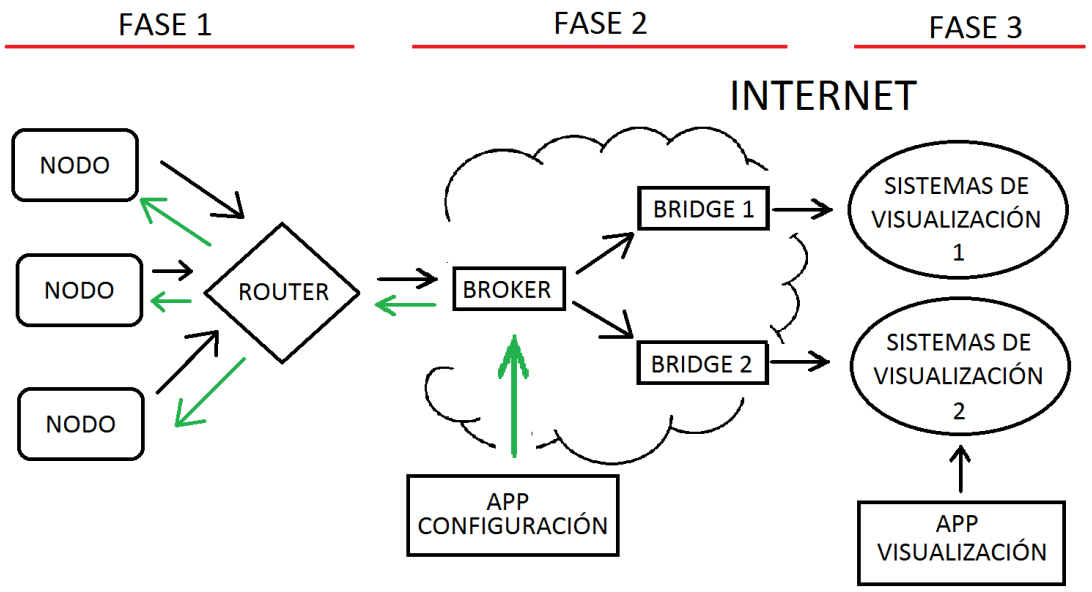

Figura 3.1 Esquema arquitectura y funcionalidad general

En la fase 1, como se observa en la Figura 3.1, el sistema Freeze Sense cuenta con una red de sensores formada por varios nodos y el *router*. Uno o varios nodos podrían representar los camiones donde el sistema ha sido instalado y en cada uno de esos camiones es necesario un *router* para las comunicaciones. Los nodos tienen un sensor de temperatura, un procesador y una batería. También poseen de una conexión Wi-Fi con la que se conectan con el *router*.

El procesador lee periódicamente la temperatura de un sensor y el módulo Wi-Fi se encarga de mandarla a un *broker*. Para hacer posible la conexión entre nodo y *broker*, es necesario contar con un *router* y que éste tenga una conexión GPRS/3G debido a la movilidad de los camiones.

Con esta arquitectura, el *gateway* no sería necesario, ya que ese papel lo hace el *router* y los nodos a través de su conexión Wi-Fi pueden mandar la información directamente al *broker*.

La fase 2, representa el servidor. El servidor tiene almacenado un servicio de un *broker* y los bridges. Existe un bridge por cada sistema de visualización que se quiera utilizar. El *broker* recibe la información de los nodos y esa información pasa a través de los bridges a los sistemas de visualización.

Por otro lado, como se muestra en la Figura 3.1 con el flujo de color verde, la aplicación de configuración, encargada de modificar los parámetros internos del nodo, manda directamente información a los nodos a través del *broker*.

Por último, en la fase 3, se encuentran los sistemas de visualización, los datos llegan desde el nodo atravesando el *broker* y su correspondiente bridge. En este punto podremos acceder a estos sistemas para visualizar la información o manipularla. También existe una aplicación para ver estos datos de una manera más accesible.

En los siguientes puntos se explicará qué arquitectura software y hardware se ha adoptado para este sistema y cómo se han tomado estas decisiones.

3.1. Arquitectura Hardware

Como hemos visto en el apartado anterior, el único punto de la arquitectura del sistema Freeze Sense que necesita componentes hardware es la red de sensores. Para que el nodo funcione necesita una serie de elementos que realicen las tareas de medición, alimentación, comunicación, etc. Este conjunto de elementos asegura un funcionamiento correcto del nodo. Estos elementos se muestran en la Figura 3.1.1:

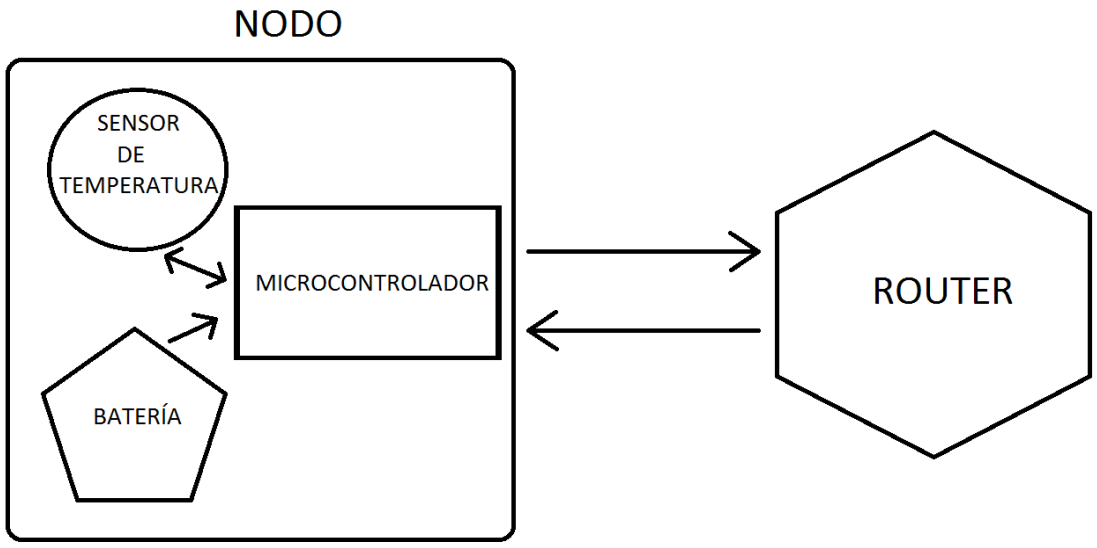

Figura 3.1.1 Esquema componentes del nodo y router

La Figura 3.1.1 representa un esquema abstracto que recalca los elementos hardware a escoger de forma separada. Estos elementos son: microcontrolador, sensor de temperatura y batería, que componen el nodo, y el *router* para poder formar una red de sensores.

Estos elementos se deben elegir de tal forma que el producto conseguido cumpla con las siguientes características:

- **Precio:** Las soluciones que se han estudiado previamente son muy caras (ver Sección 1.1), la principal característica que buscamos en los elementos que conforman el nodo es el precio.

- **Consumo:** La autonomía es muy importante en este problema, ya que los trayectos a cubrir pueden ser muy largos y es necesario un tracking completo para que nuestro proyecto sea funcional. Se buscarán elementos que tengan un consumo pequeño, ya que, a menor consumo, tendremos una mayor autonomía.

- **Desarrollo:** Se buscarán elementos en los que desarrollar el código del nodo sea más sencillo para nuestros conocimientos y que tengan un SDK que facilite el desarrollo.

- **Comunicación:** Es un requisito del proyecto que la comunicación en la red de sensores sea Wi-Fi como una prueba de concepto.

En las siguientes secciones se describe cómo se han elegido todos los elementos descritos anteriormente.

En primer lugar, se empezará por seleccionar el microcontrolador del nodo ya que es el elemento que puede limitar en mayor medida debido a su conectividad (es el encargado de hacer la conexión con el sensor y con el *router*) y a su precio ya que es el elemento más caro.

Después se seleccionará el sensor de temperatura, el cual vendrá dado por la conectividad del microcontrolador. Por otro lado, la batería, debe tener un precio reducido y ser pequeña para no aumentar el tamaño del nodo, pero sin sacrificar en gran medida su autonomía y de esta manera soportar un tracking completo de la cadena del frío.

Por último, se elegirá el *router* que debe ser de diseño industrial y con conectividad GPRS/3G.

3.1.1. Selección Del Microcontrolador

En este apartado se va a describir el proceso de selección entre los diferentes microcontroladores adecuados para la realización del proyecto. Este proceso de selección se llevó a cabo teniendo en cuenta los diferentes aspectos citados a continuación:

- **Firmware/SDK**: Se tendrá en cuenta el firmware disponible para el microcontrolador, ya que, en función del firmware, cambia el lenguaje de programación y también se tendrá en cuenta el SDK del firmware, ya que facilita el desarrollo. También son importantes los entornos de desarrollo, herramientas para el flasheo del firmware, conexión con el microcontrolador, guías y comunidad.

- **Conectividad:** Es necesario que el microcontrolador disponga de conectividad Wi-Fi ya que es un requisito y un objetivo que los nodos se comuniquen por esta tecnología.

- **MicroProcesador:** Capacidad de procesamiento y almacenamiento disponible. No es necesaria mucha potencia, ya que el programa a ejecutar no la necesita. También se tendrán en cuenta que tenga un bajo consumo.

- **Voltaje:** Rango de voltaje en el que trabaja el microcontrolador. Información necesaria para hacer una selección posterior de la batería.

- **Placa de prototipado**: Que cuente con placas comerciales de desarrollo que faciliten el desarrollo del prototipo.

En la tabla 3.1.1. podemos ver las características descritas anteriormente de una preselección de tres microcontroladores que cuentan con Wi-Fi de bajo consumo, un procesador de bajo consumo y con placas de desarrollo comerciales:

	Spark P1	**CC3200**	**ESP 12**
Conectividad	Wi-Fi: 802.11 b/g	Wi-Fi: 802.11 b/g/n	Wi-Fi: 802.11 b/g/ n
Consumo	Consumo normal: 250mA Consumo sleep: 3.2uA	Consumo normal: 190 mA Consumo sleep: <4 µA.	Consumo normal: 170 Ma Consumo SLEEP: <5uA
Voltaje de entrada	3,6 a 6,0 V	2,1 a 3,6 V	3,0 a 3,6V
Buses	*SPI, I²C*	*SPI, I²C*	*SPI, I²C, 1-Wire*
Firmware/SDK	*Particle device firmware/ particle dev*	*Free RTOS/ OpenSDK* *TI RTOS/ CC3200SDK* *WiPy basado en microPhyton/*	*NodeMCU: firmware basado en eLua/ SDK espressif* *Arduino/SDK espressif* *Microphyton/ esp-open-sdk* *freertOS/esp-open-sdk*
Precio	*12€*	*7€*	*2€*

Tabla 3.1.1 Características principales de microcontroladores

De los tres microcontroladores expuestos se ha optado por seleccionar el ESP 12[11], ahora se van a exponer las razones por las cuales se ha llevado a esta decisión

La razón principal por la que se ha seleccionado este microcontrolador es el precio, que es mucho menor que el de los otros microcontroladores y, por tanto, lo que marca la diferencia.

Por otra parte, aunque los tres microcontroladores son muy similares en cuanto a consumo, el ESP12 destaca por la inclusión del bus 1-wire el cual se adapta de mejor manera al proyecto debido a su bajo coste de implementación y que está pensado para sensores de temperatura como el que se va a utilizar. Respecto a SPI e I^2C, ambos son válidos, pero el primero está pensado para sensores que necesiten mucha velocidad y el segundo es más caro y más costoso de desarrollar puesto que necesita direccionamiento y arbitraje en el bus.

Al tener más buses hay una limitación menor para elegir los sensores. La conectividad de esta placa ha sido un factor determinante para su elección, ya que nos permite interactuar con más periféricos de distintas maneras y nos limita menos a la hora de elegir el sensor. Respecto al procesador, el ESP12 monta uno a 80 MHz lo que garantiza un consumo bajo y un rendimiento suficiente.

Por último, también hay que tener en cuenta que el ESP 12 tiene un gran número de versiones de firmware disponible, por lo que se podría cambiar el desarrollo si se encontrara algún problema. Dentro del firmware, destaca NodeMCU [12], que está basado en eLua y es el más utilizado. El lenguaje utilizado para el desarrollo en NodeMCU es Lua y dispone de una gran cantidad de documentación que nos facilitará el desarrollo y una comunidad muy activa que podría ayudarnos en los problemas que surjan durante el desarrollo.

Respecto a las placas de desarrollo para el ESP12 podemos decir que cuenta con diferentes modelos que facilitan la labor del desarrollador. En este caso, se ha utilizado la placa NodeMCU ya que se decidió desarrollar sobre este firmware y lo traía preinstalado, además de tener un puerto microusb que facilita la conexión.

También, se utilizó la placa de desarrollo Adafruit HUZZAH [13] cuya única diferencia con NodeMCU [14] es no incluir el puerto microusb.

3.1.2. Selección Del Sensor De Temperatura

Una vez elegido el microcontrolador, se analizaron una serie de sensores que más se adaptaban a las características de éste [15]. El proceso se llevó a cabo teniendo en cuenta además los siguientes factores:

- **Rango de temperatura:** Debían cubrir las temperaturas necesarias para la cadena del frío, las cuales se encuentran entre los -18ºC (esta temperatura es la fijada como estándar de congelación para la cadena de frío internacional) hasta los 8ºC.

- **Digital:** Proporciona la información procesada a través del bus y permite dormir el procesador sin que el nodo esté dormido. Además, los termómetros analógicos son más susceptibles al ruido.

Después de la pre selección de sensores que valían para alguno de los buses del microcontrolador preseleccionado, con un coste bajo y encontrándose en el rango de temperaturas necesario, se exponen a continuación los parámetros que nos ayudaron a la elección final del sensor.

	ADT7420	**ADT7310**	**AD7814**	**TMP04**	**DS18B20**
Rango de temperatura	*-40ºC/150ºC*	*-55ºC/ 150ºC*	*-55ºC/ 125ºC*	*-40ºC/150ºC*	*-55ºC/ 125ºC*
Salida	*Digital*	*Digital*	*Digital*	*Digital*	*Digital*
Consumo	*2,7V-5,5V*	*2,7V-5,5V*	*2,7V-5,5V*	*4,5V-7 V*	*3V-5,5V*
Protocolo	*I²C*	*I²C*	*I²C*	*I²C*	*1-Wire*
Precio	*3.10$*	*1.32$*	*1.00$*	*3.53$*	*2.13$*

Tabla 3.1.2 Sensores de Temperatura

Al final, optamos por elegir el sensor DS18B20 principalmente porque cumple con los requisitos que tenemos y, aunque no es el más barato, al utilizar el protocolo 1-Wire no se necesitan tantos componentes adicionales para realizar la conexión con el chip ya que únicamente posee un canal. Por el contrario, los sensores I²C poseen dos canales y, por tanto, necesitan más componentes, lo que supone un incremento en el precio final.

Como se puede ver en Figuras 3.1.2 y 3.1.3, el sensor 1-Wire solamente necesita una resistencia para conectarse mientras que los sensores I²C necesitan dos resistencias para cada línea de señal que tienen. [16]

- *Con suplementos de energía externo:*

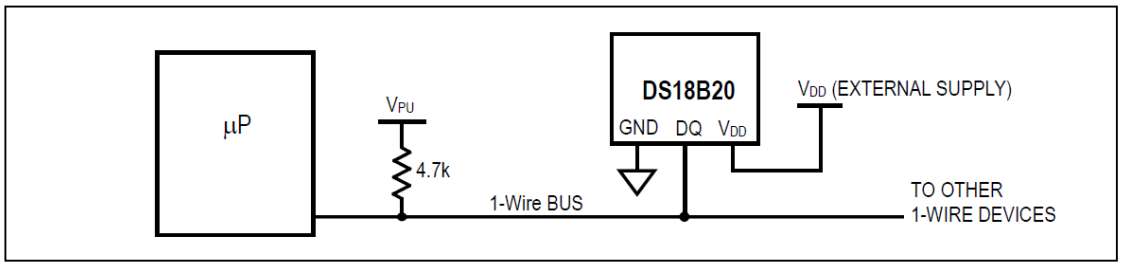

Figura 3.1.2 Esquema sensor de temperatura

- *Modo parásito:* se puede alimentar a través del bus de datos sin necesidad de un suministro externo de energía.

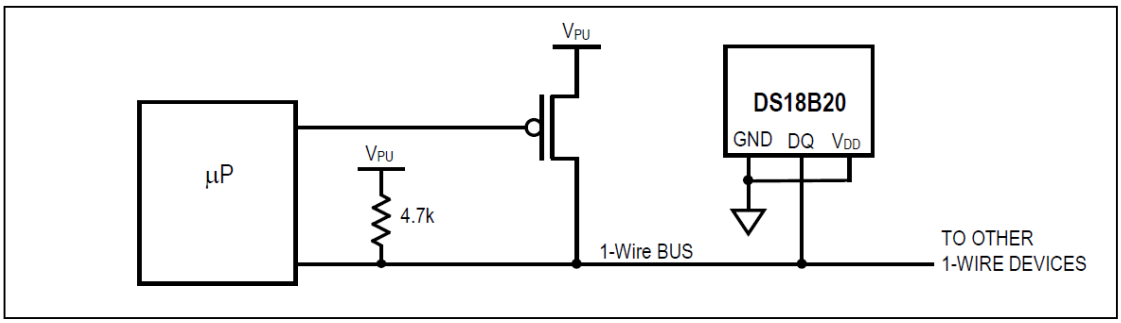

Figura 3.1.3 Esquema sensor de temperatura modo parásito

3.1.3. Selección De Batería

En este apartado se va a describir el proceso de selección entre los diferentes tipos de baterías-pilas adecuadas para la realización del proyecto. Este proceso de selección se llevó a cabo teniendo en cuenta por un lado los principales objetivos del proyecto (sobre todo coste y tamaño) y por otro lado la duración como una de las principales características. La duración va fuertemente ligada al coste de las pilas. Por ello se seleccionaron pilas que tenían una menor capacidad de carga. Como se demostrará en las pruebas de rendimiento de la pila (Ver Sección 4.2) esta selección de pilas de menor coste y por tanto menor duración no afectará en gran medida al prototipo ya que el nodo, combinado con un código que exprime al máximo la eficiencia energética, consigue unos resultados bastante buenos en cuanto a tiempo de vida de la batería y por tanto autonomía de nuestro prototipo.

En la Tabla 3.1.3. se muestra un resumen en el que aparecen los modelos que se analizaron previamente al inicio del desarrollo del proyecto, así como las principales características de cada uno de ellos.

	CR2032	**CR2450**	**LIR2450**
Voltaje	*3V*	*3V*	*3,6V*
Capacidad	*210 mAh*	*610 mAh*	*120 mAh*
Diámetro	*20 mm*	*24,5 mm*	*24,5 mm*
Recargable	*No*	*No*	*Sí*
Precio	*0.95$*	*1.95$*	*2.95$*

Tabla 3.1.3 Modelos de Baterías/Pilas

Todos los modelos que aparecen en la Tabla 3.1.3. pertenecen a la familia de pilas de tipo "botón" siendo el Litio su principal material de construcción. Se descartó desde el primer momento la selección de baterías (por ejemplo, Li Polymer 803860 [17]) con mayor capacidad (2000mAh) ya que todas ellas tenían un tamaño mucho mayor al de los modelos que aparecen en la Tabla 3.1.3. afectando a uno de los objetivos principales del proyecto citados en la sección 2 (facilidad de intercambio entre cámaras o palés), pero sobre todo se rechazaron por el coste tan elevado que tienen con respecto a las pilas de botón (del orden de 10$ mayor). Además, este ahorro en el coste de las baterías/pilas fue posible ya que debido al consumo tan bajo del prototipo desarrollado se puede asegurar una funcionalidad completa durante un largo periodo de tiempo (ver sección 4).

Como se puede comprobar en la Tabla 3.1.3., todos los modelos seleccionados para su posible implementación en el prototipo tienen características muy similares en cuanto a tamaño y material del que están fabricados.

En primera instancia se seleccionó el modelo CR2032 [18] ya que ofrece una capacidad suficiente para una gran autonomía del prototipo con el coste más reducido de los 3 modelos que aparecen en la Tabla 3.1.3. además de ser el modelo que tiene la menor dimensión de las otras 2. Tras la selección de esta pila, se realizaron una serie de pruebas iniciales para comprobar el funcionamiento del dispositivo con una alimentación diferente a la que ofrecía el puerto USB de la placa de desarrollo NodeMcu.

El código que se utilizó para estas pruebas se corresponde al desarrollado en la fase 2 del proyecto (ver sección 2). Se comprobó que en el momento que el chip ESP12 (junto con la placa de desarrollo NodeMcu) intentaba realizar una conexión Wi-Fi al *router* configurado, el chip se reiniciaba automáticamente. Como se puede comprobar en la Tabla 3.1.1 de las características del ESP12, el chip requiere de una tensión de 3V de alimentación para que funcione correctamente. Esta tensión es la que la pila podría suministrar sin problemas, pero probablemente al realizar la conexión Wi-Fi, el chip requiere de mayor corriente, haciendo que la tensión que pueda suministrar la pila descienda mucho más del mínimo que necesita el propio chip para su funcionamiento.

Es más que probable que los reinicios incontrolados a la hora de realizar la conexión Wi-Fi fueran producidos por el problema explicado anteriormente.
Por esta razón, finalmente se decidió seleccionar el modelo LIR2450 [19] ya que era de las pocas pilas que ofrecían 3.6V en un tamaño de pila tipo "botón" a un coste reducido.

Nota: este modelo de pila al ser recargable nos permitía reutilizarla multitud de veces y de esta manera no tener que adquirir un gran número de pilas no-recargables para las pruebas realizadas. En caso de lanzarse un producto al mercado basado en este prototipo, no sería requisito indispensable la capacidad de recarga llevando consigo un ahorro de coste con respecto a la pila.

3.1.4. Selección Del Router

En este apartado se va a describir el proceso de selección del *router* que cumple el papel de intermediario de las comunicaciones. Se encarga de conectar los nodos con el *broker*. Para ello, el *router* estará en la cabina del camión proporcionando una red wifi a los nodos. También es necesario que el *router* disponga conectividad a redes móviles para poder enviar los datos en tiempo real a los servidores. Este proceso de selección se llevó a cabo teniendo en cuenta los diferentes aspectos citados a continuación:

- **Durabilidad:** Al usarse en un entorno industrial con mercancía que se mueve en grandes bloques muy rápidamente, es necesario que el *router* esté fabricado con materiales duraderos y resistentes que garantice su durabilidad. También se tendrá en cuenta un tamaño compacto para ocupar el mínimo espacio.

En la Tabla 3.1.4. Se pueden ver las características descritas anteriormente de una preselección *routers* que cuentan red wifi de bajo consumo y conexión GPRS:

	Teltonika RUT500	**Robustel M1000 Pro**	**Siretta Mica**
Conectividad	*Wi-Fi LTE, DC-HSPA, EDGE, GPRS*	*Wi-Fi LTE, DC-HSPA, EDGE, GPRS*	*Wi-Fi LTE, DC-HSPA, EDGE, GPRS*
Durabilidad	*Carcasa de aluminio*	*Carcasa de aluminio*	*Carcasa de aluminio*
Accesorios	*Antena externa Wi-Fi y antena externa para las conexiones móviles*	*Antena externa Wi-Fi*	*Antena externa Wi-Fi y antena externa para las conexiones móviles*
Precio	*110€*	*214€*	*270€*

Tabla 3.1.4 Modelos de Router

Como se puede comprobar en la Tabla 3.1.4, todos los modelos, tienen características muy similares.

Los *routers* seleccionados están diseñados para uso industrial, por lo que todos tienen un cuerpo de aluminio que les permite resistir golpes y poder funcionar en entornos hostiles. En cuanto a conectividad, no ha sido un factor determinante a la hora de elegir ya que todos presentan una conectividad similar.

Al final, elegimos el *router* Teltonika RUT500 [20] porque además de ser el más barato de los que se analizaron, incluye una mayor cantidad de accesorios que los demás, siendo estos necesarios en el proyecto, como la antena externa para las conexiones móviles.

3.2. Arquitectura Software

En este capítulo se explicará la arquitectura software que se ha desarrollado/aplicado para que el funcionamiento del sistema cumpla con los objetivos marcados en la Sección 1.

3.2.1. Nodo

Una vez explicados cada uno de los elementos funcionales seleccionados para la creación de este prototipo es necesario mostrar detalladamente la arquitectura software y los modos de funcionamiento creados para el nodo, entendiendo el nodo como el conjunto que forman el chip, la pila y el sensor de temperatura.

Modos de funcionamiento

El nodo dispone de 2 modos de funcionamiento totalmente diferentes. Por un lado, el modo de funcionamiento "setup" que servirá para configurar diferentes parámetros del nodo explicados con más detalle a continuación. Por otro lado, se ha desarrollado el modo de funcionamiento "por defecto" que hará uso de los parámetros de configuración mencionados anteriormente para controlar la cadena del frío en los transportes.

a) Modo Setup

Para mejorar la autonomía y la versatilidad del prototipo se decidió desarrollar una serie de parámetros totalmente configurables por parte del usuario final a través de una de las aplicaciones desarrolladas (Ver Sección 3.2.5, Aplicaciones). De esta manera el usuario final podría aumentar la autonomía del dispositivo además de poder adaptar el funcionamiento del dispositivo a diferentes tipos de transportes o situaciones aumentando de esta manera la portabilidad del dispositivo. El acceso a este modo de configuración del dispositivo se realiza automáticamente por parte del dispositivo sin ser necesario la conexión USB con el mismo. En el momento en el que el dispositivo detecta que existe una red Wifi al alcance con el nombre "setup", se conecta con el servidor a la espera de la configuración de los parámetros. Por ello el operario que instalará los nodos en la cámara frigorífica debe habilitar una red WiFi con ese nombre (puede hacerlo con

su propio dispositivo móvil) para configurar los parámetros requeridos por el cliente. En caso de que no se realice esta configuración, el dispositivo tomará valores por defecto para los parámetros. Los parámetros configurables son los siguientes:

1. **Nombre Red Wifi:** nombre de la red WiFi a la que se conectará el dispositivo.

2. **Contraseña Red Wifi:** contraseña de la red WiFi.

3. **Umbral RSSI:** con este parámetro se indica cual es la intensidad de la señal WiFi mínima para que el dispositivo envíe datos al servidor. Con este parámetro se puede aumentar la autonomía del dispositivo ya que podemos ahorrar conexiones del nodo al *router* en situaciones en las que la señal Wi-Fi no es la adecuada y por tanto consumiendo más batería de la necesaria.

4. **Tiempo del dispositivo en estado "dormido":** el usuario indica en segundos el tiempo el cual el dispositivo se mantendrá en estado "dormido" y por tanto consumiendo el mínimo de batería. Este parámetro es práctico ya que dependiendo del tipo de alimento transportado el usuario final querrá obtener datos entre períodos más cortos o más largos de tiempo.

5. **Número de envíos secuenciales:** este parámetro permite al usuario seleccionar el número de muestras de temperatura que se enviarán en cada conexión al servidor. Por ello el dispositivo únicamente enviará las temperaturas almacenadas internamente en el nodo cuando la cantidad sea igual a este parámetro, evitando que el dispositivo se conecte al servidor cada vez que recoge una muestra de temperatura y de esta manera reducir el consumo energético del mismo en gran medida.

6. **Tiempo Reintento en caso de que la señal Wi-Fi sea menor que parámetro Nº3 (Umbral RSSI):** en caso de que el dispositivo haya comprobado que la señal WiFi es peor que la determinada por el parámetro Umbral RSSI (Nº3), cancelará la conexión al servidor y pasará a estado dormido durante un periodo de tiempo indicado por este parámetro. Por tanto, reintentará la conexión y por tanto el envío de los parámetros una vez pasados los segundos configurados mediante este parámetro.

Una vez configurados estos parámetros, el dispositivo automáticamente entrará en el modo "normal" iniciando el proceso de control de la cadena del frío.

b) Modo por defecto de funcionamiento:

Una vez configurados los parámetros explicados anteriormente, el dispositivo activa el modo de funcionamiento "por defecto". El flujo de este modo de funcionamiento se explica de manera gráfica en la Figura 3.2.1.

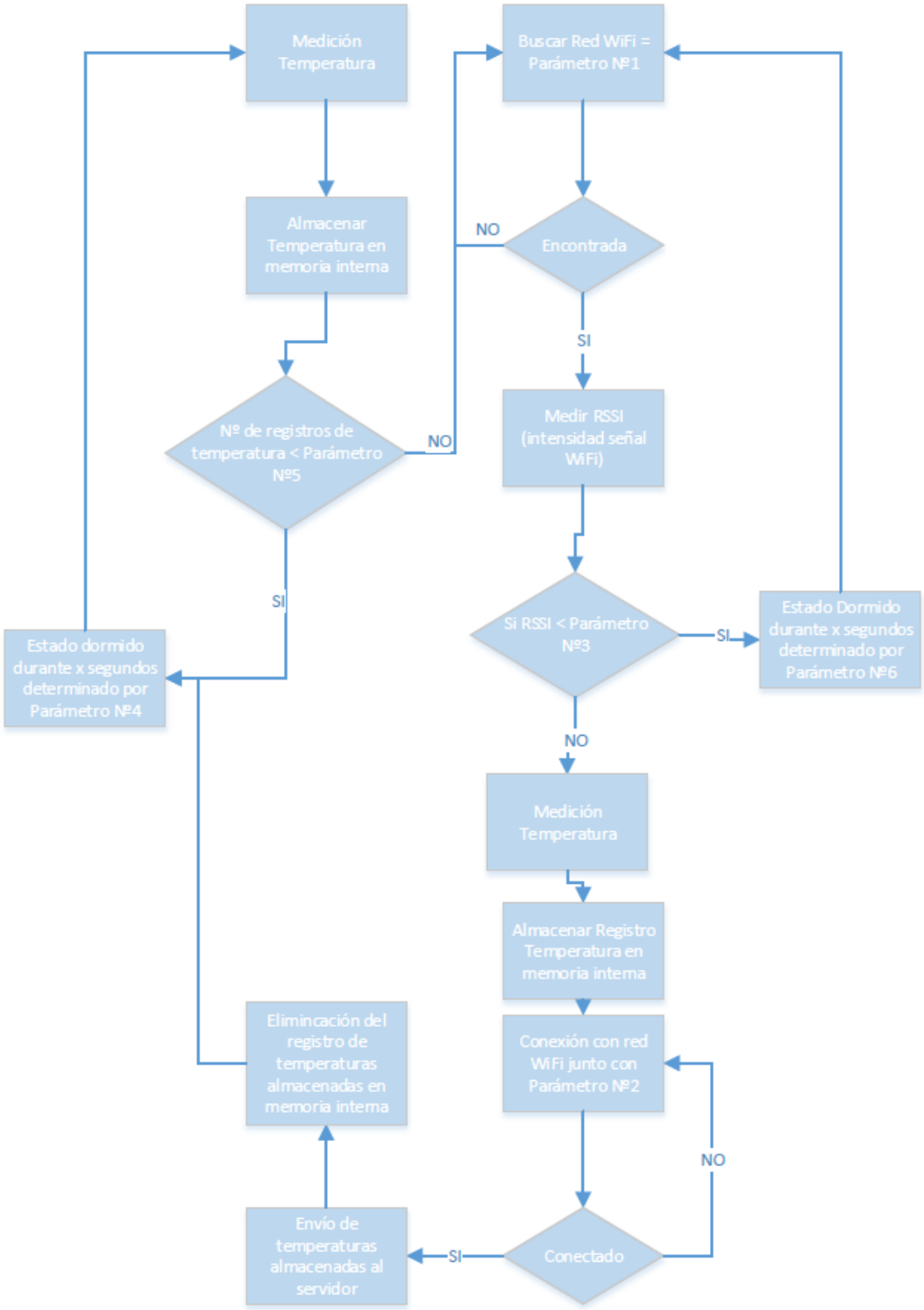

Figura 3.2.1 Diagrama de Flujo del nodo

Arquitectura Código desarrollado

Como se explicó en puntos anteriores (ver Sección 3.1.1), finalmente se escogió el firmware NodeMcu principalmente por la cantidad de documentación disponible y su facilidad de uso mediante herramientas de desarrollo específicas para este firmware escogido. El firmware NodeMcu está basado en eLua, el cual es una versión modificada del lenguaje Lua original optimizado principalmente para sistemas o dispositivos muy limitados con respecto a memoria RAM. Este firmware actúa como una capa adicional al SDK del propio chip ESP8266 evitando de esta manera ciertas incompatibilidades entre módulos del lenguaje Lua y algunas de las versiones del chip ESP8266. Una de las principales características de este SDK es que la programación debe realizarse por eventos. El principal problema que conlleva es que en el caso de que alguna de las tareas (asíncronas) pendientes de ejecutar por el SDK tarden demasiado en finalizar (en concreto 10ms), como por ejemplo la conexión Wi-Fi, el SDK automáticamente ejecuta la siguiente tarea pendiente pudiendo ocasionar graves errores en tiempo de ejecución del código. Para mitigar este problema se tuvo que desarrollar el código de manera que pudiera gestionar correctamente los callback de las funciones asíncronas y así evitar errores no previstos en tiempo de ejecución.

Para organizar el código desarrollado y facilitar de esta manera la depuración y legibilidad del código se estructuró en una serie de módulos/ficheros los cuales se encargaban cada uno de ellos de una tarea o funcionalidad concreta. Además, como se ha mostrado en la sección 2 (Fases del proyecto), se ha seguido un desarrollo incremental en el proyecto, es decir, en cada uno de los bloques temporales se han ido añadiendo nuevas funcionalidades las cuales, se corresponden prácticamente a cada uno de los módulos/ficheros de código desarrollados. Los módulos son los siguientes:

- Init
- Setup
- MqttFile
- Ds18b20
- Config
- Application

Init

Es el primer fichero que ejecuta automáticamente el firmware NodeMcu. Se encarga de inicializar desde el sistema de ficheros algunas de las variables globales como por ejemplo el nombre de la red Wi-Fi y su contraseña. Además, se encarga de comprobar si es necesario o no configurar los parámetros de configuración (Wi-Fi, contraseña, etc) y en función del resultado ejecuta el módulo mqtt o setup respectivamente

Setup

Este módulo se encarga de almacenar en variables globales los parámetros que no se cargaron en el anterior módulo init (Tiempo de espera entre muestreos, umbral de intensidad Wi-Fi, número muestreos de temperatura enviados en cada conexión al servidor). Dependiendo de los valores de umbral de intensidad Wi-Fi y del número de muestreos secuenciales, cargará el módulo ds18b20 o permanecerá dormido durante un

tiempo determinado por uno de los parámetros configurables a través de la aplicación móvil desarrollada.

Ds18b20

Realiza la comunicación entre el sensor de temperatura y el chip además del almacenamiento en un fichero interno de la temperatura tomada.

MqttFile

Este módulo es el responsable de las conexiones bidireccionales entre nodo y servidor. Por un lado se encarga de enviar al servidor las temperaturas almacenadas en el fichero interno y por otro lado de la conexión y recepción de los parámetros enviados desde el servidor al nodo. Este es uno de los módulos más importantes junto con el módulo init ya que es el que más errores puede generar en tiempo de ejecución. Como se ha explicado anteriormente, en caso de que alguna de las instrucciones enviadas al SDK tarde más de 10ms en terminar de ejecutarse, automáticamente salta a la siguiente instrucción en espera. En caso de que no se haga una buena gestión de *callbacks* asociados a estas instrucciones/funciones puede llevar a un error grave en el nodo obligándolo a reiniciarse o incluso formatearse. Como este módulo se encarga de la conexión TCP bidireccional entre servidor y nodo es muy propenso a este tipo de errores en caso de que la calidad de conexión no sea la adecuada y tarde más de lo debido. Por ello se ha realizado una buena gestión de *callbacks* asociados a las funciones específicas del protocolo MQTT consiguiendo mitigar al máximo este tipo de errores.

Config

Módulo responsable de almacenar en variables globales algunos parámetros por defecto que necesita el nodo en caso de que no pueda cargarlos o incluso no los tenga ni siquiera almacenados en alguno de los ficheros internos del nodo. Estos parámetros son el *topic* genérico donde enviar las temperaturas al servidor.

3.2.2. Protocolo De Aplicación Y Broker

Para las comunicaciones del sistema es necesario elegir un protocolo. En función del protocolo utilizado se elegirá el *broker*, ya que cada protocolo tiene su *broker* específico. Los protocolos estudiados en el capítulo 2 son válidos, pero es necesario elegir uno.
Puesto que HTTP, como se dijo en el capítulo 2 es mucho más pesado, la elección final estaría entre MQTT y CoAP. MQTT ofrece unas ventajas de cara al prototipado que CoAP no, aunque los dos tienen soporte en el SDK por lo que no es un factor diferencial, la gran diferencia entre ellos es la comunidad ya que es el protocolo más utilizado y documentado con una gran cantidad de ejemplos.
Además, existen *brokers* gratuitos y libres que pueden ser instalados en un servidor propio o utilizar uno preinstalado en la nube como AWS IoT o IBM Bluemix.

Broker utilizado

El sistema Freeze Sense utiliza Mosquitto [21], que es un bróker *open source* que implementa las versiones de protocolo MQTT.

Mosquitto es muy ligero, por lo que la carga de trabajo en el servidor es muy pequeña. Además de las características del protocolo MQTT también permite el uso de usuarios y la autenticación de los mismos, es decir, cuando un usuario quiere acceder al *broker* debe tener una contraseña asignada y los mensajes que envíe son registrados con su usuario. Para aumentar la seguridad, encripta el archivo donde están los usuarios y las contraseñas. Durante el desarrollo se han utilizado dos tipos de usuarios:

- ESP: cuando la conexión se ha hecho desde los nodos conectados.
- PC: cuando se han hecho pruebas desde otros dispositivos para agilizar el desarrollo.

La configuración del *broker* Mosquitto es sencilla, ya que consiste en editar un fichero de configuración con los parámetros que se consideren oportunos. La configuración utilizada contiene parámetros como, por ejemplo: no permitir recibir información de usuarios que no hayan sido registrados o mantener el sistema de persistencia activa para el modo de configuración de los nodos y el número de mensajes con persistencia que admite el *broker*. De ésta manera se pueden mandar varios mensajes y una vez que un usuario se suscriba recibirá esos mensajes.

3.2.3. Sistemas De Visualización

Existen multitud de plataformas que se pueden utilizar para la visualización y el tratamiento de datos. En este proyecto se han utilizado las siguientes:

ThingSpeak

ThingSpeak [22] es una plataforma web que permite la recolección de datos, su almacenamiento, su análisis y ejecutar acciones con *applets*, a través de un sistema de canales públicos o privados donde se publican los datos.

Esta plataforma monitorea datos que han sido recibidos en tiempo real. ThingSpeak crea una gráfica con la información recibida y permite exportar dicha información para su análisis con MatLab, éste es su gran potencial ya que MatLab contiene un gran número de herramientas que permiten la manipulación y el análisis de datos.

En cuanto a los *applets*, le permiten realizar acciones en función de los datos recibidos, es decir, puede realizar una acción que haya sido prefijada si recibe un dato concreto. Por ejemplo, se puede prefijar que cada vez que reciba el número 7, publique en Twitter que ha recibido ese número.

Para poder interactuar con ThingSpeak es necesario crear una cuenta. Esta cuenta te permite crear un canal, para conseguir el envío de información, ThingSpeak proporciona un número (*APIKEY*) que hace referencia a un único canal. Dentro de este canal, existe un formato de campos (*fields*) que son los responsables de separar la información que se recibe en un mismo canal.

Debido a la intervención de colaboradores de la facultad de agrónomos, se aconsejó el uso de una superficie donde almacenar los datos y donde se pudieran exportar esos datos a MatLab, ya que es la plataforma que se suele utilizar para el análisis de los datos en este tipo de proyectos. Thingspeak permite el almacenamiento de esos datos en la nube y permite el uso de esos datos con MatLab como se nos había recomendado.

La configuración utilizada para el proyecto se ha centrado en dedicar dos campos para cada nodo, uno para la temperatura y otro para el RSSI, es decir, de cada nodo se registra el RSSI y la temperatura. En la Figura 3.2.2 se puede ver como Thingspeak muestra los datos recibidos de un nodo para los campos RSSI y temperatura y los pone en formato tabla y se parados en campos. La información de cada campo puede ser exportada de forma independiente a MatLab.

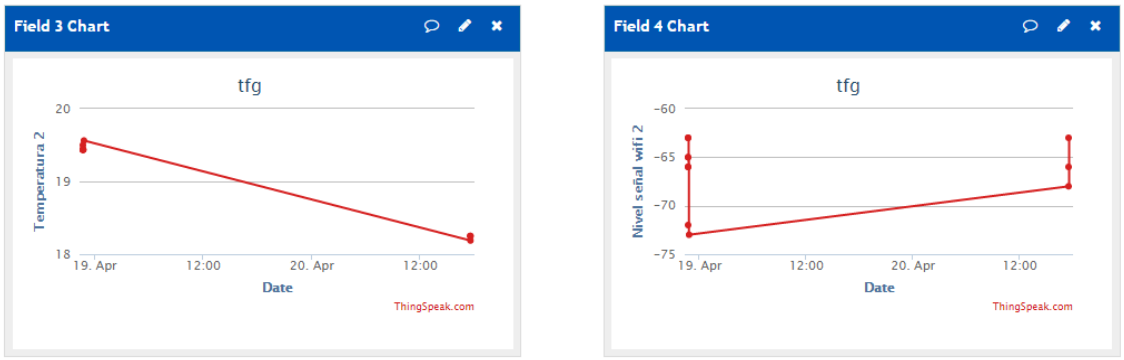

Figura 3.2.2 Gráfica en ThingSpeak

Emoncms

Se trata de una aplicación web *open source* que nos permite procesar y visualizar la temperatura, la energía, humedad y demás datos ambientales. Un añadido muy importante es que permite crear tu propio servidor Emoncms [23].

Emoncms funciona con un sistema de entradas. Cada unidad que manda información hacia Emoncms crea una entrada con una clave y un número de nodo. Con estas entradas podemos crear fuentes. Una fuente consiste en exponer la información de una manera distinta, es decir, una gráfica de puntos o de barras, la media, etc. Por último, está el apartado de *dashboard*, aquí se pueden poner las diferentes fuentes en distintos formatos en forma de widgets, como un indicador con la media de algún dato concreto o un reloj que indique el estado actual. Todos los datos que maneja Emoncms se guardan en una base de datos SQL que es accesible para el usuario.

Se decidió utilizar Emoncms ya que proporciona una visualización más amigable de los datos y un procesamiento más avanzado, además como se verá más adelante la aplicación responsable de mostrar los datos se comunica con éste.

El punto diferencial de Emoncms es la capacidad de poder instalarlo en un servidor propio, los datos de la cadena del frío son información sensible y es importante que el flujo de información no pase por terceros.

Para la configuración de Emoncms primero es necesario instalarlo en un servidor propio o utilizar la página de Emoncms, para el sistema Freeze Sense se instaló en el mismo servidor que el *broker*. Una vez Emoncms está funcionando, es necesario registrarse con un usuario y una contraseña para proteger nuestros datos. La configuración interna de Emoncms se divide en los tres apartados antes descritos: Entradas, fuentes y *dashboard*.

Las entradas se generan una vez que mandemos un dato con un nodo y una clave. En éste punto se elige como se quiere tratar la información y eso genera una fuente. En nuestro caso se guarda como una lista que utiliza MYSQL y es en tiempo real. Un ejemplo de entradas en la Figura 3.2.3.

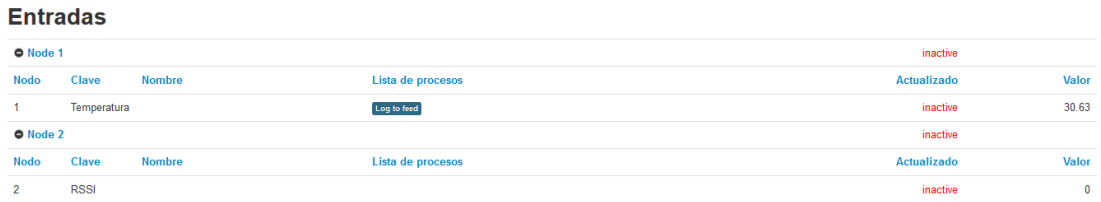

Figura 3.2.3 Entradas de Emoncms

Las fuentes se generan en función de cada uno de los tratamientos de los datos que hemos hecho en las fuentes. Éste apartado también permite descargar una tabla Excel con los datos de la gráfica. En nuestro caso tenemos una fuente con la temperatura del nodo con el formato de la Figura 3.2.4.

Figura 3.2.4 Fuentes Emoncms

Por último, está el *dashboard* y podemos configurarlo a nuestro gusto con widgets y tablas. En nuestro caso tiene una tabla y un número con la última temperatura, como se puede apreciar en la Figura 3.2.5.

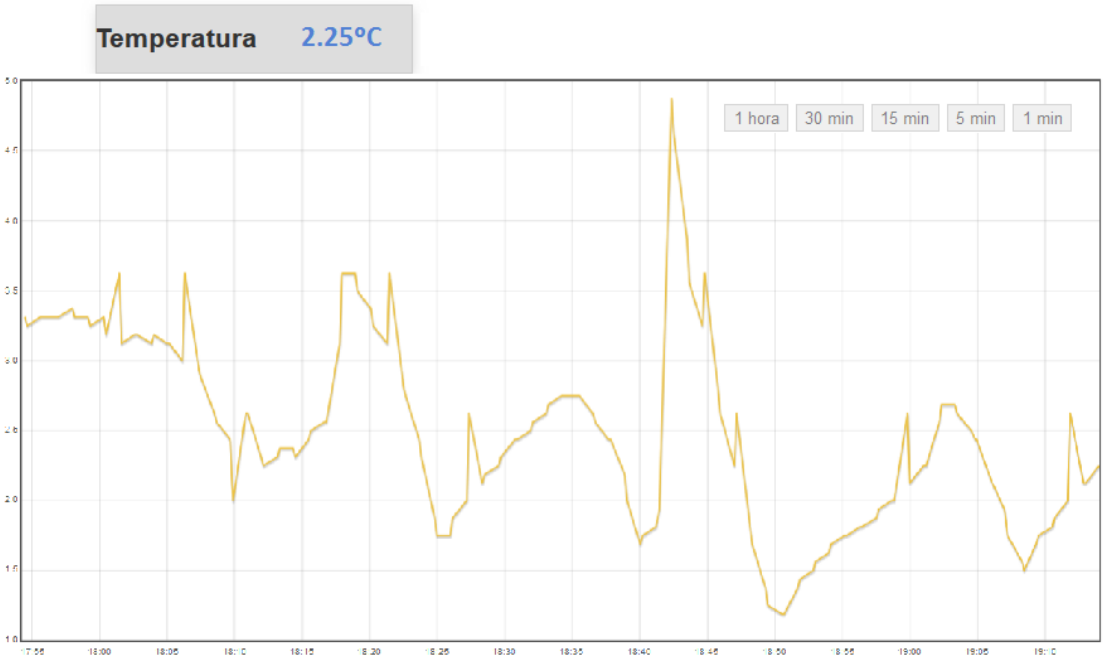

Figura 3.2.5 Dashboard Emconcms

3.2.4. Bridges

Los sistemas de visualización necesitan información para poder funcionar, es decir, es necesario que los datos que llegan al *broker* lleguen también a estos sistemas. Por este reenvío de datos se ha decidido utilizar dos programas.

1. Bridge a Thingspeak

Se ha utilizado el programa mqspeak [24] de código abierto que hace el bridge entre el *broker* y el servicio ThingSpeak.

Mqspeak está programado en el lenguaje de programación Python y hace uso de la librería paho.mqtt que implementa un cliente MQTT y sirve para poder suscribir o publicar en un bróker MQTT.

Su funcionamiento comienza con el parseo de un archivo de configuración que se separa en dos fases:

- **Configuración del bróker**: En esta fase se parsea la configuración para poder conectarse al bróker y a los posibles *topics*. Son necesarios la dirección, el puerto, usuario, contraseña y los *topics*. A cada una de estas configuraciones se le asigna una etiqueta, que más tarde nos será de ayuda para asociarlo a un canal.
- **Configuración de canales:** En esta fase se parsea la configuración del servidor ThingSpeak. Para ello es necesario identificar la *key* (clave para poder escribir sobre el canal creado anteriormente), el *field* destino y la etiqueta del *topic* que hemos puesto anteriormente.

Después de parsear el archivo crea un hilo distinto para cada uno de los *topics*, ya que tienen que estar escuchando continuamente para poder publicar los datos en thingspeak

Cuando se recibe un dato en un *topic*, se procede a su reenvío al servidor thingspeak. Para enviar los datos a través de internet, se utiliza la librería urllib y los datos de direccionamiento parseados además de la "*key*" del canal. Para encapsular la información se utiliza la librería httplib. Esta librería permite crear un paquete que se conecte con la dirección api.thingspeak.com en el puerto 80 y se envíe.

2. Bridge Emoncms

Se ha utilizado el programa mqttwarn [25] de código abierto que hace el bridge entre el *broker* y multitud de servicios entre ellos EmonCms.

El bridge está programado en el lenguaje de programación Python. Y hace uso de las librerías importadas de paho.mqtt.

Mqttwarn hace un parseo del fichero de configuración que contiene un sistema de etiquetas y destinos, que permite asociar una etiqueta a un destino. Primero parsea la información del *broker* y la etiqueta, en este caso como "emoncmbridge" ahora cada vez que queramos referirnos a esa conexión lo haremos con ese nombre. También es necesario crear unos campos para los *topics* que se quieren reenviar, un ejemplo de este caso sería temperatura/ y lo etiquetamos como "btemperatura". Después se configura el otro extremo de la comunicación, el Emoncms, en este punto es necesaria la *key* y la dirección de nuestro servidor. Por último, hay que configurar el espacio de destinos, es necesario asociar las etiquetas de los *topics* a los datos de una entrada en Emoncms, es decir "btemperatura" con el número de nodo y el nombre de la entrada.

Para el registro en el bróker y la suscripción a los *topics*, se utiliza la librería paho.mqtt de forma que se crea un cliente que queda a la escucha de los *topics* que se han configurado previamente.

Finalmente utilizando la librería urllib, adjuntando el valor de la *API-key* y la configuración de la entrada hecha antes, es enviado hasta el servidor Emoncms.

3.2.5. Aplicaciones Móviles

Ambas aplicaciones están desarrolladas sobre la plataforma Android, el principal motivo de esta elección fue su SDK por las herramientas proporcionadas que nos ofrecen facilidades a la hora de desarrollar y ejecutar las aplicaciones. El SDK está desarrollado en Java por esto, es el lenguaje de programación utilizado para desarrollar las aplicaciones. [26]

Una de las aplicaciones es la encargada de mostrar la temperatura al usuario mientras que la otra se encarga de la configuración del nodo.

¿Por qué dos aplicaciones?

Se pensó así por los dos tipos de usuarios que harán uso de ellas: el cliente, al que principalmente le interesa conocer el estado de los productos y, los técnicos encargados de la colocación de los nodos, quiénes serán los responsables de sus configuraciones internas.

1. Aplicación para la configuración de parámetros

Esta aplicación está dirigida para el técnico encargado de colocar los nodos en los camiones/tráiler y hacer así más fácil su configuración.

Con ella se pueden modificar parámetros en los nodos, los cuales serán: una ***nueva red Wifi*** a la que poder conectarse, y, en su defecto una ***nueva contraseña*** si tuviera, la ***señal de Wifi*** por debajo de la cual simplemente se almacenará la temperatura, el ***tiempo*** que se quedarán en estado ***dormido*** los nodos, el ***número de muestras de temperaturas*** que se guardarán en el nodo y por último, el número de muestras de temperatura que se enviarán en cada conexión al servidor.

El envío de los parámetros se realiza mediante la publicación a través de un *"topic"*, el cual es común para todos los nodos de un mismo camión.

Se ha utilizado una aplicación base [27], que ha sido modificada según nuestro propósito en el proyecto, que nos proporciona la comunicación con el servidor.

Para el desarrollo de la aplicación se ha utilizado una librería MQTT [28], *Java Paho Client*, que proporciona dos APIs:

- ***MqttAsyncClient***: es una API asíncrona, que notifica a través de *callbacks* cada vez que se termina de realizar una actividad. Nos permite iniciar acciones MQTT y seguir trabajando mientras éstas se realizan en segundo plano.

- ***MqttClient***: es una "capa" síncrona de la anterior que registra las funciones de la aplicación de forma síncrona. Sirve para bloquear las comunicaciones hasta que terminen por lo que facilita la lógica en la comunicación.

A parte de esta librería, también se utiliza una interfaz de ésta, *Paho Android Service*, donde la conexión MQTT se encapsula dentro de un Android Service que se ejecuta en

el contexto de la aplicación y, mantiene la conexión cuando la aplicación está cambiando de una actividad a otra.

Es necesaria esta capa de abstracción para poder recibir mensajes MQTT de forma fiable.

Arquitectura de la aplicación de parámetros

La aplicación se estructuró en diferentes clases relacionadas entre sí, las cuales son:

ActionListener: Implementa a la interfaz *IMqttActionListener*, que se encarga de informar si una acción, como publicar o conectar se han realizado o no con éxito. Cuando esta clase recibe esa acción, se encarga de notificar si se ha podido realizar.

ActivityConstants: En esta clase se encuentran todas las variables utilizadas en las demás clases.

Advanced: Muestra la actividad para una conexión más avanzada e implementa las funcionalidades de dicha actividad.

ClientConnection: Actúa como el main de la aplicación. Primero inicializa la interfaz en la que el usuario verá su id de cliente, el estado de su conexión y el nombre del servidor al que se conecta y, un botón para una nueva conexión.

Connection: Realiza las acciones de conectar y desconectar, informando en el historial de la acción realizada, si ha habido algún error en la conexión, el id del cliente conectado y el nombre del servidor al que se conecta.

ConnectionDetails: Es la encargada de mostrar las dos pestañas disponibles para el cliente: *History* y *Parameters*. La primera de ellas muestra el historial de la actividad realizada por el cliente, como los parámetros que está enviando al servidor, la segunda es la encargada de configurar y enviar los parámetros al servidor.

Connections: Se encarga de guardar todas las conexiones realizadas.

HistoryFragment: Muestra la interfaz que verá el usuario en la que se muestra su actividad realizada.

Listener: Esta clase es la encargada de realizar las acciones de conectarse, desconectarse, publicar los parámetros y crear una nueva conexión.

MqttCallbackHandler: Informa al usuario si la conexión se ha perdido y de los mensajes, añadiéndolos también al historial del cliente.

NewConnection: Se encarga de realizar una nueva conexión con los datos necesarios: id del cliente, puerto y nombre del servidor al que quiere conectarse.

Notify: Clase encargada de notificar cada acción realizada al usuario.

OpenFileDialog: Añade el archivo SSL que se elija de los archivos.

ParametersFragment: Realiza una llamada a la actividad que contiene la interfaz donde se encuentran los parámetros de configuración.

Persistence: Es la encargada de crear e ir modificando la tabla para la base de datos con los correspondientes campos.

PersistenceException: Devuelve un error cuando la conexión falla.

Respecto a las interfaces que se muestran al usuario, se han implementado a una serie de actividades (layouts) diferentes para cada una de ellas:

- **Activity_advanced:** Muestra una conexión más avanzada, en la que el usuario puede introducir su nombre de usuario, contraseña, si quiere utilizar SSL y seleccionarlo de sus archivos, establecer el tiempo de espera y el tiempo de vida para la conexión y guardar.
- **Activity_modify_parameters:** Presenta todos los parámetros para configurar, tiene los botones para desconectarse o conectarse y enviar los parámetros.
- **Activity_new_connection:** Realizar una nueva conexión: id del cliente, el puerto, el nombre del servidor. Además, puede conectarse y acceder a la pantalla de la actividad advanced.
- **Connection_text_view:** Es la actividad principal de la aplicación, en la que se muestra el cliente, el estado de la conexión al servidor en forma de lista y puede realizarse una nueva conexión.
- **List_view_text_view:** Muestra una lista con la actividad realizada por el cliente.

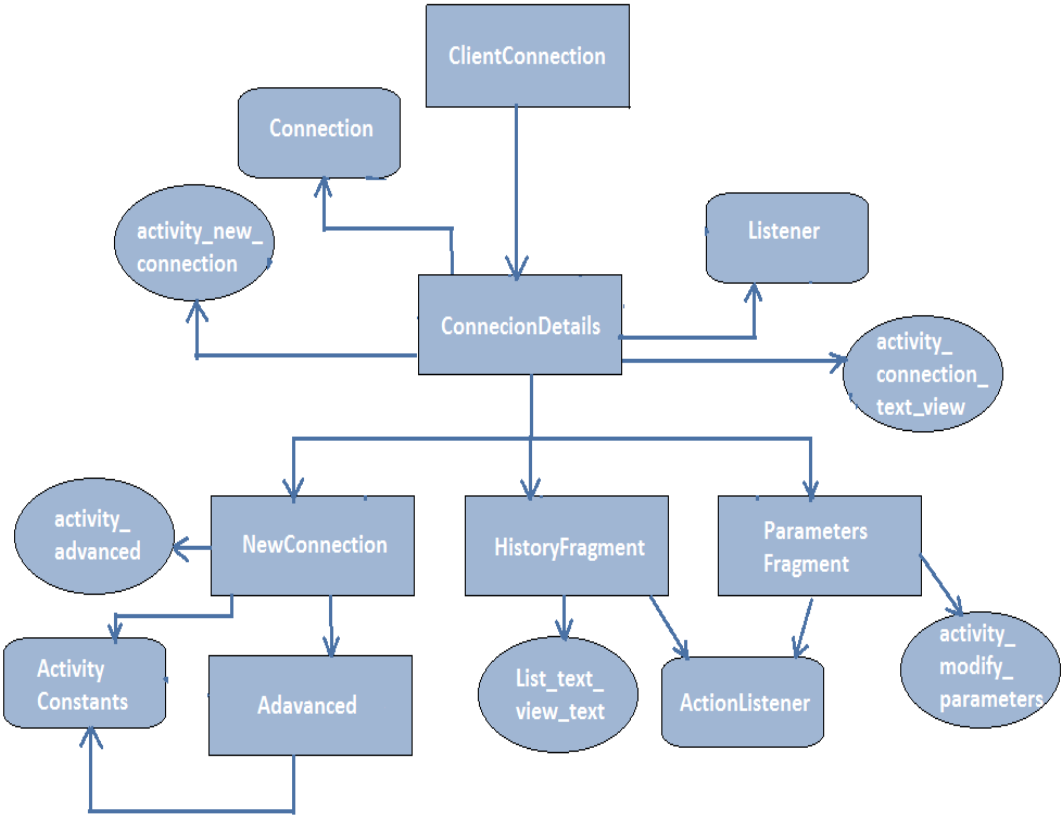

Figura 3.2.6 Diagrama de flujo de aplicación de configuración de parámetros

En la Figura 3.2.7 se muestra la interfaz con los parámetros configurables de la aplicación:

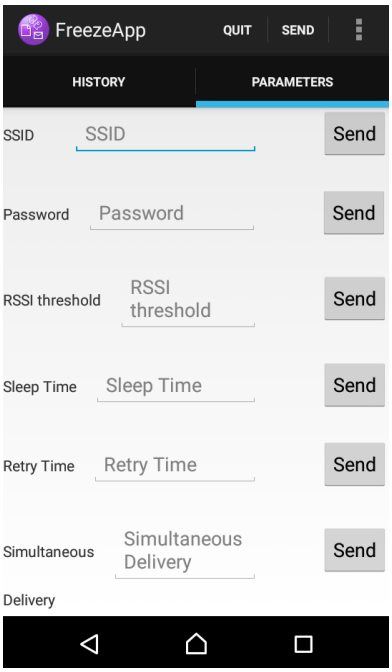

Figura 3.2.7 Aplicación de configuración

2. Aplicación para la visualización de los datos

Esta aplicación está dirigida especialmente al cliente. Con ella podrá visualizar mediante una sencilla gráfica la temperatura de los productos en tiempo real.

Esta aplicación se comunica mediante el protocolo HTTP con el servidor EmonCms del que podemos recoger los datos que envían los nodos con la temperatura correspondiente. Se ha utilizado la librería de transporte HTTP para el lado del cliente, ***HTTP Client,*** que se encarga de transmitir y recibir mensajes HTTP.

Para la visualización de la gráfica se ha utilizado la biblioteca ***MPAndroidChart*** [29], se trata de una librería que dispone de gran variedad de tipos de gráficas y bastante flexible a la hora de personalizar las gráficas.

Arquitectura de la aplicación de visualización de datos

La aplicación se estructuró en diferentes clases, son las siguientes:

HTTPClient: Clase encargada de recibir y transmitir los mensajes HTTP.

MainActivity: Responsable de inicializar la aplicación.

Temperature: Realiza la conexión con el servidor y va añadiendo los datos recogidos a la gráfica para la visualización de todos ellos.

Como se explica en la anterior aplicación, hay una serie de interfaces para el usuario que son las siguientes:

- **Activity_main:** Muestra la pantalla principal de la aplicación con un botón para acceder a la pantalla con los datos.
- **Activity_show_temperature:** Muestra la gráfica con los datos recogidos.

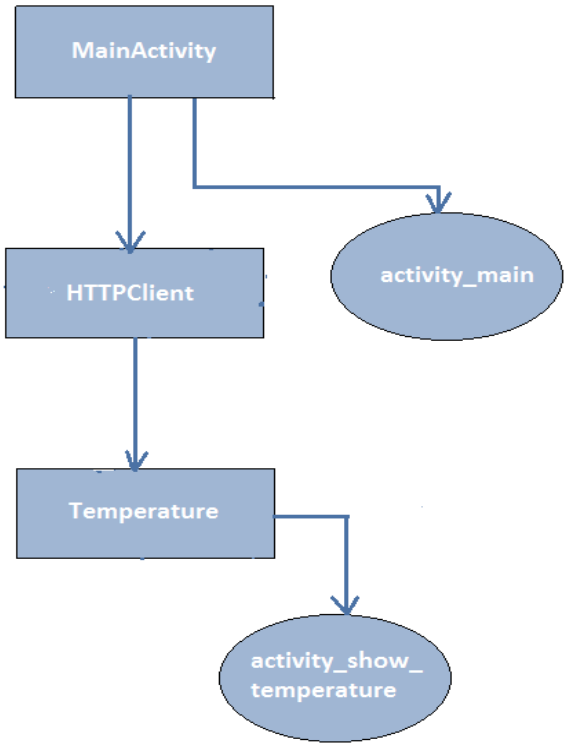

Figura 3.2.8 Diagrama de flujo de la aplicación de visualización de datos

En la Figura 3.2.9 se muestra el seguimiento de la temperatura de un nodo en forma de gráfica:

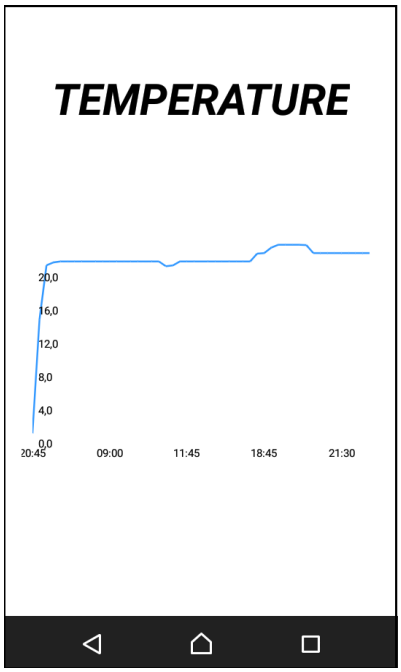

Figura 3.2.9 Aplicación de visualización de datos

4. EXPERIMENTACIÓN

Una vez finalizado el desarrollo del código instalado en el nodo, se realizaron una serie de pruebas que cubrían por un lado el comportamiento del nodo en entornos reales (cámaras frigoríficas) y por otro lado el consumo de batería del dispositivo.

4.1. Pruebas En Entornos Reales

Como parte del desarrollo, las pruebas en entornos reales tienen un papel fundamental tanto para la corrección de errores como para la observación del comportamiento del dispositivo bajo condiciones adversas. Por ello, se fijaron previamente a la realización de las pruebas una serie de objetivos que ayudarían a mejorar o corregir el funcionamiento del dispositivo. Estos objetivos son los siguientes:

- Toma de contacto con las diferentes cámaras frigoríficas
- Comportamiento del chip y la batería en este tipo de entornos
- Comprobar los posibles problemas de conectividad dentro y fuera de la cámara frigorífica.
- Analizar los datos recogidos en este tipo de entorno.

Una vez fijados estos objetivos se describe a continuación el material utilizado en las pruebas:

- **Batería:** Para estas pruebas se utilizó una batería externa de 10200 mAh que alimentaba el nodo a través de la conexión USB de la placa de desarrollo NodeMcu
- **Chip:** ESP12 con placa de desarrollo NodeMcu.
- **Router 3G:** modelo Teltonika RUT 500
- **Cámara frigorífica:** la cámara frigorífica donde se realizaron las pruebas estaba compuesta por un aislante de poliuretano como el de la Figura 4.1.1. Este tipo de cámaras frigoríficas están destinadas al transporte de productos refrigerados, los cuales mantienen una temperatura aproximada entre 2-5ºC.

Figura 4.1.1 Material de la cámara frigorífica de pruebas

Pruebas y Resultados

Prueba Inicial

A mediados de febrero, se realizó la primera prueba en cámaras frigoríficas. El principal objetivo era obtener una toma de contacto con las instalaciones y comprobar los posibles errores que surgieran en la misma.
En esta primera prueba inicial se utilizó la Versión 2 del Código (código desarrollado en Segunda Fase del Proyecto). En esta versión del código se enviaban las mediciones directamente a ThingSpeak para su posterior análisis.
Se debe hacer una mención aparte sobre las condiciones en cuanto a conectividad que nos encontramos en la facultad de Agrónomos. Al estar situadas las cámaras frigoríficas en un sótano no llegaba ningún tipo de señal exterior (vía Wi-Fi o vía 3G). Por tanto tuvimos que conectar una antena con cable de larga distancia para que llegara de alguna manera conectividad 3G del exterior.

Gracias a las condiciones adversas descritas anteriormente, esta primera prueba inicial fue muy importante a la hora de corrección de errores. Se comprobó que existía un error a nivel de código que impedía conectar el nodo a la única red Wi-Fi disponible. Es decir, si existía más de una conexión Wi-Fi disponible, no había ningún problema en cuanto a la conectividad. Por el contrario, si únicamente existía una red Wifi disponible, en este caso la que generaba el *router*, el chip no era capaz de conectarse.
Como resultado de esta primera prueba se solucionó el error de conectividad en condiciones adversas permitiéndonos enfocar las siguientes pruebas a la recogida de datos.

Prueba Final

A mediados de mayo se realizó la prueba final de nuevo en las instalaciones de Agrónomos. En este caso la prueba estaba orientada en mayor medida a la recogida de datos y su futuro análisis.

En estas pruebas se utilizó la última versión de código disponible. Es decir, se debían configurar previamente los parámetros descritos en la sección 3.2 y posteriormente, colocar el nodo en diferentes posiciones de la cámara con respecto al ventilador y así tener diferentes muestras.

Para la realización de estas pruebas se contó con la ayuda de un pequeño armario con baldas a diferentes niveles que nos permitía situar el nodo en diferentes posiciones de altura con respecto al ventilador.
A continuación, se muestran los resultados obtenidos en cada una de las posiciones de la cámara donde ubicamos el nodo.

Eje X → Hora de la muestra.
Eje Y → Temperatura obtenida.

Estantería debajo Refrigerador - Balda Inferior

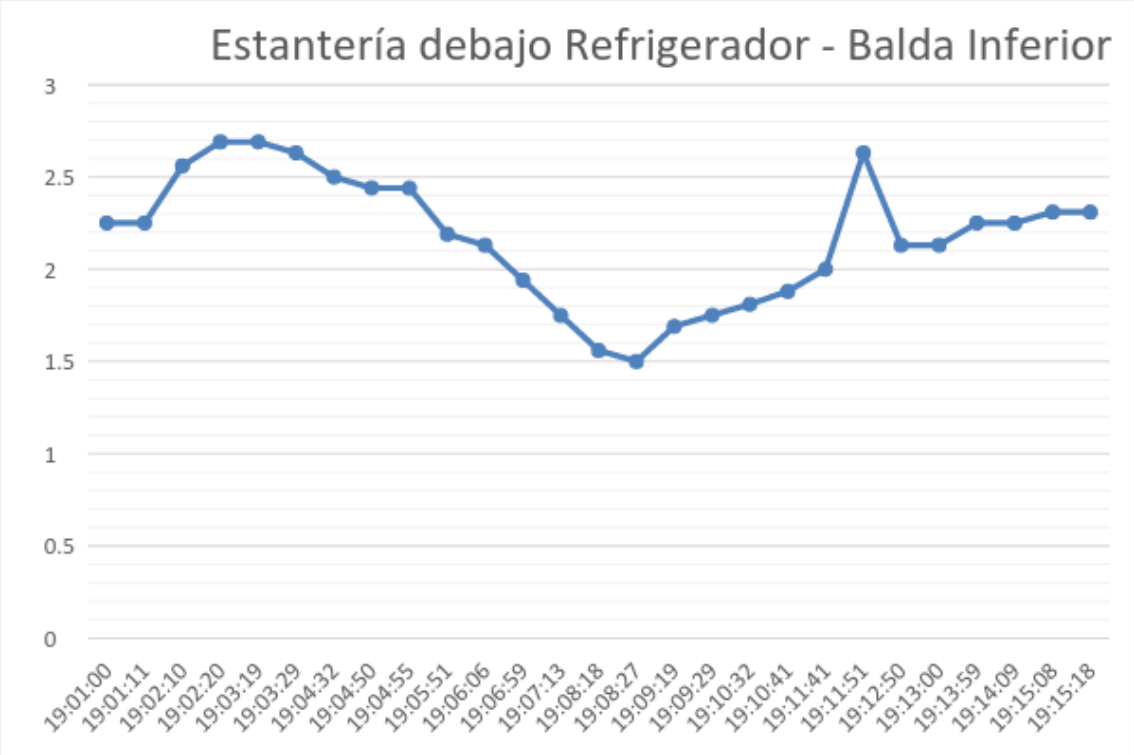

Figura 4.1.2 Gráfica debajo del refrigerador en la balda inferior

Estantería debajo Refrigerador - Balda Superior

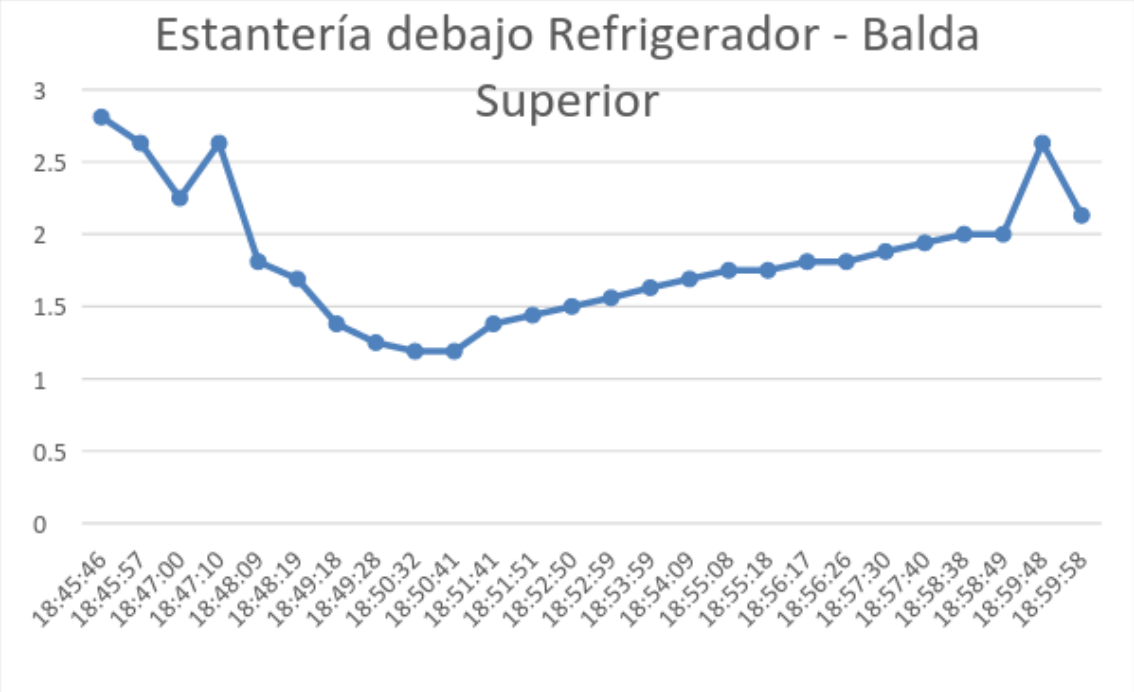

Figura 4.1.3 Gráfica debajo del refrigerador en la balda superior

Estantería Frente a Refrigerador - Balda Superior

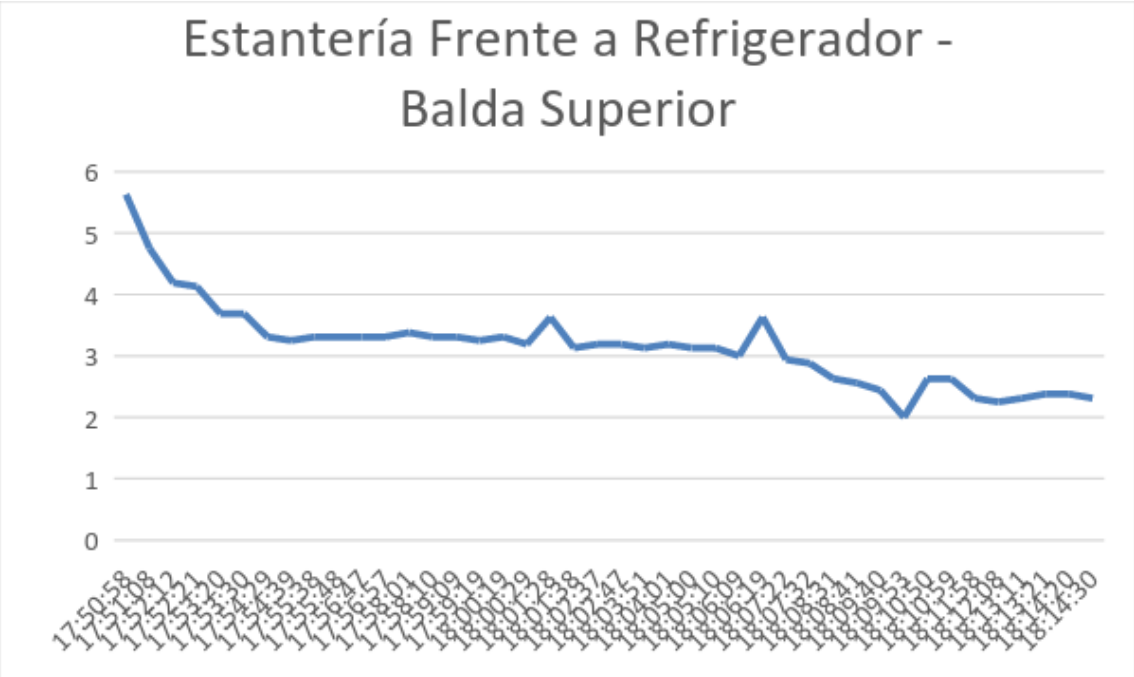

Figura 4.1.4 Gráfica frente al refrigerador en la balda superior

Estantería Frente a Refrigerador - Balda Central

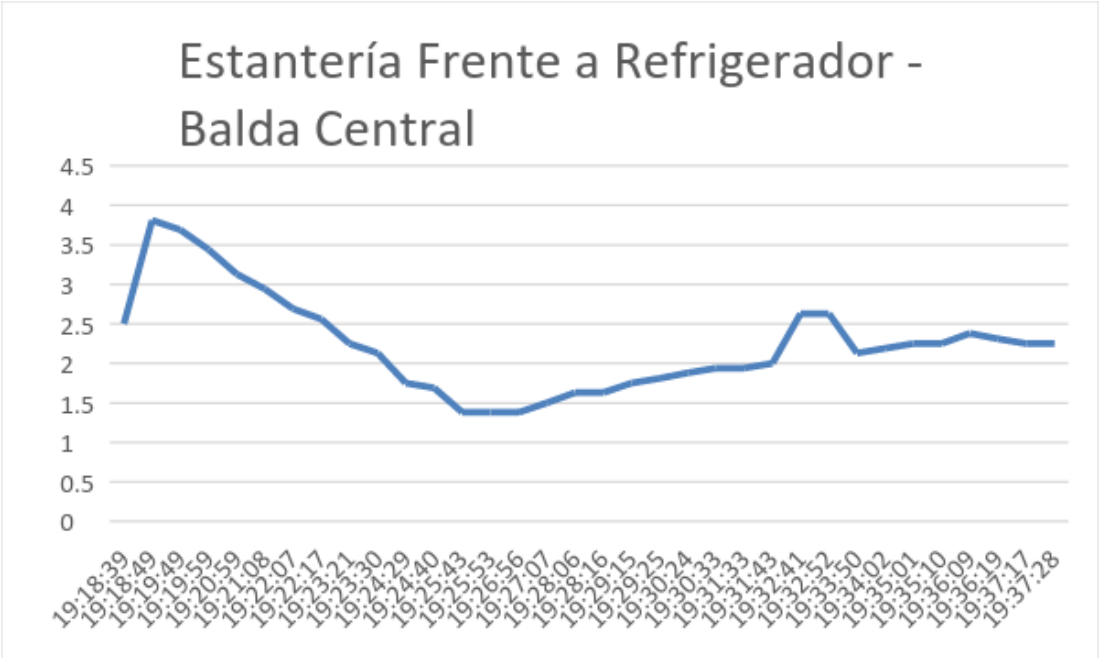

Figura 4.1.5 Gráfica frente al refrigerador en la balda central

Estantería Frente a Refrigerador - Balda Inferior

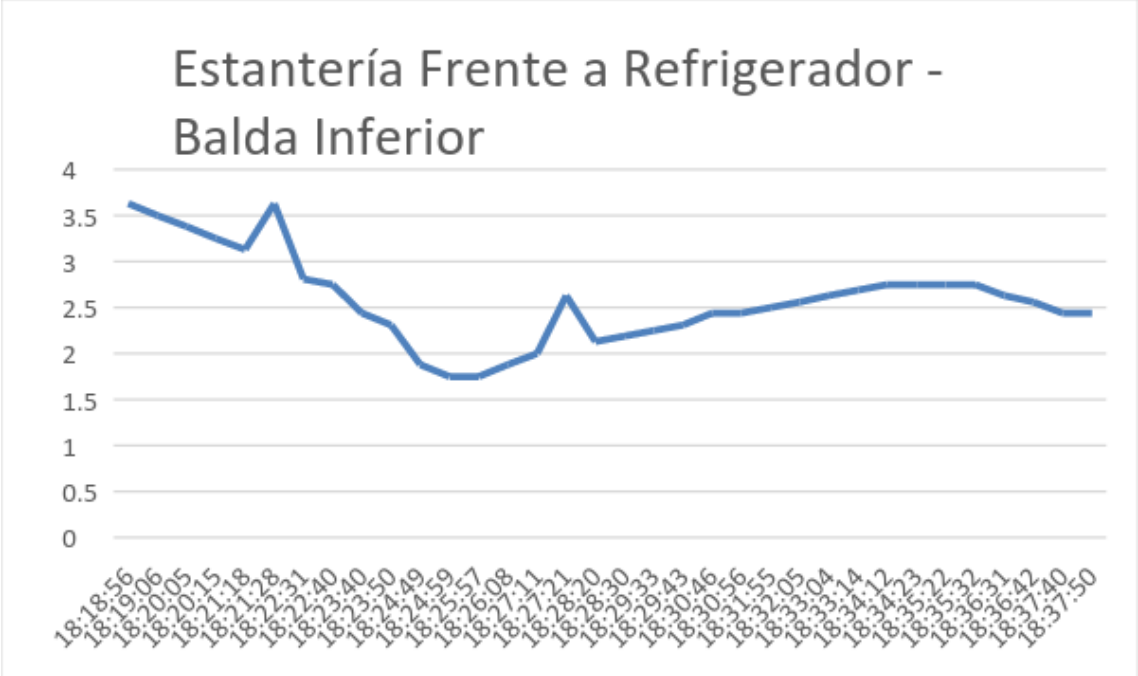

Figura 4.1.6 Gráfica frente al refrigerador en la balda inferior

Interior Cámara Frigorífica

Figura 4.1.7 Interior cámara frigorífica

Como se pueden comprobar en las gráficas mostradas anteriormente, se observa la variación de temperaturas entre unas posiciones y otras, siendo la más fría la posición inferior al ventilador del frigorífico en la balda superior del armario utilizado en las pruebas (1.19ºC de mínima).

4.2. Pruebas De Consumo

En este apartado se van a detallar las pruebas de consumo realizadas en el proyecto así como los resultados obtenidos.

Modelo Chip de pruebas:

- ESP12 con Placa Desarrollo NodeMcu

Para los casos de prueba se configuraron los parámetros de envío de tal manera que se mandaran cada 5 minutos el valor de la temperatura en ese momento. Por tanto, la placa se mantendría en estado "dormido" durante aproximadamente 5 minutos (existe un pequeño retardo entre que la instrucción se envía para ejecución y cuando realmente entra en estado "dormido") y aproximadamente durante 28 segundos el chip se mantendría encendido para calcular la temperatura, conectarse a la red WiFi y enviar los datos al servidor.

En la figura 4.2.1. se muestra la gráfica de un ciclo de funcionamiento completo, considerando ciclo completo como:

- Comienzo en estado dormido
- Arranque del chip para calculo y envío de temperatura
- Vuelta a estado dormido
- Siguiente arranque del chip para cálculo y envío de temperatura.

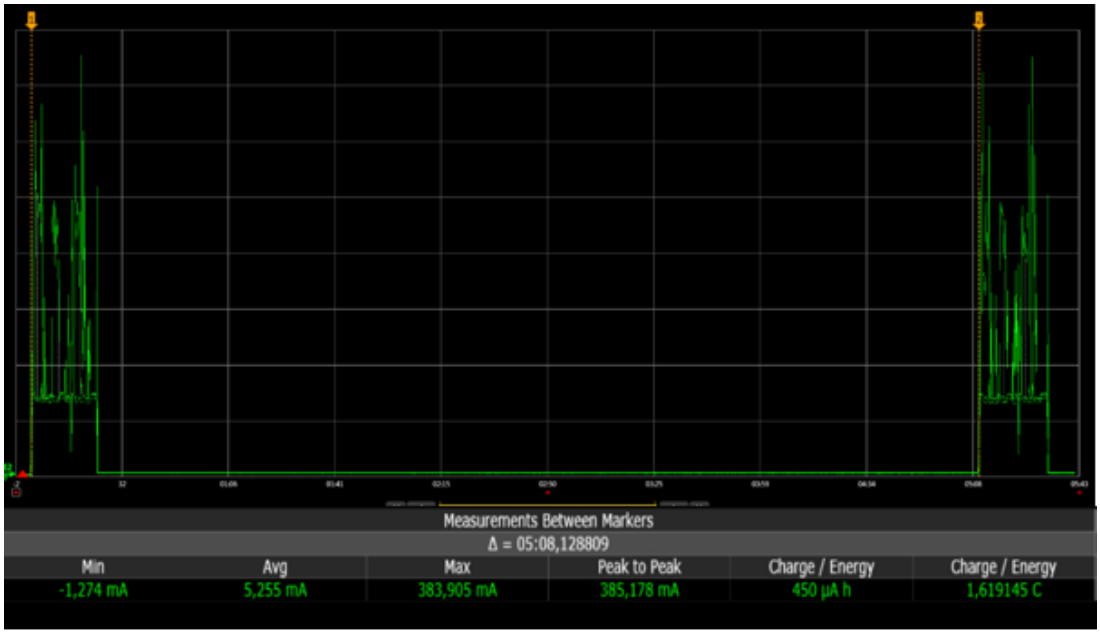

Figura 4.2.1 Gráfica de consumos del nodo

Con esta configuración de parámetros, es decir, enviando cada 5 minutos la temperatura al servidor, se puede comprobar en la imagen que arroja un consumo de 450µAh.

Por tanto, con este consumo podría llegar a tener una vida útil con este modelo de batería de aproximadamente 11 días.
En el ámbito del transporte de productos refrigerados cabe destacar que el muestreo de la temperatura cada 5 minutos podría considerarse demasiado exhaustivo. Un control de la temperatura cada 10 min sería más realista para empresas de transporte de este tipo de alimentos/objetos.

En caso de que se configuraran los parámetros de esta manera se podría alargar la vida de la batería varios días más.

Nota: Estos consumos podrían ser mejorados en gran medida ya que en estado dormido el kit que utilizamos consumía 370µA, del orden de 12 veces más que un kit sin placa de desarrollo (29 µA). Esto se producía ya que la entrada de 3,3v de la placa de desarrollo alimentaba tanto al chip ESP12 como a la placa en sí. Al entrar en estado "dormido" el chip ESP12 sí que se dormía, pero la placa de desarrollo seguía consumiendo de los 3,3v de entrada. Es por esto por lo que hay tanta diferencia entre ESP12 sin placa de desarrollo y otro con placa incluida.

5. CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se describirán tanto las conclusiones que hemos obtenido una vez finalizado el proyecto, las posibles mejoras que podrían realizarse en él, así como la viabilidad del proyecto en el ámbito comercial.

5.1. Conclusiones

Nuestro principal objetivo con la realización de este proyecto ha sido desarrollar un prototipo diseñado para el control de la cadena del frío basado en la tecnología IoT que garantizase la misma funcionalidad que las soluciones ya existentes, pero ofreciendo un coste reducido.
Con respecto a los objetivos descritos en la sección 1, hemos sido capaces de garantizar el menor coste posible sin afectar de ninguna manera a la completa funcionalidad desarrollada para este dispositivo de pequeñas dimensiones haciéndolo a su vez, más manejable a la hora de su instalación o intercambio entre cámaras frigoríficas.
Además, como ya se ha descrito anteriormente en la sección 3, se dispone de la posibilidad de poder visualizar los datos desde cualquier dispositivo ya sea móvil, tablet u ordenador, en cualquier momento.

Durante el desarrollo del proyecto nos encontramos con los inconvenientes de la falta de familiaridad con el SDK utilizado para la programación del chip y la poca documentación de la que disponíamos de éste, los cuales nos entorpecieron en el desarrollo, aunque sin el uso de este SDK, se hubiera tenido que implementar más código a la hora de desarrollar.
Respecto a la tecnología que se ha utilizado, se ha valorado continuar con ella ya que se han obtenido buenos resultados con la prueba de contexto que se ha realizado utilizando la tecnología wifi.

Una vez terminado el proyecto, podemos decir que se han cumplido con las expectativas propuestas en un principio, sin embargo, también somos conscientes de las mejoras que se necesitan para poder ofrecer un producto comercial, las cuales se detallan en el siguiente punto.

5.2. Trabajo Futuro

El hecho de que las empresas existentes no ofrezcan una solución completa a un bajo coste y, una vez realizadas las mejoras que se describen a continuación, podemos garantizar una ventaja a la hora de entrar al mercado ya que con nuestro producto aseguramos una solución completa, tanto en el control de la temperatura como en el bajo coste del producto.

Cuando hablamos de trabajo futuro nos estamos refiriendo a una serie de mejoras en todo el prototipo en su conjunto, las cuales podrían dividirse en las siguientes partes:

1. **Nodo:**

    - **Ampliación de memoria**

      A lo largo del desarrollo del proyecto, una limitación que ha estado presente ha sido la escasa memoria de la que disponía el nodo, como hemos podido ver en la sección 3.2. Por ello, creemos que una de las principales mejoras a realizar sería utilizar un nuevo nodo con un almacenamiento más amplio. Esto supondría un beneficio ya que se podría mejorar tanto el rendimiento como ampliar la funcionalidad del prototipo.

    - **Cambios en el sensor**

      Al analizar el mercado de los sensores, nos encontramos con soluciones que poseen en un mismo sensor tanto termómetro como higrómetro, el cual nos permite medir la humedad. De esta forma y con la mejora anterior, con una pequeña modificación en el código se podría ampliar la funcionalidad de nuestro prototipo.

      El que más se ajusta a esta mejora se trata del sensor DHT22, con un rango de temperatura de -40ºC/+80ºC y una precisión de ±0,5% y, respecto a la humedad posee un rango de 0-100% y una precisión de ±0,3%.
      Este sensor proporciona unas dimensiones y un precio algo más elevado que el que utilizamos, pero sigue ajustándose a los objetivos de nuestro proyecto.

    - **Añadir un botón para acceder al modo Setup**

      En el momento de la instalación del nodo el hecho de añadir un botón para que éste pueda entrar en modo Setup nos facilitaría su nueva configuración.

2. Servidor y plataformas de visualización de datos: En este punto se detallan las posibles mejoras en nuestro servidor y en las plataformas utilizadas para la visualización de los datos:

- **Servidor Mosquitto: Nuevos Sistema de Topics**

  - A la hora de referirnos a un sólo nodo, el sistema utilizado es el siguiente: *nombre topic + id del nodo*, facilitando el control de los datos recibidos por cada uno de ellos.
    Este sistema resulta muy costoso si se trata de más de un nodo ya que se realiza manualmente y requiere un pequeño tiempo de configuración. Para mejorar este sistema, se automatizaría el proceso realizando un programa que observara los *topics* del *broker*, cada vez que se creara un *topic* nuevo que se refiera a un nodo, el bridge fuera capaz de añadir ese nuevo *topic* y mandarlos a los sistemas de visualización.

- **Plataformas para la visualización de datos**

  - ThingSpeak: *Campos automáticos*

    Como ya se ha visto en la sección 3, ThingSpeak trabaja con un sistema de campos para mostrar la información, los cuales deben crearse a mano. Pero utilizando el sistema de *topics* detallado en la mejora anterior, este proceso es muy pesado a la hora de realizarlo manualmente. Por ello, se planteó la posibilidad de implementar tanto en el bridge como en la visualización de los datos la funcionalidad de automatizar la creación y asociación de un campo nuevo a cada *topic* nuevo.

  - Emoncms: *Entradas automáticas*

    Como se explica en la sección 3, Emoncms necesita información para la creación de una entrada y sus fuentes correspondientes. Sin ésta entrada, no es posible diferenciar los datos de cada uno de los nodos. Debido al sistema de *topics* descrito anteriormente, se necesita implementar una nueva funcionalidad tanto en el bridge como en las visualizaciones, la cual genere de forma automática una nueva *fuente* para cada entrada asignada a un nuevo nodo y así, poder visualizar las gráficas con los datos de este nodo.

3. Aplicaciones Móviles: En cuanto a las aplicaciones móviles, se podrían realizar las siguientes mejoras:

- **Unificación de ambas aplicaciones**

  De esta forma el usuario podría llevar un control de la temperatura del producto, así como configurar los nodos a la hora de su colocación en la cámara frigorífica del camión.

- **Visualización de los datos**

  En cuanto a la visualización, en este momento sólo se dispone de una visualización de temperaturas de un único nodo, podría ser posible reflejar la temperatura de todos los nodos de un camión, con sus correspondientes gráficas, así como una media de las temperaturas obtenidas.

- **Aplicación para iOs**

  Se podría desarrollar una versión para el sistema iOs abarcando así más usuarios.

Una vez realizadas estas mejoras, se dispondría de lo necesario para comenzar a elaborar un producto comercial.

Para el prototipo, se puede estipular su coste en 7,25 €, siendo el coste de cada uno de los componentes el siguiente: placa de desarrollo 2€, pila 3,3€ y sensor 1,95€. Estos costes a la hora de realizar los pedidos se reducirían en cada uno de los componentes ya que se realizarían encargos de grandes cantidades de cada uno de ellos.
El coste final del producto estaría en 20€/unidad pudiendo así obtener un beneficio de más de 10€ por cada producto vendido. Respecto a las dos aplicaciones serían gratis.

6. CONTRIBUCIONES

Contribución de Irene Cerro de Paz

- **Investigación:**

➢ Investigación conjunta sobre *Internet of Things*, haciendo hincapié en los dispositivos que estaban basados en esta tecnología y tenían relación con nuestro proyecto.

➢ Investigación sobre el sensor de temperatura y el chip utilizado en el proyecto. Traducción de la documentación del sensor para que nos fuera más fácil su comprensión.

➢ Investigación y me documentación sobre los distintos protocolos y *brokers* que podrían utilizarse para el proyecto.

➢ Una vez elegidos el protocolo y el bróker, se investigó la comunicación entre el bróker y el chip y las plataformas utilizadas para la visualización de los datos enviados entre el chip y el servidor, ThingSpeak y Emoncms.

➢ Investigación sobre cómo hacer aplicaciones móviles sobre la plataforma Android y su posterior desarrollo. Dentro de las aplicaciones, documentación sobre la forma de comunicar una de las aplicaciones con la plataforma Emoncms y sobre las librerías para mostrar gráficas.

➢ Por último, documentación e investigación sobre la comunicación entre la otra aplicación y nuestro servidor para el envío de parámetros al chip, mediante las librerías *Client Paho*.

- **Desarrollo:**

➢ Contribución con la instalación y configuración del bróker. Configuración junto con mi compañero Carlos de los bridges tanto a ThingSpeak como a Emoncms entre estos y nuestro servidor.

➢ Respecto a las aplicaciones móviles, se buscó junto a mi compañero Carlos una librería para la muestra de temperaturas mediante gráficas de forma que fuera más amigable ver los datos. Y la librería *HTTPClient* para la comunicación con la plataforma Emoncms. Realización también del envío de parámetros desde la aplicación a nuestro servidor mediante el sistema de *topics*.

➢ Finalmente, se colaboró de manera conjunta a la hora de realizar las pruebas tanto en el laboratorio como en las cámaras frigoríficas de la Escuela de Agrónomos.

- **Memoria:**

➢ **Capítulo 3:** Este capítulo fue escrito por mí y mis compañeros de manera conjunta, siendo mi parte escrita correspondiente al punto "*Selección del sensor y Aplicaciones móviles*".

➢ **Capítulo 5:** Todo el capítulo fue escrito por mí.

➢ **Capítulo 6:** Realicé mi parte correspondiente a las contribuciones en el proyecto.

➢ Finalmente, realicé los apartados "*Agradecimientos*, *Lista de Acrónimos y Consentimiento del proyecto*".

Contribución de Virginia Galisteo Fernández

- **Investigación:**

➢ Investigación sobre el tracking del frío, documentándome sobre su definición, los problemas y las soluciones que existían actualmente. De esta manera pude sacar conclusiones con las que seguir orientando el resto de la investigación.

➢ Investigación conjunta sobre la tecnología IoT, leyendo todo tipo de artículos de diferentes aplicaciones y servicios que proporcionaba, haciendo hincapié en los dispositivos que estaban basados en esta tecnología y tenían relación con nuestro proyecto.

➢ El siguiente paso fue investigar y documentarse sobre el nodo, sus diferentes kits de desarrollo y versiones de firmware disponibles y, el sensor de temperatura utilizados para el proyecto.

➢ Documentación de la API del nodo, de manera que me familiarizara con su multitud de funciones y así facilitar su comprensión. Además, se investigó sobre las diferentes librerías que podíamos utilizar para el nodo.

➢ Para la implementación del código que llevaría nuestro nodo se investigó sobre los lenguajes de programación que podíamos utilizar en el desarrollo, focalizándonos finalmente en el lenguaje Lua. Para el cual tuvimos que documentarnos sobre la programación por eventos, que es el modelo de programación que sigue este lenguaje.

➢ Investigación sobre los entornos de desarrollo disponibles para la implementación del código.

➢ Documentación sobre el funcionamiento del protocolo MQTT.

- Documentación sobre la comunicación entre el chip, el servidor y las plataformas que íbamos a utilizar para la visualización de los datos enviados.

- Investigación sobre los entornos de desarrollo de Android para la elaboración de las aplicaciones móviles.

- **Desarrollo:**

- Desarrollo junto con Alejandro Martin del código íntegro del nodo. Este desarrollo incluye:

    - Toma de temperaturas secuencialmente
    - Adaptación del código para permitir comunicación mediante protocolo MQTT a nivel del nodo y de esta manera habilitar la comunicación con el servidor.
    - Configuración del nodo mediante parámetros.
    - Envío de temperaturas a ThingSpeak
    - Adaptación del código para un menor consumo de batería. Esta adaptación incluye una previa investigación sobre los diferentes modos de funcionamiento con respecto al estado "dormido" del nodo.
    - Conexión al servidor para el envío de temperaturas.

- Finalmente, se colaboró de manera conjunta en las pruebas realizadas en las cámaras frigoríficas de la Escuela de Agrónomos y en las realizadas en multitud de ocasiones en los laboratorios de la Facultad de Físicas

- Análisis de resultados: realicé un estudio conjunto de los datos obtenidos en las pruebas hechas anteriormente.

- **Memoria:**

- **Capítulo 2:** Este capítulo fue escrito por mí de forma íntegra.

- **Capítulo 3:** Este capítulo fue escrito por mí y mis compañeros de manera conjunta, siendo mi parte escrita correspondiente al punto "*Selección del router*".

- **Capítulo 5:** Colaboré en el desarrollo de este capítulo.

- **Capítulo 6:** Realicé mi parte correspondiente a las contribuciones en el proyecto.

- Finalmente, realicé los apartados de "*Lista de acrónimos y Bibliografía y traducción de las conclusiones*".

Contribución de Alejandro Martín Seijas

- **Investigación:**

➢ Formé parte de la investigación de los diferentes tipos de chip ESP12 y de los diferentes kits de desarrollo disponibles para el mismo, más concretamente en el kit NodeMcu.

➢ Investigación de los diferentes entornos de desarrollo disponibles para la implementación del código.

➢ Investigación sobre las soluciones existentes actualmente en cuanto al tracking de la cadena del frío en el transporte de mercancías.

➢ Investigación de los diferentes tipos de sensores de temperatura compatibles con ESP12.

➢ Investigación de las diferentes versiones de firmware NodeMcu disponibles para el ESP12 y las diferencias existentes entre ellos a nivel de fiabilidad y funcionalidad

➢ Investigación de las técnicas de desarrollo de código Lua para NodeMcu. Se tuvo que investigar acerca de la programación por eventos ya que es el modelo que seguía el entorno Lua junto con NodeMcu.

➢ Investigación de la multitud de funciones disponibles en la Api NodeMcu y su posible utilización a nivel de código.

➢ Investigación a nivel de código Lua del funcionamiento del protocolo MQTT y su posible aplicación dentro de nuestro proyecto.

- **Desarrollo:**

Desarrollo junto con Virginia Galisteo del código íntegro del nodo. Este desarrollo incluye:

  - Toma de temperaturas secuencialmente
  - Adaptación del código para permitir comunicación mediante protocolo MQTT a nivel del nodo y de esta manera habilitar la comunicación con el servidor.
  - Configuración del nodo mediante parámetros.
  - Envío de temperaturas a ThingSpeak
  - Adaptación del código para un menor consumo de batería. Esta adaptación incluye una previa investigación sobre los diferentes modos de funcionamiento con respecto al estado “dormido” del nodo.
  - Conexión al servidor para envío de temperaturas

- **Memoria:**

➢ **Capítulo 1:** Este capítulo fue escrito íntegramente por mí.

➢ **Capítulo 3:** Este capítulo fue escrito por mí y mis compañeros de manera conjunta, siendo mi parte escrita correspondiente a los puntos "*Nodo y Arquitectura del código desarrollado*" y "*Selección de Baterías*".

➢ **Capítulo 4**: Este capítulo fue realizado integro por mí.

➢ **Capítulo 6:** Realicé mi parte correspondiente a las contribuciones en el proyecto.

➢ Finalmente, realicé los apartados de "*Lista de acrónimos, Resumen Inicial e introducción en español e inglés y Bibliografía*".

Contribución de Carlos Membrilla Cobo

- **Investigación:**

➢ Investigación sobre las soluciones existentes para el seguimiento de la cadena del frío.

➢ Investigación sobre los microcontroladores y sensores que podíamos utilizar para este proyecto.

➢ Investigación sobre el protocolo de comunicación entre el nodo y el servidor, en función de éste, hice lo mismo para el bróker que se iba a montar para la comunicación.

➢ Una vez todo elegido, se investigó y documentó sobre cómo comunicar el nodo con el bróker en código lua y también hacerlo con varios datos almacenados en el nodo.

➢ Investigación sobre los sistemas de visualización de y sobre cómo hacer un bridge para mandar los datos recibidos en el bróker a los esos sistemas.

➢ Investigación sobre cómo desarrollar una aplicación en Android, teniendo en cuenta cómo comunicar una aplicación Android con un bróker MQTT. También sobre cómo comunicar una aplicación Android con un servidor Emoncms propio.

- **Desarrollo:**

  - Colaboración en las primeras fases del código del nodo y en las primeras comunicaciones del nodo con el bróker Mosquitto y más tarde con thingspeak.

  - Me encargué de todo lo relacionado con el servidor:

    - Montaje del bróker en el servidor, su configuración y selección de usuarios para la seguridad en el acceso al bróker.
    - Diseño y configuración del sistema de *topics* del bróker.
    - Montaje del bridge entre el bróker y Thingspeak y su correspondiente configuración.
    - Montaje del servidor propio Emoncms y su correspondiente configuración.
    - Montaje de bridge entre el bróker y Emoncms y su correspondiente configuración.

  - Colaboración en el desarrollo de la aplicación FreezeApp que hace uso de las librerías Client Paho para la comunicación con el bróker.

  - Colaboración en el desarrollo de la aplicación FreezeSenseApp para mostrar la información de las gráficas del emoncms.

  - Colaboración en las pruebas hechas en casa como las pruebas hechas en las cámaras frigoríficas de la Escuela de Agrónomos.

- **Memoria:**

  - **Capítulo 2:** Colaboré en el desarrollo de éste capítulo.

  - **Capítulo 3:** Este capítulo fue escrito por mí, en su mayor parte, con la colaboración de mis compañeros.

  - **Capítulo 6:** Realicé mi parte correspondiente a las contribuciones en el proyecto.

  - Finalmente, realicé los apartados "*Índice de imágenes y Lista de acrónimos*".

7. BIBLIOGRAFÍA

[1] La cadena del frío, elemento clave en seguridad alimentaria. 18 de Diciembre de 2008. Disponible en: http://www.consumer.es/seguridad-alimentaria/sociedad-y-consumo/2008/12/18/182212.php

[2] Características Data Logger de temperatura Testo 174-T https://www.testo.com.ar/detalles_productos/0572+1560/#tab-8

[3] Características sobre etiqueta de temperatura termosensible WarmMark. Disponible en: http://www.unprecio.es/precio/etiquetas-de-temperatura-termosensibles-warmmark/

[4] Sistema de seguimiento de la cadena del frío Astrata. Disponible en: http://astratagroup.com/solutions/cold-safe/

[5] La revolución del Internet of Things, Harvard-Deusto Business Review http://www.harvard-deusto.com/la-revolucion-del-internet-of-things

[6] Definición de IoT, Wikipedia. Disponible en: https://es.wikipedia.org/wiki/Internet_de_las_cosas

[7] La logística en la era IoT, 11 de Junio de 2015. Disponible en: http://www.mercado.com.ar/notas/tecnologa/8018429/la-logstica-en-la-era-de-iot

[8] ¿Qué es CoAP?, Un poco de Java, 28 de Abril de 2013. Disponible en: https://unpocodejava.wordpress.com/2013/04/28/que-es-coap/

[9] Protocolos del Internet de las Cosas, AltranTech360, 1 Marzo 2016. Disponible en: http://equipo.altran.es/protocolos-iot-internet-de-las-cosas/

[10] MQTT and CoAP, IoT protocols, Eclipse newsletter. Disponible en: http://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[11] ESP8266 Support WIKI. Disponible en: http://www.esp8266.com/wiki/doku.php?id=start

[12] NodeMCU –An open-source firmware based on ESP8266 wifi-soc. Disponible en: http://nodemcu.com/index_en.html

[13] Adafruit HUZZAH ESP8266 Breakout. Disponible en: https://www.adafruit.com/product/2471

[14] NodeMCU –Development kit. Disponible en: http://nodemcu.com/index_en.html#fr_54747661d775ef1a3600009e

[15] Integrated Temperature Sensors. Disponible en: http://www.analog.com/en/products/analog-to-digital-converters/integrated-special-purpose-converters/integrated-temperature-sensors.html

[16] DS18B20 temperature Sensor datasheet. Disponible en:
https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[17] Lithium Ion Battery, Adafruit. Disponible en:
https://www.adafruit.com/products/2011

[18] CR2032 coin cell Datasheet. Energizer. Disponible en:
http://data.energizer.com/PDFs/cr2032.pdf

[19] Coin Cell, Lithium, 3.6 V, 120 MAH (LIR2450), Datasheet. Disponible en:
http://www.farnell.com/datasheets/1475807.pdf

[20] Router Teltonika RUT500 datasheet. Disponible en:
http://www.blauden.com/files/RUT500_flyer_EN_v1_08.pdf

[21] An Open Source MQTT v3.1/v3.1.1 Broker. Disponible en:

http://mosquitto.org/

[22] Internet Of Things - ThingSpeak. Disponible en:

https://thingspeak.com/

[23] Emoncms – site home. Disponible en:
https://emoncms.org/

[24] mqspeak: MQTT bridge. Disponible en:
https://github.com/mqopen/mqspeak

[25] mqttwarn: Subscribe to MQTT topics and notify pluggable services. Disponible en: https://github.com/jpmens/mqttwarn

[26] Android Studio. Disponible en:
https://developer.android.com/studio/index.html

[27] Using the Android MQTT client sample. Disponible en:
http://www.eclipse.org/paho/clients/android/sample/

[28] Paho MQTT library. Disponible en:
https://github.com/eclipse/paho.mqtt.java

[29] Android chart view/graph view library. Disponible en:
https://github.com/PhilJay/MPAndroidChart

8. ANEXO

Introduction

Maintaining the cold chain in the transport of perishable items such as food, it is one of the most important obligations that have to be met both logistics companies and companies which sells these products directly to consumers.

The cold chain is defined as the set of steps needed in cooling (-1ºC to 4ºC) or freezing (-18ºC to -25ºC) process in order to deliver the products for the customer with a guarantee of quality and safety. This set of steps encompass from the production initial phase, including transport step and ending with the customer sales.

As mentioned before, to fulfil the products cold chain gives to the customer a guarantee of quality and safety. Apply cold to the products totally inhibit (refrigerated products) or partially inhibits (refrigerated products) the metabolic process inside these products. In this way we can achieved the degradation of product properties delay. Furthermore, in case cold chain is broken at some of the steps, the possibility of having more microbial activity in the products increases, having more risks of suffering serious diseases (Salmonella, E.Coli, etc).

The cold chain must be met at all stages of the food production, transport and sale of the products thus different actors involved in this process should control the temperature in each of this phases. An example of this process (dairy product) and actors is shown below:

- **Raw Material supplier:** as it is well known, the milk is the main raw material of these type of products. That is why, it should be kept in control the temperature from the farms to the processing factories in order to give a quality and safety guaranteed final product.
- **Processing factory:** during the processing and production of dairy products the temperature has to be controlled in each of the elements which take part. A break in the cold chain in this phase could produce big economic losses for the company and a prestige loss as a brand.
- **Final Product Transport:** there is more probability of cold chain break during this phase because more logistic companies could take part. Moreover, not only the product destination but also where the product is going to be transported is important. In this way is easier to ensure cold chain during the product transport through places of low temperature. The control of the cold chain break should be harder in places where the temperature is very high for the product and here is where the project is going to be focused on.

Once introduced cold chain concept and have shown his problems and characteristics, in following sections will be explained the different cold chain control systems which exists currently in real life. Once shown these types and having analysed pros and cons of each, the objectives and project requirements will be explained.

Current cold chain tracking systems

Cold chain is a very important concept for both selling brand and for logistics companies. So, to control the cold chain problem, currently exist different types of systems/devices which mainly differ in the price, functionality and size. This systems can be classified into 3 groups:

- Data Loggers
- TTI Tags
- Cold Chain real time control system

Data Loggers

These systems only store the temperature periodically in its internal memory. Once the product reaches the destination, the employee downloads the data in his own PC or mobile via USB or Bluetooth connection and checks the temperature records during the transport.

- **Pros:** these type of devices offer a good battery life (500 days approx). As they have a small size, it could be coupled at box level which is going to be transported in the cold storages, but its high cost limits this possibility. Therefore, 1 or 2 devices are usually installed in each cold storage (depending on the size). In addition to these advantages, this devices are usually easy to install.
- **Cons:** the main disadvantage of these devices is that theay are only capable to periodically record temperatures which they can only be ckecked at the end of the product transport. Therefore, they cannot have a real time control of the temperature and in this way to avoid a possible cold chain break. For this disadvantage and for its high cost price, it was discarded as a possible option for this project.
- **Examples:** 174 T [2], offers up to 500 hours of battery life with a very small size. This model costs 70€, preventing in this way the coupling of multiple devices per cold storage.

TTI Tags

These tags have chemical reagents sensitive to temperature changes. Thus, when the product has a break in the cold chain, the chemical reaction starts inside the Tag, changing his own colour. Therefore, the operator can easily check if the products has suffered a break of the cold chain during the transport.

- **Pros:** these devices are very small (3-4 cm), and can be easily coupled to each box or product transported.
- **Cons:** these devices will be only able to check at the end of the transport if the product has suffered and break of the cold chain, but it does not indicate where and how long it happened. Although the cost of these tags are really small, the are single-use, forcing logistics companies to renew them.
- **Examples:** WarmMark Tag, these labels can be bought by 2€. As explained before, are single-use labels and not having the possibility to know when or how long the cold chain has be broken. [3]

Cold Chain real time control system

These systems allow real time monitoring of food transport. By this way, logistics or retail companies have the possibility to check at any time the status of the products and therefore react faster to possible changes or breaks of the cold chain. Such devices usually consist of one or more temperature sensors that are placed at different points of cooling chamber and directly connected via WiFi or Bluetooth to a router or gateway, which will send real-time measurements to the server. After this, the temperatures records could be displayed in a mobile device or web interface.

- **Pros:** these are really complete systems offering a wide range of sensors type and multitude additional services (display tools, mobile applications, etc) that facilitate both employees and logistics companies.
- **Cons:** typically these systems are usually quite expensive for the number of services and the data quality they can collect (high accuracy). Therefore, these devices are only achievable for companies with a big business volume or for transports which require a comprehensive control of their temperature (food, vaccines, etc). In addition, the installation of these types of systems are usually hard to install (see example below).
- **Example:** Astrata is a company that focuses part of his business to control the cold chain in worldwide transports. They offer a full comprehensive service, both at control temperature inside cold storages, as at display on different devices level. This system costs around 2000€ and it requires an specific installation, because the temperature sensors are directly connected to the truck or car battery via CAN BUS connection.

Project Objectives

This project is based on the last system mentioned before (real Time cold Chain system control). A number of goals will be taken in mind which will be focused on improving the advantages of this type of system (functionality and visibility) and also to minimize or delete the disadvantages they currently have.

Before setting these goals it is important to highlight the most important aspects that a company will take into account when choosing any of the previous mentioned systems.

- Cost of the transported product: Logistics Company will assume more risks and make less comprehensive cold chain control for lower cost price products.
- Temperature change sensivity: companies will take more temperature measures for products which have more sensitivity to temperature change. That is why they will use more advanced and expensive systems to control the cold chain. Products with higher temperature change sensitive are usually more expensive in terms of production and maintenance. A good example for this would be the transport of vaccines.
- Business volume of the company: the cold chain control depends largely on the size of the company and therefore the business volume they have. The higher business volume it has, the higher budget will be available to make a more comprehensive cold chain control.

These are the main objectives (ordered from highest to lowest priority) that had been defined at the start of the project.

- Maximize the relationship between cost and functionality.
- Possibility to view data in real time in any mobile device.
- WiFi connection between devices.
- Minimize the device size as possible
- Easy change between different cold storages.
- Easy installation

There are many products that cover all price and functionality possibilities. Therefore, the main objective of this project is to obtain the lowest cost without reducing the functionality of the system. This cost saving will be applied to all the prototype area, both the microprocessor and the temperature sensor, router and battery. Note that this adjustment in the cost is also extremely necessary as it is intended to place the device in each of the pallets of the truck.

Therefore, if a large cost adjustment is not done, the final price will increase making the prototype unfeasible for companies with small benefits.

Although all the efforts to make the device as cheap as possible, the future client should have available a range of facilities during the use of it. Once of the most important is to have the possibility to view (in real time) and set up certain parameters of the device without the necessity of using and specific element for the reading of this values. In this way, the logistic company or even the truck driver can check the transport temperature both from a mobile device and a web interface respectively.

WiFi connection has been chosen in order to make feasible the data view and the setup of the parameters. This will work as a concept test.

On the other hand, the prototype must also facilitate employees tasks such as cargo loading and unloading. This is really important for the viability of our project because the device will be coupled to the bottom of the pallet. As it has a small size, it facilitates the loading and unloading of the pallet for the employee minimizing the possibility of falling or loss of the device.

In addition, the small size of the device must be accompanied by the ability of recognizing the truck change or cold storage. This is really important because the same pallet could be used in different trucks and therefore the device should be able by his own to send the data in the same way and with no problems.

Although with the objectives mentioned before the main functionalities at user level are covered, the ease of the devices installation inside cold storages should be also considered. As it was mentioned in Section 1.1, very advanced and complex systems currently exist which require a non-friendly installation inside cold storages. One of these systems is Astrata, which offers a complete system of cold chain tracking, needs a small job inside the cold storage in order to connect to the power supplier (truck battery) to the device (using CAN-BUS connection).

Project Plan

To carry out efficiently the project and develop in 1 academic year estimated, it was divided into incremental phases. At the end of each of these phases and as an indispensable request to advance to the next phase, meeting with the tutors were made. These meetings served as status control of the project, in which the completed tasks of the previous phase were reviewed.

Below the main phases of the project are shown, including the characteristics of each and the time space the occupied. All these phases can be checked in detail in the Gantt chart that appears in the next page.

1st Phase: Investigation (October)

In this phase, the temperature sensor and the chip have been investigated. After analyzing the different models and checking which of them suited the best way, we proceed to initial selection of the thermometer and chip. As it was an initial phase, fortnightly meetings were fixed with tutors in order to facilitate the start of the project.

2nd Phase: Initial code development

Once investigated both the temperature sensor and the selected chip, an initial code was developed, which only measured the temperature and send it to a web tool called ThingSpeak, which will be reviewed in detail in following sections. Furthermore, at this phase an initial test was run with the previously mentioned code version with the aim of testing the device functionality using an external power supply.

3rd Phase: Work distribution and development

Before Christmas holidays a meeting was held in order to divide the future activities of each of the group members. In this way, the project was divided into 2 different parts:

- Functional node code development: here it is included the selective connection depending on the WiFi signal strength, the temperature measurement, and the sequential sending to the server.
- Development of the communication between node and server and mobile application (one for parameters configuration and other for the data visualization). In addition, at the end of this part a bridge (or link) was implemented between the server and the web data tool ThingSpeak so that in this way the future client has several options for viewing the data (from mobile application or web)

4th Phase: Testing

This was the last phase related to the prototype development. Tests in Facultad de Agronomos cold storages were made in order to check the functionality and connectivity between node and server under cold and low WiFi strength (read Section 4. Experimentación).

Also device consumption tests were made, which can be checked in section 4, Test Consumption.

5th Phase: Documentation

The main objective of this phase was to create the final document for the project.

Conclusions

In this section will be described the obtained conclusions after the developing of the project.

The main goal with this project was the development of a prototype designed to control and track the cold chain based on the IoT technology that would ensure the same functionality as existing solutions but offering a lower cost.

Regarding to the objectives described in section 1, we have been able to get the lowest cost without reducing any way the full functionality developed for this device and its small dimensions. In this way, the device turns easily to be installed and exchanged between cold storages.
Also, as is described in section 3, there is the possibility to visualize data at any time from any device, whether mobile, tablet or computer.

During the development of the project, we found the disadvantages of a not well known system (SDK) used for programming the chip and the little quantity of available documentation. This disadvantages supposed a lot of difficulties while the development, although this SDK avoid us to develop more code.
About the technology that has been used, we have considered to continue using it because of the good results obtained with the context test which, has been performed using WiFi technology.