Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería Informática, Tecnologías Informáticas

TRABAJO DE FIN DE GRADO

Demostrador de sincronizador temporal para IoT

Curso 2020/2021

Autor: Francisco Sánchez López
Tutor: Julián Viejo Cortés
Cotutor: Jorge Juan Chico

Resumen

Hoy en día, con el auge del IoT (Internet of Things) o IdC (Internet de las cosas), las RIS (Redes inalámbricas de sensores, o Wireless Sensor Network, WSN en inglés) están experimentando una gran expansión. La capacidad de monitorización de estas redes ha demostrado tener mucha utilidad en múltiples áreas como las redes eléctricas, gestión del medio ambiente, redes de transportes, procesos industriales, medicina y domótica. En muchas de estas áreas, tener una buena referencia de tiempo sobre la que situar la información obtenida a través de los sensores o sobre la que realizar tareas, puede ser crucial para el buen funcionamiento de la red, lo que significa que se precisa de protocolos de sincronización temporal. En el resto de Internet, este problema está resuelto usando protocolos como NTP (Network Time Protocol) y PTP (Precision Time Protocol). Estos protocolos son ampliamente usados y satisfacen las necesidades y requisitos del mundo de los ordenadores, pero no son protocolos pensados ni para redes de sensores ni para el IoT. Aunque este tipo nuevo de redes son muy variadas, todas tienen características comunes que difieren enormemente de las del resto de Internet. Una gran diferencia son los nodos: en el Internet convencional, los equipos son potentes ordenadores conectados a Internet a través de redes de alta capacidad, mientras que en las redes de sensores se tiene una gran cantidad de pequeños nodos con capacidades de memoria, energéticas y de cálculo muy limitadas. Estas diferencias ponen de manifiesto la necesidad de nuevos protocolos que se adapten mejor a este nuevo paradigma, teniendo en cuenta las nuevas limitaciones y aprovechando el uso de redes inalámbricas. Este trabajo empieza explorando los tipos de protocolos de sincronización temporal propuestos para diferentes redes, y estudiando sus características. Una vez analizados los diferentes tipos nos centraremos en el protocolo que nos ha parecido más interesante, FCSA (Flooding with Clock Speed Agreement), y explicaremos por qué. El objetivo del trabajo consiste en implementar un demostrador de este protocolo.

Palabras clave: redes de sensores, IoT, sincronización temporal, FCSA.

Abstract

Nowadays, with the rise of the Internet of Things (IoT), Wireless Sensor Networks (WSN) are going through a huge expansion. Monitoring capabilities of those networks are realized to be very useful in a lot of subjects like electrical networks, environment management, transports networks, industrial processes, medicine and domotics. In many of these topics, the ability to point data from sensors (or to actuators) on time with high accuracy is mandatory for success, so it means that time synchronization protocols are needed. In the conventional Internet, that issue is well solved with protocols like the NTP (Network Time Protocol) and PTP (Precision Time Protocol). These protocols are widely used and they fulfill all PC environment necessities, but their usefulness is limited in other kinds of networks. Despite this, new networks are very different between them, they share common properties that are very far from the convencional Internet. A big difference are nodes: in the conventional Internet machines are powerful computers always online through high bandwidth connections, meanwhile WSN are very tiny machines with very few resources. These differences manifest the necessity of new protocols adapted to this world taking into account new limitations and taking advantage of new possibilities. This document starts by exploring some of the proposed protocols and their features. After that, we will see in detail the most interesting protocol for us, FCSA (Flooding with Clock Speed Agreement) and why it is adequate. The goal of this project consists of implementing a demo of this protocol.

Keywords: sensors network, IoT, time synchronization, FCSA.

Índice

1. Introducción

Cualquier servicio o aplicación que funcione en una red de sensores tiene que adaptarse a las circunstancias de la misma. Aunque puede haber muchas diferencias entre estas redes y acentuarse una característica concreta en una determinada red, por lo general nos encontramos en el siguiente contexto:

- Hay un gran número de dispositivos autónomos, los llamados nodos, que están distribuidos en el espacio y se comunican inalámbricamente. Su propósito es comunicarse entre ellos para informar de la lectura de algún sensor o la detección de algún evento usando la menor energía posible para aumentar su vida útil.
- La comunicación se realiza a través de un medio compartido, donde los paquetes de datos son escuchados por todos los vecinos.
- Existen uno o más nodos maestros que reciben los paquetes del resto de nodos y los encaminan hacia el resto de Internet. A estos nodos se los conoce como "*gateway*".
- La topología de la red puede cambiar, ya sea por fallos en los nodos o por modificaciones intencionadas en la red.

Esto tiene varias consecuencias a la hora de diseñar una aplicación o elegir un protocolo de comunicaciones para estas redes. Por ejemplo, cuantos más paquetes envíe una aplicación más veces habrá que activar el transmisor de radio, lo que supone más consumo energético y mantener ocupado más tiempo el medio compartido, aumentando también el riesgo de colisión. Ahora se vuelve conveniente e interesante ajustar lo máximo posible el uso de la red y los cálculos necesarios.

Así pues, no todo son restricciones ya que estas redes presentan oportunidades que no se encuentran en comunicaciones punto a punto. Por ejemplo, en este entorno, los paquetes de datos se envían a través de un medio inalámbrico, que por su naturaleza es compartido. Esto significa que se puede aprovechar la emisión de un paquete para informar a varios nodos de una sola vez y así evitar hacer múltiples transmisiones, ahorrando uso de la red y tener que activar y desactivar el transmisor de radio varias veces.

En este nuevo paradigma, los protocolos que actualmente están muy extendidos, como NTP y PTP no son adecuados. En el caso de NTP, es un protocolo diseñado para redes de computadores donde varias conexiones cliente-servidor de forma periódica es despreciable para el funcionamiento del sistema. Como consecuencia directa, no se aprovechan las conexiones multipunto ni tampoco se optimiza el uso de las comunicaciones. Por otro lado, requiere realizar operaciones matemáticas complejas para estimar el tiempo, algo que para un ordenador personal es muy sencillo pero que para un nodo de una red de sensores puede llegar a ser un verdadero reto. En el caso de PTP, requiere *hardware* caro, complejo y voluminoso que hace inviable su despliegue masivo en una red de sensores.

1.1. Motivación

Diseñar nuevos protocolos que se adapten a las nuevas circunstancias es un proceso vital para poder contar con sincronización temporal en redes de sensores. Sin embargo este proceso no tiene por que limitarse solo a sortear las nuevas limitaciones, si no que además, dada la variedad de estas redes, pueden surgir nuevos tipos de protocolos radicalmente diferentes que serán idóneos dependiendo de la naturaleza de la red o de su propósito. Explorar estas nuevas posibilidades es un aspecto fundamental de este trabajo.

1.2. Objetivo

Nuestro objetivo principal consiste en desarrollar un demostrador de un sincronizador temporal para IoT. Para ello vamos a necesitar otros objetivos secundarios que también son necesarios de forma indirecta:

1. Hacer un **estudio previo de los protocolos actuales** y encontrar uno que se adapte a nuestras necesidades. Como se ha dicho anteriormente, los protocolos NTP y PTP no son los más aptos para redes de sensores, por tanto es necesario buscar un protocolo diferente, para ello hemos de estudiar cómo está la escena actual en este tema.

2. Buscar una **plataforma que se adapte a un entorno de IoT y redes de sensores**. El ecosistema del IoT y las redes de sensores tienen unas características particulares como se ha dicho anteriormente. Dado que estas redes pueden llegar a contar con muchos nodos, estos tienen que ser baratos y fáciles de mantener. Aunque dependerá del entorno de la red, es posible que los nodos se encuentren en una superficie extensa, donde sea necesaria una comunicación de gran alcance. También un fácil mantenimiento es la capacidad de autoprogramarse o la facilidad para reprogramar los nodos de forma externa pues puede ser que estos nodos tengan un difícil acceso, lo que puede significar que acceder físicamente a los nodos sea costoso. Todas estas consideraciones hay que tenerlas en cuenta a la hora de escoger el *hardware*.

3. Desarrollar la implementación en sí basándose en lo decidido por los otros dos objetivos anteriores. Como requisito general, la implementación debe ser compatible con un entorno IoT.

1.3. Alcance

Para definir el alcance primero hay que conocer de cuánto tiempo se dispone. Según la normativa de Trabajos Fin de Estudios, un Trabajo Fin de Grado (TFG) consta de 12 créditos, equivaliendo cada crédito a 25 horas de trabajo. Por tanto, esto supone un total de 300 horas, teniéndose que definir un alcance que se ajuste a este tiempo.

Por ello, se define el alcance en los siguientes puntos:

1. Razonar la elección de un protocolo adecuado para el propósito del proyecto, es decir, un protocolo para la sincronización temporal diseñado para redes de sensores.

2. Razonar la elección de una plataforma adecuada para este propósito.

3. Diseñar, especificar e implementar un demostrador del protocolo elegido sobre la plataforma elegida.

En resumen, el alcance del proyecto es **diseñar e implementar un demostrador de un protocolo de sincronización temporal para IoT**.

1.4. Estructura de la memoria

La memoria se organiza de la siguiente manera:

1. **Introducción**: Parte actual donde se define de forma breve el por que del trabajo, el objetivo y el alcance.
2. **Planificación**: Se detallarán las fases del proyecto, las tareas y los roles correspondientes de forma detallada.
3. **Estudio previo**: Aquí se expondrá lo encontrado acerca de redes de sensores y protocolos de sincronización. Se hará un acercamiento al IoT, a las redes de sensores con detenimiento.
4. **Implementación del demostrador**: En esta parte se diseña el demostrador basándose en lo aprendido del estudio previo. También se documentan las especificaciones usadas y generadas durante la implementación del mismo.
5. **Pruebas**: Evaluación del demostrador y los resultados.
6. **Conclusiones**: Se habla del desarrollo del proyecto en retrospectiva y se ven formas de continuar el desarrollo del proyecto.

2. Planificación

A continuación se procede a exponer los detalles de la planificación temporal empleada para la realización del proyecto.

2.1. Lista de fases y tareas

Veamos las tareas necesarias para la realización del proyecto:

- Iniciación.
    - Definir el objetivo del proyecto.
    - Definir el alcance del proyecto.
    - Esbozar las tareas a realizar.
- Planificación.
    - Definir en detalle las fases del proyecto, sus tareas y sus roles.
    - Establecer la estructura de la presente memoria.
- Realizar el estudio previo.
    - Obtener y analizar documentación relevante sobre IoT y redes de sensores.
    - Obtener y analizar documentación relevante sobre plataformas *hardware* y software para implementar una red de sensores.
    - Obtener y analizar documentación sobre protocolos de sincronización temporal para IoT.
    - Documentar el proceso en la memoria.
- Diseñar el demostrador.
    - Sopesar alternativas y elegir la arquitectura de la red.
    - Sopesar alternativas y elegir el *hardware* que se va a usar para implementar la red.
    - Sopesar alternativas y elegir un protocolo de sincronización.
    - Estudiar en detalle el protocolo de sincronización elegido.
    - Documentar el proceso en la memoria.
- Implementar el demostrador.
    - Generar las especificaciones de los formatos y estructuras a usar.
    - Ensamblar los nodos según el diseño.
    - Testear el entorno de desarrollo en los nodos.
    - Testear el módulo de comunicación de los nodos.
    - Testear la comunicación entre nodos.
    - Implementar el protocolo en los nodos.
    - Documentar el proceso en la memoria.
- Pruebas.

    - Diseñar pruebas.
    - Realizar pruebas.
    - Documentar el proceso en la memoria.
- Cierre del proyecto.
    - Valorar el resultado del proyecto respecto a los objetivos y el alcance.
    - Proponer trabajos futuros.
    - Documentar el proceso en la memoria.

2.2. Planificación temporal

Fase de iniciación

En esta fase se concreta el objetivo y el alcance del proyecto y de forma gruesa se define la planificación a seguir. Se acuerda entre los tutores y el alumno realizar reuniones periódicas de entre 1 y 2 horas una vez cada semana o cada dos, a convenir según se desarrolle el trabajo.

Tarea	**Tiempo**	**Rol**
Definir el objetivo del proyecto	2 horas	Jefe de Proyecto
Definir el alcance del proyecto	2 horas	Jefe de Proyecto
Esbozar las tareas a realizar	2 horas	Jefe de Proyecto
Total	**6 horas**	

Tabla 1.1.

Fase de planificación

En esta fase se genera la documentación referente a la planificación temporal.

Tarea	**Tiempo**	**Rol**
Definir en detalle las fases del proyecto, sus tareas y sus roles.	6 horas	Jefe de Proyecto
Establecer la estructura de la presente memoria.	5 horas	Jefe de Proyecto
Total	**11 horas**	

Tabla 1.2.

Fase de realización del estudio previo

Esta fase es crítica para el resto del desarrollo del proyecto. El resultado de esta fase determinará en gran medida el resto, sobre todo fases como el diseño del demostrador o la elección de la plataforma a usar, pues aquí se definirán las necesidades técnicas que ha de satisfacer el demostrador.

Tarea	**Tiempo**	**Rol**
Obtener y analizar documentación sobre IoT y redes de sensores	20 horas	Técnico de redes
Obtener y analizar documentación sobre plataformas *hardware* y *software* para implementar una red de sensores	20 horas	Técnico de *hardware*
Obtener y analizar documentación sobre protocolos de sincronización temporal para IoT	20 horas	Técnico informático
Documentar el proceso en la memoria	5 horas	Jefe de Proyecto
Total	**65 horas**	

Tabla 1.3.

Fase de diseño del demostrador

En esta fase se sopesan las diferentes alternativas tecnológicas para implementar el demostrador y se seleccionará la más adecuada.

Tarea	**Tiempo**	**Rol**
Sopesar alternativas y elegir la arquitectura de la red	8 horas	Jefe de Proyecto
Sopesar alternativas y elegir el *hardware* que se va a usar para implementar la red	8 horas	Jefe de Proyecto
Sopesar alternativas y elegir un protocolo de sincronización	15 horas	Jefe de Proyecto

Estudiar en detalle el protocolo de sincronización elegido	25 horas	Desarrollador
Documentar proceso en la memoria	4 horas	Jefe de Proyecto
Total	**60 horas**	

Tabla 1.4.

Fase de implementación del demostrador

En esta fase, basándose en las elecciones de la fase anterior, se procederá a desarrollar el demostrador en sí. El producto de esta fase consistirá en el propio demostrador.

Tarea	**Tiempo**	**Rol**
Generar las especificaciones de los formatos y estructuras a usar	15 horas	Desarrollador
Construir los nodos según el diseño	25 horas	Desarrollador
Testear el entorno de desarrollo de los nodos	8 horas	Desarrollador
Testear el módulo de comunicación de los nodos	8 horas	Desarrollador
Testear la comunicación entre nodos	8 horas	Desarrollador
Implementar el protocolo en los nodos	25 horas	Desarrollador
Documentar el proceso en la memoria	4 horas	Jefe de Proyecto
Total	**93 horas**	

Tabla 1.5.

Fase de pruebas

Una vez implementado el demostrador, procedemos a testear su calidad.

Tarea	**Tiempo**	**Rol**
Diseñar todas las pruebas	15 horas	Tester

Realizar pruebas	25 horas	Tester
Documentar proceso en la memoria	4 horas	Jefe de Proyecto
Total	**44 horas**	

Tabla 1.6.

Fase de cierre del proyecto

En la fase final vamos a documentar el resultado del proyecto, valorar el mismo y proponer trabajos futuros.

Tarea	**Tiempo**	**Rol**
Valorar el resultado del proyecto respecto a los objetivos y el alcance	8 horas	Jefe de Proyecto
Proponer trabajos futuros	8 horas	Jefe de Proyecto
Documentar proceso en la memoria	2 horas	Jefe de Proyecto
Total	**18 horas**	

Tabla 1.7.

2.2.1 Suma total

Fase	**Tiempo**
Iniciación	6 horas
Planificación	11 horas
Realización de estudio previo	65 horas
Diseño del demostrador	60 horas
Implementación del demostrador	93 horas
Pruebas	44 horas
Cierre del proyecto	18 horas
Total	**297 horas**

Tabla 2.

2.2.2 Suma por roles

Rol	**Tiempo**

Jefe de Proyecto	83 horas
Técnico de redes	20 horas
Técnico de *hardware*	20 horas
Técnico informático	20 horas
Desarrollador	114 horas
Tester	40 horas
Total	**297 horas**

Tabla 3.

2.3. Análisis de costes

A continuación se exponen los costes de desarrollar e implementar el proyecto. Por un lado vamos a ver el desglose por fases y roles y por otro lado el coste de los materiales:

2.3.1 Costes por roles

Rol	**Precio (€) / hora [1]**	**Tiempo**	**Total por rol**
Jefe de Proyecto	46 € / hora	83 horas	3.818 €
Técnico de redes	38 € / hora	20 horas	760 €
Técnico de *hardware*	38 € / hora	20 horas	760 €
Técnico informático	38 € / hora	20 horas	760 €
Desarrollador	28 € / hora	114 horas	3.192 €
Tester	28 € / hora	40 horas	1.120 €
Total		**297 horas**	**9.650 €**

Tabla 4.

2.3.2 Coste de los materiales

Como se verá más adelante, el demostrador puede estar compuesto de más o menos partes, por tanto habrá que ver el coste de los materiales según el número de nodos que queramos implementar. Aun así, se espera que el demostrador cuente con al menos un par de nodos, por tanto podemos fijar un coste de los materiales mínimo necesario para construir un demostrador.

En el apartado de *software*, todas las herramientas que se van a usar van a ser gratuitas, como es Arduino IDE, librerías para el modem SX1276 y el *toolchain* para el ESP8266.

Coste del *hardware*

En este caso se han usado los siguientes componentes para cada nodo:

Componente	Unidades	Precio por unidad
Wemos D1 Mini ESP8266	x1	2,72 €
SX1278	x1	2,88 €
Breadboard	x1	4,00 €
Wires	x10	0,50 €
Total		**10,1 €**

Tabla 5.

Los precios que aquí se recogen son del proveedor de donde fueron adquiridos, en este caso fue Aliexpress España, en otros proveedores puede variar.

2.4. Viabilidad del proyecto

Vamos a proceder a identificar y exponer los riesgos a los que se enfrenta el desarrollo del proyecto y a vislumbrar soluciones o posibles alternativas.

2.4.1 Identificación de riesgos

Aunque el proyecto en sí es más complejo, se pueden generalizar las tareas a las siguientes líneas generales:

- Buscar información acerca de un tema (protocolos de sincronización temporal)
- Diseñar y desarrollar un demostrador del mismo tema, lo que implica:
    - Comprender el protocolo en profundidad
    - Buscar una plataforma *software* y *hardware* sobre la que implementar el demostrador
    - Diseñar el demostrador e implementar dicho diseño

Este tipo de tareas no difieren mucho de otros proyectos donde se desarrolla algún producto. La única diferencia objetiva es la complejidad del proyecto. Vamos a ver más detalladamente los riesgos que se contemplan.

Riesgos desde el punto de vista técnico

Desde el punto de vista técnico no se plantea ningún riesgo relevante. El proyecto implica a partes iguales buscar y comprender protocolos y especificaciones para posteriormente implementar lo aprendido, osea programar y ensamblar algo de *hardware*.

El tema que se trata no es algo absolutamente puntero o en un estado embrionario, donde la documentación escasea. Dentro de los protocolos de sincronización podemos hallar gran cantidad de documentación y estudios y las plataformas para IoT están experimentando una gran expansión en los últimos años.

Dado el desarrollo del proyecto está a cargo de alguien cualificado para las tareas, no parece que vayamos a correr ningún gran riesgo en este apartado, aunque eso no es impedimento para surjan contratiempos y dificultades, como:

- Corremos el riesgo de encontrarnos ante un tema excesivamente complejo que aunque bien sea abordable, puede desbordarnos en tiempo, lo que significaría una demora en el tiempo de finalización y un sobrecoste.

Riesgos desde el punto de vista económico

En este apartado tampoco hay grandes riesgos. En el caso de un completo fracaso del proyecto, el *hardware* adquirido para la implementación del proyecto sigue siendo válido y funcional, por lo que podría ser reusado en otro proyecto o devuelto. No ocurre así con las horas empleadas antes de declarar el fracaso.

Se considera, con probabilidad muy baja, de que a la hora de implementar el demostrador necesitemos algún tipo de licencia que encarezca el proyecto o que necesitemos algún *hardware* específico para lograr los objetivos, pero esto último parece aún menos probable pues justo tratamos de evitar eso.

Riesgos desde el punto de vista temporal

Nada nuevo que decir. A excepción de la posibilidad de una excesiva complejidad que obliga a redimensionar el tiempo y/o los recursos asignados al proyecto, no se contempla nada que pueda alargar la duración del proyecto.

Resumen de riesgos

Por tanto, **el principal riesgo que identificamos es la posibilidad de una excesiva complejidad** en el tema de buscar, comprender e implementar un protocolo de sincronización temporal para IoT, ya sea porque hay mucha documentación donde buscar, porque sean algoritmos muy complejos o por que la implementación sea especialmente ardua.

2.4.2 Planes de contingencia

Explorados los riesgos, vamos a definir los planes de contingencia a seguir en caso de encontrarnos en estas situaciones: si nos encontrásemos en la situación de que la complejidad del proyecto está muy por encima de lo esperado, planteamos estas tres vías:

1. Agregar recursos al proyecto: dada la excesiva complejidad, una vía es valorar las nuevas circunstancias y ampliar los plazos de tiempo. Por esta vía, podríamos llegar a cumplir el objetivo del proyecto, a costa de sobrecostes.

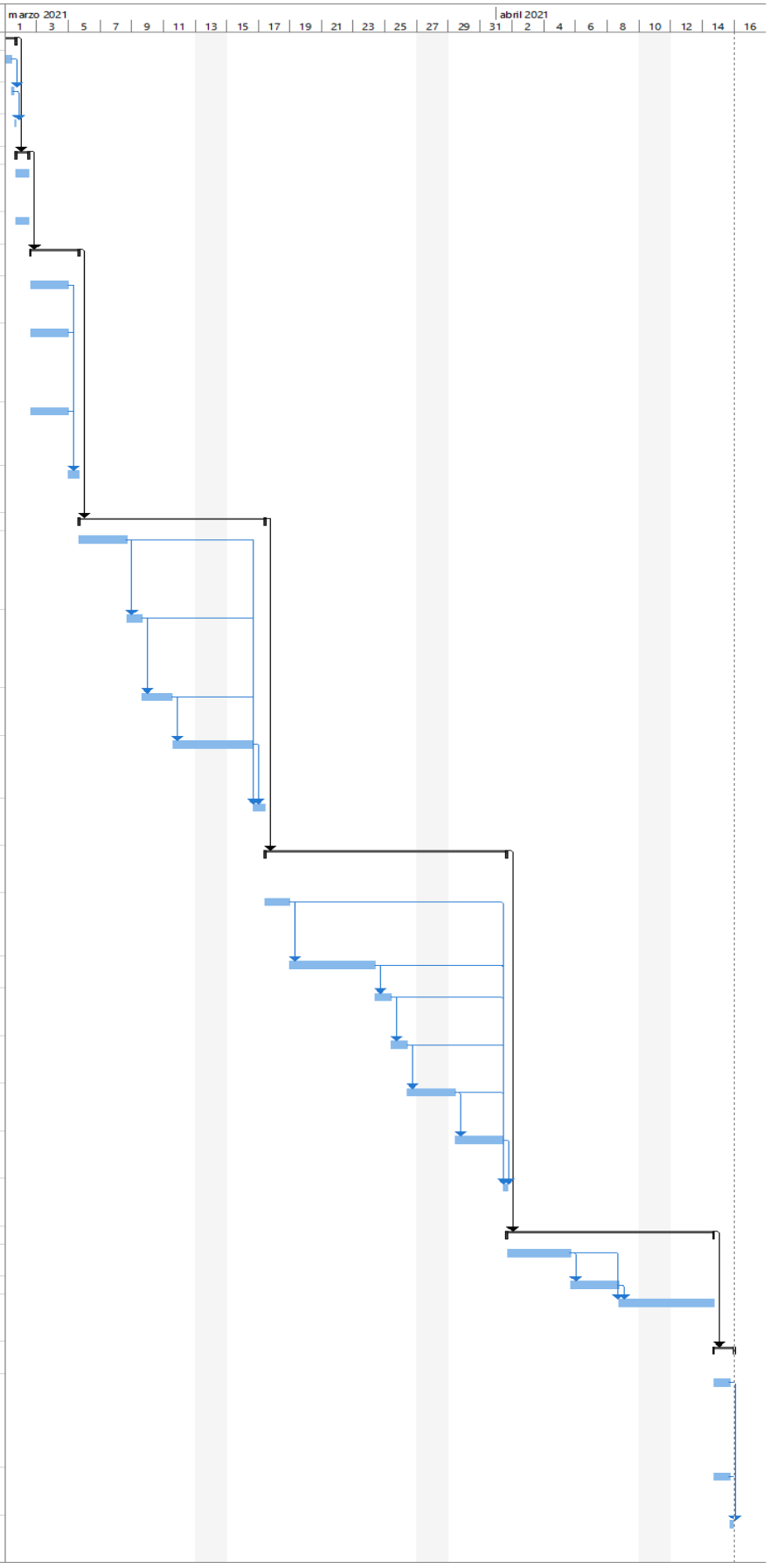
2. Reducir el alcance del proyecto: ya que la complejidad impide poder culminar el objetivo, se pueden rebajar las exigencias del alcance, reduciéndolo por ejemplo solo a diseñar el demostrador, dejando la implementación para otro proyecto. Por esta vía no tenemos sobrecostes y tiempos adicionales pero conseguimos un éxito parcial.

3. Declarar el fracaso del proyecto: en este caso no todo tienen por que ser pérdidas, puesto que dependiendo del estado del desarrollo del proyecto puede documentarse lo encontrado hasta la declaración de fracaso.

La elección del plan de contingencia adecuado dependerá en gran medida del estado del proyecto. No es lo mismo darse cuenta de la complejidad del área a estudiar al principio del estudio previo que por problemas prácticos a la hora de implementar el demostrador.

2.5. Cronograma

Id	Nombre de tarea	Duración
1	**Fase de iniciación**	**0.75 días**
2	Definir el objetivo del proyecto	2 horas
3	Definir el alcance del proyecto	2 horas
4	Esbozar tareas a realizar	2 horas
5	**Fase de planificación**	**0.75 días**
6	Definir en detalle las fases del proyecto, sus	6 horas
7	Redacción de la presente memoria	5 horas
8	**Fase de realización del estudio previo**	**3.13 días**
9	Buscar documentación sobre IoT y redes	20 horas
10	Buscar documentación sobre plataformas hardware y software para	20 horas
11	Buscar documentación sobre protocolos de sincronización	20 horas
12	Documentar el proceso en la memoria	5 horas
13	**Fase de diseño**	**7.5 días**
14	Sopesar alternativas y elegir la arquitectura de la red	8 horas
15	Sopesar alternativas y elegir el hardware que se va a usar para implementar	8 horas
16	Sopesar alternativas y elegir un protocolo	15 horas
17	Estudiar en detalle el protocolo de sincronización elegido	25 horas
18	Documentar el proceso en la memoria	4 horas
19	**Fase de implementación del demostrador**	**11.63 días**
20	Generar las especificaciones de los formatos y estructuras a usar	15 horas
21	Construir nodos según el diseño	25 horas
22	Testear el entorno de desarrollo de los nodos	8 horas
23	Testear el módulo de comunicación de los nodos	8 horas
24	Testear la comunicación entre los nodos	8 horas
25	Implementar el protocolo de comunicación	25 horas
26	Documentar el proceso en la memoria	4 horas
27	**Fase de pruebas**	**9 días**
28	Diseñar todas las pruebas	15 horas
29	Realizar pruebas	25 horas
30	Documentar el proceso en la memoria	4 días
31	**Fase de cierre del proyecto**	**1.25 días**
32	Valorar el resultado del proyecto respecto a los objetivos y el alcance	8 horas
33	Proponer trabajos futuros	8 horas
34	Documentar el proceso en la memoria	2 horas



3. Estudio previo

En este apartado se va a exponer la documentación encontrada como estudio previo para lograr los objetivos. Esto incluye información acerca de protocolos de sincronización, redes de sensores y plataformas tanto *hardware* como *software* para implementar una red de sensores y un protocolo. Primero, empezaremos viendo qué es el IoT y las redes de sensores para posteriormente explicar con más detenimiento sus detalles, principalmente sus posibles topologías y tecnologías. Para finalizar, veremos qué tipos de protocolos de sincronización hemos encontrado y cómo podemos implementarlos con las plataformas usadas.

3.1. Vista general

Para contextualizar las redes de sensores primero comencemos hablando del IoT. Puesto que es un ente que ha aparecido relativamente pronto (a principios del S. XXI) [2], aún no hay una única definición estandarizada. Cada organización tecnológica, instituto de estándares tecnológico o gran empresa tecnológica (IBM, Google, Facebook, Gartner, SAP, Wikipedia, etc.) define el IoT con sus propios matices, pero una definición general y común sería considerar al IoT como “una red de objetos físicos que contienen la tecnología para comunicarse, sin necesidad de interacción humana, entre ellos y ser capaces de interactuar con sus estados internos y externos”.

En un lenguaje más cotidiano, se trata de un Internet enfocado a las “las cosas” y no a las personas. En el Internet convencional, la información proviene en su mayoría de personas: el tráfico que se genera proviene de la interacción de seres humanos con otros humanos a través de la red; por ejemplo, cuando se envía un mensaje o un correo, se accede a una página web, se sube una foto a una red social o se publica un *tweet*.

En el *Internet de las Cosas*, los principales participantes de la red no son seres humanos sino que lo son ordenadores que de forma automática generan el tráfico en la red. En este nuevo Internet, los participantes se dedican a transmitir datos acerca de su entorno, que puede ser desde datos ambientales de una ciudad (por ejemplo, la temperatura en la calle, la luminosidad, la afluencia de tráfico y personas, el ruido, etc.) hasta datos biométricos de algún ser vivo (desde plantas hasta personas).

En la década del 2010, el IoT ha tenido un crecimiento exponencial, superando muchas expectativas. El ritmo de crecimiento del número de cosas conectadas al Internet de las Cosas aumenta cada año. Según estudios, el número de objetos que formaban el IoT en 2016 era de 8.400 millones [3], en 2018 el número de objetos conectados al IoT superó al de dispositivos móviles convencionales y se estima que en 2021 ya hay una media de 7 nodos de IoT por habitante del planeta.

3.2. Redes de sensores

El concepto de red de sensores es previo al del IoT puesto que el primer desarrollo de una red de este tipo se sitúa en la década de los 60, de mano del *Defense Advanced Research Projects Agency* (DARPA) [4], y se refiere a un red con unas características concretas.

Estas redes se definen por ser “un conjunto de dispositivos autónomos (los llamados nodos) que trabajan de forma colaborativa para recolectar información del ambiente”.

El concepto en sí no va más allá, pero en mucha literatura (y en la inmensa mayoría de redes de sensores) también se caracteriza a estas redes por el bajo coste de cada nodo y porque se comunican de forma inalámbrica (lo que sería una red inalámbrica de sensores). En este documento, las referencias a redes de sensores serán a redes de sensores inalámbricas (RIS: Redes Inalámbricas de Sensores o WSN: *Wireless Sensor Network*).

Estas redes tienen otras muchas idiosincrasias, algunas de ellas motivadas por las limitaciones computacionales y enérgicas de los nodos y otras motivadas por el uso de estas redes:

- Topología cambiante: normalmente se contemplan cambios en los nodos (en su número o en la localización de los mismos). Esto se debe a que hay “cosas” que no tienen una posición fija. Por ejemplo, los autobuses urbanos de una ciudad.

- Tolerancia a errores: no es necesario que estén funcionando todos los nodos a la vez para que la red en su conjunto funcione.

- Simplicidad de protocolos: no se implementan todas las capas de red del modelo TCP/IP o del modelo Open System Interconnection (OSI). Esto se debe a que implementar supone una complejidad que los nodos a veces no pueden asumir y que tampoco son necesarios para enviar sencillos paquetes de datos.

Otro componente fundamental de las redes de sensores es el llamado “*gateway*” que consiste en un nodo “maestro” con unas capacidades superiores al resto. Este “*gateway*” se encarga de enrutar los datos de la red de sensores hacia otra red, añadiendo las capas de red que faltan en la comunicación entre los nodos. Este nodo especial proporciona una continuidad entre la WSN y el resto de Internet (ver Fig. 1.1).

Vista esta definición, se entiende por qué las redes de sensores son una pieza clave en el desarrollo del IoT y por qué están tan íntimamente relacionadas.

A nivel operacional, una WSN no tiene que limitarse a monitorizar sino que los nodos también pueden contar con actuadores (en este caso sería una red de sensores y actuadores, WSAN, *Wireless Sensor and Actuators Network* por sus siglas en inglés). Por ejemplo, en una ciudad, una red de sensores y actuadores podrían ser las farolas: en esta red los nodos no sólo informarán de parámetros ambientales sino que también tienen como actuador la capacidad de encender y apagar la luminaria.

El tráfico de estas redes se forma de pequeños paquetes de datos del orden de kilobytes (KB) que se transmiten de forma regular con intervalos del orden de minutos. La información que contienen estos paquetes puede ser sobre sensores, marcas de tiempo, información sobre cambios en la red, etc.

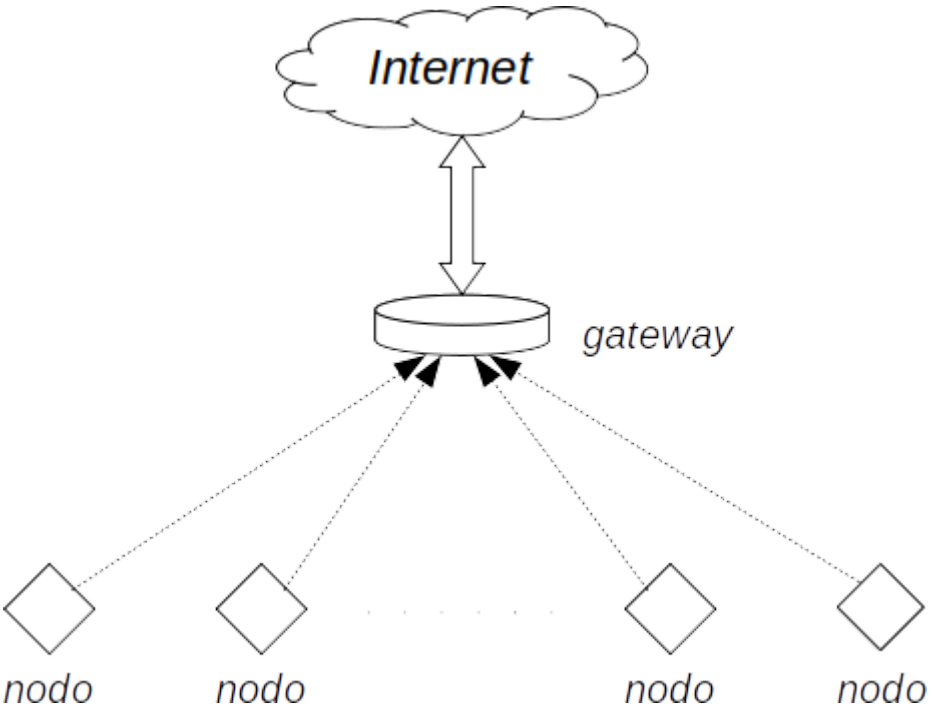


Figura 1.1. Esquema arquitectura básica de una red de sensores.

En la jerga, al conjunto de nodos se los conoce como “polvo inteligente” (*smart dust*) puesto que los nodos son algo muy pequeño, que se comunican de forma inalámbrica y que se administran de forma autónoma y en muchas ocasiones de forma oculta, como el polvo.

3.2.1 Diseño de los nodos

Aunque la variedad de los nodos de una WSN puede ser enorme, todos **suelen compartir una arquitectura común**. Esta arquitectura siempre se compone de una fuente de energía, una unidad de procesamiento, un medio de comunicación y uno o más sensores (ver Fig. 1.2).

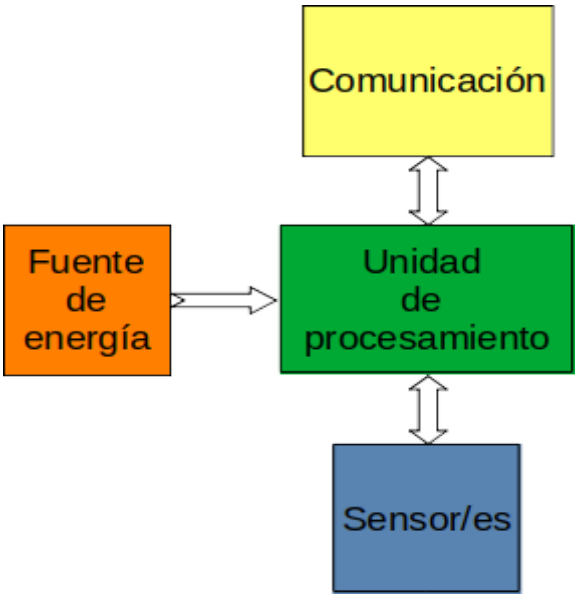


Figura 1.2. Arquitectura básica de un nodo de una red de sensores.

3.3. Tecnologías para las redes inalámbricas de sensores

A la hora de implementar una RIS hay muchos tipos de tecnologías que se pueden adoptar. Para elegir una, primero es conveniente tener claras las necesidades de una red de sensores:

- **Alcance**: aunque puede variar mucho dependiendo de donde se despliegue, en una RIS se va a necesitar comunicaciones con un alcance desde las decenas de metros hasta varios kilómetros; por lo que buscamos una **arquitectura con un alcance medio-largo**.
- **Ancho de banda**: aquí podemos afirmar que una RIS no tiene grandes demandas de ancho de banda, con llegar a velocidades de transmisión de algunos kilobits por segundo (Kbps) es suficiente; con **una tasa de transferencia baja nos basta**.
- **Consumo energético**: el consumo energético es una pieza clave en las redes de sensores. Las limitaciones energéticas de los nodos, que hacen posible que tenga una vida útil mayor, nos imponen usar **una tecnología de bajo consumo**.
- **Coste:** el coste también es un factor clave, necesitamos que sea **lo más bajo posible**.

Vistas nuestras preferencias, vamos a posicionar algunas tecnologías encontradas y a acotar las que se ajustan a nuestro interés (ver Figs. 2.1 y 2.2).

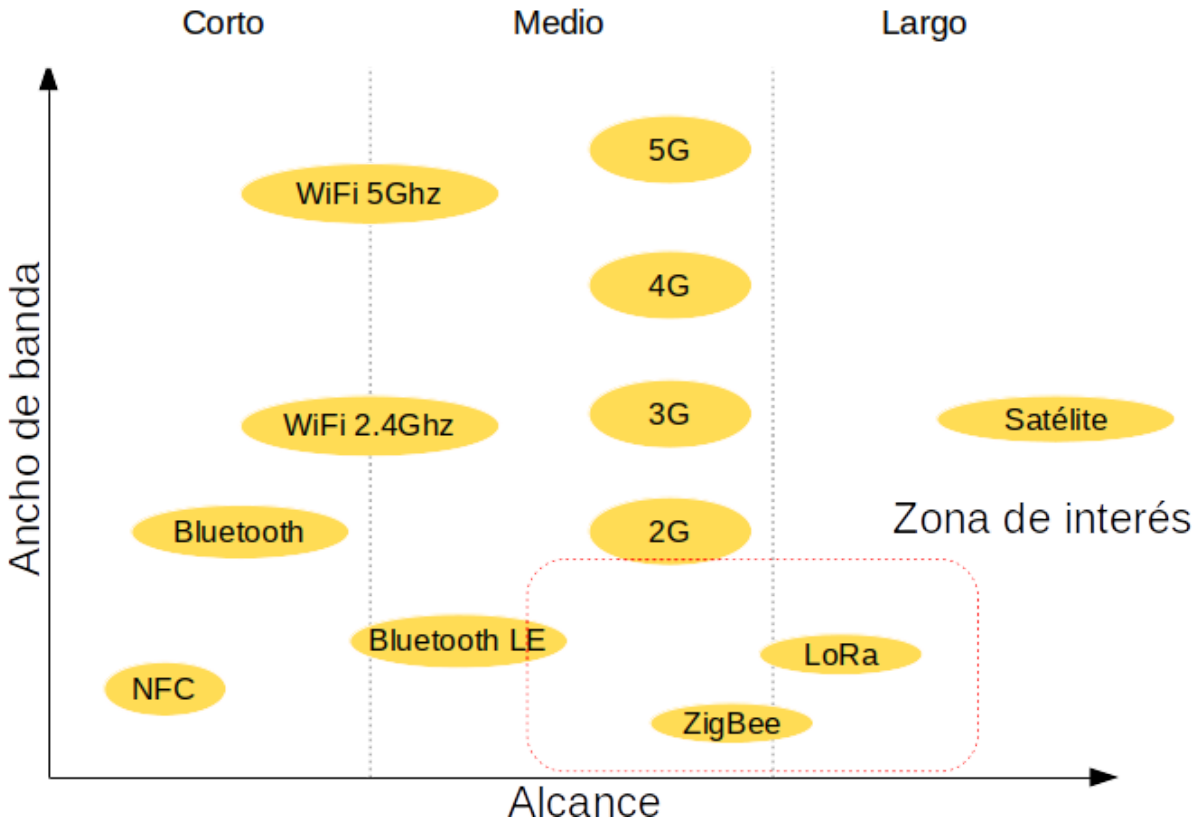


Figura 2.1. Comparativa tecnologías inalámbricas Alcance-Ancho de Banda.

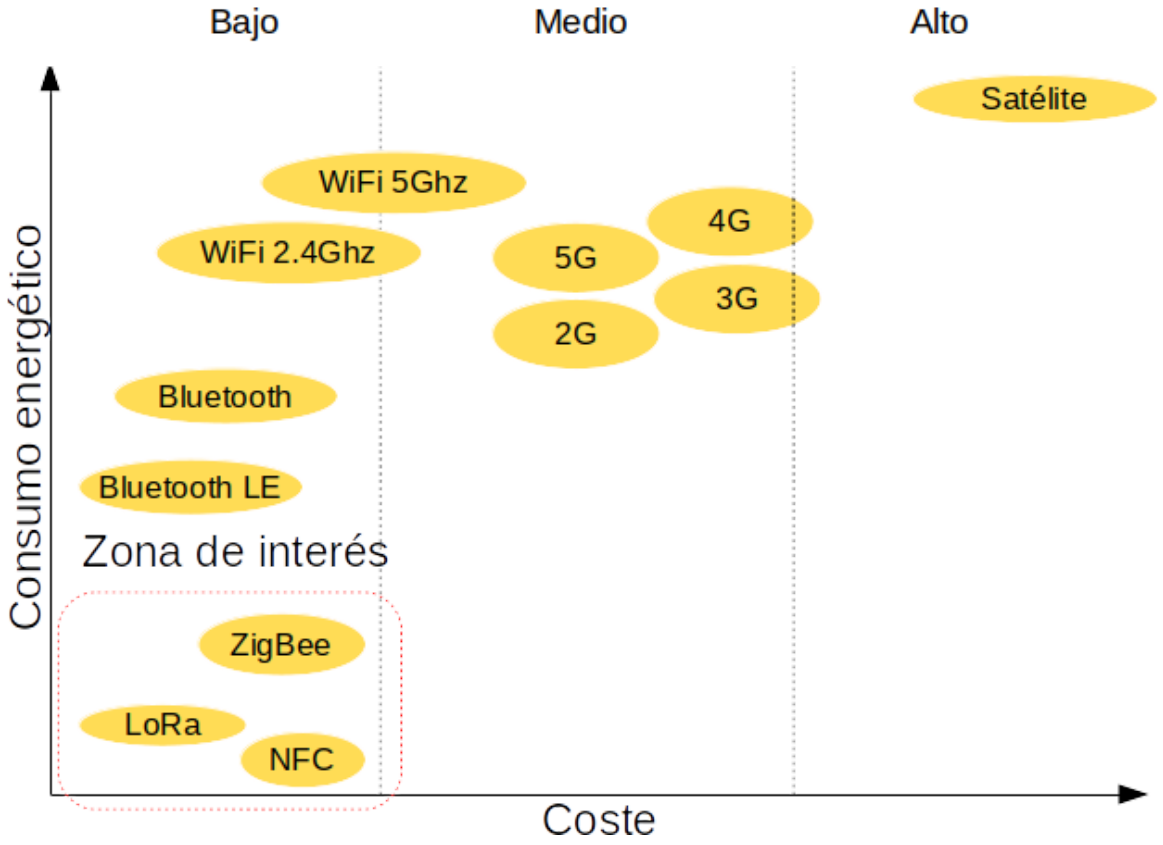


Figura 2.2. Comparativa tecnologías inalámbricas Coste-Consumo energético.

Parece claro los candidatos que tenemos. En la comparativa Alcance-Ancho de Banda (ver Fig. 2.1), está claro que las tecnologías que nos son útiles son *ZigBee* y *LoRa*. En el caso de la comparativa Coste-Consumo energético también vemos claro las candidatas: *ZigBee*, *LoRa* y NFC. La intersección entre estos dos conjuntos de candidatas, nos deja las siguientes tecnologías viables: **LoRa y ZigBee.**

En sucesivos apartados veremos cómo se decidirá cuál usar y por qué.

3.4. Arquitecturas para las redes de sensores

Ahora, vamos a necesitar buscar una arquitectura para la red de sensores que se ajuste a las necesidades. En términos de arquitectura, las necesidades son:

- **Admita medias-largas distancias**: ya se explicó en el apartado anterior
- **Protocolos sencillos**: estas redes son redes con un tráfico de poco volumen y sin problemas de enrutamiento complejo. Pilas de protocolo como TCP/IP tienen muchas prestaciones que nunca se usarán en estas redes, por lo que no hay necesidad de implementar la pila entera. En su lugar, es mejor para la red usar subconjuntos de capas del modelo TCP/IP, pues van a satisfacer las necesidades de la red a la par que no añade mucha complejidad.
- **Topología dinámica:** dado que no se puede asegurar que los nodos estén siempre en la misma posición, necesitamos que la red pueda reorganizarse.

- **Tolerancia a fallos**: la arquitectura debe admitir que los nodos fallen, pero que de igual manera la red siga funcionando, al menos de forma parcial.

Primero, empezaremos viendo una clasificación muy típica según el tamaño de la red (ver Tabla 7).

Nombre	Alcance típico	Uso típico	Ejemplo de tecnología
PAN (*Personal Area Network*)	Poco metros (a veces su alcance se define como la distancia a la que llega una voz humana)	Interconectar dispositivos en una misma sala o habitación	Bluetooth
LAN *(Local Area Network*)	Del orden de decenas de metros, pueden llegar a algunos kilómetros	Interconectar dispositivos de un edificio	WiFi
MAN (*Metropolitan Area Network*)	Del orden de decenas de kilómetros	Interconectar las redes de un pueblo o ciudad	Fixed WiMAX (IEEE 802.16), 3G, 4G
WAN (*Wide Area Network*)	Más allá de todo lo anterior	Interconectar redes en grandes distancias	Mobile WiMAX (IEEE 802.20)

Tabla 7. Comparativa arquitectura de redes.

Aunque ninguna de esta clasificación encaja exactamente con lo que buscamos, parece que lo que más se ajusta son las redes MAN y WAN. Ninguna de estas redes están pensadas específicamente para redes de sensores pero hay variantes que sí.

3.4.1 Redes LPWAN

Las arquitectura de redes LPWAN (*Low Power Wide Area Network*) han aparecido en los últimos años y están teniendo un gran desarrollo tanto de tecnologías como de protocolos. Estas redes, consideradas una variante de las redes WAN, se definen por tener:

- **Alto rango**: de varios kilómetros
- **Baja potencia**: el tráfico típico de estas redes será de poco volumen, consistirá principalmente de sencillos paquetes pequeños. No suele haber grandes cargas de tráfico.
- **Bajo coste**: los elementos que la conforman así como los protocolos están simplificados para evitar tener funciones innecesarias y así poder reducir el coste de desarrollo y de mantenimiento.

A nivel de protocolos, estas redes no implementan todas las capas del Internet convencional. En su lugar, **solo implementan tres capas: física, control de acceso al medio y aplicación**. Por tanto, puesto que no deja de ser una red de computadores, para

un supuesto usuario o un supuesto desarrollador, estas redes proporcionan al nodo una manera de comunicarse similar a la que proporciona una antena WiFi o una conexión Ethernet cableada a un ordenador personal, es decir, los nodos pueden ejecutar servicios o aplicaciones a través de estas redes. Por supuesto, un caso de aplicación que se ejecuta en estas redes es la de un algoritmo de sincronización temporal, como veremos más adelante. Aunque difiere según el protocolo concreto, la capa de aplicación de las LPWAN es homóloga a la capa de aplicación del modelo TCP/IP.

La topología de estas redes es de estrella, siendo cada vértice un nodo excepto el central que es un nodo especial llamado concentrador o "*gateway*" que se comunica con un servidor donde se almacenan los datos (ver Fig. 3). Por tanto, si queremos añadir un nodo nuevo simplemente basta con conectarlo al concentrador. Si uno falla el resto de la red sigue funcionando puesto que el concentrador sigue funcionando.

Esta arquitectura brinda nuevas oportunidades de cara a la eficiencia. Puesto que son inalámbricas y con forma de estrella, es bastante sencillo usar mensajes de difusión o "*beacons*" que son enviados por el concentrador a todos los nodos una sola vez, en vez de enviar la información a cada nodo uno por uno, aprovechando así tiempo y energía. También es posible el proceso inverso y es que un nodo difunda un mensaje a todos sus vecinos (no tiene por qué tener visibilidad con todos los nodos) de una sola vez y así evite hacer varias transmisiones, ahorrando de nuevo tiempo y energía.

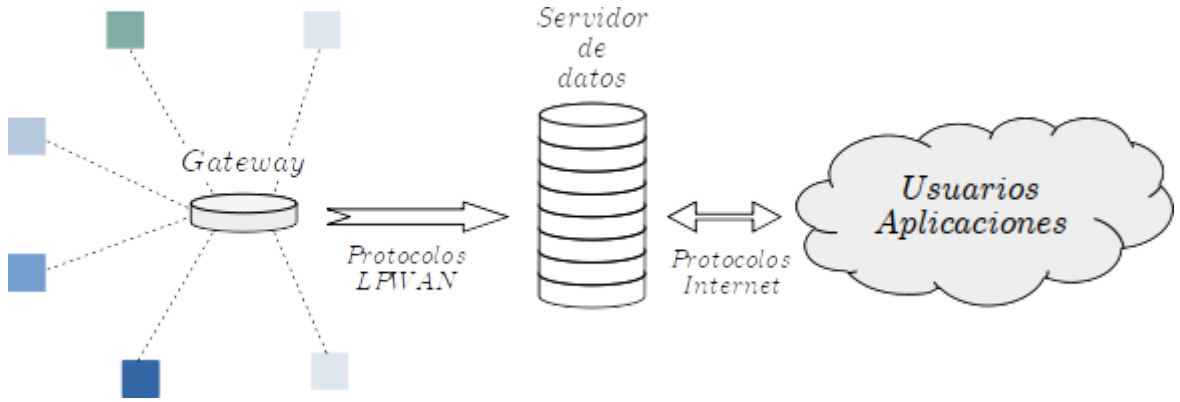


Figura 3. Esquema de una red LPWAN.

Actualmente hay distintos protocolos de redes LPWAN en activo, funcionando y con desarrollo, veamos algunas:

- **Sigfox**: esta red es de operador, osea que hay una operadora llamada Sigfox [5] que ha desplegado la infraestructura para poder usarla, de manera similar a una operadora móvil. Actualmente está desplegada en España y fue la primera de este tipo en desplegarse. Evidentemente es de pago, ofrece una tasa de transferencia máxima de 100 bytes por segundo y tiene gran parte del territorio con cobertura. Esta compañía proporciona todo el *hardware* y software de bajo nivel para su uso, es decir, proporciona el *back end*. Usa un espectro sin licencia y la compañía asegura un alcance de hasta 13 km desde sus antenas.

- **NB-IoT:** está dentro del estándar LTE [6], osea que es una red cercana a la red de telefonía habitual. En España operadoras como Orange y Vodafone ofrecen este servicio ya. Ofrece unas tasas de transferencia de 150 kilobytes por segundo

(KBps). Dado que es un protocolo tan cercano a la telefonía, el uso de su espectro electromagnético es bajo licencia.

- **LoRaWAN**: esta LPWAN no es ninguna operadora, simplemente un conjunto de estándares que conforman la pila de protocolos. Por tanto los protocolos son públicos [7], exceptuando los de la capa física. Actualmente existen en el mercado algunos modems baratos muy conocidos como la familia *Semtech* SX127x y SX126x. Además, existe una comunidad en crecimiento que despliega nodos y supernodos a modo de operadora colaborativa llamada "*MyThings*" donde se pueden conectar los nodos y volcar los datos. Como características, esta tecnología admite transferencias de hasta 10 Kbps y alcances de hasta 10 km en espacio abierto. Usa frecuencias libres de licencia.

Esta es una área que está en continuo desarrollo y es posible que en intervalos cortos de tiempo surjan nuevas plataformas y tecnologías así como el declive de las actuales.

3.5. Tecnologías para los nodos

Previamente se ha hablado de forma superficial sobre la arquitectura de los nodos de una LPWAN. En resumidas cuentas, cada nodo tiene cuatro componentes básicos:

- Unidad de procesamiento: la CPU de cada nodo, es su cerebro. Este componente puede ser una micro *Central Process Unit* (µCPU) o un *System on Chip* (SoC). Incluso puede contar con sensores, sin necesidad de que los mismos sean externos.
- Medio de comunicación: su función es acceder a la red y es controlado por la unidad de procesamiento.
- Sensor/es y/o actuador/es: se encargan de leer el estado del medio que le rodea en el caso de los sensores y de modificar el medio en caso de los actuadores. Son controlados por la unidad de procesamiento.
- Fuente de energía: como su propio nombre indica, se encarga de alimentar todos los componentes anteriores.

Veamos cada parte con más detalle, exponiendo caso concretos:

Unidad de procesamiento

Dada la naturaleza de las redes de sensores y LPWAN, la unidad de procesamiento de cada nodo debe cumplir lo siguiente:

- Ser barata.
- Consumir poca potencia (del orden de milivatios - mW).
- Ser pequeña.
- Poder interconectarse con otros sensores (para ello, debe contar con periféricos como *Serial Peripheral Interface* - SPI, *Inter-Integrated Circuit* - I2C, *Universal Asyncrhonus Receiver-Transmitter* - UART, *General Purpose Input-Output* - GPIO, etc.).

- Ser fácil de programar (incluso autoreprogramarse).
- Disponer de una memoria RAM del orden de kilobytes (KB) como mínimo para ejecutar programas sencillos.
- Contar con una memoria ROM del orden de decenas de KB para almacenar los programas y algunos datos.

Evidentemente en cada red las necesidades pueden agudizarse en algún aspecto concreto dependiendo de las exigencias de la tarea. Vamos a analizar qué dispositivos pueden cumplir estos requisitos (ver Tabla 8.1).

Tipo	CPU	SoC	MCU (*Micro Controller Unit*)
Descripción	Un procesador como tal, diseñado como unidad de procesamiento exclusivamente	Un procesador con varios periféricos integrados	Un µCPU con varios periféricos simples
Requisitos para funcionar	Una placa base con memoria RAM, almacenamiento, etc.	Una placa base con algunos componentes	Generalmente solo una fuente de alimentación
Potencia	Muy potente, varios núcleos, relojes de varios GHz, conjuntos de instrucciones muy amplios	Potentes, varios núcleos, gráfica integrada	Poco potentes, conjuntos de instrucciones muy básicos, frecuencias de MHz, sin capacidad gráfica
Consumo	Decenas de vatios (W)	Varios vatios	Desde milivatios hasta pocos vatios
Coste	Desde 10€ hasta 1000€	Desde varios € hasta ~100€	Desde céntimos de € hasta varios €
Periféricos	PCI, PCIe, USB, DDR	USB, RJ45, RMII, I2C, SPI, HDMI, VGA, eMMC, DDR	SPI, I2C, UART, GPIO
Memoria RAM	Externa, del orden de GigaBytes (GB)	Interna y/o externa, del orden de GB	Interna, desde algunos KB hasta cientos de KB
Memoria ROM	Externa, del orden de cientos de GB	Interna y/o externa, del orden de decenas GB	Interna, de cientos de KB hasta algunos MegaBytes (MB). Ampliable de forma externa
Complejidad de	El medio de	Depende del	Depende del

programación	almacenamiento de programas es externo, depende de la placa	modelo y la placa. Pueden hacer falta programadores *hardware*	fabricante, normalmente mediante los propios periféricos
Uso típico	Ordenadores	Móviles, tablets, routers, dispositivos embebidos en general	Electrodomésticos, automatismos
Páginas del datasheet	~10.000	~1.000	~100

Tabla 8.1. Comparativa tipos de Unidades de Procesamiento.

Según lo que buscamos, la mejor opción es la de los MCU. El hecho de ser baratos, sencillos de poner en marcha (puesto que solo necesitan alimentación), la simplicidad de los manuales, su coste y su consumo los hacen candidatos idóneos. Comparemos algunos MCUs del mercado (ver Tabla 8.2).

	STM32F10x	Atmega328	ESP8266	ESP32
Fabricante	STM	ATMEL	Espressif	Espressif
CPU	ARM Cortex-M3 72MHz, DMA	AVR 8-bit 20MHz	Xtensa L106 32-bits 80MHz	Xtensa L6 32-bits 160MHZ dual core
Memoria RAM	20KB	2KB	80KB	520KB
Memoria ROM	64/128KB	32KB + 2KB EEPROM	Hasta 4MB, externa	Hasta 16MB, externa
Conectividad	-	-	WiFi	WiFi, BLE
Voltaje	2.0-3.6V	1.8-5.5V	3.0-3.6V	2.2-3.6V
Consumo	~500mW	~60mW	~230mW	~230mW
Coste	Depende variante, ~3€	~2€	Depende de la placa, ~3€	Depende de la placa, ~4€
SPI	x2	x1	x2	x4
UART	x3	x1	x2	x3
GPIO	Hasta 80	x23	x32	x11
PWM	x7	x8	x8	x16
I2C	x2	x1	x1	x2
I2S	x2	-	x1	x2
Toolchains	GCC, Arduino	GCC, Arduino	Arduino IDE,	Arduino IDE,

	IDE	IDE	esp-open-sdk, Espressif SDK, MicroPython	Espressif SDK, MicroPython
Nº páginas (págs) del datasheet	~1200 págs	~150 págs	Documentación oficial pobre, mirar código	Documentación oficial pobre, mirar código

Tabla 8.2. Comparativa µCPUs.

Hablando en términos genéricos, todas las opciones parecen válidas. Quizás el ESP32 es demasiado potente para la mayoría de casos, pero puede haber otros casos donde sí sea útil. Puede ser que la opción más equilibrada sea el Atmega328, que es el MCU del archiconocido Arduino.

En el mercado existen muchísimos más MCU con características muy dispares, los aquí mostrados no son más que ejemplos muy famosos y que son representativos del panorama.

Medio de comunicación

Aunque ya hemos presentado algunas redes anteriormente (Sigfox, NB-IoT y LoRaWAN), en este apartado vamos a centrarnos solo en una, que va a ser LoRaWAN, ya que la documentación de sus módems es más accesible, usa frecuencias libres de licencia y es un estándar de red abierto.
Actualmente en el mercado hay dos familias de modems muy extendidos (hay algunos otros pero menos relevantes) que cuentan con un amplio desarrollo y librerías: SEMTECH SX172x y HOPERF RFM9x.
Ambas familias cuentan con una gama de módems (ver Fig. 4) con diversidad de potencia, encapsulados e interfaces.

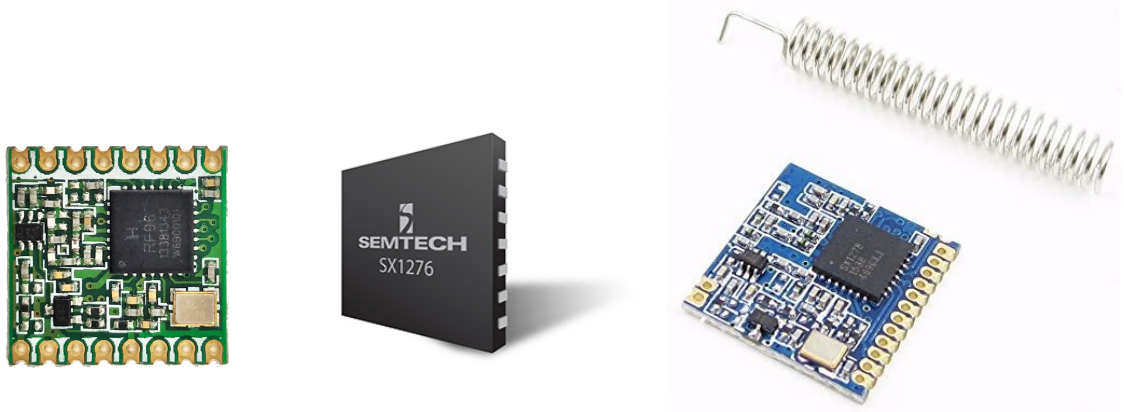


Figura 4. Imágenes de algunos módems.

Sensores y actuadores

Este es un componente que depende enormemente de la aplicación de la red. Hay infinidad de tipos de sensores y actuadores con características muy diferentes y con interfaces muy variadas (normalmente las interfaces suelen ser SPI, I2C o UART). Algunos tipos de sensores son:

- Temperatura (BMP180, BME680, GY-68)

- Presión (BMP180)
- Brújula (HMC5883L)
- GPS
- Humedad (DHT11)
- Luminosidad
- Inerciales (MPU6050 acelerómetro y giroscopio)
- Movimiento
- Infrarrojos (IR): TCRT5000L
- Ultrasonidos: HC-SR04

Por otro lado hay muchos tipos de actuadores:

- Leds
- Motores
- Servomotores
- Potenciómetros digitales
- Relés

Fuente de energía

Este componente está más en el campo de la electrónica que en el de la informatica, solo nombraremos algunas formas de alimentar un nodo:

- Red eléctrica: para el caso de que el nodo sea fijo y cuente con un suministro eléctrico, es una opción.
- Bateria recargable.
- Pila de larga duración: una pila de un solo uso que pueda dar una vida de años al nodo.
- Célula solar con o sin acumulador: una placa solar que alimenta al nodo y carga una batería para cuando no hay luz suficiente.

La opción más adecuada vuelve a depender mucho de la red en concreto y de sus circunstancias.

3.6. Protocolos de sincronización temporal

A continuación se exponen los protocolos de sincronización para redes de sensores encontrados [8].

Estos protocolos van a funcionar sobre la red que se despliegue como si fueran un servicio, es decir, se va a ejecutar sobre la capa de aplicación. Por hacer un símil, se ejecutan en lo más alto de la pila de protocolos de la LPWAN, al igual que NTP y PTP se ejecutan sobre la capa más alta de la pila TCP/IP.

El objetivo de estos protocolos es ofrecer una marca de tiempo fiable. Sin embargo, este objetivo es muy vago y puede matizarse de muchas maneras. Por otro lado, existe una amplia variedad entre estos protocolos puesto que se pueden diseñar y enfatizar sobre determinados aspectos de las LPWAN.

Primero enumeramos los protocolos encontrados para, en el siguiente apartado, clasificarlos:

- **NTP:** protocolo ampliamente conocido y desarrollado en el año 1985. Su especificación se puede encontrar en el documento RFC 5905. Utiliza servidores centrales y alcanza precisiones del orden de milisegundos.
- **PTP:** otro protocolo ampliamente conocido en el mundo de las aplicaciones industriales. Publicado en el 2002, su especificación más actual es la IEEE 1588-2019. Alcanza precisiones de nanosegundos pero requiere *hardware* dedicado.
- **TPSN (*Time-synchronization Protocol for Sensor Network*)**: es de los primeros protocolos diseñados para redes de sensores. Fue publicado en 2004.
- **FTSP (*Flooding Time Synchronization Protocol*):** este protocolo está considerado como el protocolo estándar para sincronización temporal para redes de sensores. Se basa en el mecanismo de inundación, que más adelante describiremos. Fue publicado en el 2004.
- **FCSA (*Flooding with Clock Speed Agreement*):** es una mejora de FTSP. Su mejora consiste en tener en cuenta la desviación de la frecuencia del reloj de cada nodo para obtener una precisión mayor. Fue publicado en 2014.
- **R-Sync (*Robust-Synchronization*):** este protocolo está enfocado a la tolerancia a fallos, de ahí el nombre. También se centra especialmente en el consumo energético.
- **TDP (*Time Diffusion Protocol*):** es el predecesor de los algoritmos distribuidos. Se centra tanto en la tolerancia a fallos como a cambios en la red.
- **DTSP (*Distributed Time Synchronization Protocol*):** fue el primer protocolo de sincronización temporal distribuida, lo que significa que en la red no hay una marca de tiempo absoluta sino que se calcula la marca de tiempo según la marca de tiempo de los vecinos. Publicado en 2006.
- **ATS (*Average Time Synchronization*):** fue el primer protocolo basado en el consenso de la marca temporal. Esto significa que los nodos tienen un mecanismo para decidir qué marca de tiempo es más válida para adoptarla. Publicado en 2007.

Realmente hay muchos más protocolos con matices diferentes, pero estos protocolos dan una idea bastante precisa de cómo está el estado del arte actualmente en este aspecto. También hay que decir que muchos protocolos no están completamente estandarizados (ya sea por que no se usan lo suficiente o por que son nuevos) y que lo más que se tiene es el artículo científico que describe el protocolo, no encontrando implementaciones del mismo. Incluso, a veces, hay aspectos concretos del protocolo que se pueden interpretar de maneras diferentes.

3.7. Características que definen a los protocolos de sincronización

Tras haber estudiado y analizado en detalle los protocolos anteriormente nombrados, vamos a proceder a clasificar estos protocolos según sus características y su comportamiento. Esta

clasificación se ha hecho en función a aspectos relevantes de los mismos y a cómo afectaría esto a una puesta en producción.

3.7.1 Tipo de sincronización

Parece obvio pensar que el único tipo de sincronización posible es respecto a un nodo con un reloj más preciso, o respecto a otro reloj que consideramos de referencia, pero la verdad es que esto no siempre es ni lo que se busca ni lo más conveniente.

Por tanto, podemos distinguir dos tipos de sincronización:

1. Sincronización centralizada
2. Sincronización distribuida

Sincronización centralizada

Este tipo de protocolos se centran en la idea de **minimizar la diferencia de tiempo entre un nodo de referencia** (nodo raíz o nodo maestro) **y el resto de nodos de la red**, con lo que estamos presuponiendo la existencia de dos categorías de nodos.

La diferencia entre estos dos tipos de nodos es que el nodo maestro tiene la marca de tiempo que "es considerada" como la de referencia, lo cual no implica que este nodo realmente tenga una marca de tiempo "mejor"[1]. Por tanto, surge la necesidad de establecer un algoritmo de selección del nodo maestro. Esta parte es muy influyente en el comportamiento y características del protocolo, veamos algunos casos:

- En el caso de que el algoritmo sea tan simple como que *"el nodo x es el maestro"*, tenemos la ventaja de que es muy sencillo y no añade complejidad, pero por contraprestación si el nodo x falla, la red deja de funcionar. Esta debilidad se puede mitigar, por ejemplo, añadiendo otros nodos maestros de respaldo.
- Otro caso es que el algoritmo decida el nodo maestro de forma dinámica mediante alguna heurística. Por ejemplo:
    - Cada un determinado tiempo, el nodo maestro va cambiando según un orden prefijado, es decir, se turnan.
    - Se elige como nodo maestro al nodo más comunicado de la red o al que más recursos energéticos posea.

Como ejemplos de estos protocolos tenemos: NTP, PTP, TSPN, FTSP, FCSA y R-Sync.

Sincronización distribuida

La diferencia con la familia anterior es que la marca de tiempo no se obtiene o se ajusta a una marca de tiempo fija o de referencia sino que se decide de algún modo según el resto de nodos de la red.

[1] Por mejor podemos referirnos a que tiene un reloj más preciso o con menos desviación. Por ejemplo un nodo maestro conectado a un módulo GPS que le da una marca de tiempo más precisa.

Por ejemplo, pensemos en un algoritmo de sincronización donde cada nodo calcula la marca de tiempo haciendo la media aritmética de la marca de tiempo de sus vecinos (el protocolo ATS funciona de esta forma).

Dentro de esta familia hay una gran diversidad, puesto que dependiendo del algoritmo para elegir la marca de tiempo local, resultará en comportamientos muy dispares. En consecuencia, el grado de "localidad" de la marca de tiempo se puede llevar a los extremos: si queremos una localidad "máxima" podemos centrarnos en mantener una sincronización "entre pares", donde toda la red está sincronizada entre pares y no de forma global. En dirección opuesta, podemos tener un grado de "localidad" mínimo, que llegue a toda la red.

Por comparar esta familia con la anterior, esta tiene como principal ventaja que es más resistente a fallos en los nodos puesto que no hay un único punto crítico que inutilice la red. Por contra, no va a ser posible obtener una marca de tiempo absoluta, por lo que dependiendo de la aplicación que queramos será una desventaja o no.

Los protocolos TDP, DTSP y ATS pertenecen a esta familia.

3.7.2 Modelo de reloj

Cuando se modela un reloj[2], entran en juego al menos dos variables: la velocidad del reloj y su desplazamiento. La primera se refiere a la frecuencia con que el reloj avanza y la otra al valor desde el que parte.

Dependiendo de nuestra aplicación, puede que necesitemos tener en cuenta las dos variables o solamente una de ellas.

- Si solo sincronizamos la frecuencia del reloj obtendremos una red cuyos nodos avanzan todos a la misma frecuencia pero en la que puede haber diferencias entre los relojes. Es decir, puede que dentro de esa red haya relojes que marquen horas diferentes, pero como están ajustadas sus frecuencias, el tiempo de esos relojes avanza exactamente al mismo tiempo. Un ejemplo donde solo esto es necesario es el caso de un cruce de semáforos: no nos interesa saber a qué hora un semáforo se pone en rojo y otro en verde, pero si nos interesa saber que van a hacer el cambio transcurrido el mismo tiempo.

- Por otro lado, si solo sincronizamos el desplazamiento, conseguimos los relojes sincronizados pero durante un periodo, puesto que a menos que la frecuencia real de todos los relojes de la red sea exactamente la misma, los relojes se desincronizarán. Este periodo será mayor o menor en función de la diferencia entre las frecuencias de los relojes; si la diferencia entre la frecuencia de los relojes es muy pequeña, el margen de tiempo hasta la desincronización puede ser suficiente para realizar otra corrección del desplazamiento, pero si por contra es muy grande, corregir sólo el desplazamiento no servirá de mucho.

Consecuentemente, si solo sincronizamos una de las dos variables, el protocolo será más sencillo y requerirá menos recursos.

[2] Este modelo es: $r(t) = d + ft$ donde d es el desplazamiento, f la frecuencia del reloj, t el tiempo para un observador externo y r el valor mismo del reloj.

3.7.3 Tipo de comunicación

En redes, hay al menos dos formas de comunicarse, una es la punto a punto y otra la multipunto:

- En las comunicaciones punto a punto los mensajes están dirigidos desde un punto de la red a otro punto de la red exclusivamente (por ejemplo el modelo cliente-servidor). Como se explicó anteriormente, en el caso de redes de sensores los paquetes enviados son escuchados por todos los nodos vecinos, por lo que una comunicación punto a punto no aprovecha la oportunidad de enviar información a otros nodos. No es que este modelo no sea útil o que presente problemas, sino que simplemente basar un protocolo únicamente en este modelo puede no ser la opción más eficiente. Como ejemplos de protocolo que siguen este modelo tenemos NTP y PTP.

- Para la redes multipunto surgen más posibilidades pues se aprovecha cada emisión para enviar información y por tanto la red es más eficiente. Casi todos los protocolos orientados a redes de sensores utilizan este modelo: normalmente hay un tipo de paquete genérico, sin destinatario concreto que contiene información acerca de la marca de tiempo o del nodo que envía el paquete. Como ejemplo de estos protocolos tenemos FTSP, FCSA y R-Sync.

3.7.4 Manejo de redes con nodos sin visibilidad directa entre ellos

Se conoce como red multisalto a una red en la que al menos un par de nodos necesitan de uno intermedio para comunicarse. Es decir, que la red no está directamente conectada entre sí misma. Esto es un problema si un protocolo solo contempla los nodos sobre los que tenga comunicación directa. Realmente esto no es una caracterización de los protocolos, puesto que una forma de abordar esta situación es una alternativa para cualquier protocolo que no contemple redes multisalto.

- **Clusterización de la red**: para implementar un protocolo que soporte redes inconexas, se puede segmentar la red en "*clusters*" monosalto (totalmente conectadas) y desplegar en cada clúster el protocolo.

- ***Spanning-Tree***: hay protocolos que sí contemplan las redes inconexas y se organizan modelando la red como un *spanning-tree*, donde se presupone la existencia de un nodo raíz.

Un *spanning-tree* es la representación de una red de sensores en forma de árbol, donde el nodo raíz es la raíz del árbol y los hijos de cada nodo son los nodos con los que tiene conexión directa, formándose **capas** en el árbol. En los protocolos diseñados alrededor de este concepto, el proceso de sincronización se realiza por capas: cada vez que el protocolo se ejecuta, la capa 0 (donde se sitúa el nodo maestro) sincroniza a la capa 1 (la capa inferior), la capa 1 a la 2 y así sucesivamente hasta llegar a la última capa. Es decir, la capa *n* sincroniza a la capa *n+1* partiendo desde la capa 0.

A este proceso se le conoce como inundación o *flooding* y tiene dos variantes:

- *Rapid flooding*: cada vez que un nodo de la capa n se sincroniza con la capa n-1, inmediatamente se sincroniza con la capa n+1. Puede parecer la opción más rápida

pero tiene graves inconvenientes que dependen de la topología de la red. Dado que el *spanning-tree* es una representación en forma de árbol, hay muchas conexiones entre nodos que no se representan, lo que puede provocar colisión en algunos nodos si la sincronización se realiza inmediatamente.

- *Slow flooding*: para evitar los problemas del *rapid flooding*, cada nodo de la capa n espera un periodo aleatorio antes de sincronizar a las capas inferiores (por ejemplo FTSP y FCSA).

3.7.5 Otros aspectos a tener en cuenta

Hay otras muchas características a tener en cuenta sobre los protocolos, pero que ya son clasificaciones más artificiales o más subjetivas.

Necesidades de cálculo y de memoria

Podemos establecer de forma objetiva que un protocolo necesita más o menos cálculo y memoria, pero esto es algo que depende en gran medida de la implementación.

Periodo de sincronización

Esta cuestión es común a todos los protocolos: ¿cuánto tiempo debe transcurrir entre dos sincronizaciones periódicas?

A priori parece obvio que cuanta más precisión requiera nuestra aplicación debería transcurrir un menor tiempo entre sincronizaciones. Sin embargo, esto supone un mayor uso de recursos, por tanto no responde bien a la pregunta.

Podemos continuar intentando reducir el consumo de recursos, y así poder bajar el periodo de sincronización sin gastar más recursos, pero este es precisamente el objetivo de estos protocolos.

La otra vía es aumentar el tiempo que tarda un nodo en “desincronizarse”. Volviendo a la modelación de un reloj, el único factor dependiente del tiempo es la frecuencia del reloj. Controlar esta variable es la que va a permitirnos aumentar el margen de tiempo antes de la desincronización.

Aun con protocolos que tengan en cuenta este factor, hay fluctuaciones en la variación de la frecuencia que son difíciles de manejar: la temperatura del nodo, el nivel de la batería, el desgaste de algún componente interno, etc. No hay nada que diga que no se puedan tener en cuenta estos factores y corregir la frecuencia del reloj según la temperatura ambiente, el nivel de la batería o el tiempo de funcionamiento del nodo, pero sin duda son factores muy específicos.

4. Implementación del demostrador

Una vez que tenemos una perspectiva sobre el estado actual de la tecnología y tras analizar las diferentes alternativas y posibilidades existentes para implementar el demostrador, podemos proceder a decidir qué tecnologías concretas usar, es decir, concretar el diseño del demostrador.

4.1. Diseño del demostrador

En primer lugar, conviene recordar que el objetivo final del proyecto consiste en implementar un demostrador de un sincronizador temporal para IoT.

Como se expuso en el estudio previo, estos protocolos están diseñados para desplegarse sobre una red de sensores. También, dentro de ese capítulo se explicó, que el escenario más típico dentro del IoT son las redes de sensores. Puesto que parece el diseño más común, se decide que el diseño del demostrador sea una red de sensores con el protocolo de sincronización temporal funcionando como servicio.

A continuación analizaremos más de cerca los diferentes aspectos del diseño, como el protocolo a usar, la tecnología de la red y las características de los nodos.

4.2. Implementación de la red

Partiendo del estudio previo, el tipo de red más usado son las LPWAN. Por tanto, será la arquitectura que usaremos para desplegar la red. De las tres LPWAN actualmente desplegadas en España, ambas cuentan con unas prestaciones técnicas similares pero a la hora de usarlas hay algunas diferencias:

- NB-IoT: hay que pedir licencia para usar el espectro electromagnético y darse de alta como cliente.
- Sigfox: su espectro no está licenciado, pero sí que hay que darse de alta en la red y comprar el *hardware* específico de la operadora.
- LoRaWAN: **su espectro no está licenciado y sus estándares son abiertos**. Los módems para esta red son accesibles y asequibles, además de contar con una comunidad abierta.

Puesto que los protocolos de sincronización se ejecutan en la capa de aplicación y no requieren características especiales de la red, las tres alternativas son completamente viables para implementar el protocolo de sincronización. Sin embargo, para este proyecto, conviene que la documentación de la red y la red en sí misma sean accesibles. Por esta razón *LoRaWAN* nos parece la mejor alternativa puesto que **las especificaciones de su pila de protocolos es pública** y para usar la red **no necesitamos pedir licencia ni darnos de alta en ninguna operadora**.

Para identificar a los nodos en la red LoRaWAN define un campo en la cabecera de la capa de aplicación para identificar los nodos de origen. A este campo, llamado *DevAddr*, le vamos a asignar siempre valor 0 cuando el mensaje provenga del nodo maestro y un número distinto de 0 cuando sea un nodo esclavo.

4.2.1 *Hardware* a usar

Para elegir el módem, ya vimos dos opciones válidas, ambas muy parecidas: la familia SX127x y RFM9x. En este caso vamos a elegir la opción del modem SEMTECH SX1276 [9] por razones de facilidad de compra, pero la otra opción también sería perfectamente válida. Para detallar este modem, su radio opera a una frecuencia de 433 MHz con la modulación de LoRa, que no es pública. La capa física de la LoRa está completamente implementada por *hardware*, lo que implica que se realizan de forma automática una comprobación de integridad, como especifica LoRa.

La comunicación con este modem se realiza mediante una interfaz SPI y también se pueden configurar hasta ocho líneas de interrupciones para diferentes eventos del modem, como señales de “recepción completa” (*RxDone*) o “transmisión completa” (*TxDone*).

4.2.2 Estructura de datos

En este apartado vamos a describir con detalle los tipos de datos que vamos a tener. Estos datos irán localizados en la unidad de datos de la capa de aplicación.

Marca de tiempo

Al menos de forma interna mediremos el tiempo en **milisegundos** relativos al momento en que se inició el nodo maestro. Esta medida puede ser corregida a una marca de tiempo absoluta por el *gateway* (por ejemplo a UTC gracias a un reloj GPS). No tiene sentido hablar de microsegundos puesto que se espera conseguir unas precisiones del orden de milisegundos, por tanto mayor precisión sería despreciada.

El tipo de dato a elegir para representar este valor será un entero sin signo de 32 bits.

Estructura de datos

Para dar soporte a la diferentes características que necesitamos en la red de sensores, vamos a realizar una multiplexación de los datos usando un campo que nos indique el tipo de paquetes según la siguiente estructura (ver Tabla 9).

Byte 1	*Byte 2*	...	*Byte n*
Tipo de paquete	Datos (dependiendo del tipo de paquete)		

Tabla 9. Estructura de los mensajes.

Los tipos de paquetes son los siguientes (ver Tabla 10).

Num	*Tipo*	*Descripción*
1	FCSA Beacon	Estos tipos de paquetes son para el protocolo de

2	FCSA Join	sincronización. Más adelante hay más detalles
3	Datos	Sirve para enviar información

Tabla 10. Enumeración de los tipos de paquetes.

La necesidad del tipo de paquete FCSA Join surge de ser capaz de informar al resto de la red de que un nuevo nodo se acaba de introducir en la red, lo que puede ser muy relevante durante el tiempo de ejecución del algoritmo.

Un ejemplo muy claro de esta necesidad es cuando un nodo se desconecta de la red y posteriormente se vuelve a conectar. Para el resto de nodos, es como si nada hubiera pasado, y van a procesar la información nueva proveniente del nuevo nodo, lo que potencialmente puede generar alteraciones en los cálculos.

Paquete de datos

La cabecera de este paquete contiene una marca de tiempo (que se supone que tiene que ser la marca de tiempo del evento o de los datos), la longitud de los datos y los datos en sí. (ver Tabla 11).

Byte[0,1]	*Byte[2,3,4,5]*	*Byte 6*	*...*	*Byte[6 + longitud]*
longitud: int16 sin signo (número de bytes)	Marca de tiempo: int32 sin signo	Datos[0]	Datos[...]	Datos[longitud]

Tabla 11. Estructura de los mensajes de datos con marca de tiempo.

4.3. Implementación de los nodos

Para implementar los nodos, vamos a usar un MCU. Entre las opciones barajadas (ESP32, ESP8266, STM32 y Atmega328) todas son viables, pero nos decantamos por el **ESP8266** por las siguiente razones:

- Es asequible y se puede encontrar en muchos distribuidores
- Cuenta con una amplia documentación técnica
- Múltiples *toolchains* soportan este MCU
- Tiene unas prestaciones representativas de la oferta actual del mercado
- Existen una gran cantidad de proyectos abiertos basados en este MCU.

Además, vamos a usar la placa **Wemos D1 Mini** (ver Figura 6.1) porque además cuenta con mucha oferta de módulos de expansión (*shields)*.

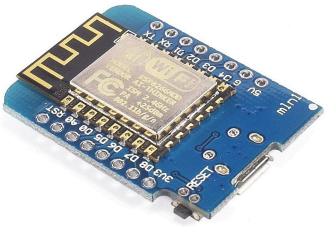


Figura 5. WeMos D1 Mini.

Para recoger eventos o información es posible usar cualquier sensor que sea compatible, en este caso, como prueba del concepto, se emplea la interrupción por *hardware* en flanco de subida de uno de los pines del microcontrolador, que provocará el envío de un paquete tipo DATA a la red.

Tanto para la depuración de los nodos como para obtener información de los paquetes DATA que llegan al *gateway*, se usa el puerto serie del microcontrolador para obtener dicha información.

En el apartado de *software*, el desarrollo se realiza mediante la herramienta **Arduino IDE,** configurada para el MCU ESP8266. El código [10] ha sido desarrollado en C++, usando las librerías de Arduino y el SX1276 [11].

Este sería el esquema de las conexiones para cada nodo (ver Figs. 6.1 y 6.2).

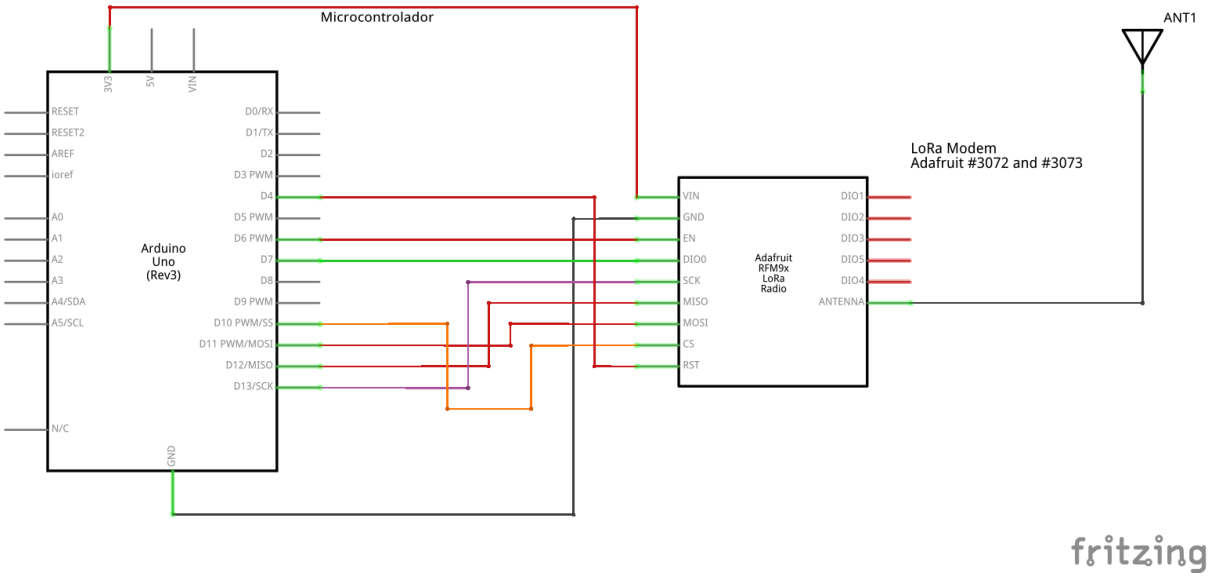


Figura 6.1. Esquema de conexiones del demostrador.

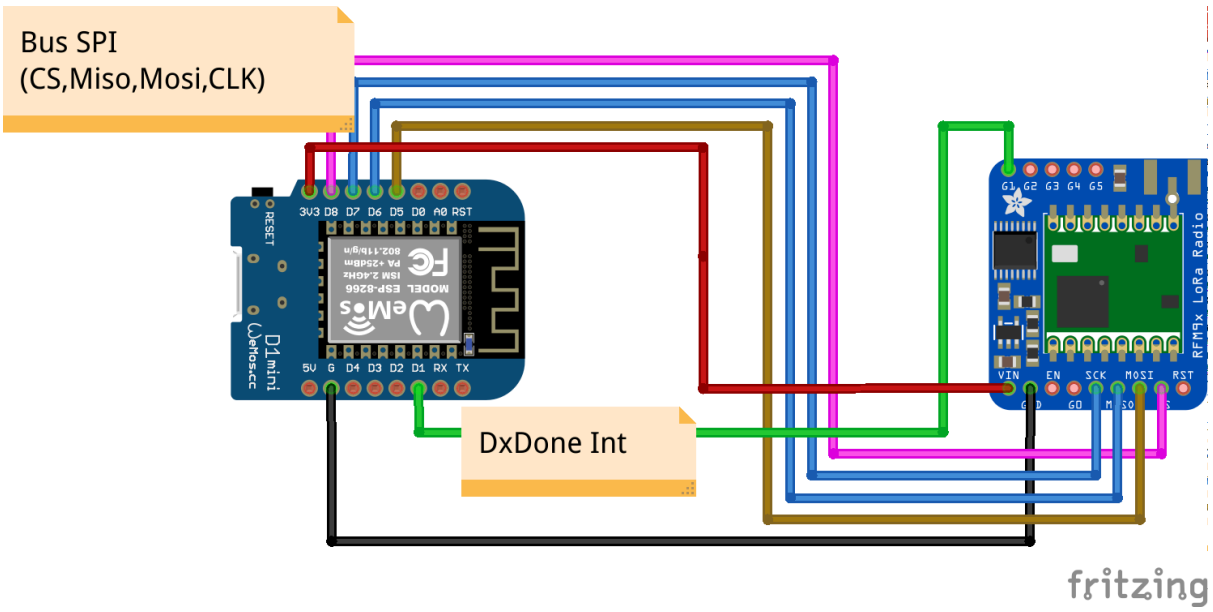


Figura 6.2. Representación gráfica de las conexiones del nodo.

El cable verde con la etiqueta *RxDone* se usa como interrupción externa para procesar la llegada de un paquete, previa configuración del modem.

4.4. Implementación del protocolo de comunicación

Ahora que ya sabemos como vamos a implementar la red del demostrador en sí misma, es momento de empezar a pensar en las aplicaciones o servicios que se van a ejecutar sobre la misma, osea hay que elegir un protocolo de sincronización.

De los protocolos vistos en el estudio previo, vemos que hay mucha variedad y que dependiendo de la labor a realizar por nuestra red nos convendría usar uno u otro.

Si por ejemplo necesitaremos hacer tareas de forma síncrona, por ejemplo, controlar varios semáforos en un cruce, lo crucial sería que los semáforos cambien su estado al mismo tiempo, y no que lo hagan a una hora concreta, por tanto ahi seria mas conveniente un protocolo de sincronización distribuida, y el desplazamiento del reloj no seria muy util ajustarlo. En esta situación GTSP o ATS serían buenos candidatos.

Por el contrario, si estamos monitorizando a que hora llega un autobús a cada parada para ofrecer mejores estimaciones, tener una marca de tiempo respecto a una hora de referencia es absolutamente crucial, así como valorar los cambios en la topología de la red. Protocolos como FCSA o FTSP serían los idóneos ahora.

En un hipotético caso en el que estuviéramos monitorizando la temperatura o humedad de un bosque extenso, con mucha vegetación frondosa donde sea difícil acceder y moverse, pensar en un protocolo que cuide el consumo energético es interesante, pues puede ahorrarnos tener que hacer mantenimientos en los nodos, que son difíciles de acceder.

Partiendo de que nuestro objetivo es desarrollar un demostrador de este tipo de protocolos, creemos que la mejor opción es elegir un protocolo típico o al menos ampliamente extendido. Según nuestro estudio previo, el protocolo **FTSP** está considerado como el protocolo *de facto* en este paradigma, por ello, es un buen candidato. Sin embargo, este

protocolo tiene un descendiente directo, relativamente actual que es **FCSA.** Puesto que este nuevo protocolo mejora al anterior y entra dentro de las posibilidades del proyecto en términos de tiempo, vamos a elegir este último para implementar el demostrador.

4.4.1 FCSA

Es un protocolo propuesto en 2014 por *Kasim Sinan Yildirim* y *Aylin Kantarci* [12]. Se caracteriza por:

- Hay un nodo central con la marca de tiempo de referencia.
- Admite redes multisalto, se apoya en un *spanning-tree*.
- *Slow flooding.*
- Tiene en cuenta tanto el desplazamiento como la frecuencia del reloj (controlar esta segunda variable es su principal diferencia respecto a FTSP).
- Modelo de comunicación de difusión.

Visto el apartado anterior podemos hacernos una idea bastante aproximada de cómo funciona el protocolo. Aun así podemos resumir que el funcionamiento de FCSA se basa en la emisión periódica por parte de cada nodo de un paquete con información referente a su reloj (relojes, como se verá a continuación) con el objetivo de proporcionar a sus vecinos la información necesaria para ajustar la frecuencia de sus relojes respecto a la de sus vecinos.

La especificación del protocolo en sí no concreta cómo hacer la elección del nodo maestro, solo se limita a presuponer que hay un nodo *x* que es el principal. Tampoco define los tiempos entre dos inundaciones consecutivas, pero como ya se vio, no es algo que depende solo del protocolo.

Sus autores sostienen con los resultados de las simulaciones, que es al menos tan eficaz y eficiente que los protocolos FTSP, PulseSync (muy parecidos con FCSA) y GTSP. Sin embargo, una desventaja que presenta FCSA es el tiempo de inicialización en redes grandes (>1000 nodos), donde puede tardar hasta 1 hora y media en completar la sincronización.

4.4.2 Reloj lógico

Para tener en cuenta la diferencia de frecuencia, FCSA crea de forma virtual un reloj que va a funcionar a una frecuencia diferente al reloj del procesador (lo llamaremos reloj *hardware*, H a partir de ahora). Este nuevo reloj se llama “reloj lógico” (L). Ambos relojes están relacionados de la siguiente manera:

$$\frac{dL}{dt} = l\frac{dH}{dt}$$

La variable l se conoce como la “ratio” del reloj lógico. Cuando hagamos referencia al valor H_u o L_u nos referiremos al valor del reloj *hardware* y del reloj lógico en el nodo u respectivamente.

4.4.3 Variables internas de cada nodo para el protocolo

Cada nodo u va a poseer las siguientes variables internas:

- seq_u. Será la secuencia más alta recibida por el nodo: *int16 sin signo*
- $base_u$ y $ultimaAct_u$. Sirven para corregir el desplazamiento del reloj lógico, se verán más detalles a continuación: *int32 sin signo*
- l_u. La ratio del reloj lógico: *double 32bits*
- N_u. Lista de los nodos vecinos. *El tamaño de cada elemento de la lista se detalla a continuación, la lista tiene un tamaño de 30 entradas.*
- Por cada nodo v vecino:
    - $h_v^u = h_u/h_v$. El cociente entre entre las frecuencias de ambos nodos: *double 32bits*
    - l_v. La ratio del nodo vecino: *double 32bits*
    - $R_v^u = [(H_{u_1}, H_{v_1}), ..., (H_{u_n}, H_{v_n})]$. Una lista de tamaño N: *cada tupla son dos int32 sin signo. Aunque en el artículo científico especifique una lista, dadas las limitaciones de memoria, será una FIFO con espacio para 30 tuplas.*

A continuación se explica el rol de cada variable.

4.4.4 Funcionamiento detallado

Para ver el funcionamiento, primero vamos a ver cómo se alcanza la sincronización de las frecuencias y luego vamos a ver cómo se ajusta el desplazamiento mediante las inundaciones.

Para ambas tareas, tal y como se dijo en la introducción, la información va a circular mediante un único tipo de paquete: es el momento de introducir el paquete "**FCSA Beacon**".

Mensaje FCSA Beacon

Este paquete es la piedra angular del protocolo, es la pieza que articula toda la comunicación entre los nodos. Contiene la siguiente información:

$$< L_u, H_u, l_u, seq_u >$$

Este mensaje será enviado de forma periódica, pudiendo ser el periodo variable. El artículo del protocolo no define ese periodo, en este caso se ha elegido un periodo aleatorio tal y como describe esta expresión:

$$Periodo(segundos) = 15 + Aleatorio(0, 7)$$

La razón de esta aleatoriedad es evitar colisiones a la hora de enviar los beacon entre los diferentes nodos.

El número de secuencia sirve para articular las rondas de inundación de la siguiente manera: cuando un nodo recibe un mensaje con un número de secuencia mayor a los que ha recibido anteriormente, este realiza todos los cálculos que hay que realizar y lo retransmite. En el caso de que sea un número menor o igual que el mayor que ha recibido, el nodo no hace nada. Este número de secuencia sólo puede ser incrementado por el nodo maestro.

Con este número, conseguimos el siguiente comportamiento: el nodo maestro quiere iniciar una nueva inundación, para ello incrementa su número de secuencia, que será el que se transmitirá en el próximo FCSA beacon. El resto de nodos, al recibir un número de secuencia superior, realizan los cálculos con los datos recibidos. Cuando a cada nodo le toque volver a enviar un beacon, estos enviarán el nuevo número de secuencia, propagando así la información de la nueva inundación.

A continuación se muestra la estructura del mensaje (ver Tabla 11).

Byte[0+4n]	*Byte[1+4n]*	*Byte[2+4n]*	*Byte[3+4n]*
Seq : *int16 sin signo*		Logical Clock: *int32 sin signo*	
continuación		Hardware Clock: *int32 sin signo*	
continuación		Lrate: *double*	
continuación			

Tabla 11. Estructura del mensaje FCSA Beacon

Ajuste de la frecuencia

La única frecuencia que se ajusta en este paso (y en todo el protocolo) es la del reloj lógico, es decir, vamos a calcular la variable $l_u(t^+)$, que representa el nuevo valor de l_u para el nodo u tras recibir un beacon en el instante t. Esto se va a calcular mediante la media aritmética de las frecuencias de los relojes lógicos de N_u respecto a u, tal y como sigue:

$$l_u(t^+) = \frac{l_u(t) + \sum_{v \in N_u} h_v^u(t) l_v(t)}{|N_u| + 1}$$

El momento de hacer este ajuste es cuando se recibe un beacon. Siendo u el nodo que recibe el paquete y v el nodo de origen, así como H_u el valor del H en el instante en el que se recibe el beacon, el primer paso consistirá en almacenar el par (H_u, H_v) en la lista R_v^u y l_u.

El siguiente paso consiste en calcular h_v^u, que es la operación más costosa computacionalmente de todo el protocolo. Este valor es la **pendiente de la recta de mínimos cuadrados de** R_v^u.

Ajuste del desplazamiento

El desplazamiento será ajustado en cada nueva ronda. Para ello, es necesario fijarse en el valor de seq cada vez que nos llega un beacon. Si el valor de seq_v es menor o igual que

seq_u (siendo u el nodo de destino y v el nodo de origen del beacon), no tendremos que hacer ningún ajuste en el desplazamiento, puesto que es un paquete de una inundación previa.

En caso contrario procederemos a actualizar nuestra variable interna referente a la secuencia seq_u , $base_u$ y $ultimaAct_u$:

1. Si $seq_v > seq_u$
2. $\quad seq_u \leftarrow seq_v$
3. $\quad base_u \leftarrow L_u$
4. $\quad ultimaAct_u \leftarrow H_u$

Cálculo de la marca de tiempo

El cálculo de la marca de tiempo del protocolo es el valor del reloj lógico, es decir, L_u. Se calcula con la siguiente expresión:

$$L_u = base_u + (H_u - ultimaAct_u)l_u$$

Esta expresión es la que nos va a dar el tiempo sincronizado.

4.5. Resumen

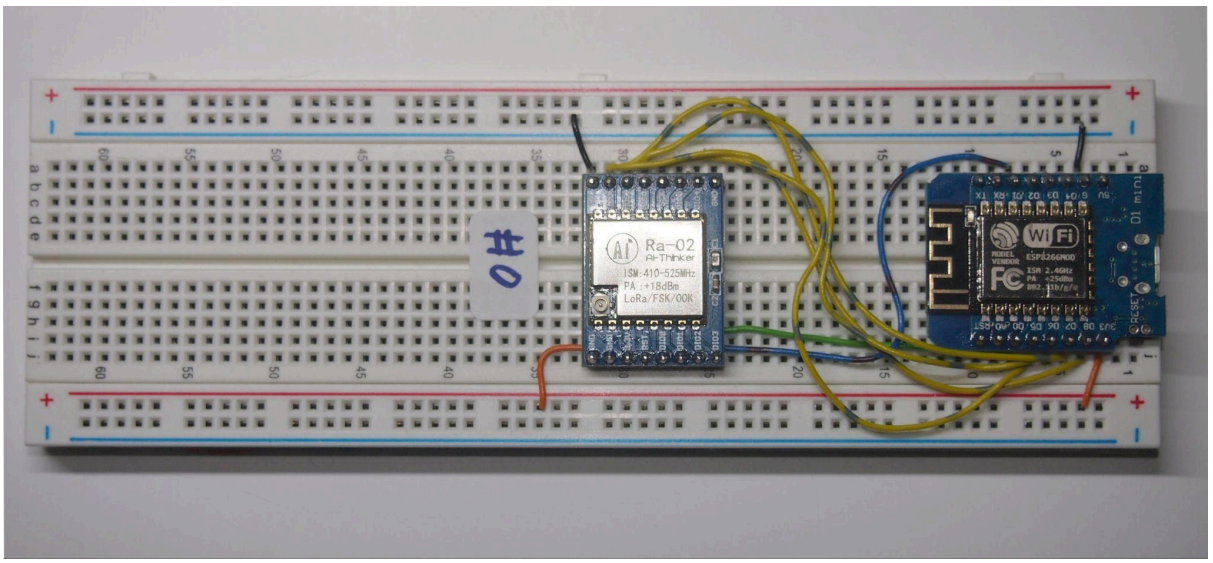
Para concluir este apartado, vamos a resumir algunos detalles de la implementación para tener una perspectiva más clara:

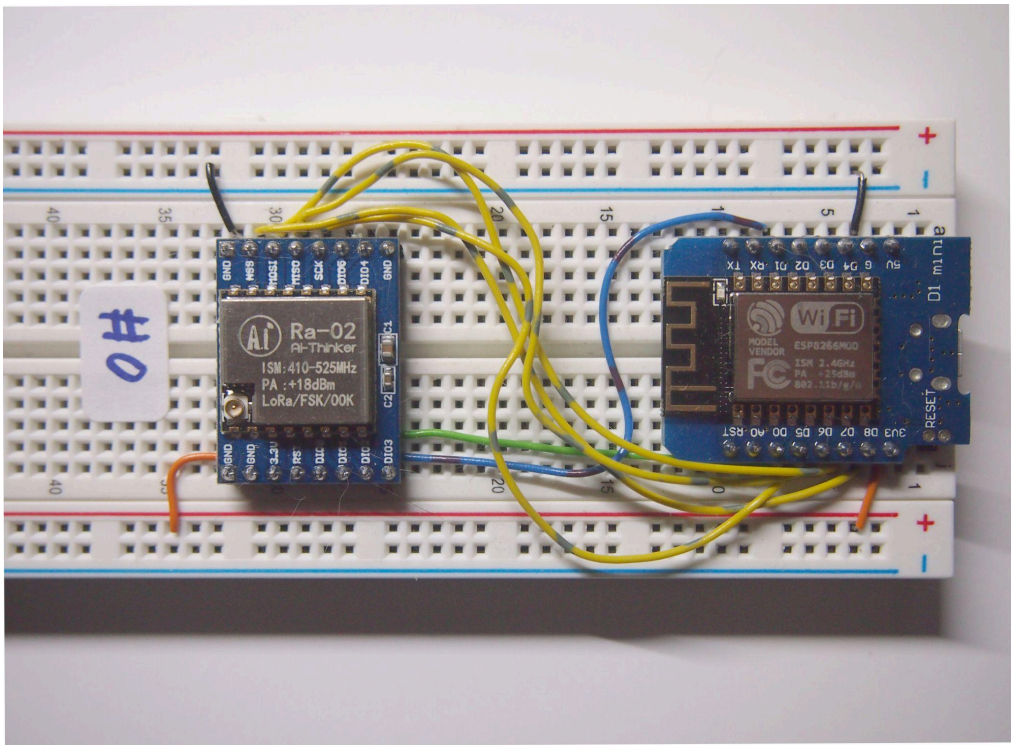
- **Implementación de la red:** la arquitectura usada es una LPWAN, en concreto usando la pila de protocolos de LoRaWAN. Dentro de LoRaWAN, vamos a asignar la dirección 0 siempre al nodo maestro o *gateway* y el resto de direcciones a los nodos esclavos. Para implementar el servicio de sincronización, los mensajes se multiplexan en la capa de aplicación gracias a un campo que determina el tiempo de paquete, tenemos los siguientes paquetes: FCSA Beacon, FCSA Join y un paquete de datos con su propia marca de tiempo. La marca de tiempo para la red se mide en milisegundos y empieza a contar cuando el *gateway* se enciende. Como *hardware*, se usa el modem SEMTECH SX1276.

- **Implementación de los nodos**: para implementar los nodos, se usa un MCU, en concreto el modelo ESP8266 en la placa WeMos D1 Mini. La conexión entre el nodo y el módem se realiza por SPI y el código ha sido programado en C++ con el entorno Arduino IDE, usando librerías de dominio público para gestionar el modem.

- **Implementación del protocolo:** el protocolo escogido es FCSA, implementado según los detalles de su artículo científico.

A continuación se añaden algunas fotografías del prototipo de los nodos.





Fotografías 1 y 2. Prototipo de nodo

En estas fotografías vemos los componentes de cada nodo, montados sobre una placa de prototipado. Veamos las partes:

- **WeMos D1 Mini:** situado a la derecha, se conecta mediante Micro USB al ordenador.
- **SX1276:** situado a la izquierda.
- **Cables:**
    - Naranja y negro: +3V y GND
    - 4 Amarillos: Bus SPI
    - Verde: señal de reset para SX1276
    - Azul: DIO0

5. Pruebas

Una vez terminado el demostrador, es el momento de hacer las pruebas para medir su rendimiento y otros factores más.

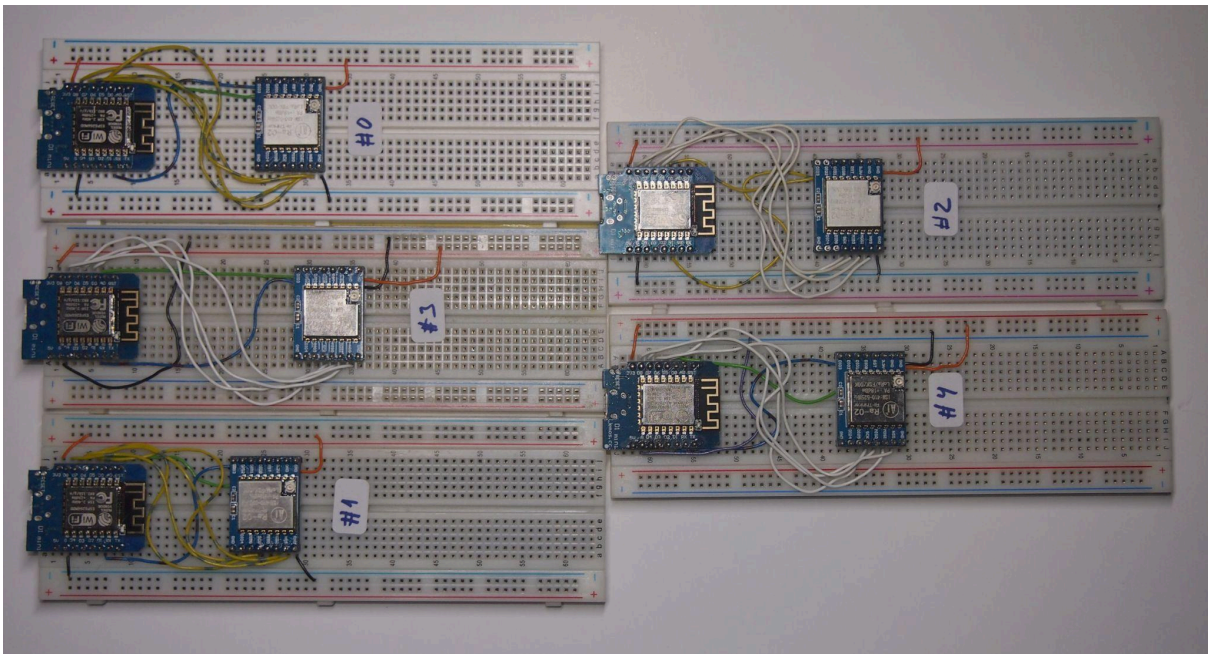
5.1. Introducción

Las pruebas a realizar sobre el demostrador serán las siguientes:

- Prueba funcional: esta prueba está orientada a comprobar que el demostrador funciona correctamente en un entorno en el que no hay perturbaciones ni alteraciones.
- Prueba de tolerancia a fallos: en esta prueba comprobaremos como el demostrador sigue funcionando a pesar de perder algunos nodos no-maestros y como la red se comporta cuando falla el nodo maestro.
- Prueba de precisión: probaremos cuánta precisión obtenemos con este algoritmo de sincronización temporal.

5.1.1 Entorno de pruebas

El entorno de pruebas se compone de 5 nodos, que conforman el demostrador, todos conectados a un ordenador mediante cables USB, desde el cual se va a inspeccionar la salida del puerto serie (el puerto serie que se menciona en el punto 4.3) de cada uno. La información relevante de las capturas será subrayada.



Fotografías 3. Entorno de pruebas

5.2. Pruebas

Procedamos a explicar en detalle cada prueba, su objetivo y sus resultados. La visualización de los resultados se realizará analizando la salida del puerto serie de cada nodo.

5.2.1 Prueba funcional

El objetivo de esta prueba consiste en comprobar que todo funciona correctamente, es decir, como en un entorno idóneo, sin perturbaciones ni fallos, los nodos que conforman el demostrador se sincronizan entre ellos correctamente. En esta prueba no vamos a tratar de medir la precisión de la sincronización.

El procedimiento se dividirá en las siguientes fases:

- Fase 1: Los nodos se encienden por separado y vemos que no se produce ninguna sincronización puesto que no hay una red como tal.
- Fase 2: Empezando por el nodo maestro, se van a ir conectando nodos hasta llegar a un total de 5 nodos conectados simultáneamente.

El resultado esperado de esta prueba es que desde el momento que haya más de un nodo conectado, estos se sincronizan entre sí.

Fase 1

Captura del nodo 0 (nodo maestro) solo en la red

```
*
** Sincronizador temporal para IoT basado en hardware **
** Francisco Sanchez - TFG - 2020/21 - DTE **
Inicializando modulo Lora OK!
Registrando callback OK
Activando escucha OK
FCSA INIT...OK
Local addr: 0x00 (L.1.1)
Enviando Join OK (L.1.2)
Inicializacion completa!
< Enviando FCSA beacon > (L.1.3)
< Timestamp: 0 minutos 2 segundos 63 milisegundos > (L.1.4)
< Timestamp: 0 minutos 7 segundos 64 milisegundos >
< Timestamp: 0 minutos 12 segundos 65 milisegundos >
< Timestamp: 0 minutos 17 segundos 66 milisegundos >
< Enviando FCSA beacon >
```

Esta captura corresponde al momento en el que el nodo se inicializa. Podemos ver información relevante como la dirección local (ver Línea 1.1), que es 0 pues la captura es del nodo maestro, el momento en el que se envía el mensaje FCSA Join (ver Línea 1.2), el envío de un FCSA Beacon (ver Línea 1.3) y la marca de tiempo del nodo (ver Línea 1.4).

La marca de tiempo se imprime cada 5 segundos. La primera vez que se imprime la marca de tiempo siempre tiene el mismo valor, y corresponde al tiempo de inicialización de cada nodo.

Captura del nodo #1 (nodo esclavo) solo en la red

```
*
** Sincronizador temporal para IoT basado en hardware **
** Francisco Sanchez - TFG - 2020/21 - DTE **
Inicializando modulo Lora OK!
Registrando callback OK
Activando escucha OK
FCSA INIT...OK
Local addr: 0x01 (L.2.1)
Enviando Join OK
Inicializacion completa!
< Enviando FCSA beacon >
< Timestamp: 0 minutos 2 segundos 63 milisegundos >
< Timestamp: 0 minutos 7 segundos 64 milisegundos >
< Timestamp: 0 minutos 12 segundos 65 milisegundos >
< Timestamp: 0 minutos 17 segundos 66 milisegundos >
< Enviando FCSA beacon >
```

La inicialización del resto de nodos es igual, variando únicamente la dirección local (ver L.2.1).

Fase 2

Ahora vamos a observar la captura del nodo maestro a medida que se van añadiendo nodos a la red.

Captura del nodo maestro cuando se enciende otro nodo y recibe un FCSA Beacon

```
< Timestamp: 1 minutos 27 segundos 18 milisegundos >

@@@ Paquete recibido @@@ (L.3.1)
< 0x01 0x01 0x05 0x00 0x00 0x00 0xad 0x54 0x01 0x00 0xcb 0x5a 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@ (L.3.2)
### FCSA BEACON ### (L.3.3)
Recv Seq: 5
Recv Logical: 87213
Recv Hardware: 23243
Recv lrate: 0.000000
Inicializando lista nodo 0x01… (L.3.4)
Local Seq: 5
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000000
######
```

```
< Timestamp: 1 minutos 32 segundos 19 milisegundos >
```

En esta captura se observa cómo se recibe un paquete de información, en concreto un FCSA Beacon. Cuando recibe algo, se muestra la información en hexadecimal (ver entre L.3.1 y L.3.2) y se imprime información acerca el tipo de paquete (ver L.3.3)

Vemos todos los datos sobre el estado de FCSA, tanto los recibidos (ver entre L.3.3 y L.3.4) como los locales (a partir de L.4). En este caso, ya que es el primer paquete que se recibe, no se puede calcular la pendiente de los mínimos cuadrados puesto que solo hay un punto. En ese caso se imprime que se está inicializando la lista para el nodo 1 (ver L.4).

Captura del nodo #0 ya inicializada la información del nodo #1.

```
< Timestamp: 1 minutos 42 segundos 21 milisegundos >

@@@ Paquete recibido @@@
< 0x01 0x01 0x05 0x00 0x00 0x00 0x86 0x97 0x01 0x00 0xa4 0x9d 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 104326
Recv Hardware: 40356
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x01… (L.4.1)
Hu/Hv: 1.000000 (L.4.2)
nodo 1 hrate 1.000000 lrate 1.000000 (L.4.3)
Local Seq: 5
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000000
######
< Enviando FCSA beacon >
< Timestamp: 1 minutos 47 segundos 22 milisegundos >
```

En este nuevo mensaje FCSA Beacon, ya vemos como efectivamente se informa de que se está calculando los mínimos cuadrados (ver L.4.1) y se imprime su valor (L.4.2). También se imprime información referente al estado interno sobre cada nodo (L.4.3).

Nodo #1 recién incorporado a la red

```
*
** Sincronizador temporal para IoT basado en hardware **
** Francisco Sanchez - TFG - 2020/21 - DTE **
Inicializando modulo Lora OK!
Registrando callback OK
Activando escucha OK
FCSA INIT...OK
Local addr: 0x01
Enviando Join OK
```

```
Inicializacion completa!
< Enviando FCSA beacon >
< Timestamp: 0 minutos 2 segundos 63 milisegundos >
< Timestamp: 0 minutos 7 segundos 64 milisegundos >
< Timestamp: 0 minutos 12 segundos 65 milisegundos >
< Timestamp: 0 minutos 17 segundos 66 milisegundos > (L.5.1)

@@@ Paquete recibido @@@
< 0x00 0x01 0x05 0x00 0x00 0x00 0x2b 0x46 0x01 0x00 0x2b 0x46 0x01
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 83499
Recv Hardware: 83499
Recv lrate: 1.000000
Inicializando lista nodo 0x00...
< Nueva ronda 0 -> 5 > (L.5.2)
Local Seq: 5
Local Base: 83499
Local LastUpdate: 19529
Local Lrate: 1.000000
######
< Timestamp: 1 minutos 26 segundos 37 milisegundos > (L.5.3)
```

En esta captura se puede observar como un nodo esclavo se incorpora a la red. Un cambio interesante es el que se produce al imprimir las marcas de tiempo. Como se mencionó antes, las marcas de tiempo se imprimen cada 5 segundos, pero se ve que el incremento de tiempo tras recibir un mensaje FCSA Beacon no corresponde con 5 segundos (ver L.5.1 y L.5.3). También se puede apreciar el salto de la ronda 0 (la inicial) a la ronda actual de la red (ver L.5.2).

Nodo #1 realizado cálculos

```
< Timestamp: 5 minutos 41 segundos 127 milisegundos >

@@@ Paquete recibido @@@
< 0x00 0x01 0x13 0x00 0x00 0x00 0xc0 0x44 0x05 0x00 0xc4 0x44 0x05
0x00 0x00 0x00 0x00 0x00 0x04 0x5d 0xdd 0x0e 0xe8 0xff 0xef 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 19
Recv Logical: 345280
Recv Hardware: 345284
Recv lrate: 0.999989
Calculado minimos cuadrados nodo 0x00...
Hu/Hv: 0.999995 (L.6.1)
nodo 0 hrate 0.999995 lrate 0.999989 (L.6.2)
< Nueva ronda 18 -> 19 > (L.6.3)
```

```
Local Seq: 19
Local Base: 345280
Local LastUpdate: 281321
Local Lrate: 0.999984
######
< Timestamp: 5 minutos 46 segundos 128 milisegundos >
```

Nodo maestro realizando cálculos

```
< Timestamp: 5 minutos 52 segundos 66 milisegundos >

@@@ Paquete recibido @@@
< 0x01 0x01 0x13 0x00 0x00 0x00 0x18 0x64 0x05 0x00 0x42 0x6a 0x04
0x00 0x00 0x00 0x00 0x00 0x9b 0x26 0xed 0xc6 0xde 0xff 0xef 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 19
Recv Logical: 353304
Recv Hardware: 289346
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x01…
Hu/Hv: 1.000003 (L.6.4)
nodo 1 hrate 1.000003 lrate 0.999984 (L.6.5)
Local Seq: 19
Local Base: 0
Local LastUpdate: 0
Local Lrate: 0.999988
######
< Timestamp: 5 minutos 57 segundos 67 milisegundos >
```

En estas dos capturas de los nodos #0 y #1 se observa como ambos nodos tienen información acerca del otro (ver L.6.2 y L.6.5).

Es interesante observar el valor de la pendiente en ambos casos (ver L.6.1 y L.6.4). Dado que no son relojes ideales y que están sujetos a variaciones en su frecuencia, observamos que la pendiente es distinta de 1.0, valor que solo se daría si ambos relojes funcionasen a la misma frecuencia exacta. Podemos ver que los valores son coherentes y ver como uno marca un valor por encima de 1 (ver L.6.4) y otro por debajo de 1 (ver L.6.1), es decir, que hay un nodo #0 observa que el reloj del nodo #1 va un poco más lento que el suyo y el nodo #1 observa que el reloj del nodo #0 va más rápido que el suyo, lo cual es un resultado coherente.

También se aprecia con claridad el paso a una ronda nueva (L.6.3).

Nodo #2 en la red, recibiendo beacons de los nodos #0 y #1

```
*
** Sincronizador temporal para IoT basado en hardware **
** Francisco Sanchez - TFG - 2020/21 - DTE **
Inicializando modulo Lora OK!
```

```
Registrando callback OK
Activando escucha OK
FCSA INIT...OK
Local addr: 0x02
Enviando Join OK
Inicializacion completa!
< Enviando FCSA beacon >
< Timestamp: 0 minutos 2 segundos 63 milisegundos > (L.7.1)

@@@ Paquete recibido @@@
< 0x00 0x01 0x31 0x00 0x00 0x00 0x21 0xc7 0x0d 0x00 0x2a 0xc7 0x0d
0x00 0x00 0x00 0x00 0x00 0x38 0xef 0xe8 0x3a 0xed 0xff 0xef 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 49
Recv Logical: 902945
Recv Hardware: 902954
Recv lrate: 0.999991
Inicializando lista nodo 0x00...
< Nueva ronda 0 -> 49 > (L.7.2)
Local Seq: 49
Local Base: 902945
Local LastUpdate: 3061
Local Lrate: 1.000000
######
< Timestamp: 15 minutos 6 segundos 948 milisegundos > (ver L.7.3)

@@@ Paquete recibido @@@
< 0x01 0x01 0x31 0x00 0x00 0x00 0xef 0xde 0x0d 0x00 0x1f 0xe5 0x0c
0x00 0x00 0x00 0x00 0x00 0xfa 0x8b 0x74 0x60 0xe8 0xff 0xef 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 49
Recv Logical: 909039
Recv Hardware: 845087
Recv lrate: 0.999989
Inicializando lista nodo 0x01...
Local Seq: 49
Local Base: 902945
Local LastUpdate: 3061
Local Lrate: 1.000000
######
< Timestamp: 15 minutos 11 segundos 949 milisegundos >
```

Vemos que el nodo 2 pasa inmediatamente de la ronda 0 a las 49 (ver L.7.2) y su marca de tiempo se ajusta instantáneamente (ver L.7.1 y L.7.2).

Captura del nodo #0

```
< Timestamp: 17 minutos 11 segundos 947 milisegundos >

@@@ Paquete recibido @@@
< 0x01 0x01 0x38 0x00 0x00 0x00 0x3f 0xd0 0x0f 0x00 0x80 0xd6 0x0e
0x00 0x00 0x00 0x00 0x00 0xd8 0xac 0x50 0xb0 0xc8 0xff 0xef 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 56
Recv Logical: 1036351
Recv Hardware: 972416
Recv lrate: 0.999974
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 0.999984
nodo 0 hrate 0.999980 lrate 0.999979 (L.8.1)
nodo 1 hrate 0.999984 lrate 0.999974 (L.8.2)
Local Seq: 56
Local Base: 1029450
Local LastUpdate: 129591
Local Lrate: 0.999959
######
< Timestamp: 17 minutos 16 segundos 948 milisegundos > (L.8.3)
```

Captura del nodo #1

```
< Timestamp: 17 minutos 41 segundos 245 milisegundos >

@@@ Paquete recibido @@@
< 0x02 0x01 0x39 0x00 0x00 0x00 0x2c 0x3f 0x10 0x00 0x1d 0x84 0x02
0x00 0x00 0x00 0x00 0x00 0xb5 0x79 0x41 0x8a 0xa7 0xff 0xef 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 57
Recv Logical: 1064748
Recv Hardware: 164893
Recv lrate: 0.999958
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 1.000013
nodo 0 hrate 0.999996 lrate 0.999976 (L.8.4)
nodo 2 hrate 1.000013 lrate 0.999958 (L.8.5)
Local Seq: 57
Local Base: 1049900
Local LastUpdate: 985967
Local Lrate: 0.999972
######
< Timestamp: 17 minutos 46 segundos 246 milisegundos > (L.8.6)
```

Captura del nodo #2

```
< Timestamp: 17 minutos 31 segundos 948 milisegundos >
```

```
@@@ Paquete recibido @@@
< 0x01 0x01 0x39 0x00 0x00 0x00 0xf1 0x13 0x10 0x00 0x35 0x1a 0x0f
0x00 0x00 0x00 0x00 0x00 0x85 0x56 0x86 0x90 0xc6 0xff 0xef 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 57
Recv Logical: 1053681
Recv Hardware: 989749
Recv lrate: 0.999973
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 0.999985
nodo 0 hrate 0.999981 lrate 0.999976 (L.8.7)
nodo 1 hrate 0.999985 lrate 0.999973 (L.8.8)
Local Seq: 57
Local Base: 1049900
Local LastUpdate: 150044
Local Lrate: 0.999958
######
< Timestamp: 17 minutos 36 segundos 949 milisegundos > (L.8.9)
```

Con estas tres capturas se puede observar que la marca de tiempo de la red es la misma, exceptuando las diferencias a la hora de obtener las capturas (ver L.8.3, L.8.6 y L.8.9).

Ahora también cada nodo informa de la información que tiene de los otros nodos. Como de momento tenemos tres nodos en la red, cada nodo va a tener información sobre los otros dos nodos que no son él mismo. Por ejemplo, en el nodo #0 deberiamos ver información sobre el nodo #1 y nodo #2 exclusivamente (ver L.8.1 y L.8.2).

Sigamos añadiendo nodos a la red.

Captura del nodo #0

```
< Timestamp: 1 minutos 17 segundos 76 milisegundos >

@@@ Paquete recibido @@@
< 0x04 0x01 0x05 0x00 0x00 0x00 0xd0 0x2f 0x01 0x00 0xae 0xa7 0x1c
0x00 0x00 0x00 0x00 0x00 0xf2 0x73 0x94 0xed 0x80 0xff 0xef 0x3f >
@@@ sAddr: 04 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 77776
Recv Hardware: 1877934
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x04...
Hu/Hv: 1.000029
nodo 1 hrate 1.000022 lrate 0.999971
nodo 2 hrate 1.000017 lrate 0.999955
nodo 3 hrate 1.000053 lrate 0.999923
```

```
nodo 4 hrate 1.000029 lrate 0.999939
Local Seq: 5
Local Base: 0
Local LastUpdate: 0
Local Lrate: 0.999977
######
```

Captura del nodo #1

```
@@@ Paquete recibido @@@
< 0x04 0x01 0x05 0x00 0x00 0x00 0xd0 0x2f 0x01 0x00 0xae 0xa7 0x1c
0x00 0x00 0x00 0x00 0x00 0xf2 0x73 0x94 0xed 0x80 0xff 0xef 0x3f >
@@@ sAddr: 04 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 77776
Recv Hardware: 1877934
Recv lrate: 0.999939
Calculado minimos cuadrados nodo 0x04...
Hu/Hv: 1.000029
nodo 0 hrate 0.999990 lrate 0.999976
nodo 2 hrate 1.000017 lrate 0.999955
nodo 3 hrate 1.000053 lrate 0.999923
nodo 4 hrate 1.000029 lrate 0.999939
Local Seq: 5
Local Base: 74803
Local LastUpdate: 3324051
Local Lrate: 0.999970
######
< Timestamp: 1 minutos 18 segundos 788 milisegundos >
```

Captura del nodo #2

```
@@@ Paquete recibido @@@
< 0x04 0x01 0x05 0x00 0x00 0x00 0xd0 0x2f 0x01 0x00 0xae 0xa7 0x1c
0x00 0x00 0x00 0x00 0x00 0xf2 0x73 0x94 0xed 0x80 0xff 0xef 0x3f >
@@@ sAddr: 04 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 77776
Recv Hardware: 1877934
Recv lrate: 0.999939
Calculado minimos cuadrados nodo 0x04...
Hu/Hv: 1.000011
nodo 0 hrate 0.999983 lrate 0.999976
nodo 1 hrate 0.999984 lrate 0.999971
nodo 3 hrate 1.000036 lrate 0.999923
nodo 4 hrate 1.000011 lrate 0.999939
Local Seq: 5
```

```
Local Base: 74803
Local LastUpdate: 2488157
Local Lrate: 0.999956
######
< Timestamp: 1 minutos 19 segundos 355 milisegundos >
```

Captura del nodo #3

```
< Timestamp: 1 minutos 17 segundos 59 milisegundos >

@@@ Paquete recibido @@@
< 0x04 0x01 0x05 0x00 0x00 0x00 0xd0 0x2f 0x01 0x00 0xae 0xa7 0x1c
0x00 0x00 0x00 0x00 0x00 0xf2 0x73 0x94 0xed 0x80 0xff 0xef 0x3f >
@@@ sAddr: 04 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 77776
Recv Hardware: 1877934
Recv lrate: 0.999939
Calculado minimos cuadrados nodo 0x04...
Hu/Hv: 0.999982
nodo 0 hrate 0.999945 lrate 0.999976
nodo 1 hrate 0.999946 lrate 0.999971
nodo 2 hrate 0.999972 lrate 0.999955
nodo 4 hrate 0.999982 lrate 0.999939
Local Seq: 5
Local Base: 74803
Local LastUpdate: 1665232
Local Lrate: 0.999922
######
```

Captura del nodo #4

```
@@@ Paquete recibido @@@
< 0x02 0x01 0x05 0x00 0x00 0x00 0x94 0x25 0x01 0x00 0xbf 0xf8 0x25
0x00 0x00 0x00 0x00 0x00 0x1a 0x41 0x3d 0x58 0xa1 0xff 0xef 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 5
Recv Logical: 75156
Recv Hardware: 2488511
Recv lrate: 0.999955
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 0.999983
nodo 0 hrate 0.999961 lrate 0.999976
nodo 1 hrate 0.999968 lrate 0.999971
nodo 2 hrate 0.999983 lrate 0.999955
nodo 3 hrate 1.000021 lrate 0.999923
Local Seq: 5
```

```
Local Base: 74803
Local LastUpdate: 1874960
Local Lrate: 0.999939
######
< Timestamp: 1 minutos 17 segundos 378 milisegundos >
```

Si nos fijamos en cada nodo, podemos ver que cada nodo contiene información del resto de nodos.

5.2.2 Prueba de tolerancia a fallos

En esta prueba, se va a comprobar la capacidad del demostrador de admitir fallos en algunos de sus nodos. Es decir, como la red puede seguir funcionando en el caso de que un nodo falle, provocando su salida de la red, y como luego vuelve a sincronizarse una vez que el nodo se reincorpora.

Evidentemente, esta intermitencia en la disponibilidad de los nodos va a tener diferentes consecuencias dependiendo de si se trata de un nodo maestro o de un nodo esclavo.

Las fases son:

- Fase 1: Partimos de una red funcionando y sincronizada.
- Fase 2: Retiramos un nodo (no el maestro) de la red y comprobamos el estado de la misma
- Fase 3: Reincorporamos el nodo desconectado
- Fase 4: Repetimos el proceso con el nodo maestro

El resultado esperado es que cuando se retire un nodo que no sea el maestro, la red siga funcionando y en el momento en que se vuelva a conectar, se sincronice de nuevo.

La red, en su versión actual, no cuenta con mecanismo alguno para detectar que un nodo se ha salido de la red, por tanto si un nodo se desconecta, el resto de nodos simplemente dejarán de recibir nuevos mensajes FCSA Beacon de ese nodo y seguirán haciendo los cálculos con los datos existentes del nodo que se desconecta. Es decir, los nodos no informarán de la baja de otro nodo.

Cuando se desconecte el nodo maestro, la red seguirá funcionando como se acaba de explicar, pero la red funcionará de forma degradada hasta finalmente degradarse, pues no hay ningún nodo que continúe con las inundaciones.

No ocurre lo mismo cuando un nodo se incorpora a la red, momento en el cual si se envía un mensaje FCSA Join informando de la entrada de un nuevo nodo en la red.

En el caso de que se reconecte un nodo esclavo, el resto de nodos simplemente eliminará los datos previos de este nodo, mientras que si se reincorpora el nodo maestro, se limpiarán todos los datos previos, incluida la marca de tiempo de la red.

Fase 1

Captura del nodo #0

```
@@@ Paquete recibido @@@
< 0x01 0x01 0x04 0x00 0x00 0x00 0xad 0xed 0x00 0x00 0x44 0xea 0x00
0x00 0x00 0x00 0x00 0x00 0x30 0x90 0xce 0xfd 0x06 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 4
Recv Logical: 60845
Recv Hardware: 59972
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 1.000000
nodo 1 hrate 1.000000 lrate 1.000007
nodo 2 hrate 1.000016 lrate 0.999995
Local Seq: 4
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000010
######
< Timestamp: 1 minutos 2 segundos 75 milisegundos > (L.1.1)
```

Captura del nodo #1

```
@@@ Paquete recibido @@@
< 0x02 0x01 0x04 0x00 0x00 0x00 0x06 0xe5 0x00 0x00 0x73 0xdc 0x00
0x00 0x00 0x00 0x00 0x00 0xf7 0x86 0x45 0x29 0xf5 0xff 0xef 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 4
Recv Logical: 58630
Recv Hardware: 56435
Recv lrate: 0.999995
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 1.000000
nodo 0 hrate 1.000000 lrate 1.000017
nodo 2 hrate 1.000000 lrate 0.999995
Local Seq: 4
Local Base: 57350
Local LastUpdate: 56477
Local Lrate: 1.000007
######
< Enviando FCSA beacon >
< Timestamp: 1 minutos 2 segundos 948 milisegundos > (L.1.2)
```

Captura del nodo #2

```
@@@ Paquete recibido @@@
```

```
< 0x01 0x01 0x04 0x00 0x00 0x00 0xad 0xed 0x00 0x00 0x44 0xea 0x00
0x00 0x00 0x00 0x00 0x00 0x30 0x90 0xce 0xfd 0x06 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 4
Recv Logical: 60845
Recv Hardware: 59972
Recv lrate: 1.000007
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 1.000000
nodo 0 hrate 0.999973 lrate 1.000017
nodo 1 hrate 1.000000 lrate 1.000007
Local Seq: 4
Local Base: 57350
Local LastUpdate: 55154
Local Lrate: 0.999997
######
< Timestamp: 1 minutos 4 segundos 208 milisegundos > (L.1.3)
```

Comparando las distintas marcas de tiempo de la red (ver L.1.1, L.1.2 y L.1.3), así como la información de cada nodo respecto a los demás, comprobamos que la red está sincronizada.

Fase 2

Partiendo de una red de 3 nodos sincronizados, vamos a retirar uno de la red, en concreto el #2 y ver que sucede.

Captura del nodo #0 cuando se desconecta el nodo #2

```
< Timestamp: 2 minutos 52 segundos 98 milisegundos >
< Timestamp: 2 minutos 57 segundos 99 milisegundos >
< Timestamp: 3 minutos 2 segundos 100 milisegundos >
Momento en el que se retira el nodo #2
< Timestamp: 3 minutos 7 segundos 101 milisegundos >

@@@ Paquete recibido @@@
< 0x01 0x01 0x0a 0x00 0x00 0x00 0x46 0xdb 0x02 0x00 0xdd 0xd7 0x02
0x00 0x00 0x00 0x00 0x00 0x4d 0xac 0x21 0x18 0x08 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 10
Recv Logical: 187206
Recv Hardware: 186333
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 1.000000
nodo 1 hrate 1.000000 lrate 1.000008
nodo 2 hrate 1.000016 lrate 0.999992
```

```
Local Seq: 10
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000008
######
< Enviando FCSA beacon >
< Timestamp: 3 minutos 12 segundos 102 milisegundos >
< Timestamp: 3 minutos 17 segundos 103 milisegundos >
< Timestamp: 3 minutos 22 segundos 104 milisegundos > (L.2.1)

@@@ Paquete recibido @@@
< 0x01 0x01 0x0b 0x00 0x00 0x00 0x14 0x1c 0x03 0x00 0xab 0x18 0x03
0x00 0x00 0x00 0x00 0x00 0x81 0xb6 0xcb 0x5d 0x07 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 11
Recv Logical: 203796
Recv Hardware: 202923
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 1.000000
nodo 1 hrate 1.000000 lrate 1.000007
nodo 2 hrate 1.000016 lrate 0.999992
Local Seq: 11
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000008
######
< Enviando FCSA beacon >
```

Captura del nodo #1 cuando se desconecta el nodo #2

```
< Timestamp: 2 minutos 47 segundos 969 milisegundos >

@@@ Paquete recibido @@@
< 0x02 0x01 0x0a 0x00 0x00 0x00 0x35 0x98 0x02 0x00 0xa3 0x8f 0x02
0x00 0x00 0x00 0x00 0x00 0x43 0xf1 0xdf 0x7f 0xef 0xff 0xef 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 10
Recv Logical: 170037
Recv Hardware: 167843
Recv lrate: 0.999992
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 1.000013
nodo 0 hrate 1.000000 lrate 1.000010
nodo 2 hrate 1.000013 lrate 0.999992
Local Seq: 10
Local Base: 167701
```

```
Local LastUpdate: 166828
Local Lrate: 1.000008
######
< Timestamp: 2 minutos 52 segundos 970 milisegundos >
< Timestamp: 2 minutos 57 segundos 971 milisegundos >
< Timestamp: 3 minutos 2 segundos 972 milisegundos >
Momento en el que se retira el nodo #2
< Enviando FCSA beacon >
< Timestamp: 3 minutos 7 segundos 973 milisegundos > (L.2.2)

@@@ Paquete recibido @@@
< 0x00 0x01 0x0b 0x00 0x00 0x00 0xb9 0xe2 0x02 0x00 0xb8 0xe2 0x02
0x00 0x00 0x00 0x00 0x00 0x9f 0x8c 0xca 0x8f 0x08 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 11
Recv Logical: 189113
Recv Hardware: 189112
Recv lrate: 1.000008
Calculado minimos cuadrados nodo 0x00...
Hu/Hv: 1.000000
nodo 0 hrate 1.000000 lrate 1.000008
nodo 2 hrate 1.000013 lrate 0.999992
< Nueva ronda 10 -> 11 > (L.2.3)
Local Seq: 11
Local Base: 189113
Local LastUpdate: 188240
Local Lrate: 1.000007
######
```

A primera vista vemos que no hay percepción por parte del resto de nodos de la desconexión del nodo #2. Las marcas de tiempo siguen avanzando en los nodos restantes (ver L.2.1 y L.2.2) e incluso, después de la desconexión se siguen produciendo nuevas inundaciones y cambios de ronda (ver L.2.3). También los nodos siguen haciendo los cálculos con los datos ya existentes del nodo #2.

Fase 3

En esta fase vamos a reincorporar el nodo #2 a la red.

Captura del nodo #0 cuando se incorpora de nuevo el nodo #2

```
< Timestamp: 7 minutos 43 segundos 29 milisegundos >
< Timestamp: 7 minutos 48 segundos 30 milisegundos >
< Timestamp: 7 minutos 53 segundos 31 milisegundos > (L.3.1)
< Enviando FCSA beacon >

@@@ Paquete recibido @@@
< 0x02 0x02 >
```

```
@@@ sAddr: 02 | len: 2 @@@
### FCSA JOIN ### (L.3.2)
Nodo: 2 (L.3.3)
Reiniciando nodo...
< Timestamp: 7 minutos 58 segundos 32 milisegundos > (L.3.4)
< Timestamp: 8 minutos 3 segundos 33 milisegundos >
< Timestamp: 8 minutos 8 segundos 34 milisegundos >
```

Captura del nodo #1 cuando se incorpora de nuevo el nodo #2

```
< Timestamp: 7 minutos 42 segundos 158 milisegundos >
< Enviando FCSA beacon >
< Timestamp: 7 minutos 47 segundos 159 milisegundos >
< Timestamp: 7 minutos 52 segundos 160 milisegundos > (L.3.5)

@@@ Paquete recibido @@@
< 0x01 0x01 0x1a 0x00 0x00 0x00 0x31 0x38 0x07 0x00 0xc7 0x34 0x07
0x00 0x00 0x00 0x00 0x00 0xce 0x2c 0x24 0x0c 0x05 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 26
Recv Logical: 473137
Recv Hardware: 472263
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 1.000004
nodo 0 hrate 1.000004 lrate 1.000005
nodo 2 hrate 1.000016 lrate 0.999992
Local Seq: 26
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000008
######

@@@ Paquete recibido @@@
< 0x02 0x02 >
@@@ sAddr: 02 | len: 2 @@@
### FCSA JOIN ### (L.3.6)
Nodo: 2 (L.3.7)
Reiniciando nodo...
< Timestamp: 7 minutos 57 segundos 162 milisegundos > (L.3.8)
< Enviando FCSA beacon >
< Timestamp: 8 minutos 2 segundos 164 milisegundos >
< Timestamp: 8 minutos 7 segundos 165 milisegundos >
```

Captura del nodo #2

```
*
** Sincronizador temporal para IoT basado en hardware **
```

```
** Francisco Sanchez - TFG - 2020/21 - DTE **
Inicializando modulo Lora OK!
Registrando callback OK
Activando escucha OK
FCSA INIT...OK
Local addr: 0x02
Enviando Join OK
Inicializacion completa!
< Enviando FCSA beacon >
< Timestamp: 0 minutos 2 segundos 63 milisegundos >
< Timestamp: 0 minutos 7 segundos 64 milisegundos >

@@@ Paquete recibido @@@
< 0x01 0x01 0x1a 0x00 0x00 0x00 0x16 0x73 0x07 0x00 0xac 0x6f 0x07
0x00 0x00 0x00 0x00 0x00 0xce 0x2c 0x24 0x0c 0x05 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 26
Recv Logical: 488214
Recv Hardware: 487340
Recv lrate: 1.000005
Inicializando lista nodo 0x01… (L.3.9)
< Nueva ronda 0 -> 26 > (L.3.10)
Local Seq: 26
Local Base: 488214
Local LastUpdate: 11346
Local Lrate: 1.000000
######
< Timestamp: 8 minutos 8 segundos 933 milisegundos > (L.3.11)
< Timestamp: 8 minutos 13 segundos 934 milisegundos >
```

Por un lado, vemos como la marca de la red se ha mantenido y que el nodo #2 se ha sincronizado con la red (ver L.3.4, L.3.8 y L.3.11). También se aprecia como la marca de tiempo sigue avanzando en los nodos #0 (ver L.3.1 y L.3.4) y #1 (ver L.3.5 y L.3.8) como de normal.

Por otro lado, en ambos nodos vemos cómo se detecta el mensaje FCSA Join del nodo #2 (ver L.3.2,L.3.3 y L.3.6,L.3.7).

Por último, el nodo #2 se incorpora con éxito a la red, pues ha obtenido la ronda actual (L.3.10), está inicializando los datos de los otros nodos (L.3.9) y ha cogido la marca de tiempo de la red (L.3.11).

Fase 4

Ahora vamos a hacer lo mismo con el nodo maestro.

Captura del nodo #1

```
< Timestamp: 5 minutos 27 segundos 123 milisegundos >
```

```
@@@ Paquete recibido @@@
< 0x00 0x01 0x17 0x00 0x00 0x00 0x74 0xfe 0x04 0x00 0x58 0x23 0x06
0x00 0x00 0x00 0x00 0x00 0x69 0x3e 0x3f 0xc5 0x0d 0x09 0xea 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 23
Recv Logical: 327284
Recv Hardware: 402264
Recv lrate: 0.813605
Calculado minimos cuadrados nodo 0x00...
Hu/Hv: 0.999996
nodo 0 hrate 0.999996 lrate 0.813605
nodo 2 hrate 1.000012 lrate 0.813584
< Nueva ronda 22 -> 23 >
Local Seq: 23
Local Base: 327284
Local LastUpdate: 402354
Local Lrate: 0.813604
######
Momento en el que desaparece el nodo maestro
< Enviando FCSA beacon >
< Timestamp: 5 minutos 31 segundos 181 milisegundos >
< Timestamp: 5 minutos 35 segundos 250 milisegundos >

@@@ Paquete recibido @@@
< 0x02 0x01 0x17 0x00 0x00 0x00 0x1c 0x22 0x05 0x00 0x8b 0x4f 0x06
0x00 0x00 0x00 0x00 0x00 0xdd 0x00 0xf6 0x43 0xeb 0x08 0xea 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ### (L.4.1)
Recv Seq: 23
Recv Logical: 336412
Recv Hardware: 413579
Recv lrate: 0.813589
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 1.000012
nodo 0 hrate 0.999996 lrate 0.813605
nodo 2 hrate 1.000012 lrate 0.813589
Local Seq: 23
Local Base: 327284
Local LastUpdate: 402354
Local Lrate: 0.813602
######
< Timestamp: 5 minutos 39 segundos 318 milisegundos >
< Timestamp: 5 minutos 43 segundos 387 milisegundos >
< Enviando FCSA beacon >
< Timestamp: 5 minutos 47 segundos 456 milisegundos >
< Timestamp: 5 minutos 51 segundos 525 milisegundos >
```

Captura del nodo #2

```
< Timestamp: 5 minutos 27 segundos 68 milisegundos >

@@@ Paquete recibido @@@
< 0x00 0x01 0x17 0x00 0x00 0x00 0x74 0xfe 0x04 0x00 0x58 0x23 0x06
0x00 0x00 0x00 0x00 0x00 0x69 0x3e 0x3f 0xc5 0x0d 0x09 0xea 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 23
Recv Logical: 327284
Recv Hardware: 402264
Recv lrate: 0.813605
Calculado minimos cuadrados nodo 0x00...
Hu/Hv: 0.999983
nodo 0 hrate 0.999983 lrate 0.813605
nodo 1 hrate 0.999986 lrate 0.813582
< Nueva ronda 22 -> 23 >
Local Seq: 23
Local Base: 327284
Local LastUpdate: 402359
Local Lrate: 0.813582
######
Momento en el que desaparece el nodo maestro
@@@ Paquete recibido @@@
< 0x01 0x01 0x17 0x00 0x00 0x00 0x1f 0x07 0x05 0x00 0x5a 0x2e 0x06
0x00 0x00 0x00 0x00 0x00 0x68 0xdb 0xff 0x2b 0x0c 0x09 0xea 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ### (L.4.2)
Recv Seq: 23
Recv Logical: 329503
Recv Hardware: 405082
Recv lrate: 0.813604
Calculado minimos cuadrados nodo 0x01...
Hu/Hv: 0.999986
nodo 0 hrate 0.999983 lrate 0.813605
nodo 1 hrate 0.999986 lrate 0.813604
Local Seq: 23
Local Base: 327284
Local LastUpdate: 402359
Local Lrate: 0.813589
######
< Timestamp: 5 minutos 31 segundos 126 milisegundos >
< Timestamp: 5 minutos 35 segundos 195 milisegundos >
< Enviando FCSA beacon >
< Timestamp: 5 minutos 39 segundos 264 milisegundos >
< Timestamp: 5 minutos 43 segundos 332 milisegundos >
```

Las marcas de tiempo no se descontrolan por ahora y los nodos siguen intercambiando mensajes (ver L.4.1 y L.4.2) pero no se reciben mensajes del nodo #0 ni nuevas rondas.

El siguiente paso es volver a conectar el nodo maestro.

Captura del nodo #0

```
*
** Sincronizador temporal para IoT basado en hardware **
** Francisco Sanchez - TFG - 2020/21 - DTE **
Inicializando modulo Lora OK!
Registrando callback OK
Activando escucha OK
FCSA INIT...OK
Local addr: 0x00
Enviando Join OK
Inicializacion completa!
< Enviando FCSA beacon >
< Timestamp: 0 minutos 2 segundos 63 milisegundos >
< Timestamp: 0 minutos 7 segundos 64 milisegundos >

@@@ Paquete recibido @@@
< 0x02 0x01 0x01 0x00 0x00 0x00 0xb9 0x1d 0x00 0x00 0x82 0xc6 0x09
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 1
Recv Logical: 7609
Recv Hardware: 640642
Recv lrate: 0.000000
Inicializando lista nodo 0x02...
Local Seq: 1
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000000
######
< Timestamp: 0 minutos 12 segundos 65 milisegundos >

@@@ Paquete recibido @@@
< 0x01 0x01 0x01 0x00 0x00 0x00 0x5a 0x3c 0x00 0x00 0x1b 0xe5 0x09
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 01 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 1
Recv Logical: 15450
Recv Hardware: 648475
Recv lrate: 0.000000
Inicializando lista nodo 0x01...
Local Seq: 1
```

```
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000000
######
< Timestamp: 0 minutos 17 segundos 66 milisegundos > (L.5.1)
```

Captura del nodo #1

```
@@@ Paquete recibido @@@
< 0x00 0x02 >
@@@ sAddr: 00 | len: 2 @@@
### FCSA JOIN ###
Nodo: 0 (L.5.2)
< @!@! Critical Error: Gateway REJOIN @!@! >
Reset FCSA…
Reiniciando nodo...
FCSA INIT...OK

@@@ Paquete recibido @@@
< 0x00 0x01 0x01 0x00 0x00 0x00 0xd1 0x07 0x00 0x00 0xd1 0x07 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 1
Recv Logical: 2001
Recv Hardware: 2001
Recv lrate: 1.000000
Inicializando lista nodo 0x00...
< Nueva ronda 0 -> 1 >
Local Seq: 1
Local Base: 2001
Local LastUpdate: 635026
Local Lrate: 1.000000
######
< Timestamp: 0 minutos 4 segundos 165 milisegundos > (L.5.3)
```

Captura del nodo #2

```
@@@ Paquete recibido @@@
< 0x00 0x02 >
@@@ sAddr: 00 | len: 2 @@@
### FCSA JOIN ###
Nodo: 0 (L.5.4)
< @!@! Critical Error: Gateway REJOIN @!@! >
Reset FCSA...
Reiniciando nodo...
FCSA INIT...OK

@@@ Paquete recibido @@@
```

```
< 0x00 0x01 0x01 0x00 0x00 0x00 0xd1 0x07 0x00 0x00 0xd1 0x07 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 1
Recv Logical: 2001
Recv Hardware: 2001
Recv lrate: 1.000000
Inicializando lista nodo 0x00...
< Nueva ronda 0 -> 1 >
Local Seq: 1
Local Base: 2001
Local LastUpdate: 635034
Local Lrate: 1.000000
######
< Timestamp: 0 minutos 4 segundos 95 milisegundos > (L.5.5)
```

En la primera captura se ve que el nodo maestro se inicializa correctamente, así como una marca de tiempo reiniciada (L.5.1).

También se informa como si fuera un error crítico, que ha recibido un mensaje FCSA Join proveniente del nodo maestro (ver L.5.2 y L.5.4) y se informa de que se hace un reinicio del protocolo.

Esperamos unos minutos y vemos si la red sigue operativa:

Captura del nodo #0

```
@@@ Paquete recibido @@@
< 0x02 0x01 0x0b 0x00 0x00 0x00 0x66 0x0d 0x03 0x00 0x37 0xb6 0x0c
0x00 0x00 0x00 0x00 0x00 0xf7 0x21 0x10 0xec 0x07 0x00 0xf0 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 11
Recv Logical: 200038
Recv Hardware: 833079
Recv lrate: 0.000000
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 1.000014
nodo 1 hrate 1.000003 lrate 1.000021
nodo 2 hrate 1.000014 lrate 1.000008
Local Seq: 11
Local Base: 0
Local LastUpdate: 0
Local Lrate: 1.000023
######
< Timestamp: 3 minutos 22 segundos 113 milisegundos >
< Enviando FCSA beacon >
< Timestamp: 3 minutos 27 segundos 114 milisegundos >
```

```
< Timestamp: 3 minutos 32 segundos 115 milisegundos >
```

Captura del nodo #1

```
@@@ Paquete recibido @@@
< 0x02 0x01 0x0b 0x00 0x00 0x00 0x66 0x0d 0x03 0x00 0x37 0xb6 0x0c
0x00 0x00 0x00 0x00 0x00 0xf7 0x21 0x10 0xec 0x07 0x00 0xf0 0x3f >
@@@ sAddr: 02 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 11
Recv Logical: 200038
Recv Hardware: 833079
Recv lrate: 1.000008
Calculado minimos cuadrados nodo 0x02...
Hu/Hv: 1.000012
nodo 0 hrate 0.999993 lrate 1.000026
nodo 2 hrate 1.000012 lrate 1.000008
Local Seq: 11
Local Base: 186352
Local LastUpdate: 819383
Local Lrate: 1.000019
######
< Timestamp: 3 minutos 24 segundos 199 milisegundos >

@@@ Paquete recibido @@@
< 0x00 0x01 0x0c 0x00 0x00 0x00 0x31 0x1e 0x03 0x00 0x2d 0x1e 0x03
0x00 0x00 0x00 0x00 0x00 0x33 0xdc 0x2d 0x5d 0x18 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 12
Recv Logical: 204337
Recv Hardware: 204333
Recv lrate: 1.000023
Calculado minimos cuadrados nodo 0x00...
Hu/Hv: 0.999993
nodo 0 hrate 0.999993 lrate 1.000023
nodo 2 hrate 1.000012 lrate 1.000008
< Nueva ronda 11 -> 12 > (L.6.1)
Local Seq: 12
Local Base: 204337
Local LastUpdate: 837368
Local Lrate: 1.000018
######
< Timestamp: 3 minutos 29 segundos 200 milisegundos >
< Timestamp: 3 minutos 34 segundos 201 milisegundos >
```

Captura del nodo #2

```
< Enviando FCSA beacon >
```

```
< Timestamp: 3 minutos 24 segundos 127 milisegundos >

@@@ Paquete recibido @@@
< 0x00 0x01 0x0c 0x00 0x00 0x00 0x31 0x1e 0x03 0x00 0x2d 0x1e 0x03
0x00 0x00 0x00 0x00 0x00 0x33 0xdc 0x2d 0x5d 0x18 0x00 0xf0 0x3f >
@@@ sAddr: 00 | len: 26 @@@
### FCSA BEACON ###
Recv Seq: 12
Recv Logical: 204337
Recv Hardware: 204333
Recv lrate: 1.000023
Calculado minimos cuadrados nodo 0x00...
Hu/Hv: 0.999981
nodo 0 hrate 0.999981 lrate 1.000023
nodo 1 hrate 0.999988 lrate 1.000021
< Nueva ronda 11 -> 12 > (L.6.2)
Local Seq: 12
Local Base: 204337
Local LastUpdate: 837378
Local Lrate: 1.000007
######
< Timestamp: 3 minutos 29 segundos 128 milisegundos >
< Timestamp: 3 minutos 34 segundos 129 milisegundos >
```

Si miramos la información que imprime cada nodo vemos que la marca de tiempo de la red es la misma, lo que significa que la red está sincronizada, es decir, ha vuelto a funcionar aunque sea con una marca de tiempo reiniciada.

Otro indicio de que la red vuelve a funcionar es que se siguen produciendo inundaciones (ver L.6.1 y L.6.2).

5.2.3 Prueba de precisión

En esta última prueba, vamos a comprobar con qué precisión se sincronizan los nodos. Esto lo vamos a evaluar generando un evento en todos los nodos de forma simultánea y analizando la marca de tiempo obtenida por cada nodo.

El análisis se basa en lo siguiente:

- Por cada nodo i se va a registrar un marca de tiempo t_i^j por cada evento j, dicho evento consistirá en el flanco de subida de una señal que está conectada a todos los nodos y que se activa mediante un generador de onda cuadrada, con un periodo en alto de 100 milisegundos (ms) y en bajo de 9900 ms.
- Dado que FCSA es un protocolo de sincronización centralizada, donde hay una marca de tiempo absoluta que es la del nodo maestro, vamos a calcular las diferencias por cada evento entre cada nodo y el nodo maestro. Es decir, por cada nodo $i \neq 0$ y cada evento j vamos a calcular $t_i^j - t_0^j$. Posteriormente se analizarán y mostrarán los resultados.

Las capturas correspondientes a la obtención de eventos son las siguientes:

Captura del nodo #0 de los eventos, en este caso, con origen en el nodo #3

```
@@@ Paquete recibido @@@
< 0x03 0x03 0x07 0x00 0xfe 0x3f 0xf4 0xde 0x00 0x00 0x65 0x76 0x65
0x6e 0x74 0x6f 0x31 >
@@@ sAddr: 03 | len: 17 @@@
#### DATA PACKET ####
sAddr: 3 | len: 7
Content:
< JSON {"saddr":"3","ts":57076,"content":"evento1"} >
< Timestamp: 1 minutos 2 segundos 14 milisegundos >
< Timestamp: 1 minutos 7 segundos 15 milisegundos >
Evento
< JSON {"saddr":"0","ts":67288,"content":"evento2"} > (L.1.1)

@@@ Paquete recibido @@@
< 0x01 0x03 0x07 0x00 0xfe 0x3f 0x77 0x06 0x01 0x00 0x65 0x76 0x65
0x6e 0x74 0x6f 0x31 >
@@@ sAddr: 01 | len: 17 @@@
#### DATA PACKET #### (L.1.2)
sAddr: 1 | len: 7
Content:
< JSON {"saddr":"1","ts":67191,"content":"evento1"} >
```

Podemos observar como se recibe un paquete de datos (ver L.1.2) con la información sobre el evento. También podemos ver como el nodo #0 reporta la detección de un evento (ver L.1.1)

Captura del nodo #0 filtrando líneas con la palabra JSON

```
< JSON {"saddr":"0","ts":18876,"content":"evento2"} >
< JSON {"saddr":"1","ts":18795,"content":"evento1"} >
< JSON {"saddr":"4","ts":18795,"content":"evento1"} >
< JSON {"saddr":"2","ts":18795,"content":"evento1"} >
< JSON {"saddr":"3","ts":18796,"content":"evento1"} >
< JSON {"saddr":"0","ts":28989,"content":"evento2"} >
< JSON {"saddr":"2","ts":28903,"content":"evento1"} >
< JSON {"saddr":"1","ts":28902,"content":"evento1"} >
< JSON {"saddr":"4","ts":28902,"content":"evento1"} >
< JSON {"saddr":"1","ts":39016,"content":"evento1"} >
< JSON {"saddr":"4","ts":39016,"content":"evento1"} >
< JSON {"saddr":"3","ts":39016,"content":"evento1"} >
...
```

Tras haber comprobado que se recibe información de los eventos con su correspondiente marca de tiempo, se generan un total de 115 eventos y se almacenan las marcas de tiempo de cada evento por cada nodo.

Ahora que tenemos estos datos, podemos pasar a clasificarlos y a ordenarlos (ver Tabla 12).

# Evento	Nodo #0	Nodo #1	Nodo #2	Nodo #3	Nodo #4
1	24609	24521	24527	24527	24522
2	34723	34636	34641	34641	34635
3	44838	44740	44744	44743	44742
4	54951	54858	54857	54857	-
5	65066	64968	64967	64968	64967

Tabla 12. Ejemplo de las marcas de tiempo

Evento #3			
Nodos		**Nodo #0**	**Nodo#n - Nodo#0**
#1	44740	44838	-98
#2	44744	44838	-94
#3	44743	44838	-95
#4	44742	44838	-96
	Media de las diferencias:		**-95.75(ms)**

Tabla 13. Ejemplo de las operaciones realizadas por cada evento, caso del evento #3.

Una vez mostrado un ejemplo representativo del cálculo en cada evento (ver Tabla 13), vamos a proceder a realizar el mismo cálculo sobre todos los eventos, para posteriormente poder visualizarlos (ver Fig. 7).

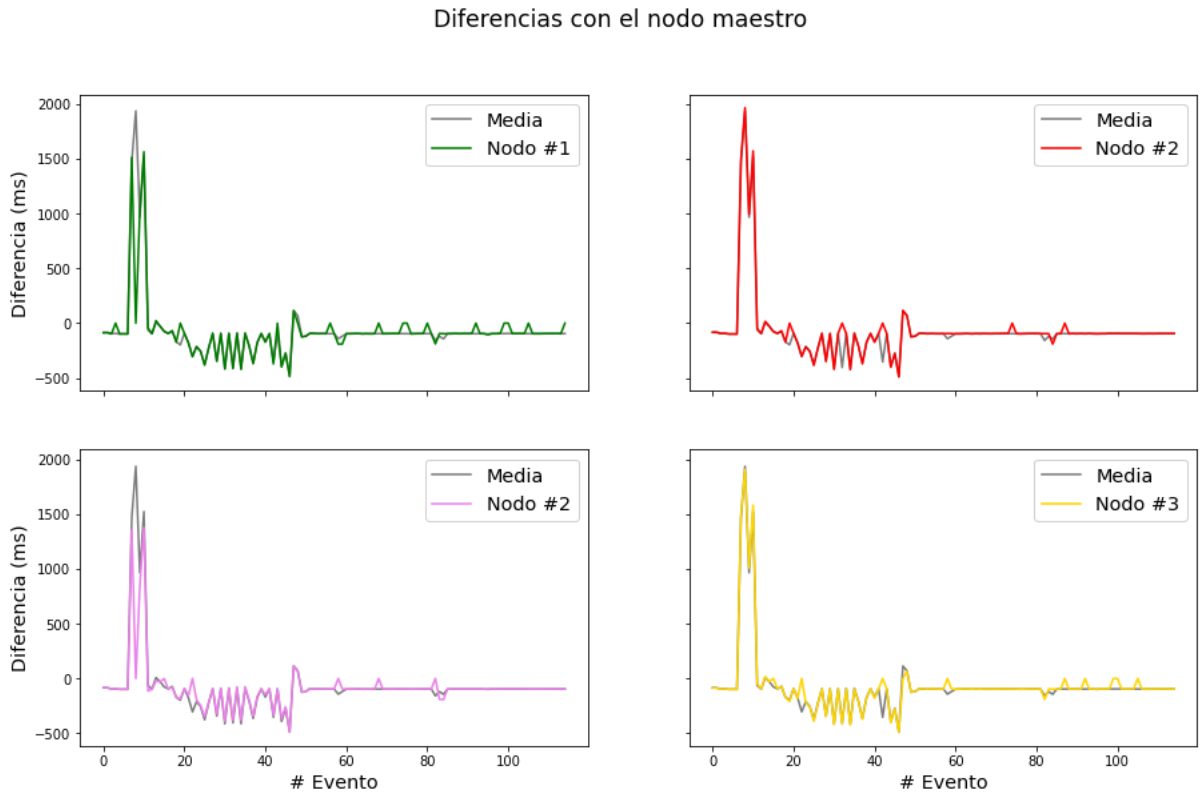


Figura 7. Visualización de las diferencias entre las marcas de tiempo del nodo maestro y cada nodo.

En cada gráfica vemos la marca de tiempo de cada nodo junto con la media del resto de nodos.

Ahora que contamos con los datos, podemos analizarlos y obtener medidas precisas (ver Tabla 14).

Análisis sobre la lista de medias de diferencias	
Media	**-72.21 (ms)**
Media (sobre valores absolutos)	**177.98 (ms)**
Mediana	**-95.75 (ms)**
Máximo	**1936 (ms)**
Mínimo	**-488.25 (ms)**
Mínimo (sobre valores absolutos)	**8.5 (ms)**
Percentil 25%	**-98.37 (ms)**
Percentil 50%	**-95.75 (ms)**
Percentil 75%	**-94.70 (ms)**
Percentil 25% (sobre valores absolutos)	**95 (ms)**

Percentil 50% (sobre valores absolutos)	**95.75 (ms)**
Percentil 75% (sobre valores absolutos)	**119.126 (ms)**
Intervalo de confianza 95%	**[-97.26 , -44.74] (ms)**

Tabla 14. Resultados de la prueba

Si observamos los datos (ver Tabla 14) y las gráficas (ver Fig. 7) **se puede concluir que el demostrador mantiene una precisión estable menor que 100 milisegundos**, aunque en ocasiones sufre perturbaciones. Esta diferencia entre el nodo maestro y el resto de los nodos podría ser usada, por ejemplo, para realizar una calibración de la red y así obtener mejor precisión.

6. Conclusiones

Como último capítulo y para dar cierre a este trabajo, se van a tratar de forma objetiva los resultados del trabajo, se van a proponer trabajos futuros y el autor va a dar su visión subjetiva del mismo.

6.1 Resumen y conclusiones

Como principal resultado del trabajo, tenemos el demostrador en sí mismo. Es decir, **se ha conseguido implementar un protocolo de sincronización que se ejecuta sobre una red de sensores, incluida la red en sí misma**. Como se vio en las pruebas, el demostrador es funcional y tiene una precisión típica inferior a los 100 milisegundos, con margen de mejora. Colateralmente, tenemos otros objetivos cumplidos dentro de este trabajo:

- Se han analizado diferentes protocolos de sincronización para redes de sensores.
- Se han analizado varios tipos de redes para el IoT y redes de sensores.
- Se han analizado diferentes tecnologías inalámbricas para redes de sensores.
- Se han analizado varias plataformas para implementar los nodos de una red de sensores.

Siendo el objetivo del trabajo *desarrollar un demostrador de un sincronizador temporal para IoT*, podemos decir que **el trabajo ha concluido con éxito**.

6.2 Trabajos futuros

Dado que el tiempo a invertir en este trabajo es limitado, hay muchas cosas muy susceptibles de añadir al proyecto que no se han podido añadir pero que pueden continuar en futuros desarrollos. Veamos algunos ejemplos:

- **Añadir soporte *hardware* para aumentar la precisión**: añadiendo algún periférico auxiliar que ayude a marcar el momento en el que se envían y se reciben los paquetes con más precisión puede contribuir sustancialmente a darle más precisión al demostrador.
- **Continuar desarrollando el puente entre la LPWAN e Internet a través del *gateway*,** por ejemplo emplear algún software para almacenar y visualizar los datos recogidos por la LPWAN (como InfluxDB y Grafana).
- **Añadir un mecanismo de calibración:** como se vio en la prueba de precisión, en un porcentaje significativamente alto de eventos, la desviación entre el nodo maestro y el resto de nodos era la misma o se encontraba en un intervalo reducido. Este parámetro podría ser estudiado en la red e intentar corregir todas las marcas de tiempo en función a este dato.
- **Añadir sensores reales a los nodos**: Realmente esta es la verdadera finalidad de una red de sensores inalámbrica, pero que aquí hemos obviado.

- **Perfeccionar FCSA**:
    - Incluir un algoritmo para la rotación del nodo maestro o elegir uno nuevo en caso de fallo de este.
    - Optimizar el cálculo de la pendiente, quizás apoyándose en una FPU o conociendo mejor la ALU del microcontrolador.
    - Emplear la interrupción TxDone del módem para obtener marcas de tiempo más precisas a la hora de enviar los beacons.
    - Determinar una heurística para el tiempo de envío de beacons.

6.3 Opinión personal

A continuación voy a comentar mi visión personal y subjetiva sobre este trabajo. A primera vista, digo que el proyecto ha finalizado con éxito puesto que el alcance se ha llevado a cabo (no sin pena ni gloria) según los plazos.

La parte de estudio previo ha sido especialmente interesante por el apartado de protocolos de sincronización puesto que el alumno autor no tenía mucha idea de la variedad de protocolos de sincronización para redes de sensores.

También ha sido interesante acercarse a las redes de sensores y las redes LPWAN, pues personalmente me llama mucho la atención la infraestructura para la monitorización de “cosas” como ciudades o campos, pues creo que el IoT supondrá (si no lo supone ya) una gran revolución histórica.

7. Referencias

1. Consejería de Agricultura, Pesca y Desarrollo Rural (2018). “Informe final sobre la consulta preliminar del mercado *Perfiles Profesionales Ámbito Informático*”. Descargado de: https://tfc.eii.us.es/TFG/APP/connector/0/46/href/informe_precios_perfiles_informaticos.pdf (fecha de consulta: 31 de Agosto de 2021)

2. Periódico The Guardian. “The internet of things”. Enlace: https://www.theguardian.com/technology/2003/oct/09/shopping.newmedia (fecha de consulta: 21 de Agosto de 2021)

3. Web www.iotworldonline.es. “Las grandes estadísticas del Internet de las Cosas (IoT)”. Enlace: https://www.iotworldonline.es/las-grandes-estadisticas-del-internet-de-las-cosas-iot/ (fecha de consulta: 21 de Agosto de 2021)

4. Wikipedia ES. “Redes de sensores”. Enlace: https://es.wikipedia.org/wiki/Red_de_sensores#Historia (fecha de consulta: 22 de Agosto de 2021)

5. Web oficial SigFox. “Sigfox - The Global Communications Service Provider for the Internet of Things (IoT)”. Enlace: https://www.sigfox.com/en (fecha de consulta: 24 de Agosto de 2021)

6. Wikipedia ES. “Narrowband IoT”. Enlace: https://en.wikipedia.org/wiki/Narrowband_IoT (fecha de consulta: 24 de Agosto de 2021)

7. LoRa Alliance. “LoRaWAN® Specification v1.1” . Enlace: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/ (fecha de consulta: 24 de Agosto de 2021)

8. Multidisciplinary Digital Publishing Institute - MPDI. “Overview of Time Synchronization for IoT Deployments: Clock Discipline Algorithms and Protocols”. Enlace: https://www.mdpi.com/1424-8220/20/20/5928 (fecha de consulta: 10 de Abril de 2021)

9. Semtech. “SX1276 | 137MHz to 1020MHz Long Range Low Power Transceiver”. Enlace: https://www.semtech.com/products/wireless-rf/lora-core/sx1276 (fecha de consulta: 10 de Febrero de 2021)

10. GitLab. “Sincronizado Temporal IoT Hardware” . Enlace: https://gitlab.com/franlego98/sincronizador_temporal_iot_hardware

11. GitHub. “Arduino LoRa - An Arduino library for sending and receiving data using LoRa radios”. Enlace: https://github.com/sandeepmistry/arduino-LoRa (fecha de consulta: 20 de Mayo de 2021)

12. IEEE Xplore. “Time Synchronization Based on Slow-Flooding in Wireless Sensor Networks”. Enlace: https://ieeexplore.ieee.org/document/6463406/ (fecha de consulta: 15 de Abril de 2021)