Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Escuela de Ingeniería Informática
Universidad de Oviedo

MEMORIA DEL TRABAJO FIN DE MÁSTER

Sistema para la gestión remota de comunicaciones en dispositivos inteligentes

Autor:
Karol Mateusz Ciok

Director
Jordán Pascual Espada

Tabla de contenido

Índice de tablas

Índice de figuras

1. Resumen

Resulta complejo encontrar una definición de Internet de las Cosas - Internet of Things (IoT) que sea aceptada de forma completa por la mayoría de los investigadores. Se podría decir que IoT hace referencia a la interconexión de objetos físicos a través de una red. El objetivo es que estos objetos interactúen entre ellos y/o con otros sistemas informáticos [1].

En recientes años, sistemas enmarcados dentro del término de Internet de las Cosas, han cambiado la manera en la que vivimos, trabajamos o hacemos negocios en muchas áreas, incluso en aquellas que hasta hace poco parecían improbables. Algunas de las áreas que se pueden beneficiar de su aplicación son, servicios de la salud, ciudades inteligentes, automatización industrial, agricultura inteligente, sistemas de transporte inteligentes, logística inteligente o respuesta a emergencias.

Sin embargo, en muchos casos, los sistemas IoT son soluciones innovadoras, la creación del sistema tiene un alto grado de incertidumbre y restricciones, por lo que a menudo es necesario probar diferentes enfoques y arquitecturas para alcanzar una forma adecuada de crear el sistema IoT. Por ello los prototipos y desarrollos rápidos tienen un papel importante en la creación de este tipo de sistemas.

La construcción de prototipos es algo habitual en la ingeniería del software. Un prototipo se podría definir de forma simple como un producto no final que incluye una parte de las características o funcionalidades del sistema. Un aspecto importante sobre la creación de prototipos IoT es conseguir reducir el tiempo empleado en su creación [2]. Los prototipos permiten probar la viabilidad o la eficiencia del sistema en una etapa temprana. Los sistemas IoT se basan en la integración de múltiples tecnologías y elementos. Determinar si los dispositivos, la arquitectura y las tecnologías utilizadas permitirán la creación de una solución que se adapte a los requerimientos puede resultar desafiante [3].

Otro aspecto importante del desarrollo de sistemas IoT es la capacidad de actualización o de producir cambios en el software de los dispositivos involucrados. Debido al alto grado de incertidumbre de los sistemas IoT, incluso después de una extensa fase de diseño con valoraciones de diferentes enfoques, un sistema ya en ejecución también puede requerir cambio o actualizaciones, por ejemplo añadir un nuevo dispositivo, cambiar la dirección de un servicio, etc.

Los enfoques más tradicionales para realizar las actualizaciones en el software de los dispositivos IoT conllevan varios costes que en algunos casos pueden suponer un problema para el sistema. Uno de los mecanismos de actualización más comunes es la actualización OTA (Over the Air). En este tipo de actualizaciones se sustituye el programa del dispositivo por e uno nuevo enviado a través de la red. En muchos casos esta sustitución implica parar la ejecución del dispositivo durante un tiempo. Este tipo de actualizaciones también puede tener un impacto en el consumo de recursos del dispositivo ya que, en algunos sistemas IoT, los dispositivos tienen como fuente de energía una pila y actualizaciones constantes aumentarían el consumo.

Este trabajo de investigación propone una novedosa alternativa que permite crear sistemas IoT capaces de hacer cambios remotos en la comunicación de los dispositivos de una manera ágil y rápida. Nuestra propuesta permite realizar algunos de los cambios en la comunicación más comunes que se hacen durante la fase de desarrollo y mantenimiento de sistemas IoT minimizando el tiempo que el dispositivo permanece inactivo. Nuestra propuesta permite cambiar el destino del envío de los datos, enviar los datos a varios destinos o cambiar la frecuencia de envío de los datos.

Mediante la aplicación de la propuesta se busca permitir la posibilidad de realizar cambios en la comunicación de forma ágil y con poco coste para los desarrolladores, por ejemplo, se tiene un

sistema IoT en el cual participan varios dispositivos con sensores de temperatura que envían los datos a un servicio centralizado. Este servicio analiza los datos e interacciona con otro dispositivo que controla el sistema de refrigeración. Si en algún momento se quiera cambiar el servicio, bastaría con enviar un comando para cambiar el destino al que se envían los datos.

Otro ejemplo, con una topología diferente podría ser un sistema en malla donde hay múltiples dispositivos que se envían datos entre ellos. Si en algún momento se quisieran añadir nuevos dispositivos al sistema, sería suficiente con indicar mediante comandos remotos que algunos dispositivos además de enviar los datos a los destinatarios actuales, también los envíen a los nuevos dispositivos.

Nuestra solución, que hemos denominado Flex-Request, se utiliza en los programas que se cargan en los dispositivos. Cuando el dispositivo se arranca, se conecta en segundo plano con el servidor de configuración y se queda a la escucha de cambios. Los cambios se envían al servidor de configuración usando comandos específicos. Cuando el dispositivo detecta un cambio, se procesa el comando recibido y se aplica el cambio en la configuración sin parar la ejecución del programa.

Hemos creado una implementación concreta de la propuesta, con parte de la funcionalidad principal para poder evaluar la propuesta. En el sistema de ejemplo se tiene un dispositivo IoT que captura datos de temperatura y un servidor que los recibe. Hemos diseñado tres experimentos para evaluar nuestra propuesta. Cada experimento corresponde con un caso de uso de las principales funcionalidades.

En cada experimento se han medido varios aspectos comparando con varias alternativas. El primer aspecto es la complejidad del programa que se ha medido basándose en el tamaño del programa utilizando un sistema de utilizado en anteriores trabajos de investigación. En este aspecto nuestra propuesta tiene la menor complejidad de todas las alternativas.

Otro aspecto evaluado es la complejidad de realizar el cambio. A diferencia de la complejidad de implementación, la realización del cambio puede implicar otras acciones adicionales (uso de herramientas de configuración, comandos, etc). Por ello para estimar la complejidad de cambio se ha tenido en cuenta todas las acciones realizadas por el desarrollador con el teclado y ratón. Nuestra propuesta tiene la segunda complejidad de cambio más baja para el segundo experimento y la tercera complejidad de cambio más baja para el primer y tercer experimento.

También se ha medido el tiempo de inactividad del dispositivo cunando se realiza la actualización. En este caso, nuestra propuesta no ha tenido diferencias significativas en los dos primeros experimentos, pero en el tercero el tiempo de inactividad fue significativamente más corto.

Por último, aunque optimizar el tráfico de red no sea un objetivo de esta investigación, se ha medido este aspecto porque podría ser un factor importante en algunos sistemas IoT. Se ha medido el tráfico de red durante un periodo de tiempo sin hacer cambios de configuración y haciendo cambios en la comunicación para cada alternativa. Nuestra propuesta, a pesar de tener el tráfico más alto en ambos casos, la diferencia con otras alternativas está entre 4% - 12% durante un período de uso normal, sin cambios en la comunicación y 0,4% - 9% cuando se realizaron cambios.

Palabras clave: Internet de las cosas, Sensores, Configuración remota, Actualización del software, Protocolo MQTT.

2. Introducción

2.1. Internet de las Cosas

Resulta complejo encontrar una definición de Internet de las Cosas - Internet of Things (IoT) que sea aceptada de forma completa por la mayoría de los investigadores. Se podría decir que IoT hace referencia a la interconexión de objetos físicos a través de una red. El objetivo es que estos objetos interactúen entre ellos y/o con otros sistemas informáticos [1]. Lo realmente importante no es definir de forma exacta el término, sino comprender el concepto y saber que los sistemas IoT suponen la integración de aspectos del mundo físico y digital.

En recientes años, sistemas de Internet de las Cosas, han cambiado la manera en la que vivimos, trabajamos o hacemos negocios en muchas áreas, incluso en aquellas que hasta hace poco parecían improbables. Este avance en el uso de los sistemas IoT es debido en parte al abaratamiento de los costes físicos de su instalación y también a la investigación relativa a Internet of Things. Algunas de las áreas que se pueden beneficiar de su aplicación son, entre otras, servicios de la salud, ciudades inteligentes, automatización industrial, agricultura inteligente, sistemas de transporte inteligentes, logística inteligente o respuesta a emergencias [4][5]. Sin embargo, la cantidad de campos que pueden acomodar una solución de IoT es enorme y continúa expandiéndose a medida que más dispositivos alcanzan las capacidades de conexión a Internet. Para el año 2020, se espera que más de 50 mil millones de dispositivos estén conectados a Internet de las cosas [6].

2.2. Prototipado y desarrollo de sistemas IoT

En muchos casos, los sistemas IoT son soluciones innovadoras, la creación del sistema tiene un alto grado de incertidumbre y restricciones, por lo que a menudo es necesario probar diferentes enfoques y arquitecturas para obtener una forma adecuada de crear el sistema IoT. Por ello los prototipos y desarrollos rápidos tienen un papel importante en la creación de sistemas IoT.

La construcción de prototipos es algo habitual en la ingeniería del software. Un prototipo se podría definir de como un producto no final que incluye una parte de las características o funcionalidades del sistema. En muchos casos, los prototipos se pretenden desarrollar de una forma ágil, rápida y sin invertir excesivos recursos en su desarrollo con el objetivo de probar la viabilidad de un sistema, realizar una prueba de concepto, entregárselo a un cliente para validar requisitos u obtener retroalimentación, etc.

Los prototipos y desarrollos rápidos tienen un papel importante en la creación de sistemas IoT [2]. Los prototipos permiten probar la viabilidad o la eficiencia del sistema en una etapa temprana. Los sistemas IoT se basan en la integración de múltiples tecnologías y elementos y por ello determinar si los dispositivos, la arquitectura y las tecnologías utilizadas permitirán la creación de una solución que se adapte a los requerimientos puede resultar desafiante [3].

2.3. Mantenimiento y modificación de sistemas IoT

Los dispositivos IoT generalmente obtienen datos de su entorno cercano utilizando los sensores. Estos dispositivos también pueden realizar acciones, utilizando actuadores como por ejemplo relés. En los sistemas IoT, los dispositivos se distribuyen en redes para intercambiar datos y realizar tareas juntos. La comunicación entre dispositivos y servicios es uno de los puntos clave de los sistemas IoT. Sin embargo, cuando aumenta el número de dispositivos y servicios, también lo hace la dificultad de realizar cambios en las comunicaciones. En muchos casos, los sistemas IoT están compuestos de muchos dispositivos pequeños dispersos en varias localizaciones. Debido a eso, algunas modificaciones pueden implicar muchas actualizaciones.

Pongamos el ejemplo de un sistema que tiene una serie de dispositivos que envían los datos registrados a un servicio centralizado. En un momento dado surge la necesitada de cambiar este

servicio por uno nuevo. Esto podría suponer cambiar la IP del servicio al que los dispositivos envían los datos. Por ello los desarrolladores tendrían que actualizar el software de todos los dispositivos a los que afectaría el cambio.

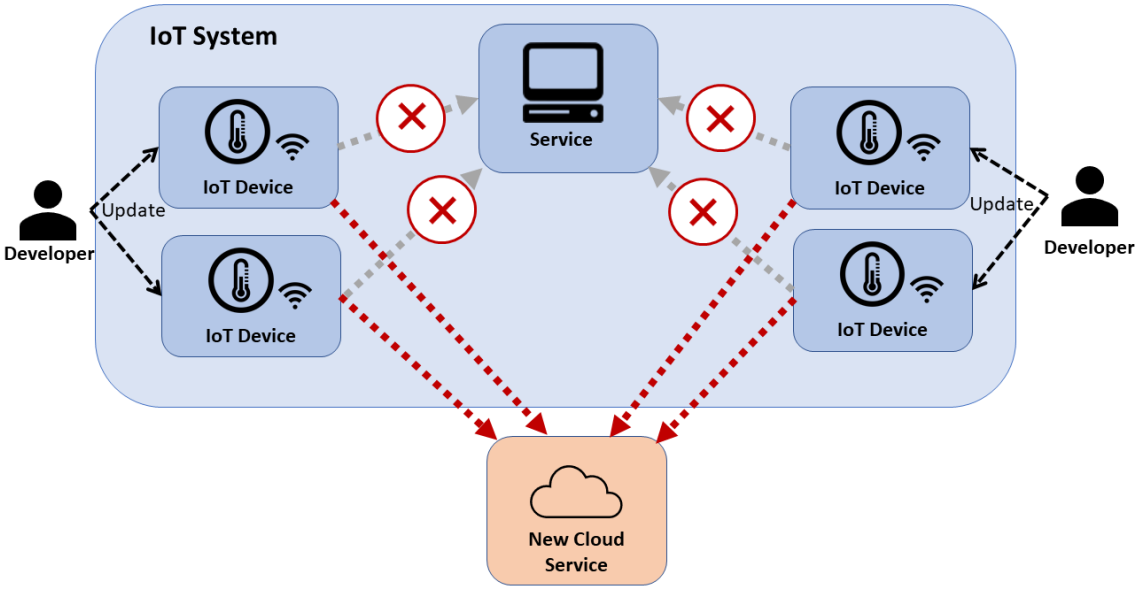

Figura 1. Actualización de dispositivos IoT

A pesar de los potenciales beneficios del uso de dispositivos IoT, la adopción por la industria de aplicaciones innovadoras de sensores y/o actuadores a gran escala es lenta. Una de las razones principales es la falta de soporte integrado para el mantenimiento de dichos dispositivos. Aunque los estándares brindan muchas opciones para configurar los protocolos de red, es difícil cambiar la configuración en tiempo de ejecución. Esto implica que toda la red necesita ser reconfigurada *offline* si se requieren cambios [7]. Los sistemas IoT pueden basarse en muchas topologías de comunicación, como cliente-servidor, árboles, mallas, P2P, etc. En muchos casos, los sistemas IoT son soluciones innovadoras, la creación del sistema tiene un alto grado de incertidumbre y restricciones, por lo que a menudo es necesario probar diferentes enfoques y arquitecturas para obtener una forma adecuada de crear el sistema IoT.

Debido a este grado de incertidumbre, desde el inicio del desarrollo del software para esos dispositivos, es interesante que los desarrolladores puedan realizar cambios en la comunicación de una manera eficiente y rápida. Esos cambios podrían traer ventajas para los desarrolladores en diferentes fases del diseño e implementación del sistema IoT. Por ejemplo, estos cambios rápidos permitirían a los desarrolladores realizar prototipado y despliegue de alternativas de manera rápida para encontrar las más adecuadas para las necesidades del sistema.

Haciendo referencia al ejemplo anterior, si los desarrolladores tuvieran una manera más eficiente de cambiar el proveedor de nube, esta tarea llevaría menos tiempo con lo que se podría probar más proveedores para analizar sus ventajas o desventajas.

Conseguir eficiencia en estos cambios, no solamente permitirían un desarrollo más rápido si no que también favorecerían el mantenimiento y evaluación del sistema ya desplegado. Por ejemplo, en un sistema con topología de malla, donde una serie de dispositivos se comunican unos con otros, si lo desarrolladores puede cambiar el eficientemente el destinatario con el que se comunica un dispositivo, se podrían añadir o quitar dispositivos del sistema sin tener que dejarlo parado o fuera de servicio total o parcialmente para reconfigurar las comunicaciones.

La complejidad del software en los dispositivos IoT Puede llegar a ser relativamente baja, en muchos casos, los dispositivos no requieren realizar tareas muy complejas. Por ejemplo, algunos dispositivos se limitan a obtener datos a través de un sensor y enviarlos a un servidor. Sin embargo, la complejidad del software del dispositivo podría estar condicionada por algunos factores, como el lenguaje de programación, la longitud del código y particularidades del *framework* utilizado para acceder al hardware y las comunicaciones del dispositivo [8][9]. Sistemas IoT tienden a tener cierta incertidumbre y poder crear software de manera más rápida, permite a los programadores realizar cambios más eficientes y evaluar más alternativas con el objetivo de elegir la más adecuada para sus necesidades [10][11].

Otro aspecto crítico a la hora de hacer cambios cuando el sistema está en producción es el tiempo de inactividad o perdida de datos tras un cambio. La aplicación de los cambios podría requerir que un dispositivo tuviera que detener el programa actual y ejecutar uno nuevo. Además, a veces el acceso físico a los dispositivos puede llegar a ser una tarea de mucho coste, bien porque estén situados en sitios con difícil acceso (por ejemplo, un techo), o bien porque el sistema disponga de muchos dispositivos que deben ser actualizados. Esto provoca que la actualización se deba enviar de forma remota lo que hace que el proceso de actualización sea aún más lento. Por ello, hay un período de tiempo en el que el dispositivo podría no recopilar ni enviar los datos del sensor [12] tras una actualización.

Un ejemplo de este problema podrías ser una planta de energía que utiliza dispositivos IoT con sensores para monitorizar el funcionamiento esta. En este caso, la recopilación y el análisis de los datos podría ser fundamental para administrar de manera segura la planta y evitar la pérdida de datos potencialmente importantes.

Hoy en día existen varias alternativas diferentes para mitigar el coste de tener que realizar cambios en algunos aspectos de los sistemas IoT. Estas alternativas podrían aplicarse para realizar cambios en la comunicación entre dispositivos y servicios en aspectos que tienen que ver con la comunicación entre dispositivos y servicios. La mayor parte de estas alternativas deben ser consideradas en la fase de diseño del sistema IoT, ya que pueden requerir diseñar o construir el sistema y el software de los dispositivos de una forma específica. El uso varias de estas alternativas incluyen restricciones al sistema que pueden implicar el uso de dispositivos con unas determinadas características, el uso de ciertas tecnologías o la aplicación de arquitecturas de comunicación concretas. Las soluciones más utilizadas se pueden agrupar en las siguientes categorías:

- Actualización de software total o parcial basada en OTA. Este tipo de actualizaciones se podrían utilizar para hacer cualquier tipo de cambio ya que ya que puede actualizar de forma completa todo el software del dispositivo. Esto puede resultar en un alto coste de tiempo y tráfico de red ya que dependen directamente del tamaño del programa a actualizar. Aunque hay algunas soluciones que intentan minimizar el tamaño de la actualización [10] o [11], muchas actualizaciones OTA aún necesitan reiniciar el dispositivo para aplicar la actualización. Las soluciones de OTA parciales si permiten disminuir el tráfico de red ya que solamente se envía un paquete con los cambios entre el programa actual y el nuevo, sin embargo, proceso solo es compatible con dispositivos que ejecutan sistemas operativos relativamente complejos.
- Dispositivos administrados por una plataforma IoT que utiliza un servidor proxy o un servidor de puerta de enlace. En este tipo de soluciones las peticiones entre los dispositivos pueden pasar por un proxy por lo que podría ser posible realizar algunos cambios en las comunicaciones del sistema solamente modificando el proxy. Existen varias soluciones basadas en el uso de Proxies en entornos IoT [7] y [13].

- Dispositivos que utilizan software modular. El software modular está diseñado de tal forma que el propio software puede actualizar algunas de sus partes. Existen muchas versiones de software modular tanto dentro del entorno IoT como fuera, algunos ejemplos podría ser el *node-supervisor*[1] o el *hotswap*[2]. Esas soluciones también tienen algunas desventajas. El *node-supervisor* reinicia el programa cuando uno de los archivos cambia. Esto provocaría que se pierda el estado de ejecución del programa. Por otro lado, *hotswap* cuando detecta un cambio, vuelve a cargar los módulos afectados por el cambio sin detener el programa. Sin embargo, en este caso, los programas deben realizarse de forma modular, lo que podría implicar una arquitectura de aplicación más compleja y ademar obligar al desarrollador a prever todos los posibles cambios futuros.
- Dispositivos que consultan sus parámetros de comunicación desde servicios externos. En esta categoría podemos incluir servicios como *Firebase Remote Config*[3]. Con este tipo de servicios los desarrolladores pueden guardar algunos parámetros de la aplicación en la nube y consultarlos dentro del programa. Esto permite cambiar posteriormente estos parámetros sin necesidad de modificar el programa. La principal desventaja de esos servicios es que solamente permiten guardar parámetros muy específicos generalmente de tipos simples (valores numéricos, cadenas de texto, valores booleanos, etc). Además, la utilización de este tipo de servicios obliga a desarrollar el software de una manera concreta, comúnmente más compleja que un desarrollo tradicional, ya que es necesario integrar la conexión con el servicio remoto dentro del programa desarrollado e interpretar los diferentes valores de la configuración en la lógica implementada.

El objetivo principal de este trabajo de investigación es proporcionar una alternativa que permita crear sistemas IoT capaces de realizar cambios remotos en la comunicación de los dispositivos de una manera ágil y rápida, permitiendo realizar algunos de los cambios en la comunicación más comunes que se hacen durante la fase de desarrollo y mantenimiento de sistemas IoT como cambiar el destino del envío de los datos, enviar los datos a varios destinos o cambiar la frecuencia de envío de los datos. Para que se considere apropiado, la solución debe cumplir con los siguientes aspectos:

- La complejidad de implementar software capaz de modificar de forma remota sus aspectos de comunicación debe ser reducida. La utilización de la propuesta no debe implicar a los desarrolladores altos costes adicionales en comparación con el desarrollo de un software tradicional para un dispositivo IoT.
- El coste de realizar un cambio de comunicación de un dispositivo debe ser reducido. Los desarrolladores deberían poder realizar cambios de forma rápida y eficiente en comparación con las alternativas actuales.
- Finalmente, la propuesta debe minimizar el tiempo que el dispositivo IoT está apagado o que no funciona correctamente después de hacer un cambio en la comunicación. En algunos de los sistemas IoT es común que después de un cambio o actualización en el software de un dispositivo, este dispositivo permanezca inactivo por un tiempo. Este tiempo de inactividad debe ser corto.

[1] node-supervisor - https://github.com/petruisfan/node-supervisor
[2] hotswap - https://github.com/rlidwka/node-hotswap
[3] Firebase Remote Config - https://firebase.google.com/docs/remote-config

3. Trabajo relacionado

3.1. Internet de las Cosas

Un microcontrolador es un circuito integrado único que al menos contiene los elementos necesarios de un sistema informático completo. Puede contener módulos periféricos adicionales, como un temporizador. Un microcontrolador debe tener más que solo una CPU, un programa y una memoria de datos para poder resolver problemas del mundo real. Además, debe contener hardware que permita acceder a información del mundo exterior. Una vez que haya reunido información y la haya procesado, también debe ser capaz de efectuar cambios en mundo exterior [14].

Los dispositivos son una fuerte fuente de heterogeneidad en los sistemas IoT. Hay cientos de microcontroladores diferentes con una variedad de características. Algunos de estos dispositivos tienen funciones concretas y se adquieren ya programados (por ejemplo, SmartEdge Agile[4], Dashcam Smart Camera[5], etc.) Otros dispositivos son programables y ampliables, lo que permite que los desarrolladores puedan crear soluciones a medida: Siemens S, Mitsubishi FX, Kunbus PR, Arduino, Raspberry Pi e Intel Galileo [15] [16]. Por lo general, tienen una serie de pines GPIO (entrada/salida de propósito general) para permitirles conectar periféricos.

Estos microcontroladores son la base necesaria para el siguiente paso en la escalera: los objetos inteligentes. Los objetos inteligentes son objetos físicos/digitales autónomos aumentados con capacidades de detección, procesamiento y comunicación. Ejecutan aplicaciones que les permiten realizar funciones de diversos tipos, desde registrar datos hasta interactuar con humanos. Los objetos inteligentes pueden percibir, registrar e interpretar lo que ocurre dentro de ellos y del mundo, actúan por su cuenta, se comunican entre sí e intercambian información con las personas [17].

Los objetos inteligentes pueden ser muy diferentes y pueden clasificarse de muchas maneras diferentes, en función de diferentes características como, capacidad informática, habilidades de comunicación, nivel de inteligencia, propósito, etc [5].

Los dispositivos poseen un sistema operativo más o menos potente dependiendo de sus características técnicas. Todos los dispositivos soportan la ejecución de programas, los cuales pueden estar escritos en uno o varios lenguajes de programación diferentes dependiendo de las características del dispositivo. El lenguaje de programación y el framework del dispositivo permiten acceder de forma abstracta a sus componentes físicos: interfaces serial, memoria flash, módulo para el manejo de pines, etc.

3.2. Dispositivos conectados

Un aspecto importante de los dispositivos IoT es su capacidad de comunicación. Las tecnologías de comunicación soportadas por los dispositivos pueden ser muy variadas, algunas de las más comunes son las siguientes:

- **WiFi** es la tecnología de comunicación inalámbrica más popular, se basa en el estándar IEEE 802.11xx (IEEE 802.11, IEEE 802.11ac, IEEE 802.11b, IEEE 802.11g e IEEE 802.11n). Esta tecnología habilita la conexión con redes inalámbricas a distancias reducidas o medias. Las conexiones WiFi están soportadas por una gran cantidad de dispositivos. Las redes WiMaX, basadas en el estándar IEEE 802.16, permiten ofrecer

[4] SmartEdge https://www.avnet.com/wps/portal/integrated/solutions/capabilities/smartedge-agile/
[5] Dashcam https://www.nonda.co/products/dash-cam

distancias de conexión mucho mayores que el WiFi tradicional. Este tipo de comunicación suele trabajar sobre IPv4 o IPv6 a nivel de red, TCP o UDP.

- **Bandas celulares**. Es popularmente conocida por ser la base de las redes de comunicación de los teléfonos móviles, pero también está presente en muchos dispositivos IoT que necesitan conectarse a una plataforma a través de largas distancias. A pesar de permitir transferir grandes cantidades de datos, y el consumo de energía suele ser alto.
- **Bluetooth**. Es una tecnología de comunicación de rango corto, es comúnmente usada en productos de carácter personal, como teléfonos móviles, periféricos, sensores para el cuerpo o la ropa, etc. Existe la variación Bluetooth Low-Energy (BLE) pensada principalmente para reducir el consumo energético. Bluetooth dispone de gran variedad de protocolos propios con diferentes propósitos, entre los que se encuentran Service Discovery Protocol (SDP), para el descubrimiento de dispositivos; Object Exchange (OBEX) para el intercambio de información; Bluetooth Network Encapsulation Protocol (BNEP), para el envío de paquetes por redes personales, etc.
- **ZigBee** es un tipo de red inalámbrica que parte del estándar IEEE 802.15.4. Su principal activo es el bajo consumo de energía y la baja capacidad de computación que necesitan los dispositivos para utilizarla. Su uso principal se encuentra en la industria, aunque cada vez resulta más común encontrar este tipo de redes en los hogares inteligentes. ZigBee utiliza protocolos de aplicación propietarios[6]. Destaca el Application Framework (AF), para acceder al control del dispositivo, y el ZigBee Device Object (ZDO), para descubrimiento de dispositivos y gestión de red.
- **Lora / LoRaWan**[7]. Low Power Wide Area Network es una tecnología de comunicación para redes inalámbricas de gran tamaño, por ejemplo, grandes ciudades. Ha sido diseñado para soportar redes con millones de pequeños dispositivos conectados.

Además de la variación de las tecnologías de comunicación soportadas también existen multitud de protocolos de comunicación diferentes, cada uno con sus particularidades, que es necesario conocer bien para escoger el protocolo más adecuado en cada caso [18]. Algunos de los protocolos de aplicación más frecuentemente utilizados en los sistemas IoT son los siguientes:

- **Hypertext Transfer Protocol (HTTP).** Es el principal protocolo de aplicación en la web. Se basa en TCP/IP y permite la creación de sistemas cliente-servidor. Los dispositivos IoT que utilizan HTTP suelen implementar un cliente y enviar peticiones periódicas al servidor. Es útil para enviar grandes cantidades de información, aunque no es efectivo para sistemas en tiempo real que requieran realizar comunicaciones en orden de milisegundos.
- **Message Queue Telemetry Transport (MQTT)**. Se ideó para dispositivos restringidos y redes que disponen de poco ancho de banda, alta latencia o poco confiables. Su uso es indicado para dispositivos de poca potencia. Está basado en mecanismos de publicación/suscripción. Lo más común es que MQTT se base en TCP, en el que hace uso del reconocimiento de sesión continua; no obstante, existen algunas implementaciones sobre UDP. Su uso principal es el de grandes redes de dispositivos pequeños que necesitan supervisión. No está pensado para comunicación dispositivo-dispositivo, ni para multidifusión. Por lo general tiene retardo, por lo que su uso en algunos tipos de aplicaciones que requieren tiempo real puede no ser adecuado. MQTT-

[6] Zigbee https://www.sciencedirect.com/topics/engineering/zigbee-protocol
[7] Lora https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf

SN [19] es una variación de este protocolo, modificado para adaptarlo a redes de sensores. Entre algunas de sus ventajas figura un sistema de acceso por ID a los temas del mecanismo de publicación/suscripción; esto puede permitir ahorrar ancho de banda.

- **Constrained Application Protocol (CoAp)**. Protocolo de aplicación ideado para ser usado en nodos y redes de baja potencia y con pérdida. Suele ser utilizado en dispositivos con poca capacidad de computación o que pretenden reducir su consumo de batería. CoAp está basado en RESTful e identificadores de recursos, y manipula esos recursos a través de una interfaz de programación de aplicaciones (API) uniforme. La versión común de CoAP se basa en UDP y trata de reproducir algunas de las funcionalidades de TCP, y establece un sistema de intercambio de solicitudes y respuestas, las cuales se realizan de forma asíncrona.
- **Object Linking and Embedding for Process Control - Unified Architecture** (**OPC-UA)**. Surge a partir del estándar OPC, con el cual es retro compatible. Está basado en cliente-servidor, en los servicios web SOAP y está construido sobre TCP. Incluye diversas consideraciones de seguridad que lo convierten en un protocolo muy seguro.
- **Data Distribution Service** (**DDS**). Se basa en mecanismos de publicación-suscripción, permitiendo establecer prioridades en el transporte. Puede basarse en varios protocolos, aunque lo más común es encontrarlo sobre UDP. Puede llegar a ofrecer un buen nivel de seguridad a través de TLS, DTSL y DDS. Su principal campo de aplicación se encuentra en sistemas de tiempo real.
- **Advanced Message Queuing Protocol** (**AMQP**). Es un protocolo de aplicación que da soporte a la comunicación dispositivo-dispositivo mediante mecanismos publicación/suscripción. Está basado en TCP. Este protocolo viene del sector financiero, contempla la seguridad y las transacciones entre sus características.

3.3. Plataformas y desarrollo de sistemas IoT

La gestión de dispositivos IoT, especialmente si su número es alto, conlleva varios retos. Algunos de estos son los siguientes [20]:

- **Heterogeneidad.** Los dispositivos IoT pueden tener diferentes características, funcionalidades, condiciones de operación. Por ello, habilitar una manera de integrar varios dispositivos de manera eficiente supone un gran reto. Este reto aumenta de magnitud si el número de dispositivos es grande y están organizados en redes complejas.
- **Escalabilidad.** Con el rápido crecimiento de la utilización de los sistemas IoT, aumenta el número de dispositivos y con ello la cantidad de datos producido. Gestionar esta cantidad de dispositivos y datos es otro reto de los sistemas IoT.
- **Seguridad o Privacidad.** Debido al gran número de dispositivos y su heterogeneidad garantizar la autenticación, la consistencia y protección de los datos es un problema principal de los sistemas IoT.
- **Posibilidad para reconfigurar el dispositivo dinámicamente sin apagar el dispositivo**. Aunque los estándares ofrecen muchas opciones para configurar los protocolos de red, es casi imposible cambiar la configuración en tiempo de ejecución. Esto implica que toda la red necesita reconfigurarse fuera de línea si se requieren cambios [7].

Para proporcionar una solución a algunos de estos retos se han desarrollado multitud de plataformas IoT. Estas plataformas son el conjunto del software y la infraestructura de servicios que sirven para dar soporte a la integración y la lógica de negocio del sistema.

No hay una estructura única y común para las plataformas IoT, estas pueden estar formadas por diferentes arquitecturas y módulos con funcionalidades muy diversas.

- **El módulo de conectividad**. Hace posible que los diferentes dispositivos establezcan una comunicación con la plataforma. Dentro de esta conexión se podrían llegar a aplicar mecanismos de normalización o procesamiento de datos.
- **El módulo de gestión de dispositivos.** Es el registro central de los dispositivos conectados a la plataforma. Desde este módulo se suelen controlar aspectos de configuración, seguridad, incluso de actualización remota.
- **Gestión de datos y almacenamiento.** La mayoría de las plataformas están provistas de un sistema de almacenamiento de datos potente y escalable, dado que en muchos casos gestionan grandes cantidades de información provistas por los dispositivos.
- **Módulos de funcionalidad o servicio**. Los más comunes son la definición y ejecución de reglas de negocio (basado en eventos, dispositivos, datos, etc.), sistemas de análisis e inteligencia artificial, visualización de datos, los sistemas para el prototipado o la ejecución de aplicaciones.
- **Habilitación de aplicaciones externas.** Muchas plataformas implementan APIS y SDKS para que sus servicios puedan ser utilizados en aplicaciones de terceros.

Algunos ejemplos de plataformas IoT son las siguientes:

- **ScriptIoT** [21] tiene como uno de sus objetivos que usuarios con poca experiencia en el desarrollo de aplicaciones IoT puedan crearlas sin demasiado esfuerzo. Esta solución propone un lenguaje de script para definir la lógica del sistema. La solución destaca que los sistemas creados poseen un buen rendimiento, siendo este el principal factor que se considera en la evaluación del sistema. La propuesta no contempla entre sus características mecanismos para simplificar la implementación de la lógica de los dispositivos o su posterior actualización. La lógica de los dispositivos y la lógica general se implementan en partes muy diferentes del sistema, lo que implica el uso de un lenguaje para los dispositivos y el uso del lenguaje de script propuesto para la lógica del sistema.
- **Giang** [22] examina el desarrollo de aplicaciones IoT desde la perspectiva de la computación en la niebla «Fog Computing». Existen múltiples definiciones de Fog computing; se trata un paradigma horizontal de recursos físicos y/o virtuales que van desde donde se crean los datos hasta donde se almacenan y utilizan. La solución propuesta se basa en una plataforma que permite especificar la lógica central de sistemas IoT mediante el lenguaje gráfico Node-Red. También se proporcionan mecanismos para poder conectar e intercambiar datos entre dispositivos heterogéneos con diferentes protocolos de aplicación. Al igual que en ScriptIoT no se contempla simplificar la implementación de la lógica de negocio de los dispositivos.
- **Mainetti** [23] propone una plataforma que da soporte a las principales necesidades de la creación de aplicaciones IoT utilizando un lenguaje visual en el editor ClickScript. Este lenguaje no cuenta con excesiva expresividad, permite sobre todo la creación de sistemas centrados en la visualización de datos. A partir de la especificación realizada con el lenguaje se genera una aplicación web.

- **Wot Architecture**[8] es una arquitectura para la construcción de sistemas IoT que se encuentra en la fase de candidata a recomendación en el W3C. Uno de los principales puntos fuertes de la propuesta es la abstracción de los elementos físicos y virtuales que forman el sistema en «cosas WoT - Wot Thing». Estas «cosas» se modelan utilizando metadatos normalizados, de forma que se establece claramente cómo procesarlas y representarlas. Gracias al formato normalizado, se proponen mecanismos de representación directa de las «cosas Wot» a través de HTML y otros lenguajes. El objetivo más ambicioso es promover la estandarización del mecanismo de modelado, de esta forma los servicios o usuarios consumidores pueden entender rápidamente la representación de los elementos.

Además de las plataformas propuestas en los trabajos de investigación, hoy en día existen, varias plataformas IoT comerciales entre las que se encuentran las siguientes:

- **Microsoft Azure IoT Central**[9]**.** Dispone de un gran conjunto de soluciones y herramientas para la creación de sistemas IoT. Entre sus características, Microsoft destaca la posibilidad de crear sistemas IoT completos en poco tiempo, facilitando la conexión de dispositivos, la supervisión y la administración de múltiples recursos IoT. Para conectar un dispositivo a la plataforma, el desarrollador debe crear el programa que define la lógica del dispositivo y la conexión con Azure. La plataforma dispone de algunas librerías que permiten simplificar las labores de conexión. Sus principales puntos fuertes son la escalabilidad, la seguridad y un acuerdo a nivel de servicio 99,9.
- **Google Cloud IoT Core**[10]**.** Se encuentra entre las plataformas que más funcionalidad y servicios ofrecen. Entre sus muchas características se incluyen la gestión y configuración de dispositivos conectados, gateways y la traducción de protocolos. Los dispositivos conectados a la plataforma pueden ser gestionados de múltiples maneras, se incluyen mecanismos para gestionar las actualizaciones OTA (si estas son soportadas por el dispositivo). El programa que ejecuta el dispositivo debe implementar la conexión con la plataforma, para ello se dispone de varias alternativas, siendo Google Pub/Sub una de las más populares.
- **Amazon AWS IoT**[11]**.** dispone de un gran catálogo de funcionalidades, entre las que se incluye la gestión de dispositivos, la gestión de datos y diversos mecanismos para especificar la lógica central del sistema. El desarrollador debe implementar la lógica de negocio y las comunicaciones de cada dispositivo que desee incluir en el sistema. Amazon provee librerías específicas que permiten simplificar la labor de comunicación con su plataforma IoT.
- **Cloudino**[12]**.** Ofrece una librería para simplificar la implementación de las labores de comunicación del dispositivo con la plataforma. La solución está destinada a dispositivos Arduino y promueve el uso de un módulo de comunicación inalámbrica que la propia empresa suministra. Cuando los dispositivos son conectados a la plataforma, se ofrecen numerosas posibilidades de gestión, entre ellas, la posibilidad de gestionar de forma sencilla las actualizaciones OTA. Permite especificar el nuevo programa de tres formas diferentes: usando el lenguaje nativo de Arduino, un lenguaje gráfico y una versión del framework de Arduino en JavaScript.

[8] Web de la recomendación https://www.w3.org/TR/wot-architecture/#sec-wot-architecture
[9] Web oficial https://azure.microsoft.com/es-es/services/iot-central/
[10] Web oficial https://cloud.google.com/iot-core/
[11] Web oficial https://aws.amazon.com/es/iot-core/
[12] Web oficial http://www.cloudino.io/

- **Thethings.io**[13]**.** Cubre muchos de los aspectos relativos a la creación de sistemas IoT. Se presenta como una de las plataformas más simples para la creación de sistemas IoT. Los desarrolladores son los encargados de implementar el programa que conecta los dispositivos a la plataforma. La plataforma gestiona los dispositivos conectados a ella y todos los datos que producen o consumen y también contempla la gestión de actualizaciones remotas OTA (si el dispositivo las soporta).

3.4. Importancia del prototipado y coste de desarrollo

La importancia del prototipado y desarrollo rápido en los sistemas IoT es algo que está muy presente en los trabajos de investigación. Existen multitud de propuestas orientadas principalmente al prototipado y desarrollo rápido en sistemas IoT [3][24][25][26].

El prototipado y desarrollo rápido también son características que a menudo se intentan destacar en el sector comercial, tanto en las plataformas IoT como en el desarrollo de software para los propios dispositivos.

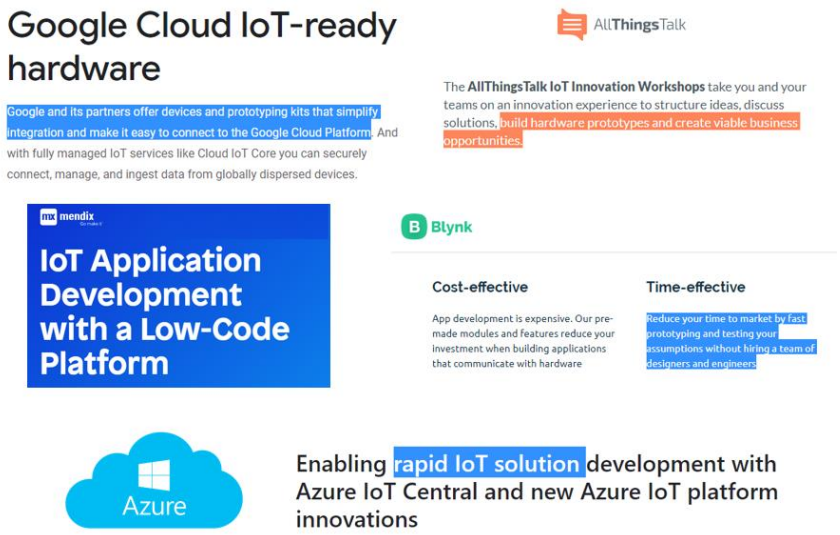

Figura 2. Plataformas IoT comerciales resaltando su capacidad para el prototipado y desarrollo rápido

Varios trabajos de investigación tratan de evaluar los aspectos que contribuyen a facilitar el desarrollo de un sistema IoT. Patel [27] realiza una revisión de las características de los sistemas IoT y de cómo estas afectan a su complejidad de desarrollo. Algunas de las características que se señalan son las siguientes:

- **El objetivo**: solo sensores, o sensores y actuadores.
- **El dinamismo de la topología**: estática con dispositivos fijos, o dinámica, donde la cantidad de dispositivos puede variar.
- **La escala**: número de dispositivos implicados en el sistema.
- **La heterogeneidad**: grado de similitud de los dispositivos que componen el sistema, no solo desde el punto de vista del tipo de dispositivo, sino desde a la lógica de negocio. Una red de dispositivos es totalmente homogénea si son del mismo tipo y ejecutan el mismo software, ese se considera el caso más sencillo.

[13] Web oficial https://thethings.io/

- **Modo de interacción**: el modo de iteración puede ser continuo, dirigido por eventos, o bajo demanda (recibe peticiones para realizar acciones).
- **Lógica de la aplicación**: la lógica es compuesta cuando se sintoniza información de varios dispositivos diferentes; es lógica simple cuando todos los dispositivos son homogéneos.

La mayor parte de estas características dependen en gran parte del diseño que se realice del sistema. En algunos casos, en la fase de diseño se podrá elegir la alternativa que permita obtener un desarrollo más ágil. En muchos casos, la elección en el diseño está condicionada por requerimientos de algún tipo (funcionalidad, dispositivo, arquitectura, protocolo, etc.) o aspectos a priorizar [28] como, la seguridad, el consumo de energía, etc.

Existen numerosas investigaciones que proponen diferentes soluciones relativas al prototipado o desarrollo rápido de sistemas IoT. Algunos ejemplos son los siguientes:

- **Datta** [29] propone un framework para la creación de sistemas IoT que permite reducir el tiempo del ciclo de desarrollo. La API propuesta unifica el acceso a la mayor parte de funcionalidades comunes de los sistemas IoT (por ejemplo, registro y gestión de dispositivos, seguridad, mecanismos de publicación-suscripción basados en diferentes protocolos de aplicación, procesamiento de datos, etc.). Según el estudio, parece que utilizar la API propuesta podría reducir hasta en un 72% el número de líneas de código en comparación con una implementación manual. La propuesta no ofrece mecanismos propios para simplificar la implementación de la lógica de los dispositivos o su posterior actualización, dejando que estas labores recaigan sobre los programadores. La lógica de los dispositivos y la lógica general se implementan en partes muy diferentes del sistema, lo que también puede implicar el uso de diferentes lenguajes de programación y frameworks.
- **Lee** [3] propone un sistema de prototipado de aplicaciones IoT. La propuesta plantea abstraer la lógica de negocio de los dispositivos, de forma que pueda ser escrita una única vez incluso para dispositivos de diferentes tipos. Propone una extensión del lenguaje C++ para la programación de los dispositivos. La programación agnóstica de los dispositivos es una técnica que se propone en otras investigaciones y soluciones comerciales (como los contenedores de Zerynth[14] o Balena.io[15]). Se trata de una técnica efectiva para agilizar el desarrollo del software de los dispositivos y facilitar la reutilización. La parte negativa es que no está soportada por muchos de los dispositivos comúnmente usados, sobre todo por aquellos pequeños, de bajo coste y menos potentes.
- **OpenWINO** [30] está compuesto por un hardware específico y una plataforma IoT. El dispositivo Wino tiene pequeñas dimensiones y bajos consumos de energía; el chip de la placa soporta la integración con el ecosistema Arduino. Una de las principales ventajas del dispositivo es que resulta posible cambiar de forma rápida el hardware del transceptor de radio (WiFi, Bluetooth BLE, Zigbee, NFC, etc.) y la librería software que le da soporte. Las librerías que dan soporte a la comunicación del dispositivo tienen un gran nivel de abstracción, por lo que pueden simplificar el desarrollo. La propuesta se centra únicamente en la parte del dispositivo, no hace referencia a la construcción del resto del sistema IoT.
- **Patel** [25] propone un sistema de alto nivel para habilitar la creación de aplicaciones IoT. El enfoque propuesto separa la lógica de negocio de los conceptos técnicos relacionados

[14] Web oficial https://www.zerynth.com/
[15] Web oficial https://www.balena.io/

con IoT. Utiliza un framework de macro programación basado en ingeniería dirigida por modelos y combinado con generación automática de código. El enfoque propuesto permite que las aplicaciones puedan ser especificadas de forma muy rápida, pero las posibilidades de definir la lógica compleja se reducen enormemente. Su principal innovación consiste en la separación de la lógica del sistema en roles o perfiles, de forma que personal con diferentes perfiles pueda trabajar de forma simultánea en la definición del sistema.

- **NB-IoTalk** [31] es una solución para el desarrollo rápido de aplicaciones IoT. La plataforma está centrada principalmente en la creación de aplicaciones IoT que tienen que ver con la geolocalización. La solución incluye todo lo necesario para interconectar dispositivos por medio de un sistema de publicación/suscripción. Ofrece la posibilidad de crear aplicaciones geoespaciales de forma rápida sobre su capa superior. Uno de los aspectos más innovadores de la solución es la gestión interna de los dispositivos que realiza la plataforma, dividiéndolos en unidades mínimas, por cada sensor o actuador. Combina esta división con un mecanismo de etiquetado para definir (identificación, datos geográficos, tiempo, batería y nivel de privacidad). Este modelo ayuda a que la aplicación pueda simplificar la identificación y situación de los dispositivos, sobre todo en aquellos sistemas que manejan gran cantidad de dispositivos homogéneos.

3.5. Actualización del software del dispositivo

Otro aspecto importante para el mantenimiento de los sistemas IoT en ejecución es la actualización del software del dispositivo. Las actualizaciones pueden ser necesarias por motivos diferentes, desde la corrección de bugs hasta cambios en la funcionalidad del software.

En algunos sistemas IoT las actualizaciones manuales en las que se conecta directamente con el dispositivo para cambiar el software son poco prácticas. Los dispositivos pueden estar colocados en lugares poco accesibles (por ejemplo, en el tejado) o el sistema puede estar compuesto de un número elevado de dispositivos. Los mecanismos de actualización remota de dispositivos son una solución para este problema. Uno de los mecanismos comunes para las actualizaciones remotas en los dispositivos IoT son las actualizaciones OTA.

La actualización OTA consiste en enviar de forma remota una nueva versión del software. Este nuevo software debe reemplazar la versión actual. La mayoría de los dispositivos IoT con cierta capacidad informática pueden usar sistemas operativos que permiten este tipo de actualización. Debido a que en la actualización se envía el programa nuevo en su totalidad este tipo de actualizaciones puede tener algunas inconvenientes como alto tiempo de descarga o alto consumo de energía. Por otro lado, el hecho de que se tenga que sustituir el programa en ejecución por uno nuevo provoca que haya un tiempo en el que el dispositivo está inactivo.

Varios trabajos de investigación proponen variaciones a las actualizaciones clásicas para resolver o minimizar estos problemas. Existen diferentes enfoques, algunos más cercanos las OTA clásicas mientras que otras permitiendo realizar modificaciones remotas de software diferentes centradas en aspectos concretos.

- **Chandra** [32] propone un sistema de actualizaciones OTA basadas en el protocolo Lightweight Mesh (LWMech) sobre una arquitectura P2P sobre topología de malla. La contribución principal es disminuir el consumo de energía mediante la utilización de un protocolo ligero y el envío de datos eficiente al utilizar una arquitectura P2P sobre una topología de malla que elimina tráfico redundante.
- **Chang** [12] propone un modelo de actualizaciones de software dinámicas para no perder el estado de la ejecución del dispositivo a la hora de aplicar la actualización. Para habilitar

las actualizaciones dinámicas las funcionalidades básicas se implementan como tareas que se manejan por un ligero motor de control de flujo.

- **Mukhtar** [10] presenta un mecanismo para realizar las actualizaciones de manera remota. Se centra en la transmisión y el uso eficiente de la memoria. Su mecanismo se basa en movimientos de memoria para solamente enviar las diferencias entre el programa del dispositivo y el nuevo programa. Utilizando su propuesta no se necesita guardar las imágenes de los programas en buffers lo que disminuye el use de la memoria.

3.6. Soluciones basadas en un proxy

Existen muchas soluciones basadas en un uso de algún elemento intermedio que haga la función del proxy entre los dispositivos IoT y un servicio centralizado. Varias de las plataformas IoT explicadas en apartados anteriores utilizan un proxy que hace de puerta de enlace con los dispositivos IoT. El uso de un proxy puede facilitar la realización de diferentes tareas tales como como la conexión de dispositivos, la supervisión y la administración de múltiples recursos IoT, etc.

Sin embargo, el uso de proxies puede ser aplicado con diferentes propositos. Existen investigaciones que utilizan este enfoque con otros objetivos. Algunos ejemplos son los siguientes:

- **PRTA** [33] es una propuesta que utiliza un proxy con el objetivo de proporcionar seguridad adicional al sistema. Este trabajo se basa en el uso de reencriptación del proxy (PRE) donde un servidor en la nube se encarga de almacenar y reencriptar reduciendo el coste de los dispositivos. Todos los dispositivos están bajo el control del servidor de autorización que ayuda a garantizar seguridad y fiabilidad de los datos.
- **Jin** [13] propone un sistema IoT basado en proxy para facilitar la gestión de diversos dispositivos IoT heterogéneos. El sistema se compone de tres capas: proveedor de servicios que proporciona unos servicios comunes (registro, gestión, monitoreo, etc), capa del proxy para homogeneizar el acceso a los dispositivos y la capa de los dispositivos IoT.
- **Banaie** [34] propone una política de cache basada en proxy. Los dispositivos actualizan el registro de la cache enviando peticiones POST. Si la petición del usuario que accede al proxy detecta que este ha superado el tiempo de vida, el propio proxy consulta a los dispositivos para actualizar sus registros.

3.7. Comandos y configuración remotos

Existen soluciones comerciales que permiten configuración remota. Algunas soluciones como Firebase Remote Config [35] permiten configurar parámetros específicos de la aplicación en la nube y ofrecen una API para actualizar esos parámetros dentro de la aplicación si los parámetros configurados se modifican.

Otras soluciones poseen un sistema operativo o firmware base admiten comandos de configuración, sobre todo de administración remota. Algunos ejemplos podrían ser:

- SD Remote Command[16] es parte de Suse Manager diseñado para ayudar a los equipos de DevOps y Operaciones de TI a reducir la complejidad de gestionar los activos de TI con una sola herramienta para administrar sistemas Linux en una variedad de arquitecturas de

[16] https://documentation.suse.com/external-tree/en-us/suma/4.0/suse-manager/reference/systems/system-details/sd-remote-command.html

hardware, hipervisores, así como plataformas de contenedores, IoT y en la nube. Permite ejecutar comandos remotos en el sistema seleccionado.

- Dell Comand[17]. Este producto consta de una interfaz de línea de comandos (CLI) y una interfaz gráfica de usuario (GUI) para configurar varias funciones de la BIOS.

- Comandos remotos de dispositivos Cisco[18]. Estos dispositivos permiten cambiar la configuración existente y guardarla de forma que se mantenga después de un reinicio.

Los comandos AT son utilizados como una interfaz software para módulos inalámbricos. Los comandos AT se definen como parte del estándar 3GPP bajo 3GPP TS 27.007. Eso implica que todos los módulos inalámbricos que operan en redes celulares son necesarios para admitir comandos AT. Estos comandos proporcionan una interfaz para interactuar con el módulo de comunicación para realizar multitud de tareas, como obtener información del dispositivo / fabricante, enviar/recibir llamadas telefónicas, iniciar llamadas de datos, etc [36][37].

Además de las soluciones comerciales existen también trabajos de investigación que utilizan la configuración o control remoto, por ejemplo:

- **Barron-Gonzalez** [38] propone una extensión del servicio de control remoto y configuración capaz de estandarizar un procedimiento general dentro de la rama más nueva del estándar ISO/IEEE11073 llamada X73 para dispositivos de salud personal (X73PHD).
- **Nikolaidis** [39] presenta una solución para realizar la configuración remota de dispositivos IoT centrados en un entorno de hogar.
- **Seo** [40] presenta usa solución para el control remoto de un robot de propósito militar. La propuesta permite controlar el movimiento del robot así como manipulación del equipo adicional como detector de minas.

Los sistemas de configuración remota vía comandos se suelen utilizar directamente sobre el sistema operativo. No se ha encontrado ninguna solución que facilite un mecanismo para que los desarrolladores puedan actualizar vía comandos remotos algunas características relevantes de las aplicaciones software que desarrollan, entre estas características se podrían encontrar aspectos relativos a la comunicación.

[17] https://www.dell.com/support/article/es-es/sln311302/dell-command-configure?lang=en
[18] https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus5000/sw/configuration/guide/cli/CLIConfigurationGuide/begin.html

4. Propuesta Flex-Request

Este trabajo de investigación propone una novedosa plataforma, basada en el uso de un servicio centralizado de gestión de configuraciones de comunicación y una librería de comunicación con capacidad de configuración remota. La librería se integra en el desarrollo del software que se ejecuta en los dispositivos IoT y gestiona la comunicación entre propio dispositivo y otros dispositivos y/o servicios. Esta librería permite modificar dinámicamente algunos aspectos de la comunicación del software ejecutado en el dispositivo, sin la necesidad de modificar el código del programa y sin utilizar ningún componente intermedio (como proxy). Esta librería se puede usar como una librería estándar para enviar peticiones a través de Internet, pero también permite a los desarrolladores cambiar de forma rápida y remota algunos de los aspectos de la comunicación:

- **Cambiar el de destino de las peticiones**. En la mayor parte de sistemas IoT los dispositivos envían datos a servicios o a otros dispositivos. La capacidad de comunicación es una parte fundamental de los dispositivos IoT. Normalmente cada dispositivo conoce los datos del servicio o dispositivo con el que debe comunicarse, por ejemplo, su IP. Sí en algún momento se quiere cambiar de servicio o la estructura de comunicación entre los dispositivos puede ser necesario acceder al software de dispositivo y cambiar el destino de las peticiones.
- **Reenviar las peticiones a múltiples servicios o dispositivos, además del destino principal.** En ocasiones tanto en fase de prototipado, desarrollo o mantenimiento puede ser necesario evaluar o comparar el uso de diferentes servicios (por ejemplo, dos plataformas de cloud computing de almacenamiento y análisis de datos), el envío múltiple de la misma información permite realizar de forma sencilla esta evaluación con costes muy reducidos para el desarrollador. Este tipo de envíos también permiten duplicar de forma muy rápida datos en servicios de pruebas o prototipos. Los patrones de comunicación basados en un dispositivo que se comunica con 1-N dispositivos son comunes en algunos sistemas IoT poco centralizados, donde cada dispositivo puede comunicarse con 1-N dispositivos.
- **Cambiar la frecuencia de envío de las peticiones.** La frecuencia de las comunicaciones entre un dispositivo y un servicio u otro dispositivo puede variar enormemente dependiendo de los objetivos del sistema. Algunos sistemas tienden a minimizar estos envíos para reducir el tráfico de red o el consumo de energía de los dispositivos. En cambio, en otros sistemas el envío se realiza de forma muy frecuente ya que manejan datos que deben ser procesados en tiempo real. La frecuencia ideal de envío de datos para un mismo sistema no tiene por qué ser constante, puede requerirse un cambio de la frecuencia de envíos por múltiples motivos, sobrecargas de servicio, inoperatividad de alguna de las partes del sistema, factores ambientales que modifican puntualmente las mediciones, condiciones externas o físicas no previstas, etc.

La solución propuesta busca que sea posible realizar cambios como los citados anteriormente de forma rápida y sin implicar modificaciones del software, ejecución de un nuevo programa o reinicio del dispositivo.

4.1. Arquitectura

La plataforma propuesta tiene dos partes. La parte principal es la **librería** de comunicación que se utiliza dentro del programa del dispositivo y la otra es el **servidor de configuración**.

Para la comunicación de los cambios entre los desarrolladores, el servidor de configuración y los dispositivos, se utiliza el protocolo MQTT. Message Queuing Telemetry Transport (del inglés

Transporte de telemetría de colas de mensajes) es un protocolo de red abierto, OASIS e ISO (ISO/IEC 20922) ligero, de publicación y suscripción que transporta mensajes entre dispositivos. Está diseñado para conexiones con ubicaciones remotas donde se requiere una "huella de código pequeña" o el ancho de banda de la red es limitado [41].

Debido al protocolo utilizado, el servidor de configuración es un agente MQTT que expone un *topic* en el que se pueden publicar las actualizaciones de configuración. Los dispositivos que tienen cargado un programa que utiliza la librería, se conectan a este *topic* y se quedan escuchando, esperando a que se publiquen cambios de configuración.

La librería propuesta se utiliza en los programas ejecutados en los dispositivos IoT para realizar todas las comunicaciones con otros dispositivos y/o servicios. Cuando se inicia el programa, la librería hace dos tares principales:

- En primer plano, realiza las tareas de comunicación que ha programado el desarrollador, por ejemplo enviar un dato procedente de un sensor a un servicio centralizado para que sea almacenado.
- En segundo plano, se conecta al **servidor de configuración** y permanece a la escucha de instrucciones para realizar cambios en la comunicación. Este proceso es automático e imperceptible para el desarrollador. Cuando se detecta una orden de cambio en la comunicación, se procesa la orden para determinar la naturaleza del cambio. Una vez procesado se realiza de forma automática un cambio en la configuración interna de la librería de forma que se modifica la comunicación actual.

En la figura 1 se muestra el esquema de comunicación entre la librería y el servidor de configuración.

1) Los dispositivos IoT ejecutan un programa que utiliza la librería para enviar los datos capturados al Servicio 1. La librería se conecta al **servidor de configuración** y escucha cambios.
2) Un desarrollador publica un cambio al servidor de configuración para que los dispositivos IoT pasen a enviar los datos al Servicio 2.
3) La librería dentro de los dispositivos IoT detecta el cambio de configuración que el desarrollador envió, procesa el comando y aplica el cambio de comunicación.
4) Una vez se haya procesado el comando recibido por los dispositivos, los dispositivos comienzan a enviar los datos capturados al Servicio 2.

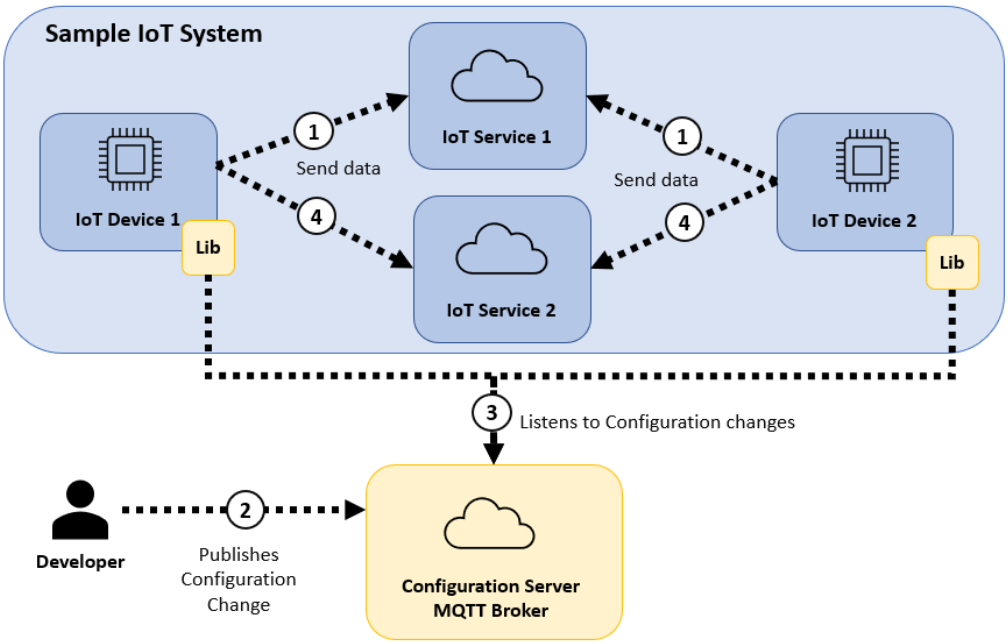

Figura 3. Esquema conceptual de la propuesta

4.2. Librería de comunicación

Uno de los objetivos de esta investigación es no aumentar la complejidad de desarrollo respecto a un desarrollo normal de un software para un dispositivo IoT. Por ello el diseño de la librería propuesta se ha basado en el estándar WebAPI [41].

La librería propuesta define un método *request* que acepta los siguientes argumentos:

- **url** *[opcional]*: url o IP del servidor de destino. Debe ser una URL bien formada incluyendo el protocolo. Cuando se pasa a la llamada, se parsea con URL de WebAPI [9] y se sobrescribe con las propiedades establecidas explícitamente en el objeto de *options*.
- **options** *[obligatorio]*: objeto con los siguientes parámetros de la petición:
  - **id** *[obligatorio]*: id de la petición. Se usa para diferenciar los diferentes usos de la librería dentro del mismo programa.
  - **data** *[obligatorio]*: función asíncrona que devuelve los datos que se enviarán en la petición.
  - **host** *[opcional, por defecto "localhost"]*: host del servidor de destino.
  - **protocol** *[opcional, por defecto "http:"]*: protocolo utilizado.
  - **method** *[opcional, por defecto "GET"]* - método http utilizado.
  - **port** *[opcional, por defecto 80]*: puerto de destino.
  - **requestFrecuency** *[opcional, por defecto 1000]* - frecuencia (en milisegundos) con la que se envían las peticiones. Es un mecanismo interno de la librería; los programadores no necesitan poner el método dentro de un bucle u otro mecanismo de iteración para que las peticiones se envíen continuamente con la frecuencia establecida.
  - **forwardUrls** *[opcional, por defecto un array vacío]*: array de URLs a las que se reenviarán las peticiones.
- **Función de procesamiento de respuesta** *[obligatorio]*: función que se ejecutará cuando se reciba la respuesta.

Para usar la librería, los desarrolladores deben incluirla dentro del programa. A continuación, los desarrolladores tienen que configurar la comunicación con el servidor de configuración para todo el programa indicando la URL en la que está el servidor de configuración. Una vez realizadas estas acciones, ya se puede usar el método ofrecido por la librería para enviar peticiones a través de Internet. Cada uso del método de *request* debe identificarse con un **identificador de la petición** para permitir la modificación independiente de cada uso. Si dentro de un mismo programa se utiliza la librería 2 veces, pero se quiere poder modificar cada uno por separado, al tener distintos identificadores, se podrán enviar comandados con identificadores diferentes y aplicar cambios sobre uno u otro.

A modo de prueba para validar conceptualmente la propuesta se ha realizado una implementación de la librería en Node.js. La figura 4 representa un ejemplo de un programa que utiliza esta implementación de la librería propuesta. El programa de ejemplo realiza lo siguiente:

- Línea 1 – se añade al programa la dependencia necesaria para leer datos del sensor.
- Línea 2 – se añade al programa la implementación de la librería.
- Líneas 3-5 – se configura la dirección del servidor de configuración.
- Líneas 8-27 – se realiza la primera petición.
    - Líneas 9-20 – objeto con las opciones de la petición.
    - Líneas 21-26 – función de procesamiento de la respuesta.
- Líneas 30-49 – se realiza la segunda petición a otro servicio.
    - Líneas 31-42 – objeto con las opciones de la petición.
    - Líneas 43-47 – función de procesamiento de la respuesta.

```
01: const sl = require("node-dht-sensor");
02: const FlexRequest = require("flex-request");
03: const factory = FlexRequest({
04:   masterServerUrl: "mqtt://192.168.0.27:5000",
05: });
06:
07: // Sending the temperature
08: factory.request(
09:   {
10:     id: 1,
11:     protocol: "http:",
12:     host: "192.168.0.27",
13:     port: 6001,
14:     data: new Promise((resolve, reject) => {
15:       resolve({
16:         temperature: sl.read(11, 4).temperature,
17:       });
18:     }),
19:     method: "POST",
20:   },
21:   (err, res, body) => {
22:     if (err) {
23:       return console.log(`Error: ${err.message}`);
24:     }
25:     console.log(`${body}`);
26:   }
27: );
```

```
28:
29: // Sending the humidity to another server
30: factory.request(
31:   {
32:     id: 2,
33:     protocol: "http:",
34:     host: "158.54.0.100",
35:     port: 7000,
36:     data: new Promise((resolve, reject) => {
37:       resolve({
38:         humidity: sl.read(11, 4).humidity,
39:       });
40:     }),
41:     method: "POST",
42:   },
43:   (err, res, body) => {
44:     if (err) {
45:       return console.log(`Error: ${err.message}`);
46:     }
47:     console.log(`${body}`);
48:   }
49: );
```

Figura 4. Ejemplo de programa utilizando la propuesta

La figura 5 representa un diagrama de secuencia más detallado de un ejemplo de uso. Al iniciarse este programa hará dos cosas. En segundo plano, abrirá una conexión con el servidor (*connect*) de configuración mediante el protocolo MQTT y se suscribirá para escuchar a cambios de configuración (*subscribe*). En primer plano, leerá la temperatura del sensor y la enviará a http://192.168.0.27:6001 (*request con id=1*) y también leerá la humedad del sensor y la enviará a http://158.54.0.100:7000 (*request con id=2*); ambos se enviarán cada segundo. Cuando se recibe una respuesta, se registrará en la consola. Si un desarrollador envía un el comando "`set 1 160.155.0.200:8000`" al servidor de configuración (*publish*), la librería detecta este cambio, analiza el comando recibido (*parseCommand*) y aplicará el cambio (*applyChange*). Cada método *request* cuyo identificador es igual al recibido en el comando cambiará su configuración de comunicación. En este ejemplo, el método con id=1 pasa a enviar los datos a http://160.155.0.200:8000. De esa manera, podemos tener muchos dispositivos en los que el método *request* tiene el mismo identificador, y cuando se realiza el cambio de configuración con un identificador específico, todos los dispositivos que envían peticiones con el identificador utilizado cambiarán su configuración.

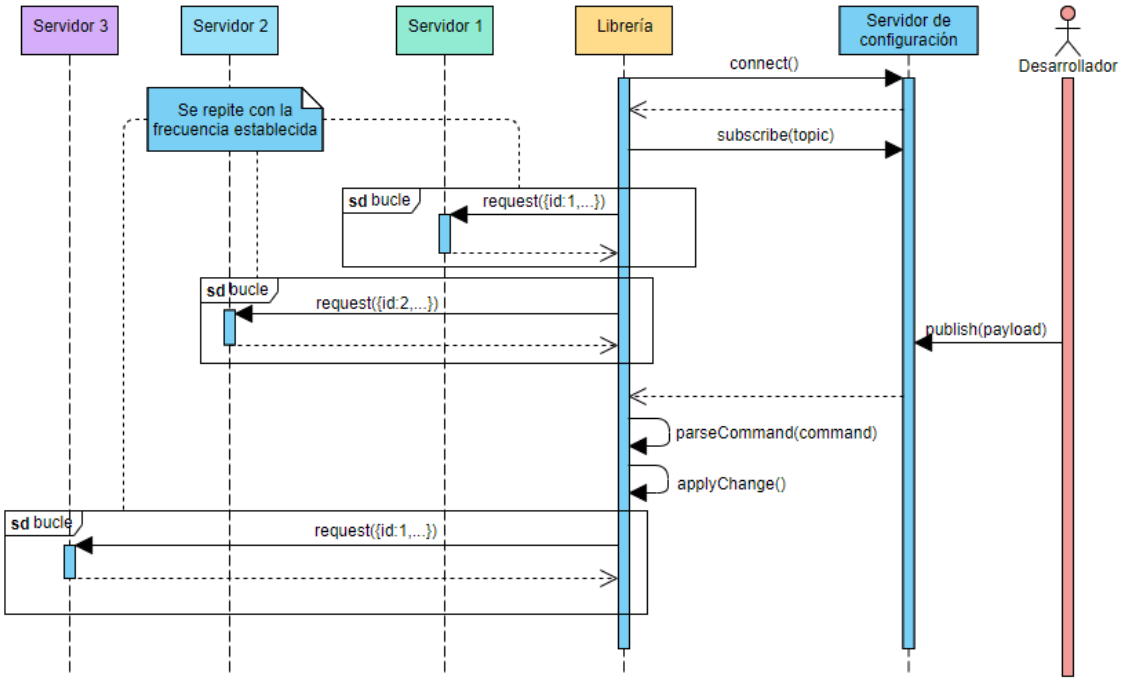

Figura 5. Diagrama de secuencia de un ejemplo de uso

Para cambiar un parámetro de comunicación, el desarrollador o administrador del sistema IoT debe enviar una petición MQTT al servidor de configuración con un comando específico que habilite el cambio requerido. En la experimentación realizada en esta investigación se ha enviado la petición mediante la ejecución de un script desde la consola. Sin embargo, para sistemas en producción se podría crear una consola de administración con una interfaz web que facilitaría estas acciones. El servidor de configuración publica el comando en todos los dispositivos que están escuchando. Para cubrir los cambios de comunicación, se han diseñado los siguientes comandos:

- *set <id> <destinationURL>*: cambia la URL del servidor de destino de la petición con identificador igual al del argumento *<id>*. Ese comando puede ser útil para cambiar el servidor al que se envían los datos en un esquema Cliente-Servidor porque se ha decidido cambiar el servidor actual a uno nuevo o cambiar el dispositivo con el que se realiza la comunicación en un esquema de malla de varios dispositivos IoT.
- *set master-server <serverURL_1> <serverURL_2> ... <serverURL_N>* se utiliza parea para cambiar la dirección del servidor de configuración. Se ofrece la posibilidad de añadir N direcciones para escuchar cambios de comandos de varios servidores.
- *forward <id> <forwardURL_1> <forwardURL_2> ... <forwardURL_N>*: reenvía la petición con el identificador *<id>* a las URLs de los argumentos *<forwardURL>*. Puede tener tantas URL de reenvío como se desee, separadas por espacios. Este comando podría utilizarse, por ejemplo, si es necesario separar el procesamiento de datos en diferentes servidores o comunicar un mismo dispositivo con varios actuadores.
- *set <id> frequency <frequencyValue>*: cambia la frecuencia del envío de la petición al valor especificado en el argumento *<frequencyValue>* (en milisegundos). Este comando podría resultar útil para reducir el impacto en el tráfico de la red. Por ejemplo, puede no ser necesario enviar los datos con la misma frecuencia todo el tiempo. Puede haber momentos que requieran más precisión que otros.

Además de estos comandos principales se plantean dos modificadores siguientes que se pueden utilizar en cualquiera de los comandos anteriores:

- *wait <waitTime>:* hace que el cambio de configuración se aplique con un retardo especificado en el valor del argumento <*waitTime*> en milisegundos. Por ejemplo:

  set 1 139.23.23.48 wait 4000 cambiaría la dirección del servidor de destino 4 segundos más tarde.

- *start <dateTime>:* hace que el cambio de configuración se ejecute en la fecha indicada en el argumento <*dateTime*> con formato *MM/dd/yyyy_HH:mm*. Por ejemplo:

  set 1 139.23.23.48 start 07/30/2020_20:45 cambia el destino el día 30 de julio de 2020 a las 20:45.

Con el objetivo de evaluar la propuesta, se ha realizado una implementación de esta en Node.js que admite un conjunto de los comandos diseñados, suficientes para poder llevar a cabo y evaluar casos de usos significativos explicados a continuación.

5. Casos de uso

A continuación, presentamos algunos casos de uso de las principales funcionalidades de nuestra propuesta: cambiar la IP del servidor de destino, reenviar peticiones a otro servidor y cambiar la frecuencia de envío.

Para presentar los casos de uso, diseñamos un pequeño ejemplo con funcionalidad limitada, lo suficiente para ilustrar un posible uso y poder realizar posteriormente una evaluación. Este ejemplo simula el registro de la temperatura y la humedad dentro de un almacén. El sistema diseñado tiene varios dispositivos con conectividad a Internet que están equipados con sensores de temperatura y humedad. Esos dispositivos capturan la temperatura y la humedad en múltiples ubicaciones del almacén y envían los datos registrados a un servicio centralizado donde se almacenan y analizan. Esos análisis generados son utilizados por los gerentes de almacén para evaluar los cambios en la infraestructura, la posición de la maquinaria y las puertas, la ubicación del equipo de refrigeración, etc.

5.1. Caso de uso 1 - Cambiar el servidor de destino de los clientes

La compañía propietaria del almacén decide cambiar su servicio centralizado por el de otro proveedor de *cloud computing*. Ese cambio implica la reconfiguración de todos los dispositivos IoT que registran la temperatura y la humedad para enviar los datos al nuevo servicio. La compañía quiere probar varios proveedores de *cloud computing* de para elegir el que mejor cubra sus necesidades.

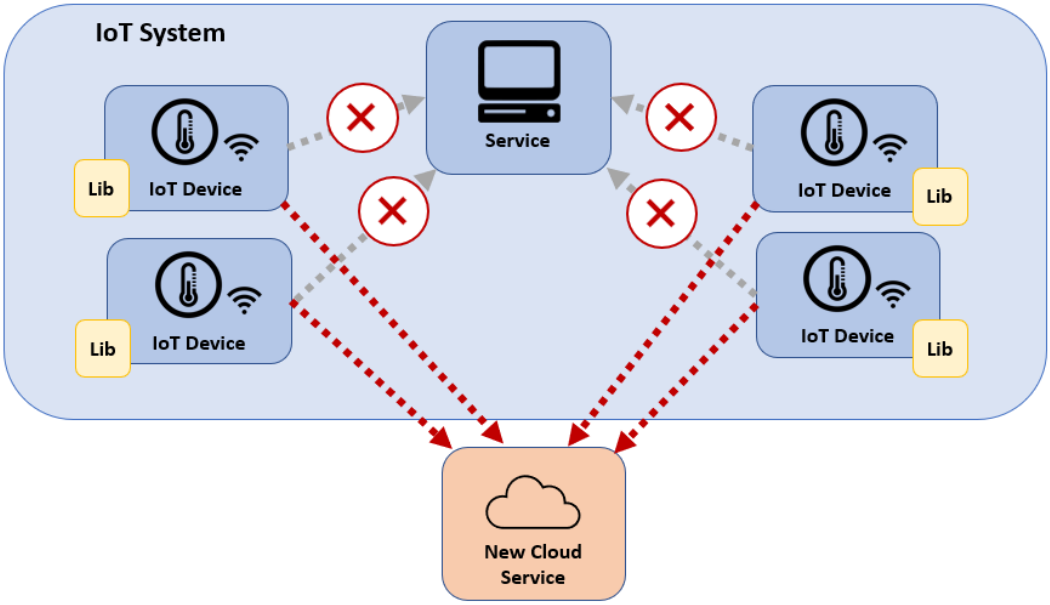

Figura 6. Esquema del cambiar el servidor de destino de los clientes

Una vez que el programa se carga en el dispositivo IoT y se ejecuta, comienza a enviar peticiones al servidor de destino. Para cambiar la IP de destino, debemos enviar una petición MQTT al servidor de configuración con el siguiente comando: *set 1 http://192.168.0.100:7071*. Cuando la librería que está escuchando los cambios de configuración detecta un nuevo cambio, actualiza la IP del servidor de destino y comienza a enviar datos al nuevo destino.

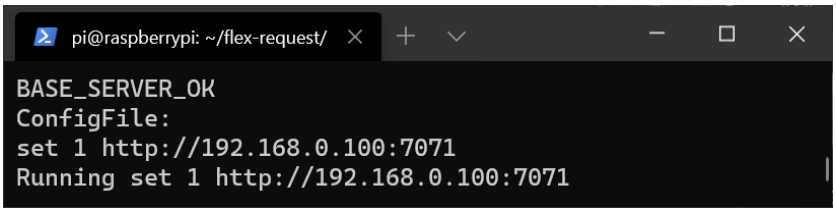

Figura 7. Cambio de configuración recibido y aplicado por la librería cargada en el dispositivo IoT

5.2. Caso de uso 2 - Reenviar las peticiones a otro servidor

En otro caso de uso, la compañía detecta un punto dentro del almacén donde la gestión de la temperatura es crítica. Para superar ese problema, se adquiere un nuevo dispositivo que puede administrar el aire acondicionado en función de la temperatura. Es necesario que los dispositivos que están cerca de ese punto crítico registren y envíen la temperatura a ese nuevo dispositivo, además de enviar los datos al servicio centralizado principal.

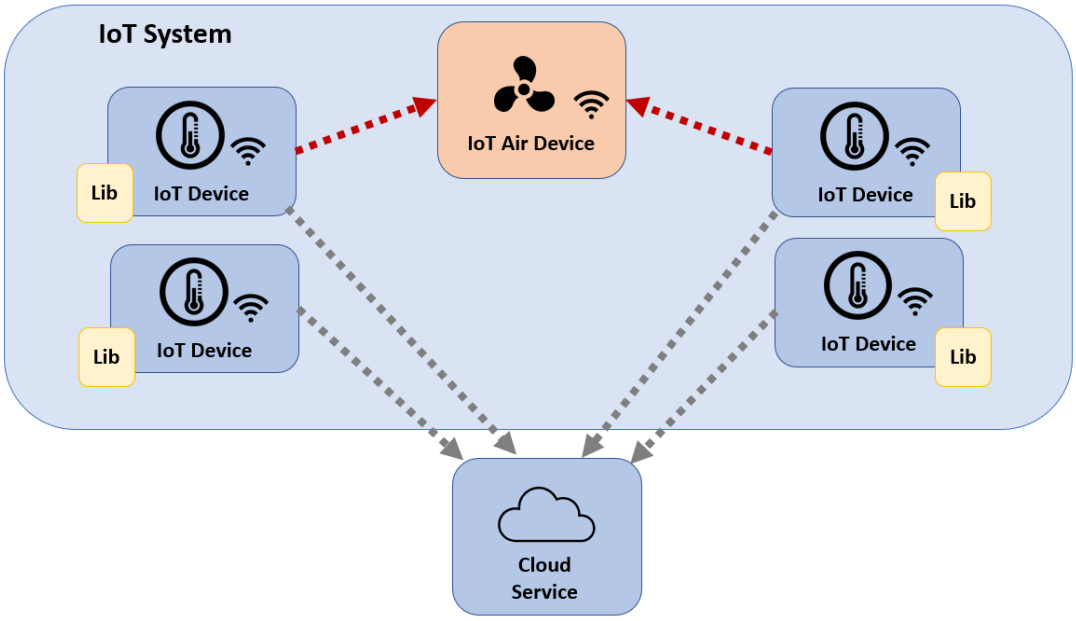

Figura 8. Esquema para reenviar peticiones a otros dispositivos

En este caso, para realizar ese cambio, se debe enviar el siguiente comando: *forward 1 http://192.168.0.200:4044*. Como en el caso de uso anterior, cuando la librería detecta una nueva configuración, guarda la dirección de reenvío localmente y comienza a reenviar cada petición enviada a esta nueva dirección.

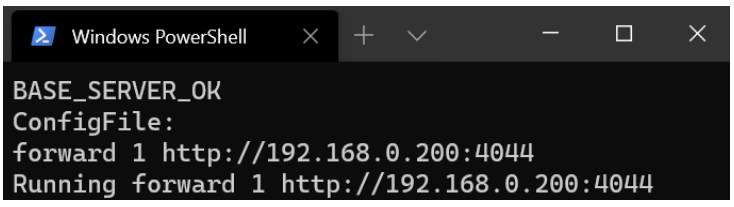

Figura 9.Cambio de configuración recibido y aplicado por la librería cargada en el dispositivo IoT

5.3. Caso de uso 3 - Cambiar la frecuencia de envío

Con el reciente aumento de personal y la nueva maquinaria, se ha detectado que los cambios de temperatura son mucho más pronunciados y rápidos que antes. Esos cambios rápidos de temperatura requieren aumentar la frecuencia con los datos de temperatura que se envía al servicio

centralizado. La compañía determina que este aumento de la frecuencia de envío es necesario, aunque significa mayores costes de procesamiento para el servidor y un mayor consumo para los dispositivos.

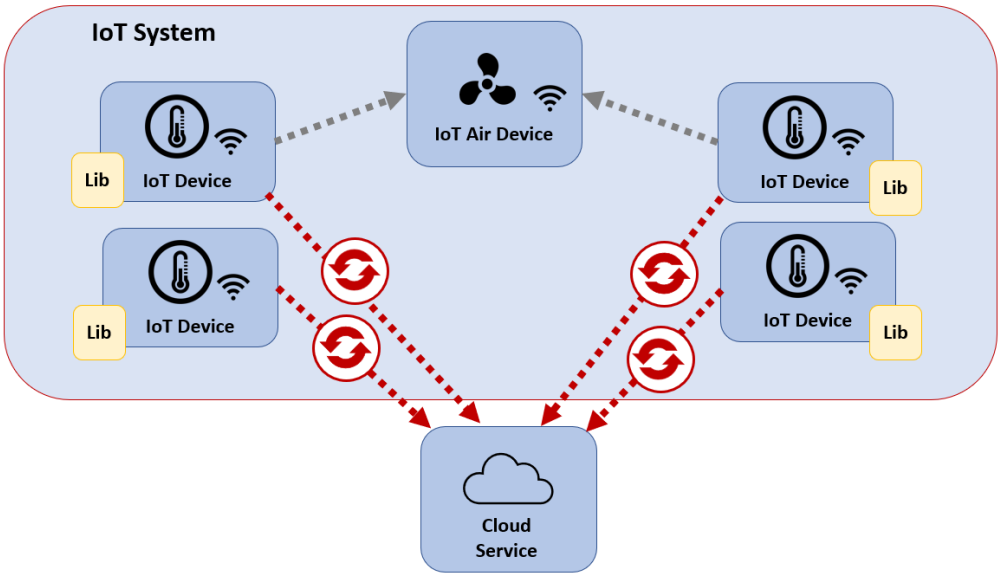

Figura 10. Esquema para cambiar la frecuencia de envío de peticiones

Para cambiar la frecuencia utilizando nuestra solución, los administradores responsables del sistema deben enviar el siguiente comando: *set 1 frequency 500*. En este comando, *"500"* es el número de milisegundos entre cada dato enviado. Cuando la librería detecta una nueva configuración, guarda el valore de la frecuencia localmente y comienza a enviar peticiones cada 500 milisegundos.

Figura 11. Cambio de configuración recibido y aplicado por la librería cargada en el dispositivo IoT

5.4. Sistemas IoT Descentralizados

Los casos de usos anteriores que fueron implementados y puestos a prueba, están basados en sistemas en los que dispositivos se comunican con servicios centralizados. Sin embargo, la propuesta es igualmente útil para apoyar otros cambios de comunicación en sistemas descentralizados. En muchos casos, los sistemas IoT tienen otras topologías, como por ejemplo una malla, donde cada dispositivo IoT puede comunicarse con uno o más dispositivos.

Nuestra propuesta podría usarse para realizar modificaciones en esas interacciones de forma remota. Por ejemplo, si tenemos un dispositivo detector que se comunica con un actuador, podríamos cambiar el actuador por otro o hacer que el detector interactúe con dos actuadores.

Otro caso en el que nuestra propuesta puede ser útil es agregar nuevos dispositivos a los sistemas IoT. Podríamos hacer que los dispositivos que ya tenemos empiecen a interactuar con los que queremos agregar a nuestra malla. Por ejemplo, si queremos agregar un nuevo controlador de luz y ya tenemos un detector de presencia en funcionamiento, podríamos hacer que el detector envíe los datos a ese controlador de luz además del destino donde está enviando los datos actualmente.

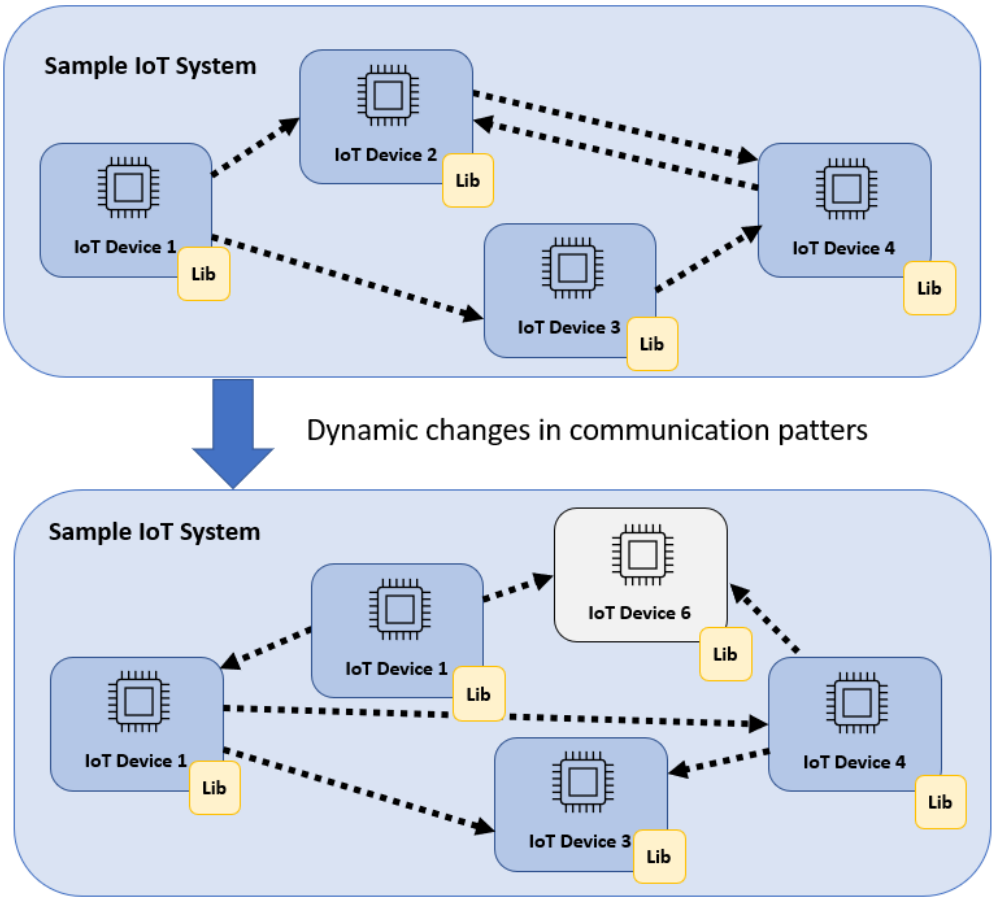

Figura 12. Cambios dinámicos en un sistema IoT con topología de malla

6. Evaluación

6.1. Introducción

En el proceso de evaluación se hará uso del prototipo de la implementación de la propuesta creado para la plataforma Node.js / JavaScript. Durante este proceso se medirán los aspectos que guarden relación con los objetivos marcados al inicio de la investigación. En la evaluación se utilizarán sistemas IoT que requieran cambios de comunicación comunes, estos sistemas estarán basados en los tres casos de uso presentados en la sección anterior:

- **Cambiar el servidor de destino de los clientes.** La versión inicial del sistema tiene un dispositivo IoT que envía datos a un servicio. Después del cambio, el dispositivo debe enviar los datos a otro servicio diferente.
- **Reenviar peticiones a otro servidor.** La versión inicial del sistema tiene un dispositivo IoT que envía datos a un servicio. Después del cambio, el dispositivo también debe enviar los datos a otro nuevo servicio adicional.
- **Cambiar la frecuencia de envío.** La versión inicial del sistema tiene un dispositivo IoT que envía datos a un servicio cada segundo. Después del cambio, el dispositivo debe enviar los datos cada 0,5 segundos.

6.2. Alternativas evaluadas

Se implementará un sistema basado en cada uno de los tres casos de uso utilizando las siguientes alternativas:

- **Actualización OTA.** La OTA (Over the Air) es una de las formas más comunes de actualizar el software de un dispositivo IoT. Consiste en enviar de forma remota una nueva versión del software. Este nuevo software debe reemplazar la versión actual. La mayoría de los dispositivos IoT con cierta capacidad informática pueden usar sistemas operativos que permiten este tipo de actualización. Muchas actualizaciones de OTA requieren reiniciar el dispositivo para aplicar la actualización.
- **OTA Parcial.** Una versión optimizada de la OTA anterior. Este proceso permite generar un paquete más pequeño ya que solamente contiene los cambios que se deben aplicar en el programa. Este proceso solo es compatible con dispositivos que ejecutan sistemas operativos complejos. A menudo requiere reiniciar el dispositivo para aplicar la actualización.
- **Software Parcialmente Actualizable.** Este es un software que ha sido especialmente diseñado desde su inicio para admitir actualizaciones parciales de algunos de sus componentes. En este caso, la actualización del software depende de sí misma, no del sistema operativo del dispositivo. Esta solución depende en gran medida de la estrategia de implementación del software, pero siempre requiere una implementación más compleja que el software sin actualizaciones periódicas. Este tipo de actualización puede o no requerir reiniciar el dispositivo. En nuestros experimentos hemos optado por conectarse directamente al dispositivo mediante una conexión ssh y realizar los cambios necesarios.
- **Proxy.** Un proxy intermedio hace que todos los dispositivos se comuniquen con el proxy, por lo tanto, solo modificando la implementación del proxy se podría modificar algunos aspectos de comunicación de todo el sistema IoT.

- **Firebase.** Servicios como Firebase permiten la inclusión de configuraciones remotas, que se pueden obtener desde el software del dispositivo. Los desarrolladores generalmente almacenan los ajustes de configuración en Firebase (como cadenas de conexión, valores de configuración, etc.). El software del dispositivo debe saber cómo obtener y procesar esos valores de configuración. Al igual que el "Software Parcialmente Actualizable", este tipo de solución requiere una implementación más compleja del software del dispositivo.
- **Flex-Request.** La propuesta de este trabajo. Como se explicó anteriormente, utiliza comandos especializados para cambiar algunos aspectos de la comunicación de los dispositivos.

6.3. Entorno de evaluación

Para evaluar nuestra solución, hemos utilizado una placa Raspberry Pi 3 Modelo B como dispositivo IoT y un ordenador actuando como servidor. En Raspberry Pi, hemos instalado el sistema operativo Raspbian GNU/Linux 10 (buster) y la versión 12.16.1 de Node.js como plataforma de ejecución. Ordenador tiene el sistema operativo Windows 10 Pro x64 versión 1909 con una CPU Intel Core i5-9600KF de 3.70GHz y 16 GB de RAM. Para acceder al dispositivo Raspberry y transferir fácilmente los programas utilizados en la evaluación de los experimentos, hemos utilizado WinSCP.

6.4. Aspectos a evaluar

Según los objetivos de la investigación, la propuesta debe permitir lo siguiente:

1) Cambios en la comunicación de los dispositivos, implicando un bajo coste de implementación. Por esta razón, durante el proceso de evaluación, calcularemos la complejidad de la implementación en cada caso.
2) Los cambios en las comunicaciones deben realizarse de forma rápida y sencilla. Por esta razón, durante el proceso de evaluación, mediremos qué tan complejo es para los desarrolladores realizar cambios en la comunicación.
3) Minimizar el tiempo que el dispositivo no funciona correctamente después de un cambio. Por esta razón, durante el proceso de evaluación, mediremos cuánto tiempo le toma a un dispositivo aplicar el cambio y comienza a estar activa la nueva configuración.

Además, debemos analizar el tráfico de red en el sistema IoT. Las alternativas evaluadas podrían tener diferentes impactos en el tráfico de la red. El objetivo de esta investigación no es optimizar el tráfico de red, pero esto podría ser un factor importante en algunos sistemas IoT.

6.4.1. Complejidad de implementación

Esta es la primera medición de todas. Intenta estimar la complejidad de implementación inicial del sistema IoT antes de aplicar el cambio de configuración, que modificará el esquema de comunicación inicial. En este proceso de evaluación, estimaremos la complejidad de implementación del sistema IoT inicial para los tres experimentos utilizando las seis alternativas.

Medir la complejidad de la implementación del programa es una tarea complicada. La complejidad depende del número de aspectos considerados. Las estimaciones más simples de complejidad se basan únicamente en el tamaño del código. En este caso, partimos de un sistema de estimación de complejidad utilizado en trabajos de investigación anteriores [9] [42]. Este sistema considera lo siguiente:

- C - Tamaño. Número de caracteres del programa, espacios incluidos
- L - Número de líneas de código en el programa.
- F - Número de funciones utilizadas, pero no implementadas por el programador

- NM - Número de módulos estándar utilizados incluidos en la distribución Node.js
- NE - Número de módulos utilizados no incluidos en la distribución Node.js y que tuvieron que instalarse con gestor de dependencias

A cada uno de los aspectos medidos se le asignó un peso en función de su importancia.: C=1, L=2, F=3, NM=4 y NE=5. La puntuación global para cada alternativa evaluada es la suma de la puntuación de cada aspecto.

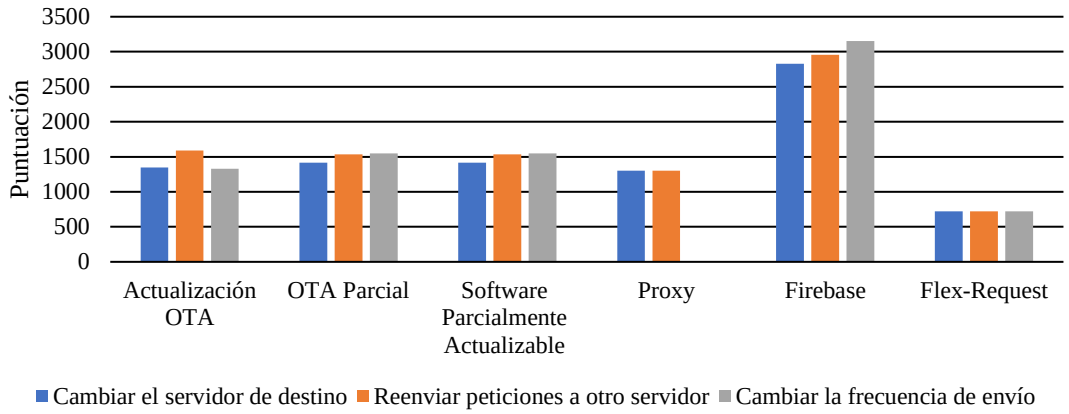

Figura 13. Complejidad de implementación de cada alternativa en cada experimento

La Figura 13 muestra los resultados de la complejidad de implementación para cada alternativa en los tres experimentos. Firebase tiene la mayor complejidad en todos los experimentos debido a que se debe agregar código adicional para conectarse al servicio Firebase Remote Config. La Actualización OTA, la OTA Parcial y el Software Parcialmente Actualizable tienen una complejidad de implementación parecida con diferencias en puntuación no mayores a 100 puntos entre ellos en cada experimento. Proxy tiene la misma complejidad en los dos primeros experimentos debido a que el dispositivo IoT es el que envía la petición y el Proxy solo redirige o reenvía la petición a otro servidor. Hemos determinado que no podemos usar Proxy para implementar el cambio de frecuencia de envío ya que este cambio se debe producir en el origen del envío de datos que es el propio dispositivo. Nuestra propuesta tiene la menor complejidad de implementación en todos los experimentos.

6.4.2. Complejidad de cambio de configuración

Esta es la segunda medición. Evalúa qué tan complejo es hacer un cambio sobre la versión inicial del sistema.

A diferencia de la implementación del software, la aplicación de cambios sobre el esquema de comunicación no solo puede implicar cambios en el código, sino que a veces requiere el uso de herramientas de configuración, comandos, etc. Debido a estas particularidades, hemos incluido en esta estimación de complejidad todas las acciones realizado por el desarrollador con un ratón y un teclado.

Definimos la complejidad del cambio de configuración como las acciones que se deben realizar para aplicar un cambio de configuración. Para medir esta complejidad, se midieron las siguientes acciones:

- K - Número de pulsaciones de teclas
- MLB: número de clics con el botón izquierdo del ratón
- MRB: número de clics con botón derecho del ratón
- MDC: número de doble clics

- MWS: número de desplazamiento de la rueda del ratón
- MM: movimiento del ratón en centímetros

Todos estos aspectos se midieron con el software Mousotron. Para cada aspecto, hemos asociado los siguientes pesos: K=1, MLB=2, MRB=1, MDC=1, MWS=1 y MM=0,5. La puntuación global para cada alternativa evaluada es la suma de las puntuaciones de cada aspecto. Por cada cambio que implica enviar una petición a través de Internet para cambiar la configuración, hemos creado un script. Por ello, la acción de enviar la petición con el cambio se traduce en escribir el comando para ejecutar el script.

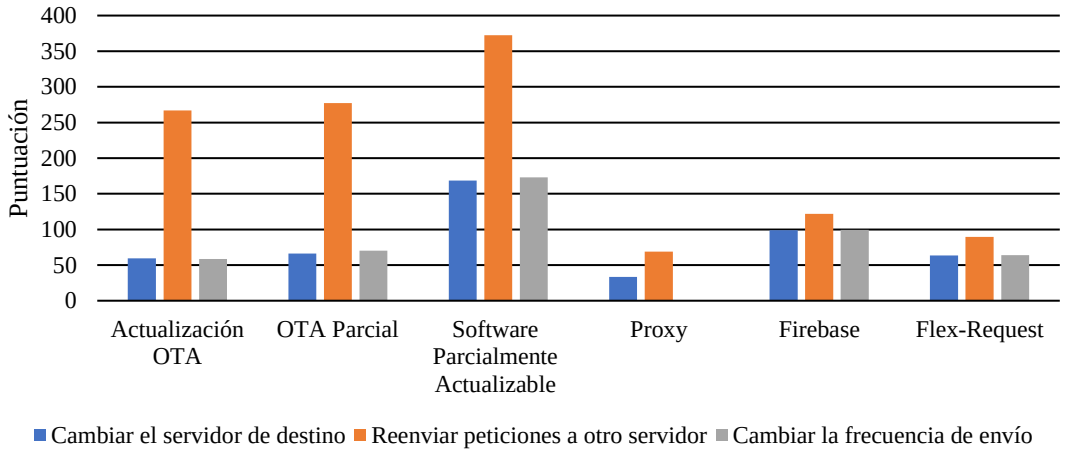

Figura 14. Complejidad de cambio de configuración de cada alternativa en cada experimento

Las medidas de la complejidad de cambiar de configuración de la figura 14 muestran que Proxy tiene la complejidad más baja para cambiar la dirección de destino y reenviar peticiones. Esto puede ser debido a que el Proxy estaba en la misma PC utilizada para ejecutar los experimentos. Por lo tanto, esos cambios solo implican cambiar y guardar archivos locales sin conectarse a otras máquinas a través de Internet. Determinamos que Proxy no se puede usar en el tercer experimento ya que la frecuencia de envío debe modificarse en los propios dispositivos IoT que envían las peticiones. La menor complejidad de cambio para el tercer experimento es la Actualización OTA, ya que solo necesita cambiar el valor de la propiedad que almacena la frecuencia de envío y ejecutar el script que envía la actualización. La Actualización OTA, la OTA Parcial y el Software Parcialmente Actualizable tienen una complejidad de cambio significativamente mayor en el segundo experimento que el resto de las alternativas. Esto se debe a las modificaciones que hay que realizar en la implementación para reenviar la petición a otro servidor. Nuestra propuesta tiene la segunda complejidad de cambio más baja para el segundo experimento y la tercera complejidad de cambio más baja para el primer y tercer experimento. A pesar de no tener la complejidad de cambio, se acerca a las óptimas.

6.4.3. Tiempo de inactividad del dispositivo

Esta es la tercera medición que evalúa cuánto tiempo pasa hasta que el dispositivo comienza a funcionar con la nueva configuración después de que se haya realizado un cambio de configuración.

La aplicación de una actualización en un software en ejecución significa que durante el período en que se aplica la actualización del dispositivo, el dispositivo puede estar inactivo. Puede estar totalmente detenido, seguir ejecutando el programa, pero sin enviar datos o seguir ejecutando el software, pero enviando los datos utilizando la configuración anterior. Por eso, medimos el tiempo que le toma al dispositivo aplicar la nueva configuración y comenzar a enviar datos usándola.

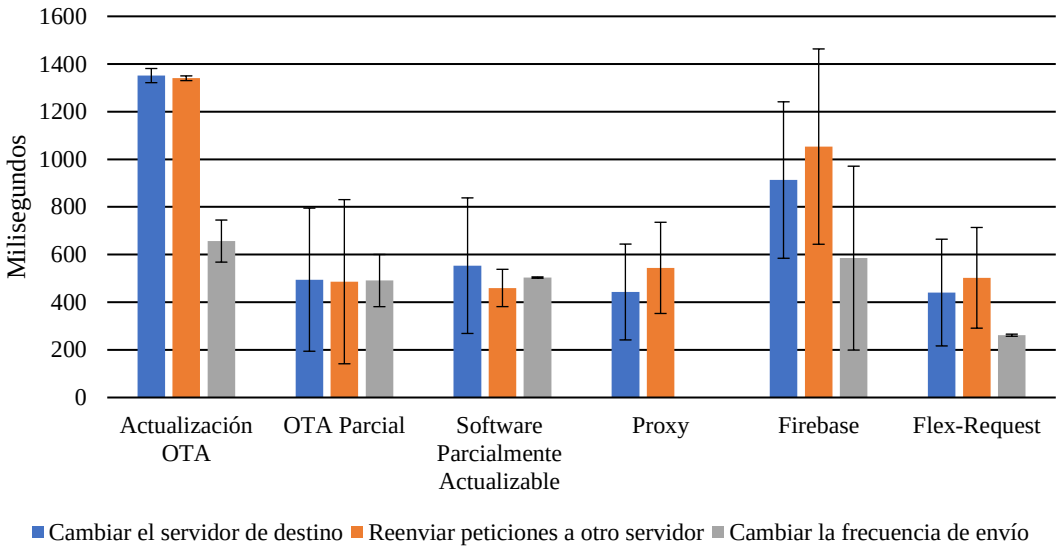

Figura 15. Tiempo hasta aplicar el cambio de configuración

Los resultados de medir el tiempo que tarda el dispositivo en aplicar el cambio se muestran en la figura 15. La única alternativa que requería detener completamente el programa en ejecución era la Actualización OTA. Los tiempos que programa no se estaba ejecutando para cada experimento son: 346,30 ± 8,49 ms, 321.30 ± 7,85 ms y 356,10 ± 20,59. Otras alternativas no requerían que el programa se detuviera totalmente. Esto permitiría, por ejemplo, seguir recolectando y almacenando los datos para enviarlos después de que se haya realizado la actualización.

La Actualización OTA tiene el tiempo de inactividad más largo en cada experimento. En el tercer experimento, el tiempo es significativamente menor porque, como se explicó, se ha medido el tiempo hasta que el dispositivo comienza a enviar datos utilizando la nueva configuración. Por lo tanto, en el caso de la Actualización OTA y Firebase, donde los tiempos requeridos para recibir el cambio son los más largos, la frecuencia de envío de los datos tiene un impacto en los resultados obtenidos. En el tercer experimento, se ha cambiado la frecuencia de 1 segundo a 0,5 segundos, y debido a ello, los primeros datos enviados con una nueva configuración ocurren antes. Como se explicó anteriormente, en el tercer experimento, determinamos que la solución Proxy no se puede aplicar. Nuestra propuesta no tuvo diferencias significativas en los experimentos 1 y 2 con OTA Parcial, Software Parcialmente Actualizable y Proxy, sin embargo, en el tercer experimento, nuestra propuesta realizó el cambio significativamente más rápido.

6.4.4. Tráfico de red

Esta es la última medición realizada sobre los casos de usos. Evalúa cuánto tráfico de red consume un sistema IoT desarrollado con las diferentes alternativas en un intervalo de tiempo de 5 minutos. Este tiempo podría servir para hacer una extrapolación del consumo en periodos de actividad más prolongados. Primero, se analiza un período de tiempo de operación sin hacer cambios de configuración. A continuación, un período de tiempo equivalente, pero esta vez haciendo un cambio de configuración. Esto nos permitirá ver el tráfico de red habitual para los sistemas IoT desarrollados con diferentes tecnologías y el coste de hacer un cambio en la comunicación.

Para medir el tráfico de la red, utilizamos el software Wireshark. Medimos el tamaño total (en bytes) de los paquetes enviados entre el dispositivo IoT y los servidores en un intervalo de tiempo de 5 minutos. En las evaluaciones cuando se realizó el cambio de configuración, el cambio siempre se realizó un minuto desde el inicio de la medición, porque agregar el reenvío de peticiones a otros servidores o cambiar la frecuencia de envío implica la modificación del número total de las peticiones enviada y, por lo tanto, el tamaño total de los paquetes enviados. Para que

la medición sea fiable el momento de realizar el cambio debe ser el siempre exactamente el mismo en todos los escenarios.

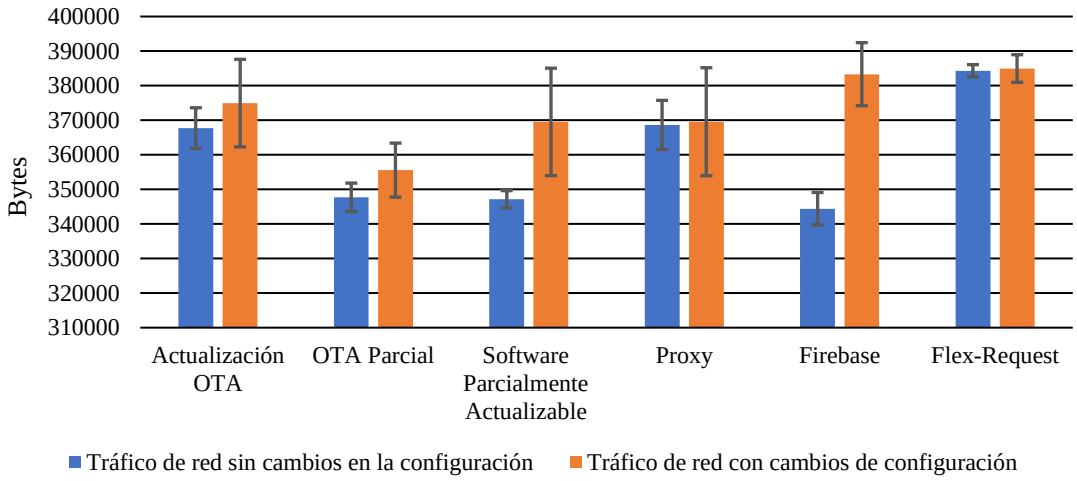

Figura 16. Tráfico de red con y sin cambios de configuración

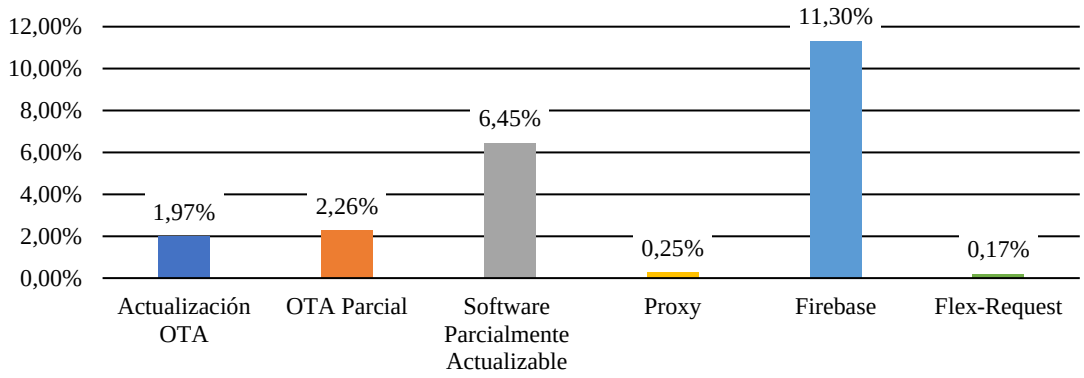

Figura 17. Diferencia relativa en el tráfico de red sin y con cambios de configuración

La Figura 17 muestra las mediciones de tráfico de la red entre el dispositivo IoT y el servidor al que envía los datos. Se ha medido el uso del tráfico de cada alternativa en un período de 5 minutos sin hacer cambios de configuración, y luego se ha vuelto a medir el mismo período haciendo un cambio de configuración.

En el primer caso, sin cambios, nuestra propuesta tiene el mayor tráfico de red, seguida por Proxy y Actualización OTA. Aunque la propuesta tiene el mayor uso de tráfico, la diferencia relativa con otras alternativas es solo entre 4% - 12%.

En la segunda parte realizó un cambio de configuración, nuestra propuesta sigue teniendo el mayor tráfico de red. Sin embargo, es seguido de cerca por Firebase y el Software Parcialmente Actualizable, que tienen un tráfico de red significativamente mayor en relación con el caso sin cambios de configuración. En este caso, la diferencia relativa en el tráfico de red con otras alternativas es entre 0.4% - 9%.

La Figura 13 muestra el aumento relativo en el tráfico de red para cada alternativa entre mediciones sin cambios y con cambios. Firebase tiene el menor uso de tráfico sin cambios con solo 344405 bytes; sin embargo, el aumento relativo en el tráfico para el caso con cambios es 11.30%, el más alto. El Software Parcialmente Actualizable tiene un aumento relativo del 6.45% que puede deberse a que, para realizar el cambio, se tiene que conectarse directamente al dispositivo a través de ssh. Aunque la propuesta tiene el tráfico de red más alto: 384301,40 bytes

sin cambio de configuración y 384952,60 bytes con cambios, el aumento de tráfico relativo es el más bajo, con solo 0.17%. Esto indicaría que si el sistema va a requerir cambios frecuentes el coste sobre las comunicaciones podría llegar a ser más bajo que el de otras alternativas.

6.4.5. Resumen de evaluación

En este apartado se incluye una tabla resumen con los resultados de la evaluación para los tres aspectos relativos a los objetivos de esta investigación.

Tabla 1. Resumen de los resultados de la evaluación

	Complejidad de implementación			Complejidad de cambio			Tiempo inactividad		
	Caso de uso 1	Caso de uso 2	Caso de uso 3	Caso de uso 1	Caso de uso 2	Caso de uso 3	Caso de uso 1	Caso de uso 2	Caso de uso 3
Actualización OTA	1347	1590	1327	59,5	267	58,5	1351,4	1340,1	656,8
OTA Parcial	1416	1535	1548	66	277	70	494,8	486,1	490,9
Software Parcialmente Actualizable	1416	1535	1550	168,5	372,5	173	553,4	459,9	503,4
Proxy	1301	1301		33,5	69		442,9	544,2	
Firebase	2828	2956	3155	99	122	99,5	912,9	1053,2	585,2
Flex-Request	720	720	720	63,5	89,5	64	440,5	502,5	261,7

La Tabla 1 contiene un resumen de los resultados obtenidos en la evaluación. Se utiliza el siguiente código de colores:

Verde	el mejor resultado o con una diferencia menor de 10% con el mejor
Amarillo	resultado entre un 10% y un 50% peor que el mejor
Rosa	resultado con una diferencia mayor del 50% con el mejor
Rojo	no aplica

7. Conclusiones y trabajo futuro

Este trabajo de investigación se centra en proporcionar una alternativa que permita la creación de sistemas IoT capaces de modificar de forma remota algunos aspectos de comunicación de los dispositivos de una manera rápida y ágil. Esta característica facilitará la creación de prototipos, permitirá probar diferentes arquitecturas durante la fase de desarrollo de un sistema IoT y agilizará la realización de cambios en producción. Permitir cambios ágiles es especialmente útil en sistemas con incertidumbre, ya que permite explorar más alternativas y realizar más pruebas. Durante la fase de uso y explotación de los sistemas IoT, los cambios en la ejecución también pueden ser frecuentes, debido a la introducción de nuevos dispositivos, servicios, cambios en el entorno físico, etc.

La propuesta para abordar este problema fue el diseño de un servicio de gestión de configuraciones de comunicación y una librería de comunicación con capacidad de configuración remota. Se ha desarrollado una implementación específica de esta librería en Node.js. El sistema propuesto permite al desarrollador realizar diferentes cambios en la comunicación de los dispositivos de forma remota utilizando comandos. Estos cambios no requieren la actualización del software del dispositivo.

Para facilitar que la propuesta pudiera ser usada de forma ágil en el desarrollo de sistemas esta no debería aumentar la complejidad de implementación del software de dispositivos IoT. En función de los casos de uso evaluados, un software desarrollado con la propuesta podría tener una complejidad de implementación significativamente menor que el software desarrollado con otras alternativas. En los casos analizados, La estimación de la complejidad de la implementación inicial del sistema se redujo en alrededor del 40% - 50%. Estos resultados indican que en algunos escenarios la propuesta puede servir para implementar con un coste bajo sistemas IoT capaces de soportar cambios remotos en sus comunicaciones.

Otro de los objetivos de la evaluación era medir el coste de realizar cambios de comunión en un sistema IoT en ejecución. Entre las alternativas analizadas, la que condujo a cambios en la comunicación del dispositivo con menor coste fue el Proxy. Aun así, esta alternativa no tenía suficiente funcionalidad para realizar cambios complejos, como los cambios especificados en el caso de uso 3. El coste de realizar cambios utilizando la propuesta no tuvo una diferencia significativa en comparación con tres de las alternativas, y este coste fue mucho menor que el de las otras dos alternativas restantes.

Un aspecto critico de los cambios de comunicación era minimizar el tiempo que el dispositivo IoT está apagado o no funciona correctamente después de realizar un cambio. Solo una de las alternativas analizadas requiere reiniciar el dispositivo después de la actualización (OTA). El tiempo que tomó la propuesta para aplicar el cambio no tuvo diferencias significativas con otras alternativas en los casos de uso 1 y 2. Sin embargo, en el tercer caso de uso, el tiempo para aplicar el cambio fue entre 45% y 60% más corto.

Aunque la investigación no buscaba optimizar el tráfico de red, este fue medido en cada uno de los casos de uso, dado su potencial aspecto critico en algunos sistemas IoT. La propuesta obtuvo un aumento en el tráfico de red en comparación con las otras alternativas en los casos evaluados. Este aumento fue entre 4% - 12% durante un período de uso normal, sin cambios en la comunicación. Cuando se realizaron estos cambios, la diferencia en el tráfico de red para la propuesta se redujo a 0.4% - 9%. Según los resultados, la propuesta siempre tiene más consumo de tráfico de red que otras alternativas, pero el aumento relativo del tráfico de red por cada cambio realizado fue solo del 0,17%, mucho menos que en otras alternativas.

Los resultados muestran que esta propuesta podría ser adecuada para muchos sistemas de IoT que por diferentes motivos pueden requerir aplicar cambios a la comunicación de sus dispositivos en fase de prototipado, desarrollo o producción.

Las futuras líneas de investigación estarán relacionadas para incluir se centrarían en permitir comunicaciones entre dispositivos y servicios heterogéneos, facilitando cambios automáticos de protocolos de datos (CoAP, AMQP, MQTT, etc.), los esquemas de comunicación y reducir el impacto de la propuesta en la red de tráfico.

8. Difusión de resultados

La investigación realizada y sus resultados se han descrito en un artículo científico que se ha enviado a la revista: *Future Generation Computer Systems* cuyo factor de impacto es: 5.768 (consultado el día 04/07/2020).

La revista se puede consultar en el siguiente enlace:

https://www.sciencedirect.com/journal/future-generation-computer-systems.

El artículo científico se encuentra en el Anexo 4. Artículo científico.

9. Planificación y presupuesto

9.1. Identificación de interesados

Los interesados de este proyecto son:

- **Autor del proyecto** - Alumno que realiza y defiende la investigación ante el tribunal.
- **Director del proyecto** - Profesor encargado de dirigir al alumno en la ejecución de la investigación.
- **Tribunal del proyecto** - Profesores encargados de evaluar la investigación realizada por el alumno.

9.2. OBS

La organización y el reparto de las tareas del proyecto se estructura de la siguiente manera.

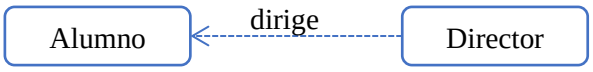

Figura 18. Organización de los participantes del proyecto

- **Alumno** - encargado de realizar y defender el proyecto ante un tribunal.
- **Director** – encargado de dirigir y apoyar al alumno en la ejecución del proyecto. No participa directamente en la implementación y la ejecución de este.

9.3. PBS

Durante la ejecución del proyecto se deberán producir los siguientes entregables:

- **Código de la implementación** – todos los ficheros con el código, la configuración y las claves de servicios externos utilizados en la implementación del proyecto.
- **Código de la implementación de los experimentos planteados** – todos los ficheros con el código, la configuración y las claves de servicios externos utilizados en la implementación de los experimentos utilizados para evaluar la solución planteada en el proyecto.
- **Hojas Excel con los resultados de la ejecución de los experimentos** – 3 hojas Excel, una por cada experimento, con los resultados de la evaluación de la propuesta.
- **Artículo de la investigación** – artículo científico en el que se presenta el proyecto de investigación realizado.
- **Memoria del proyecto** – documento con la memoria del proyecto. Consiste en una versión de la ampliada de los apartados del artículo, así como apartados adicionales con aspectos de gestión del proyecto.

9.4. WBS – Planificación inicial

9.4.1. Calendarios del proyecto

Para este proyecto se define un calendario del proyecto que diferencia dos tipos de semanas de trabajo diferentes:

- Semanas con clases presenciales: 13/09/2019 – 23/02/2020

Tabla 2. Horas de trabajo de semanas con clases presenciales

Día	*Horas de trabajo*
Lunes	0h
Martes	0h

Miércoles	0h
Jueves	0h
Viernes	1h
Sábado	2h
Domingo	2h

Estas semanas se dedicarán solamente al estudio de las alternativas y la realización de pruebas de concepto de posibles soluciones antes de decidir la línea de investigación final y el objetivo de la investigación.

- Semanas sin clases: 24/02/2020 – 31/07/2020

Tabla 3. Horas de trabajo semanales sin clases

Día	*Horas de trabajo*
Lunes	4h
Martes	4h
Miércoles	4h
Jueves	4h
Viernes	4h
Sábado	6h
Domingo	6h

9.4.2. Planificación inicial

El proyecto comienza el día 23 de septiembre con la primera reunión con el director para seleccionar el tema del proyecto de entre los propuestos por él. Este mismo día se realiza la planificación inicial del proyecto a muy alto nivel.

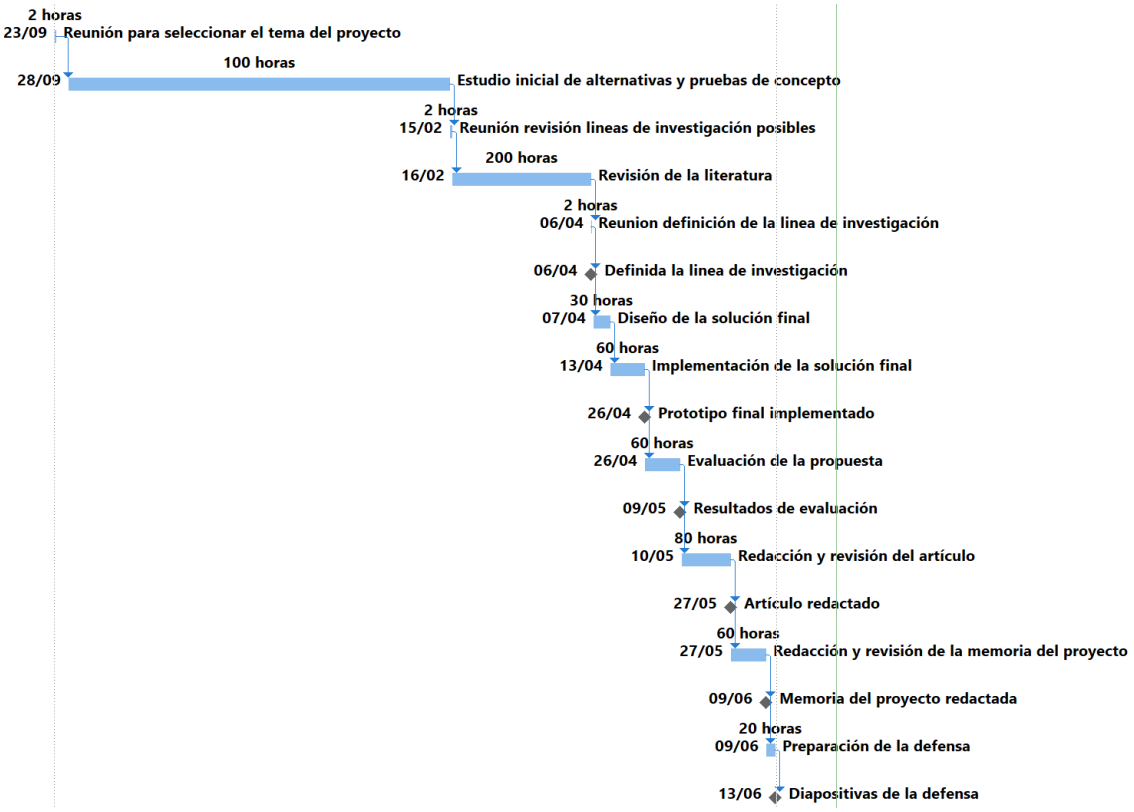

Figura 19. Planificación inicial del proyecto

Durante las siguientes semanas hasta finalizar las clases se planea, siguiendo el calendario propuesto, realizar el estudio de alternativas y hacer pruebas de concepto de aspectos de los que se tiene menos conocimientos como la comunicación mediante el protocolo MQTT o la lectura de los datos de un sensor de temperatura de una placa Raspberry.

El día 15 de febrero, tras finalizar las clases presenciales, se planea tener otra reunión con el director para determinar las posibles líneas de investigación dentro del tema y empezar la revisión de la literatura relacionada hasta el día 6 de abril. Este día se planea otra reunión con el director para definir la línea de investigación final y los experimentos para la evaluación de esta.

Entre 7 de abril y 26 de abril se realizará el diseño y la implementación de la propuesta final. Seguida de la evaluación de esta hasta 9 de mayo.

A partir del día 9 de mayo se realizará la redacción del artículo y la memoria del proyecto para finalizar con la preparación de la defensa.

9.5. Riegos

9.5.1. Plan de gestión de riesgos

El plan de gestión de riesgos completo se puede encontrar en el Anexo 1.

9.5.2. Identificación de riesgos

A continuación, se enumeran la identificación inicial de los riesgos. La hoja con el análisis en detalle se encuentra en el Anexo 2.

- Requerir más tiempo del planificado en la búsqueda de literatura relacionada
- Encontrarse con una literatura más extensa de lo estimado
- Dificultad para definir un objetivo adecuado dentro de la línea de investigación seleccionada
- Surgimiento de un trabajo de igual contribución
- Defecto en el PC de trabajo
- Requerir significativamente más tiempo del planificado para el aprendizaje de alguna de las tecnologías necesarias para poder diseñar una solución
- Diseño de una solución que resulta muy poco apropiada para el cumplimiento de los objetivos
- Problemas de implementación del prototipo de la propuesta diseñada
- Prototipado: Defecto en el hardware de la placa
- Prototipado: Defecto en sensores o periféricos
- Prototipado: Problemas de conectividad con los dispositivos IoT
- Evaluación: Errores en el diseño de los experimentos
- Evaluación: Errores en la medición de los datos
- Asignatura suspensa
- Falta de servicio de internet

9.6. Presupuesto inicial

El precio de venta de cada perfil que participa en el proyecto es el siguiente.

Tabla 4. Precio de venta de los perfiles que participan en el proyecto

Perfil	Precio Venta
Autor	35,00 €
Director	55,00 €

Este proyecto tiene los siguientes costes de material: placa Raspberry y sensores para poder realizar la evaluación de la propuesta.

Tabla 5. Costes materiales del proyecto

Descripción	Cantidad	Unidad	Precio unidad	Total
Compra de una placa Raspberry	1	articulo	29,29 €	29,29 €
Compra de un kit de sensores	1	kit	24,29 €	24,29 €
Total				**53,58 €**

Para el presupuesto inicial del proyecto se cogen las tareas de la planificación inicial, se multiplican por el porcentaje de participación del perfil y se suman los costes de cada perfil. Este calcula da un subtotal de 21.890,00 €.

A esto se añaden los materiales y el 5% de costes indirectos que ascienden a 53.58 € y 1.094,50 € respectivamente, lo que resulta en un presupuesto total de 23.038,08 €.

Tabla 6. Presupuesto inicial del proyecto

Nivel de esquema	Nombre de tarea	Horas	Autor	Director	Total
1	Reunión para seleccionar el tema del proyecto	2	100%	100%	180,00 €
1	Estudio inicial de alternativas y pruebas de concepto	100	100%		3.500,00 €
1	Reunión revisión líneas de investigación posibles	2	100%	100%	180,00 €
1	Revisión de la literatura	200	100%		7.000,00 €
1	reunión definición de la línea de investigación	2	100%	100%	180,00 €
1	Diseño de la solución final	30	100%		1.050,00 €
1	Implementación de la solución final	60	100%		2.100,00 €
1	Evaluación de la propuesta	60	100%		2.100,00 €
1	Redacción y revisión del artículo	80	100%		2.800,00 €
1	Redacción y revisión de la memoria del proyecto	60	100%		2.100,00 €
1	Preparación de la defensa	20	100%		700,00 €
Subtotal		**616**			**21.890,00 €**

Costes materiales	**53,58 €**
Costes indirectos (5%)	**1.094,50 €**

Total	**23.038,08 €**

10. Ejecución del proyecto

10.1. Plan de seguimiento

El seguimiento del proyecto se realizará utilizando la herramienta MS Project. En esta herramienta se definirá un nuevo proyecto al que se añadirán las tareas, hitos y recursos que participarán en la ejecución del proyecto. Cada día se deberá actualizar el progreso de la tarea realizada.

Las posibles incidencias que ocurran en el proyecto deberán ser registradas en una bitácora de incidencias descrita en el apartado siguiente.

Durante el desarrollo del proyecto se revisará la planificación en los siguientes puntos del proyecto:

- Planificación inicial del apartado WBS – Planificación inicial
- Después de la definición de la línea de investigación
- Después de la evaluación de la propuesta
- Planificación final del apartado Planificación final

10.1.1. Línea base inicial

La línea base inicial es igual a la planificación inicial definida en el apartado *Planificación inicial*. Como el proyecto lo realiza una única persona, las tareas se definen siguiente un orden lineal con dependencias *Fin a comienzo*, una tarea no empieza hasta que no se haya finalizado la anterior.

10.1.2. Línea base después la definición de la línea de investigación

La reunión para definir la línea de investigación final se produce el día 9 de abril lo que supone 3 días de retraso respecto a la planificación inicial.

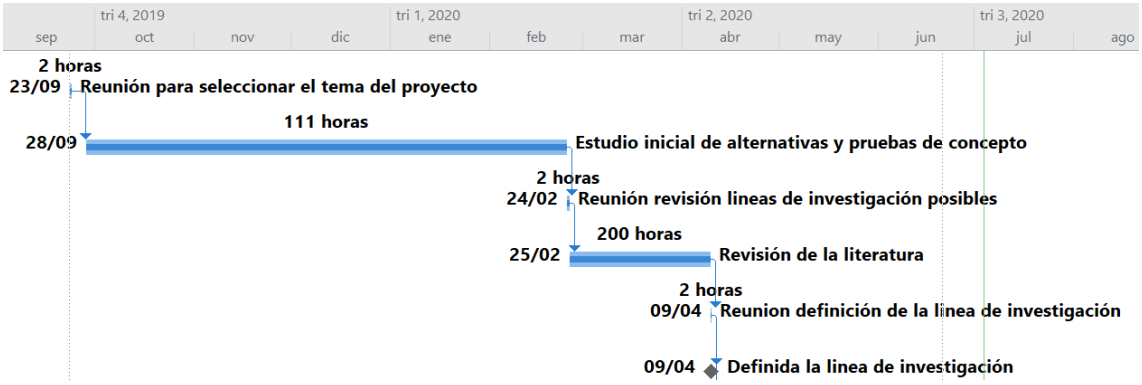

Figura 20. Línea base después la definición de la línea de investigación

Durante las semanas con clases presenciales se ha realizado el estudio de alternativas y pruebas de concepto, donde se ha analizado los aspectos más desconocidos de proyecto como el protocolo MQTT o la lectura de sensores.

Tras esta reunión se definen las tareas de las siguientes fases con mayor detalle y reevalúa la estimación inicial de horas.

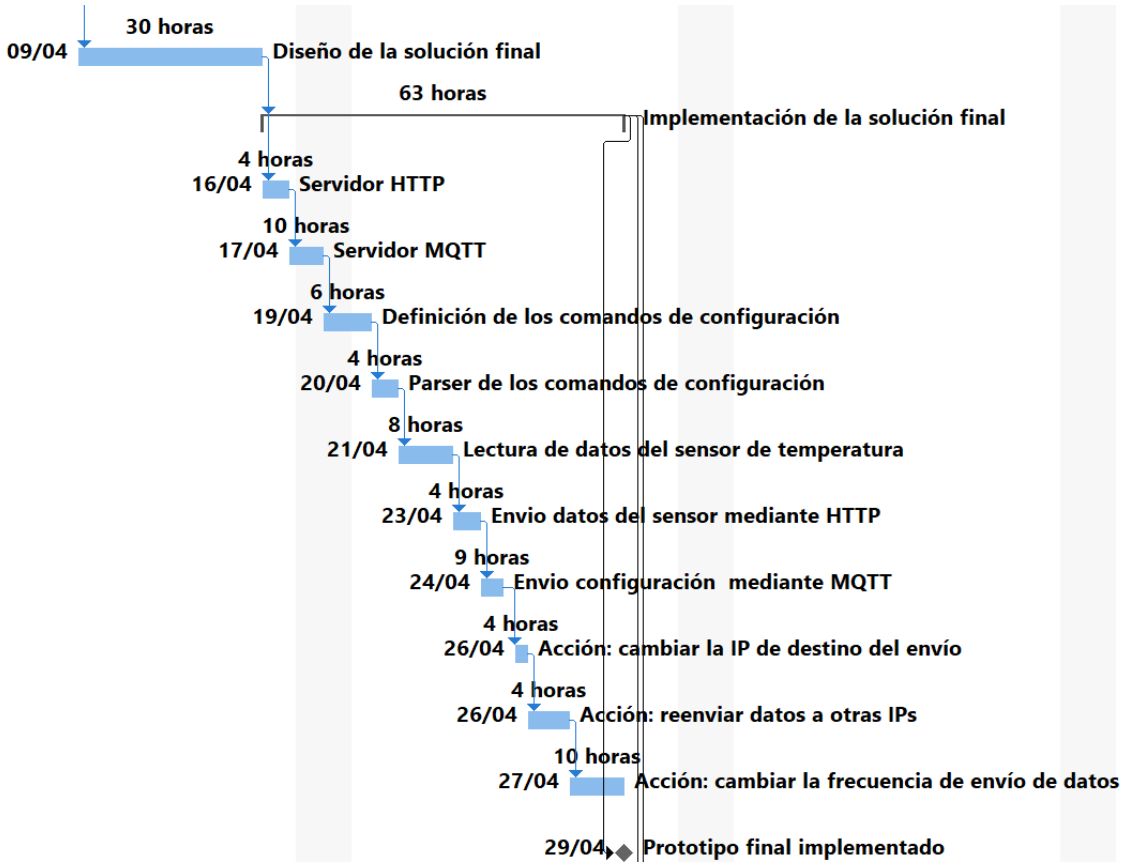

Figura 21. Tarea de implementación de la solución de la solución de la línea base después la definición de la línea de investigación

La tarea de implementación de la solución final pasa a tener 63 horas.

La tarea de la evaluación de la propuesta de la línea base inicial se desglosa en dos: implementación de los experimentos y la propia evaluación de la propuesta.

La implementación de los experimentos se estima en 42 días según los calendarios establecidos, de las cuales el primer experimento se estima en 20 horas y los dos siguientes al ser modificaciones sobre el primero tan solo en 12 horas y 10 horas.

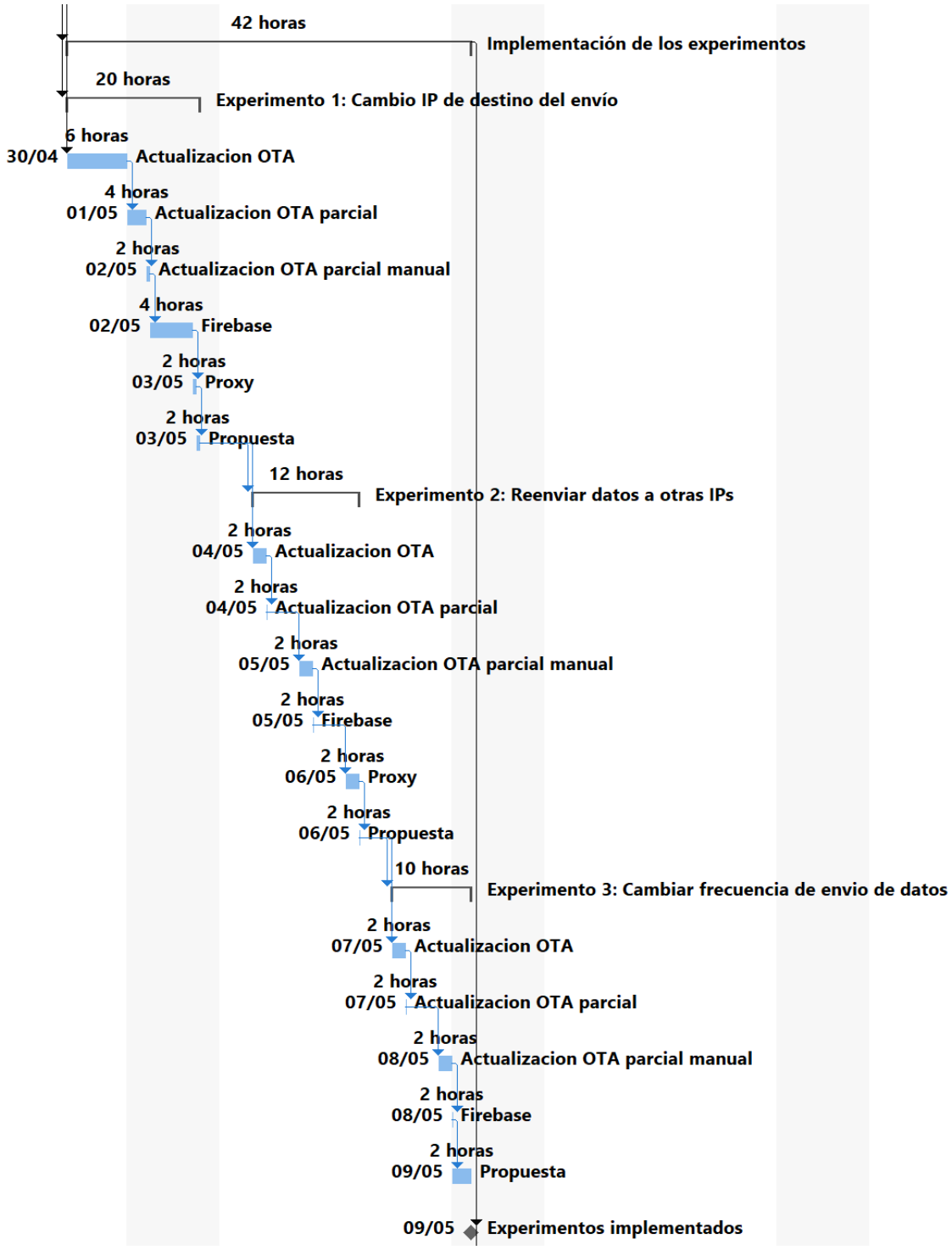

Figura 22. Tarea de implementación de los experimentos de la línea base después la definición de la línea de investigación

Con esta estimación se finalizarían los aspectos de implementación el día 9 de mayo para disponer hasta finales de mayo para realizar la evaluación. Esta tarea también se separa en tres partes de 24 horas cada una ya que en cada experimento se miden los mismos aspectos.

Con la reestimación de las tareas de evaluación, esta parte pasa a acabar el día 24 de mayo en lugar de 12 de mayo según la estimación inicial.

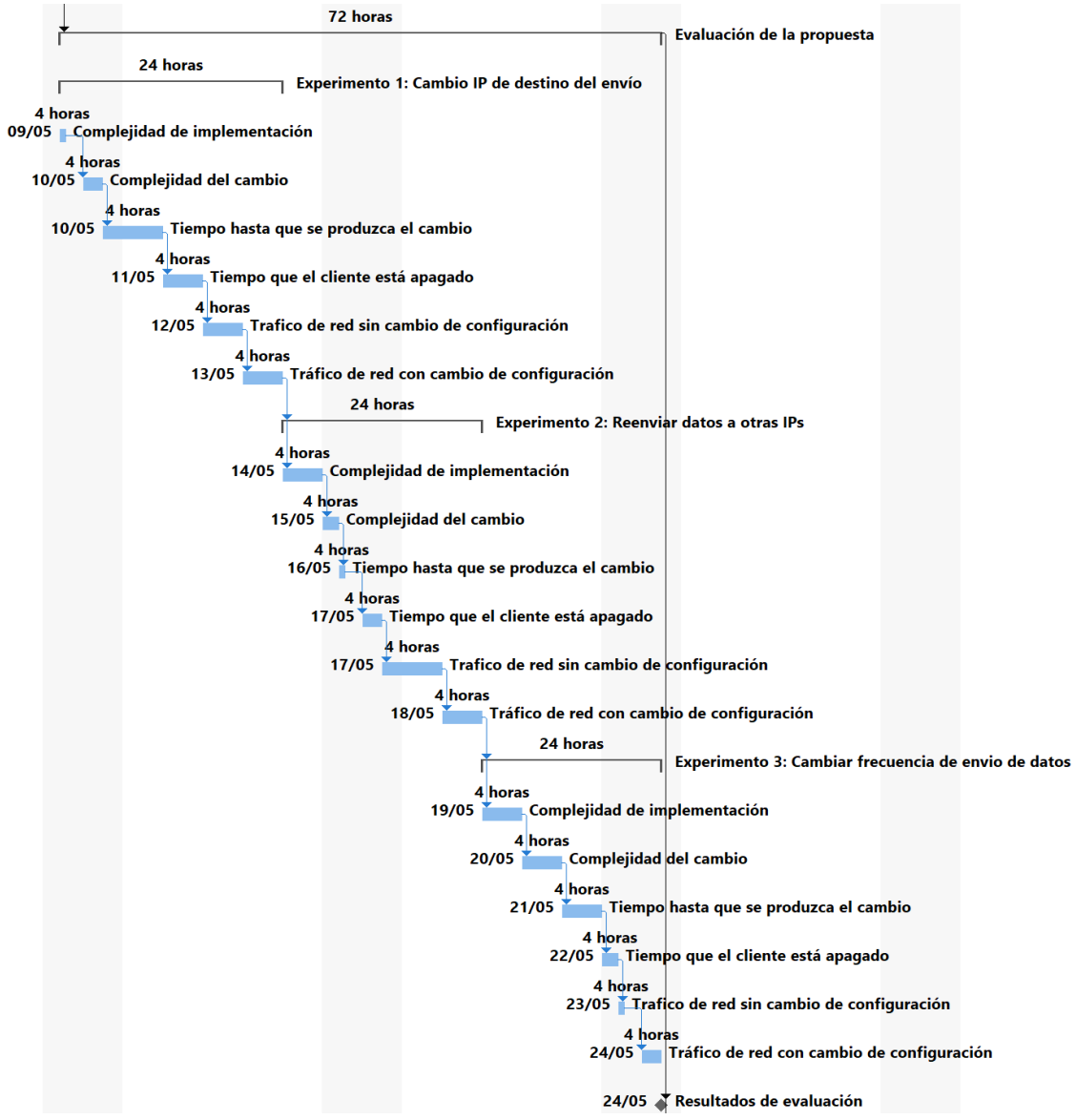

Figura 23. Tarea de evaluación de la propuesta de la línea base después la definición de la línea de investigación

Al redefinir las tareas de implementación y evaluación de la propuesta, estas dos pasan de finalizar el día 24 de mayo. Como la fecha límite para la entrega de la memoria del proyecto es el día 26 de junio en principio no se necesita ajustar las horas de las tareas de documentación para poder llegar al plazo de entrega de la memoria de la convocatoria de julio del curso 2019/2020.

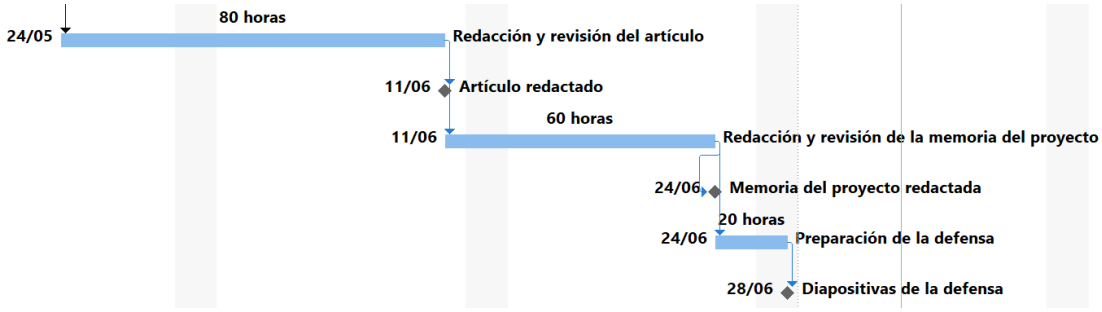

Figura 24. Tareas de documentación de la línea base después la definición de la línea de investigación

10.1.3. Línea base después la evaluación de la solución propuesta

La fecha real de finalización de la evaluación de la propuesta es 6 de junio lo que supone una desviación de 13 días respecto de la línea base anterior.

La tarea de implementación de la solución final se ha reducido de 63 horas a tan solo 34 horas. Como se ha pensado anteriormente, esto se debe en gran parte a que durante las semanas de clases presenciales se ha dedicado tiempo para revisar aspectos más desconocidos que puedan dar problemas a la hora de implementarlos.

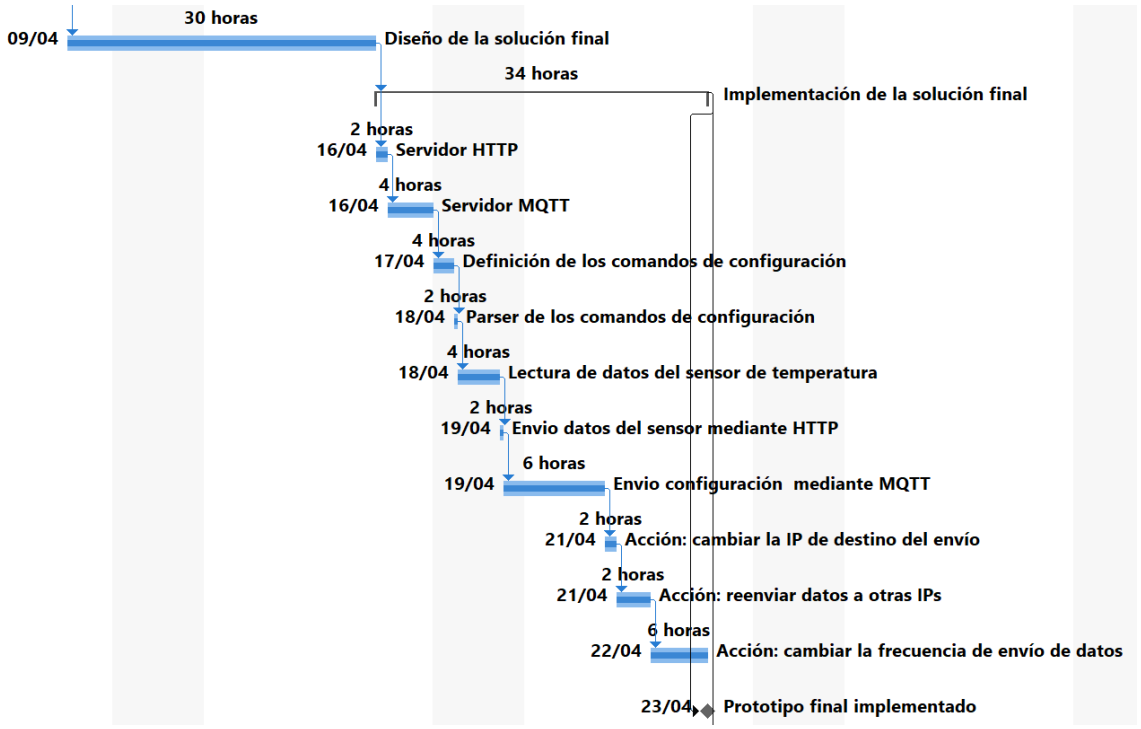

Figura 25. Tareas de implementación de la solución final después de la evaluación de la propuesta

La tarea de implementación de experimentos se ha alargado 6 horas más debido al aumento en las tareas de Actualización OTA y Firebase del primer experimento que han resultado algo más complicadas. Las demás tareas de los experimentos 2 y 3, no han tenido aumentos en coste. Esto se debe a que como se había pensado inicialmente, al ser modificaciones sobre el experimento 1 su coste se reduce considerablemente.

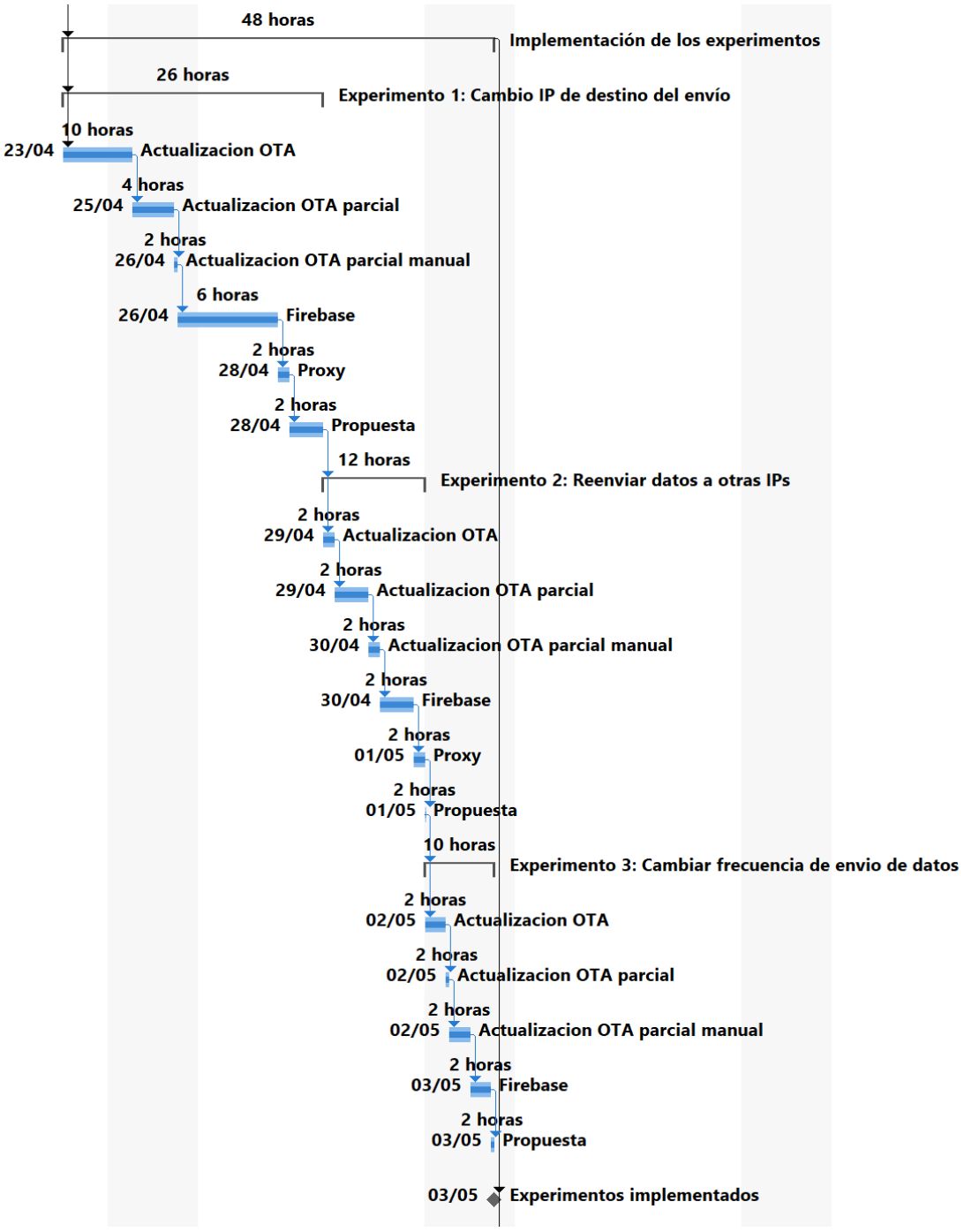

Figura 26. Tareas de implementación de los experimentos después de la evaluación de la propuesta

En la tarea de evaluación de los experimentos, es donde se ha producido el mayor retraso respecto de la planificación de la línea base anterior. Esta tarea ha pasado de su estimación anterior de 72 horas a llevar 154 horas. Esto se debe a que a la hora de estimar las horas para estas tareas no se han tenido en cuenta aspectos como: fallos de conectividad entre el dispositivo y el ordenador, posibles errores o cambios en la implementación de los experimentos, la repetición de las mediadas o el propio tiempo que lleva ejecutar un experimento.

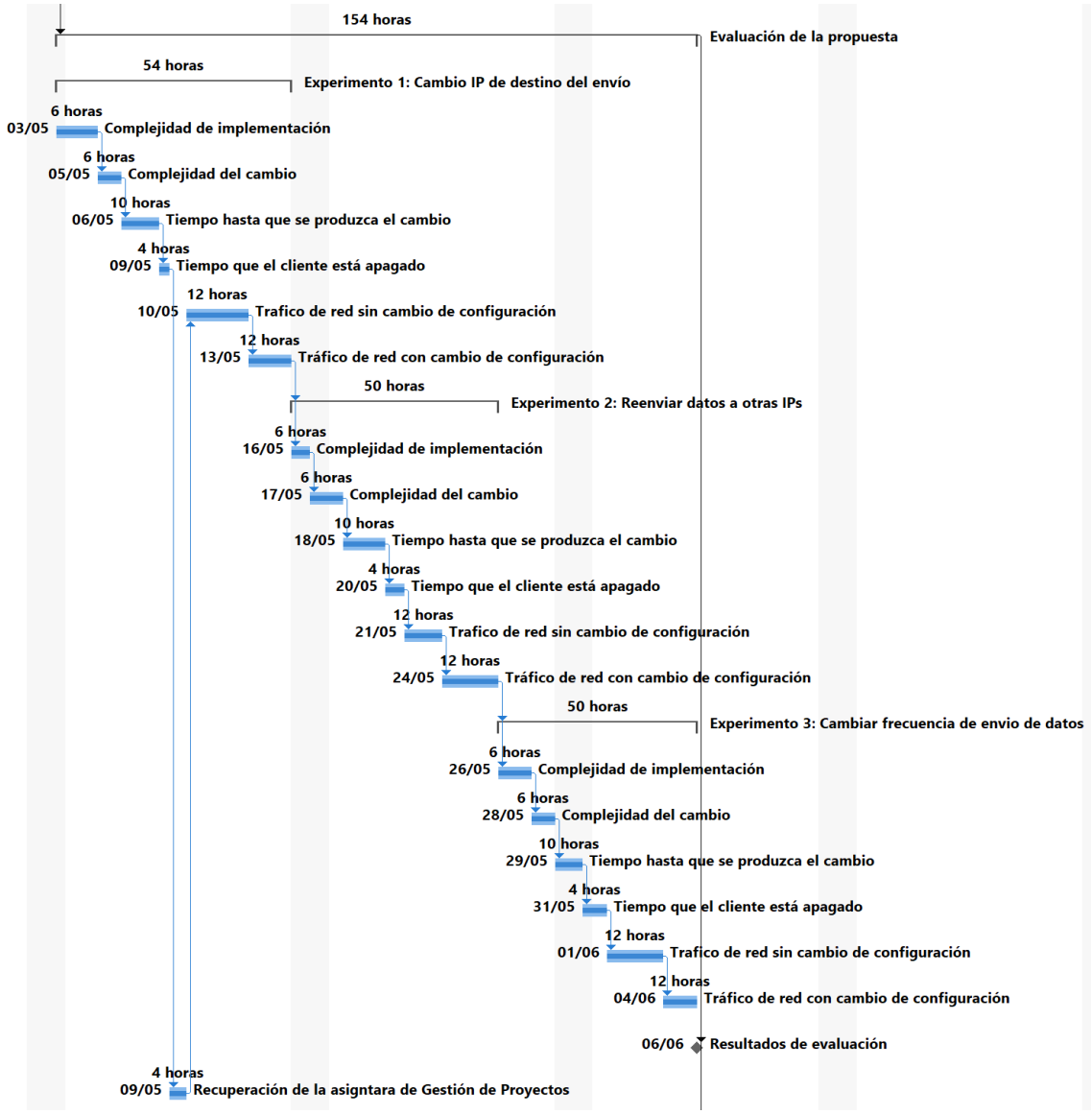

Figura 27. Tareas de evaluación de los experimentos después de acabar la evaluación

El mayor impacto de tiempo de esta tarea está en las subtareas de medición de tráfico de red que se había estimado con 2 horas para cada una sin tener en cuenta que hay que medir el tráfico de red de un intervalo de tiempo varias veces para poder calcular medias y desviaciones típicas.

También hay que destacar la aparición de la tarea *Recuperación de la asignatura de Gestión de Proyectos* que está directamente relacionada el riesgo *14 – Asignatura suspensa*.

Las tareas de documentación no se han tenido que modificar en cuanto a la duración debido a que se ha ampliado el plazo de presentación de la memoria del proyecto hasta el día 8 de julio. Por ello las tareas de documentación y preparación de la defensa simplemente se han movido para empezar el día 6 de junio en lugar del día 24 de mayo como se había planificado en la línea base anterior.

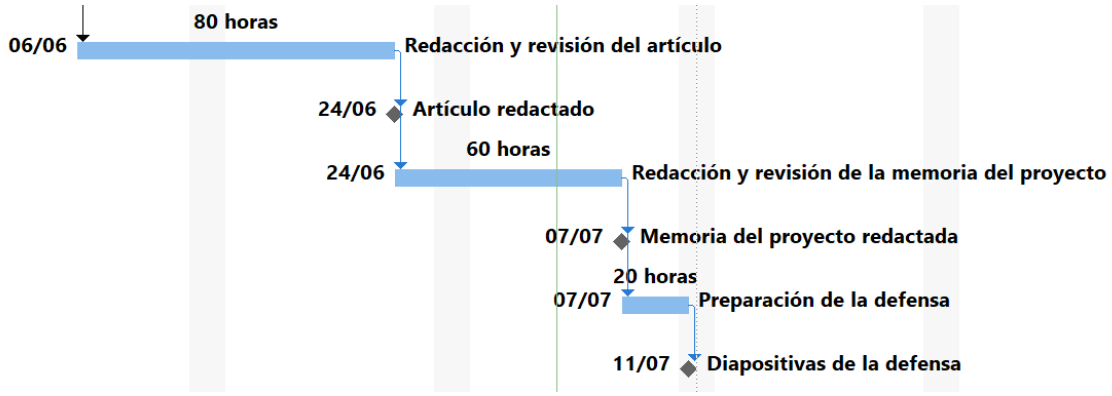

Figura 28. Tareas de documentación después de acabar la evaluación

10.1.4. Línea base final del proyecto

En la línea base final, solamente se pudieron haber producido en la fase de documentación ya que las demás fases ya se han ejecutado. En el periodo que comprende, desde la última fase y hasta el momento de entrega de la documentación, no se han producido cambios en la planificación. Queda fuera de esta documentación comentar la última tarea de preparación de la defensa que se producirá posteriormente a la entrega de la documentación.

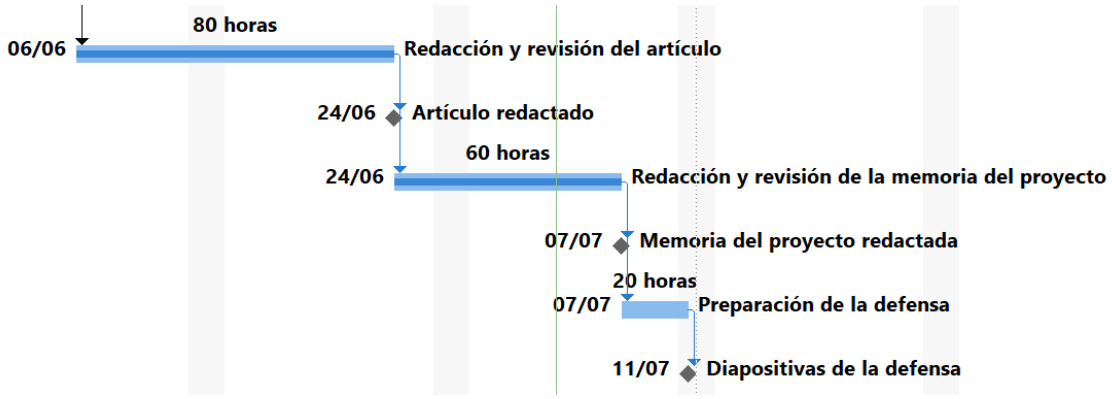

Figura 29. Tareas de documentación de la línea base final

10.2. Bitácora de incidencias del proyecto

A lo largo del proyecto se han producido las siguientes incidencias.

Tabla 7. Bitácora de incidencias del proyecto

Descripción	Fecha
Asignatura suspensa	09/05/2020
Problemas de conectividad Raspberry	21/05/2020

10.3. Seguimiento de riesgos

Se realiza una revisión de los riesgos en las siguientes fases del proyecto:

- Análisis de los riesgos inicial adjunto en el Anexo 2
- Análisis de los riesgos después de la definición de la línea de investigación
- Análisis de los riesgos después de la evaluación de la propuesta
- Análisis de los riesgos final

10.3.1. Análisis de los riesgos después de la definición de la línea de investigación

En el punto en el que se define la línea de investigación final, según la planificación, ya se ha realizado una fase de estudio de alternativas y pruebas de concepto. En esta fase, se ha podido analizar las tecnologías existentes y realizar pruebas con las escogidas para la implementación de la solución final. Gracias a eso, los riesgos referentes a la implementación del prototipo han disminuido su la probabilidad de ocurrir y por ello su impacto.

De igual manera se ha podido realizar las pruebas de conectividad de la placa y el sensor de temperatura haciendo que la probabilidad de defecto en la placa, sensor o que se produzcan problemas de conectividad también disminuye.

Por otro lado, al finalizar esta fase, ya se ha realizado una revisión de la literatura relacionada por lo que también disminuye la probabilidad de ocurrir de los riesgos relacionados con la propia investigación.

Finalmente, este punto aparece un nuevo riego de impacto medio-alto que es enfermedad por COVID-19.

Tabla 8. Análisis de los riesgos después de la definición de la línea de investigación

ID	Nombre	Responsable	Probabilidad	Impacto				Impacto
				Presup.	Planific.	Alcance	Calidad	
1	Requerir más tiempo del planificado en la búsqueda de literatura relacionada	Autor del proyecto	Muy Baja	Bajo	Alto	Medio	Alto	0,04
2	Encontrarse con una literatura más extensa de lo estimado	Autor del proyecto	Baja	Bajo	Alto	Medio	Alto	0,12
3	Dificultad para definir un objetivo adecuado dentro de la línea de investigación seleccionada	Autor del proyecto	Baja	Bajo	Alto	Medio	Crítico	0,12
6	Requerir significativamente más tiempo del planificado para el aprendizaje de alguna de las tecnologías necesarias para poder diseñar una solución	Autor del proyecto	Muy Baja	Medio	Crítico	Medio	Medio	0,08
7	Diseño de una solución que resulta muy poco apropiada para el cumplimiento de los objetivos	Autor del proyecto	Baja	Bajo	Alto	Alto	Alto	0,12
8	Problemas de implementación del prototipo de la propuesta diseñada	Autor del proyecto	Muy Baja	Bajo	Crítico	Alto	Alto	0,08

9	Prototipado: Defecto en el hardware de la placa	Autor del proyecto	Muy Baja	Medio	Alto	Alto	Bajo	0,04
10	Prototipado: Defecto en sensores o periféricos	Autor del proyecto	Muy Baja	Bajo	Alto	Alto	Bajo	0,04
11	Prototipado: Problemas de conectividad con los dispositivos IoT	Autor del proyecto	Baja	Bajo	Alto	Medio	Medio	0,12
16	Enfermedad COVID-19	Autor del proyecto	Alta	Bajo	Alto	Medio	Alto	0,28

10.3.2. Análisis de los riesgos después de la evaluación de la propuesta

Durante esta fase ha ocurrido el riesgo *15 – Asignatura suspensa* lo que ha provocado que se dedicaran horas para recuperar la signatura suspensa. Al finalizar esta fase, ya se conocen las notas de todas las asignaturas por lo que la probabilidad de ocurrir de este riego pasa a ser muy baja.

Finalmente, al día 5 de junio la situación del COVID-19 está bastante controlada por lo que se disminuye la probabilidad de enfermedad a medio.

Tabla 9. Análisis de los riesgos después de la evaluación de la propuesta

ID	Nombre	Responsable	Probabilidad	Impacto				Impacto
				Presup.	Planific.	Alcance	Calidad	
15	Asignatura suspensa	Autor del proyecto	Muy Baja	Bajo	Alto	Bajo	Bajo	0,04
17	Enfermedad	Autor del proyecto	Media	Bajo	Alto	Medio	Alto	0,20

10.3.3. Análisis de los riesgos final

En el análisis final de los riesgos se destacan los riesgos de mayor impacto que se han mantenido durante todo el desarrollo del proyecto y que no se han podido reevaluar a menor impacto.

Tabla 10. Análisis de los riesgos final

ID	Nombre	Responsable	Probabilidad	Impacto				Impacto
				Presup.	Planific.	Alcance	Calidad	
3	Surgimiento de un trabajo de igual contribución	Autor del proyecto	Baja	Bajo	Bajo	Medio	Crítico	0,24
6	Defecto en el PC de trabajo	Autor del proyecto	Baja	Alto	Crítico	Alto	Bajo	0,24
17	Enfermedad	Autor del proyecto	Media	Bajo	Alto	Medio	Alto	0,20

Durante todo el desarrollo del proyecto se ha mantenido el riesgo *6 – Defecto en el PC de trabajo* con un impacto medio-alto porque un fallo en el equipo podría suponer un retraso importante llegando hasta el punto de no poder entregar el trabajo en la convocatoria de junio.

Tampoco se ha bajado el impacto del riesgo *3 – Surgimiento de un trabajo de igual contribución* ya que hasta el momento de entregar puede surgir un trabajo en cualquier momento.

Finalmente, el tema del COVID-19 aunque parece que se va estabilizando sigue existiendo la posibilidad de enfermar lo que igual que el fallo en el PC de trabajo puede provocar que no se llegue a entregar.

11. Cierre del proyecto

11.1. Planificación final

La planificación final del proyecto se compone de 71 tareas. Entre ellas se separan en:

- 7 hitos
- 9 tareas resumen
- 55 tareas normales

La duración del proyecto es de 743. La planificación del proyecto se puede separar en las 5 fases siguientes:

11.1.1. Definición de la línea de investigación

Esta fase comprende desde el inicio hasta tener definida la línea de investigación final. Comprende un periodo muy largo, desde 23 de septiembre de 2019 hasta 9 de abril de 2020. Esto se debe a que durante el tiempo en el que ha habido clases presenciales se ha realizado un estudio de alternativas y pruebas de concepto sin avanzar en lo que sea el desarrollo de la solución final.

En esta fase también están las reuniones con el director para orientar el proyecto.

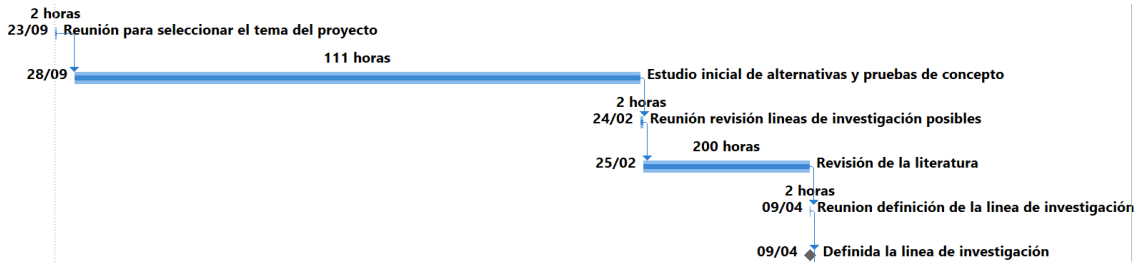

Figura 30. Fase de definición de la línea de investigación de la planificación final del proyecto

11.1.2. Implementación de la solución final

Esta fase comprende las tareas de diseño y la implementación de la solución final. Tal como se comenta en el apartado Análisis de los riesgos después de la evaluación de la propuesta, al haber realizado el periodo de estudio de alternativas y pruebas de concepto la propia implementación no llevado demasiado tiempo.

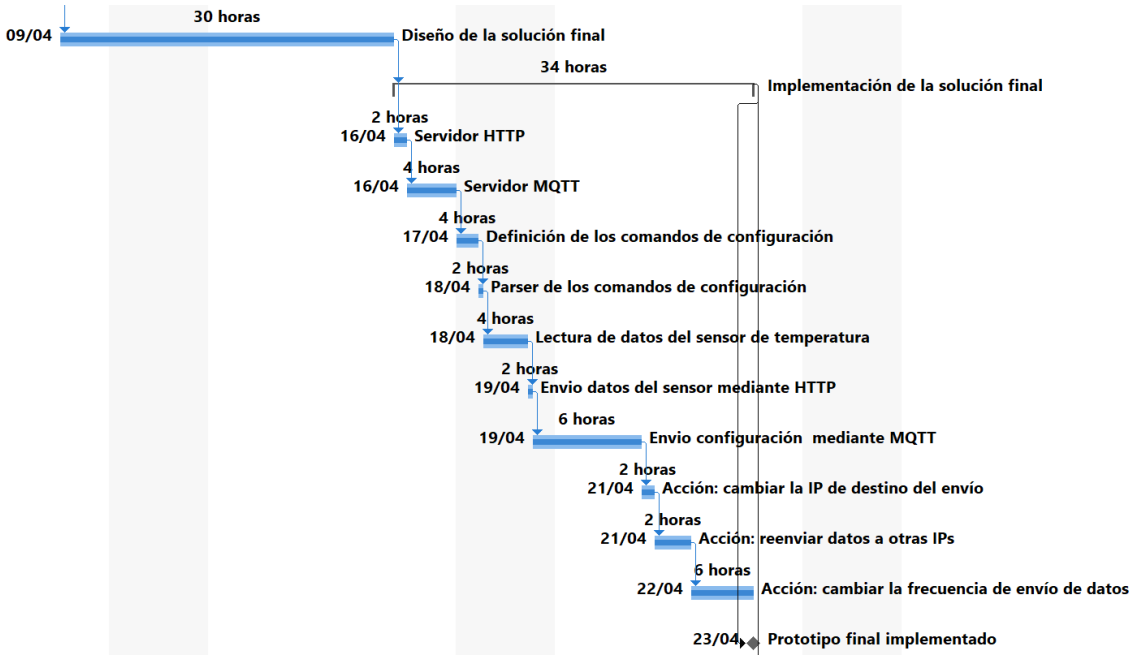

Figura 31. Fase de implementación de la solución de la planificación final

11.1.3. Fase de implementación de los experimentos

En esta fase se agrupan las tareas de desarrollos necesarios para realizar los experimentos para evaluar a la propuesta de la investigación. Se plantean 3 experimentos. El primero es el que más tiempo ha llevado implementarlo mientras que los demás eran más rápidos ya que una vez se había implementado la estructura base solamente había que realizar los cambios necesarios para los otros dos experimentos.

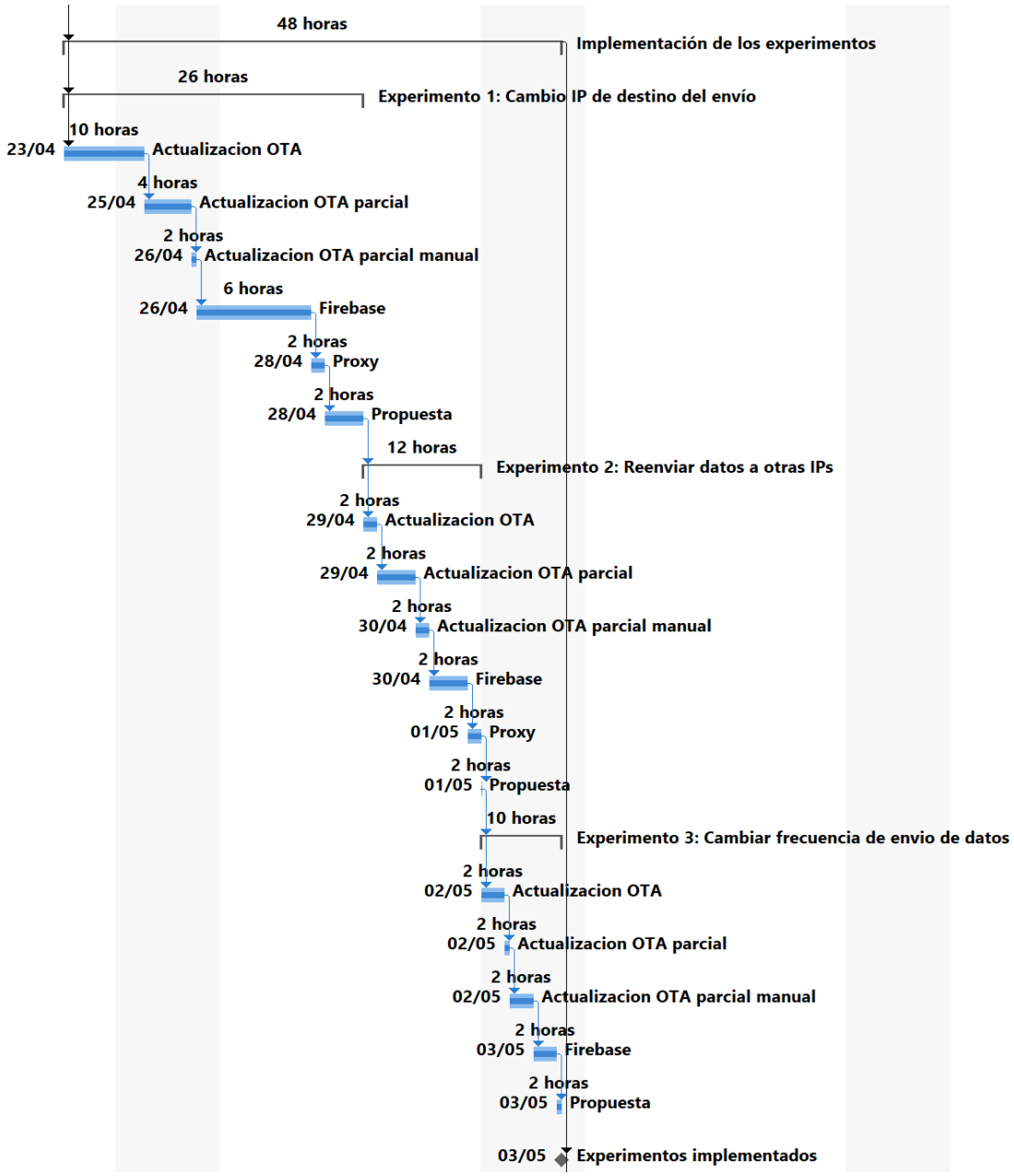

Figura 32. Fase de implementación de los experimentos de la planificación final

11.1.4. Fase de evaluación de la propuesta

La siguiente fase es la fase en la que se ejecutan los experimentos y se recogen los datos para poder realizar el posterior análisis y evaluación de la solución propuesta en la investigación. Es una de las fases que más tiempo ha llevado debido sobre todo a que los experimentos de mediciones de tráfico de red han llevado bastante tiempo.

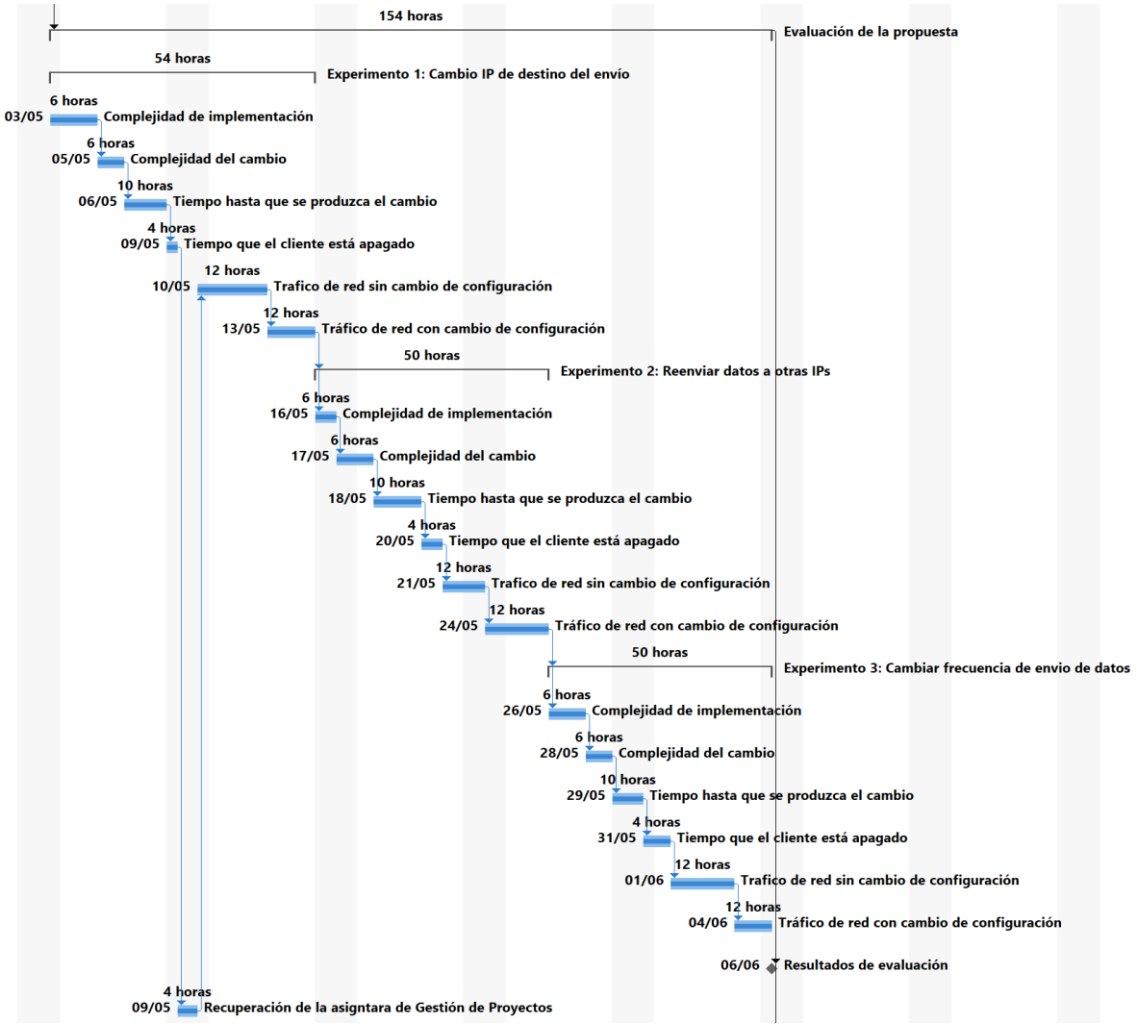

Figura 33. Fase de evaluación de la propuesta de la planificación final

11.1.5. Fase de documentación

La última fase es la fase de documentación del proyecto. Incluye la redacción del artículo y la memoria del proyecto, así como la preparación de la defensa.

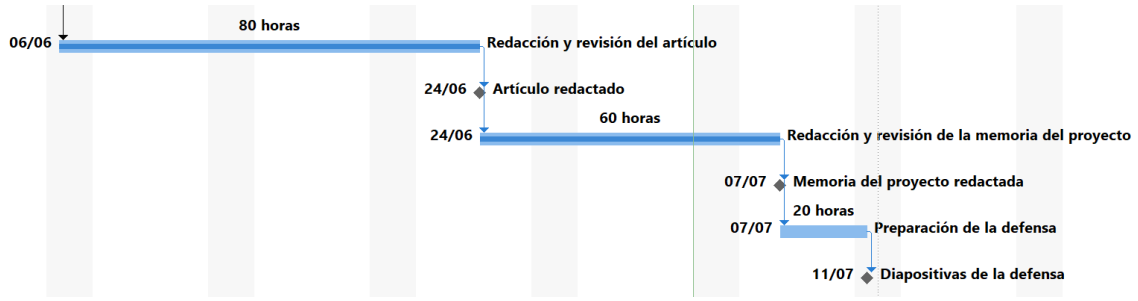

Figura 34. Fase de documentación de la planificación final

11.2. Presupuesto final de costes

Los costes materiales finales del proyecto ascienden a sen han mantenido en 53,58 €.

Tabla 11. Costes materiales finales

Descripción	Cantidad	Unidad	Precio unidad	Total

Compra de una placa Raspberry	1	articulo	29,29 €	29,29 €
Compra de un kit de sensores	1	kit	24,29 €	24,29 €
Total				**53,58 €**

El presupuesto final del proyecto asciende a 27.852,33 €.

Tabla 12. Presupuesto final del proyecto

Nivel de esquema	Nombre de tarea	Horas	Resumen	Autor	Director	Total
1	Reunión para seleccionar el tema del proyecto	2	No	100%	100%	180,00 €
1	Estudio inicial de alternativas y pruebas de concepto	111	No	100%		3.885,00 €
1	Reunión revisión líneas de investigación posibles	2	No	100%	100%	180,00 €
1	Revisión de la literatura	200	No	100%		7.000,00 €
1	Reunión definición de la línea de investigación	2	No	100%	100%	180,00 €
1	Diseño de la solución final	30	No	100%		1.050,00 €
1	**Implementación de la solución final**	**34**	**Sí**	**100%**		**1.190,00 €**
2	Servidor HTTP	2	No	100%		70,00 €
2	Servidor MQTT	4	No	100%		140,00 €
2	Definición de los comandos de configuración	4	No	100%		140,00 €
2	Parser de los comandos de configuración	2	No	100%		70,00 €
2	Lectura de datos del sensor de temperatura	4	No	100%		140,00 €
2	Envio datos del sensor mediante HTTP	2	No	100%		70,00 €
2	Envio configuración mediante MQTT	6	No	100%		210,00 €
2	Acción: cambiar la IP de destino del envío	2	No	100%		70,00 €
2	Acción: reenviar datos a otras IPs	2	No	100%		70,00 €
2	Acción: cambiar la frecuencia de envío de datos	6	No	100%		210,00 €
1	**Implementación de los experimentos**	**48**	**Sí**	**100%**		**1.680,00 €**
2	**Experimento 1: Cambio IP de destino del envío**	**26**	**Sí**	**100%**		**910,00 €**
3	Actualización OTA	10	No	100%		350,00 €
3	Actualización OTA parcial	4	No	100%		140,00 €
3	Actualización OTA parcial manual	2	No	100%		70,00 €
3	Firebase	6	No	100%		210,00 €
3	Proxy	2	No	100%		70,00 €
3	Propuesta	2	No	100%		70,00 €

2	**Experimento 2: Reenviar datos a otras IPs**	**12**	**Sí**	**100%**		**420,00 €**
3	Actualización OTA	2	No	100%		70,00 €
3	Actualización OTA parcial	2	No	100%		70,00 €
3	Actualización OTA parcial manual	2	No	100%		70,00 €
3	Firebase	2	No	100%		70,00 €
3	Proxy	2	No	100%		70,00 €
3	Propuesta	2	No	100%		70,00 €
2	**Experimento 3: Cambiar frecuencia de envio de datos**	**10**	**Sí**	**100%**		**350,00 €**
3	Actualización OTA	2	No	100%		70,00 €
3	Actualización OTA parcial	2	No	100%		70,00 €
3	Actualización OTA parcial manual	2	No	100%		70,00 €
3	Firebase	2	No	100%		70,00 €
3	Propuesta	2	No	100%		70,00 €
1	**Evaluación de la propuesta**	**154**	**Sí**	**100%**		**5.390,00 €**
2	**Experimento 1: Cambio IP de destino del envío**	**54**	**Sí**	**100%**		**1.890,00 €**
3	Complejidad de implementación	6	No	100%		210,00 €
3	Complejidad del cambio	6	No	100%		210,00 €
3	Tiempo hasta que se produzca el cambio	10	No	100%		350,00 €
3	Tiempo que el cliente está apagado	4	No	100%		140,00 €
3	Trafico de red sin cambio de configuración	12	No	100%		420,00 €
3	Tráfico de red con cambio de configuración	12	No	100%		420,00 €
2	**Experimento 2: Reenviar datos a otras IPs**	**50**	**Sí**	**100%**		**1.750,00 €**
3	Complejidad de implementación	6	No	100%		210,00 €
3	Complejidad del cambio	6	No	100%		210,00 €
3	Tiempo hasta que se produzca el cambio	10	No	100%		350,00 €
3	Tiempo que el cliente está apagado	4	No	100%		140,00 €
3	Trafico de red sin cambio de configuración	12	No	100%		420,00 €
3	Tráfico de red con cambio de configuración	12	No	100%		420,00 €
2	**Experimento 3: Cambiar frecuencia de envío de datos**	**50**	**Sí**	**100%**		**1.750,00 €**
3	Complejidad de implementación	6	No	100%		210,00 €
3	Complejidad del cambio	6	No	100%		210,00 €

3	Tiempo hasta que se produzca el cambio	10	No	100%		350,00 €
3	Tiempo que el cliente está apagado	4	No	100%		140,00 €
3	Trafico de red sin cambio de configuración	12	No	100%		420,00 €
3	Tráfico de red con cambio de configuración	12	No	100%		420,00 €
1	Recuperación de la asignatura de Gestión de Proyectos	4	No	100%		140,00 €
1	Redacción y revisión del artículo	80	No	100%		2.800,00 €
1	Redacción y revisión de la memoria del proyecto	60	No	100%		2.100,00 €
1	Preparación de la defensa	20	No	100%		700,00 €
Subtotal		**743**				**26.475,00 €**

Coster materiales	**53,58 €**
Costes indirectos (5%)	**1.323,75 €**

Total	**27.852,33 €**

Bibliografía

[1] A.B. Chebudie, R. Minerva, D. Rotondi, Towards a definition of the Internet of Things (IoT), 2014.

[2] G. Guan, W. Dong, Y. Gao, Jiajun Bu, Towards rapid and cost-effective prototyping of IoT platforms, in: 2016 IEEE 24th International Conference on Network Protocols (ICNP), IEEE, 2016: pp. 1–5. https://doi.org/10.1109/ICNP.2016.7785320.

[3] G. Lee, S. Heo, B. Kim, J. Kim, H. Kim, Rapid prototyping of IoT applications with Esperanto compiler, in: Proceedings of the 28th International Symposium on Rapid System Prototyping Shortening the Path from Specification to Prototype - RSP '17, ACM Press, New York, New York, USA, 2017: pp. 85–91. https://doi.org/10.1145/3130265.3138857.

[4] D. Zhang, J. Wan, C.-H. (Robert) Hsu, A. Rayes, Industrial technologies and applications for the Internet of Things, Computer Networks. 101 (2016) 1–4. https://doi.org/10.1016/j.comnet.2016.02.019.

[5] M.A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, Middleware for Internet of Things: A Survey, IEEE Internet of Things Journal. 3 (2016) 70–95. https://doi.org/10.1109/JIOT.2015.2498900.

[6] L.M. Vaquero, L. Rodero-Merino, Finding your Way in the Fog, ACM SIGCOMM Computer Communication Review. 44 (2014) 27–32. https://doi.org/10.1145/2677046.2677052.

[7] P. Ruckebusch, J. van Damme, E. de Poorter, I. Moerman, Dynamic Reconfiguration of Network Protocols for Constrained Internet-of-Things Devices, in: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, 2016: pp. 269–281. https://doi.org/10.1007/978-3-319-47075-7_31.

[8] V. Antinyan, M. Staron, A. Sandberg, Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time, Empirical Software Engineering. 22 (2017) 3057–3087. https://doi.org/10.1007/s10664-017-9508-2.

[9] J.P. Espada, V.G. Díaz, R.G. Crespo, O.S. Martínez, B.C.P. G-Bustelo, J.M.C. Lovelle, Using extended web technologies to develop Bluetooth multi-platform mobile applications for interact with smart things, Information Fusion. 21 (2015) 30–41. https://doi.org/10.1016/j.inffus.2013.04.008.

[10] H. Mukhtar, B.W. Kim, B.S. Kim, S.S. Joo, An efficient remote code update mechanism for wireless sensor networks, Proceedings - IEEE Military Communications Conference MILCOM. (2009). https://doi.org/10.1109/MILCOM.2009.5379862.

[11] M.L. Chiang, T.L. Lu, Two-Stage Diff: An efficient dynamic software update mechanism for wireless sensor networks, Proceedings - 2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing, EUC 2011. (2011) 294–299. https://doi.org/10.1109/EUC.2011.74.

[12] Y.C. Chang, T.Y. Chi, W.C. Wang, S.Y. Kuo, Dynamic software update model for remote entity management of machine-to-machine service capability, IET Communications. 7 (2013) 32–39. https://doi.org/10.1049/iet-com.2012.0459.

[13] W. Jin, D.H. Kim, IoT device management architecture based on proxy, Proceedings of 2017 6th International Conference on Computer Science and Network Technology, ICCSNT 2017. 2018-Janua (2018) 84–87. https://doi.org/10.1109/ICCSNT.2017.8343663.

[14] R. Bannatyne, G. Viot, Introduction to microcontrollers. I, in: Northcon/98. Conference Proceedings (Cat. No.98CH36264), IEEE, n.d.: pp. 238–248. https://doi.org/10.1109/NORTHC.1998.731542.

[15] K.J. Singh, D.S. Kapoor, Create Your Own Internet of Things: A survey of IoT platforms., IEEE Consumer Electronics Magazine. 6 (2017) 57–68. https://doi.org/10.1109/MCE.2016.2640718.

[16] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, IEEE Communications Surveys & Tutorials. 17 (2015) 2347–2376. https://doi.org/10.1109/COMST.2015.2444095.

[17] G. Kortuem, F. Kawsar, V. Sundramoorthy, D. Fitton, Smart objects as building blocks for the Internet of things, IEEE Internet Computing. 14 (2010) 44–51. https://doi.org/10.1109/MIC.2009.143.

[18] N. Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP, in: 2017 IEEE International Systems Engineering Symposium (ISSE), IEEE, 2017: pp. 1–7. https://doi.org/10.1109/SysEng.2017.8088251.

[19] A. Stanford-Clark, H. Truong, MQTT For Sensor Networks (MQTT-SN), 2013.

[20] C. Sarkar, A.U. Nambi S. N., R.V. Prasad, A. Rahim, R. Neisse, G. Baldini, DIAT: A Scalable Distributed Architecture for IoT, IEEE Internet of Things Journal. 2 (2015) 230–239. https://doi.org/10.1109/JIOT.2014.2387155.

[21] H.-C. Hsieh, K.-D. Chang, L.-F. Wang, J.-L. Chen, H.-C. Chao, ScriptIoT: A Script Framework for and Internet-of-Things Applications, IEEE Internet of Things Journal. 3 (2016) 628–636. https://doi.org/10.1109/JIOT.2015.2483023.

[22] N.K. Giang, M. Blackstock, R. Lea, V.C.M. Leung, Developing IoT applications in the Fog: A Distributed Dataflow approach, in: 2015 5th International Conference on the Internet of Things (IOT), IEEE, 2015: pp. 155–162. https://doi.org/10.1109/IOT.2015.7356560.

[23] L. Mainetti, V. Mighali, L. Patrono, P. Rametta, S.L. Oliva, A novel architecture enabling the visual implementation of web of Things applications, in: 2013 21st International Conference on Software, Telecommunications and Computer Networks - (SoftCOM 2013), IEEE, 2013: pp. 1–7. https://doi.org/10.1109/SoftCOM.2013.6671847.

[24] S.K. Datta, C. Bonnet, Easing IoT application development through DataTweet framework, in: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), IEEE, 2016: pp. 430–435. https://doi.org/10.1109/WF-IoT.2016.7845390.

[25] P. Patel, D. Cassou, Enabling high-level application development for the Internet of Things, Journal of Systems and Software. 103 (2015) 62–84. https://doi.org/10.1016/j.jss.2015.01.027.

[26] D. Mazzei, G. Baldi, G. Montelisciani, G. Fantoni, A full stack for quick prototyping of IoT solutions, in: 2016 Cloudification of the Internet of Things (CIoT), IEEE, 2016: pp. 1–5. https://doi.org/10.1109/CIOT.2016.7872915.

[27] T. Nepomuceno, T. Carneiro, T. Carneiro, C. Korn, A. Martin, A GUI-based Platform for Quickly Prototyping Server-side IoT Applications, in: Smart SysTech 2018; European Conference on Smart Objects, Systems and Technologies, 2018: pp. 1–9.

[28] B. Javed, M.W. Iqbal, H. Abbas, Internet of things (IoT) design considerations for developers and manufacturers, in: 2017 IEEE International Conference on Communications Workshops (ICC Workshops), IEEE, 2017: pp. 834–839. https://doi.org/10.1109/ICCW.2017.7962762.

[29] S.K. Datta, C. Bonnet, Extending DataTweet IoT Architecture for Virtual IoT Devices, in: 2017 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2017: pp. 689–694. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.108.

[30] A. van den Bossche, R. Dalce, T. Val, OpenWiNo: An open hardware and software framework for fast-prototyping in the IoT, in: 2016 23rd International Conference on Telecommunications (ICT), IEEE, 2016: pp. 1–6. https://doi.org/10.1109/ICT.2016.7500490.

[31] Y. Lin, H. Tseng, L. Chen, NB-IoTtalk: A Service Platform for Fast Development of NB-IoT Applications, IEEE Internet of Things Journal. (2018) 1–1. https://doi.org/10.1109/JIOT.2018.2865583.

[32] H. Chandra, E. Anggadjaja, P.S. Wijaya, E. Gunawan, Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development, in: 2016 22nd Asia-Pacific Conference on Communications (APCC), IEEE, 2016: pp. 115–118. https://doi.org/10.1109/APCC.2016.7581459.

[33] M. Su, B. Zhou, A. Fu, Y. Yu, G. Zhang, PRTA: A Proxy Re-encryption based Trusted Authorization scheme for nodes on CloudIoT, Information Sciences. 527 (2020) 533–547. https://doi.org/10.1016/j.ins.2019.01.051.

[34] F. Banaie, J. Misic, V.B. Misic, Priority-Based Caching Policy at a Hybrid IoT Proxy, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018: pp. 1–6. https://doi.org/10.1109/ICC.2018.8422307.

[35] H.G. Barron-Gonzalez, M. Martinez-Espronceda, S. Led, L. Serrano, C. Fischer, M. Clarke, New use cases for remote control and configuration of interoperable medical devices, in: 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), IEEE, 2013: pp. 4787–4790. https://doi.org/10.1109/EMBC.2013.6610618.

[36] Télit wireles solutions, AT Commands Reference Guide, (n.d.). https://www.sparkfun.com/datasheets/Cellular Modules/AT_Commands_Reference_Guide_r0.pdf.

[37] Q. Zhicong, L. Delin, W. Shunxiang, Analysis and Design of A Mobile Forensic Software System Based on AT Commands, in: 2008 IEEE International Symposium on Knowledge Acquisition and Modeling Workshop, IEEE, 2008: pp. 597–600. https://doi.org/10.1109/KAMW.2008.4810559.

[38] H.G. Barron-Gonzalez, M. Martinez-Espronceda, J.D. Trigo, S. Led, L. Serrano, Proposal of a novel remote command and control configuration extension for interoperable Personal Health Devices (PHD) based on ISO/IEEE11073 standard, in: 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, IEEE, 2014: pp. 6312–6315. https://doi.org/10.1109/EMBC.2014.6945072.

[39] A.E. Nikolaidis, S.S. Papastefanos, G.I. Stassinopoulos, M.-P.K. Drakos, G.A. Doumenis, Automating Remote Configuration Mechanisms for Home Devices, IEEE Transactions on Consumer Electronics. 52 (2006) 407–413. https://doi.org/10.1109/TCE.2006.1649657.

[40] B. Seo, S.H. Kim, H. Choi, The remote command and control system for outdoor robot, in: 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), IEEE, 2013: pp. 303–304. https://doi.org/10.1109/URAI.2013.6677371.

[41] Mozilla, MDN web docs - Web API reference: URL, (n.d.). https://developer.mozilla.org/es/docs/Web/API/URL (accessed June 30, 2020).

[42] G. Cueva-Fernandez, J.P. Espada, V. García-Díaz, C.G. García, N. Garcia-Fernandez, Vitruvius: An expert system for vehicle sensor tracking and managing application generation, Journal of Network and Computer Applications. 42 (2014) 178–188. https://doi.org/10.1016/j.jnca.2014.02.013.

Anexos

Anexo 1. Plan de gestión de riesgos

1. Cambios

1.1. Cambios en la versión 1.0

- Versión inicial

2. Metodología

Este Plan de Gestión de Riesgos es el plan para el conjunto de los riesgos identificados para el proyecto.

Los riesgos serán identificados por todos los interesados del proyecto durante todo el tiempo de duración de este. Los riesgos evolucionan y cambian por lo tanto se deberá mantenerlos actualizados en el momento.

La metodología para la gestión del riesgo tiene dos aspectos diferentes:

1. **Metodología General**, utilizada para el conjunto del Plan de Gestión de Riesgos.
2. **Metodología de Gestión de Riesgos**, utilizada para el ciclo de vida de todos los riesgos y las interrelaciones entre los riesgos identificados.

2.1. Metodología general

La metodología general describe el proceso de gestión de riesgos para el conjunto del proyecto, desde el inicio hasta el final de los trabajos relacionados con el proyecto.

1. **Identificación inicial de riesgos**. Se realiza una Tormenta de Ideas para identificación de los riesgos iniciales del proyecto.
2. **Elaboración del Plan de Gestión de Riesgos**. Se crea el Plan de Gestión, los criterios para la gestión de riesgos, el Registro de Riesgos y la Hoja de Datos de riesgos para todos los riesgos priorizados.
3. **Monitorización de Riesgos.** Durante todo el desarrollo del proyecto se realizan:
    a. **Evaluación de riesgos Regular**. De vez en cuando, el riesgo debe ser evaluado con el fin de asegurar que no representan una amenaza para el avance del proyecto.
    b. **Actualización del Plan de Gestión de Riesgos**. En todo momento se debe tener actualizado el Plan de Gestión de Riesgos. Se debe tener en cuenta que hay riesgos que pueden desaparecer en algún momento del desarrollo del proyecto y otro muchos que aparecerán sin que antes se hayan tenido en cuenta.
4. **Informe Final de Riesgos**. Al final del proyecto, una voluntad informe final muestra toda la evolución del riesgo.

2.2. Metodología de Gestión de Riesgos

Los aspectos más técnicos de la gestión de riesgos están involucrados en este aspecto de la metodología. Para ese proyecto se definen seis pasos:

1. **Planificar la Gestión de Riesgos**: Es el proceso por el cual se define cómo realizar las actividades de gestión de los riesgos para un proyecto.
2. **Identificar los Riesgos**: Es el proceso por el cual se determinan los riesgos que pueden afectar el proyecto y se documentan sus características.
3. **Realizar el Análisis Cualitativo de Riesgos**: Es el proceso que consiste en priorizar los riesgos para realizar otros análisis o acciones posteriores, evaluando y combinando la probabilidad de ocurrencia y el impacto de dichos riesgos.
4. **Planificar la Respuesta a los Riesgos**: Es el proceso por el cual se desarrollan opciones y acciones para mejorar las oportunidades y reducir las amenazas a los objetivos del proyecto.

5. **Monitorizar y Controlar los Riesgos**: Es el proceso por el cual se implementan planes de respuesta a los riesgos, se rastrean los riesgos identificados, se monitorizan los riesgos residuales, se identifican nuevos riesgos y se evalúa la efectividad del proceso contra riesgos a través del proyecto

2.3. Conceptos generales

Con el fin de evitar la ambigüedad, se deben definir tres conceptos utilizados en la Gestión de Riesgos:

- **Riesgo**: es cualquier evento que pueda causar un efecto en el proyecto. El efecto bien podría ser una amenaza o podría ser una oportunidad. La primera de ellas debe ser evitada y la segunda debe ser explotada.
- **Factor de riesgo**: Los factores de riesgo son circunstancias asociadas con el riesgo de que aumenta la posibilidad de conseguir los efectos descritos por el riesgo. Por lo general, los riesgos son imposibles o difíciles de medir o controlar directamente y las decisiones se toman controlando los factores de riesgo con el fin de evitar o minimizar (o potenciar) los efectos de riesgo.
- **Indicadores de Riesgo**: Los indicadores de riesgo son elementos asociados al riesgo y / o sus factores, que se puede medir y sirve como información acerca de la posibilidad de que se produzca el efecto del riesgo.

3. Herramientas y tecnologías

Se utilizarán diferentes técnicas para la gestión de riesgos.

3.1. Tormenta de Ideas

Esta técnica será utilizada en un principio para la identificación de la lista principal de los riesgos. Consiste en la discusión acerca de los principales objetivos del proyecto y todas las cosas que pueden ir mal.

3.2. Las evaluaciones periódicas

Regularmente, todos los indicadores serán evaluados de acuerdo con el plan establecido en la hoja de riesgo. Todos estos resultados se discutirán en las reuniones de riesgos.

4. Roles y responsabilidades

Se prevén lo siguientes roles:

- **Responsable de riesgos**: Responsable de la gestión de los riesgos es el propio autor del proyecto.
- **Identificador de riesgos**: Tanto el autor del proyecto como su director.

5. Categorías de riesgos

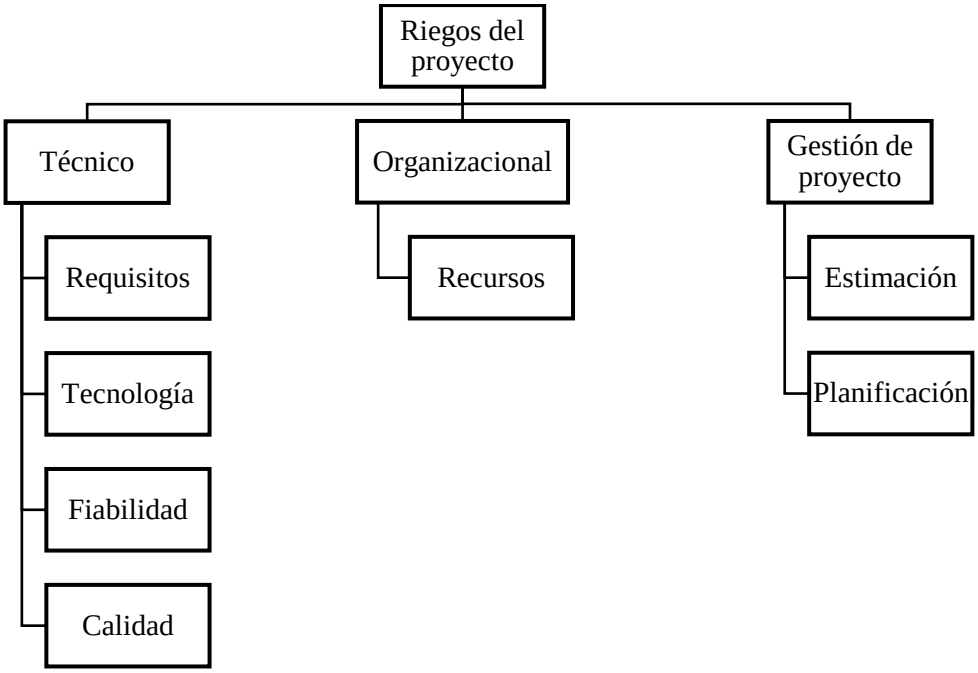

Figura 35. Categorías de los riegos

6. Definiciones de probabilidad

Tabla 13. Definiciones de probabilidad

Probabilidad	Definición
Muy Baja	(0%..20%] El valor usado en la matriz de probabilidad e impacto es: 10%
Baja	(20%..40%] El valor usado en la matriz de probabilidad e impacto es: 30%
Madia	(40%..60%] El valor usado en la matriz de probabilidad e impacto es: 50%
Alta	(60%..80%] El valor usado en la matriz de probabilidad e impacto es: 70%
Muy Alta	(80%..100%) El valor usado en la matriz de probabilidad e impacto es: 90%

7. Definiciones de impacto por objetivos

Tabla 14. Definiciones de impacto por objetivos

Condiciones definidas para las escalas de impacto de un riesgo sobre los objetivos principales de proyecto					
Objetivos del proyecto	Escalas relativas o numéricas				
	Muy bajo / 5%	Bajo / 10%	Moderado / 20 %	Alto / 40%	Muy alto / 80%

Coste	Incremento del coste insignificante	Incremento del coste <10%	Incremento entre 10-20%	Incremento entre 20-40%	Incremento del coste >40%
Tiempo	Incremento de tiempo insignificante	Incremento de tiempo <5%	Incremento de tiempo entre 5-10%	Incremento del tiempo entre el 10-20%	Incremento de tiempo >20%
Alcance	Reducción del alcance inapreciable	Afectadas áreas poco importantes del alcance	Afectadas áreas importantes del alcance	Reducciones del alcance inaceptables para el cliente	El resultado final del proyecto ni es realmente útil
Calidad	La degradación de la calidad es inapreciable	Solo las aplicaciones muy exigentes se ven afectadas	La reducción de la calidad requiere la aceptación del cliente	Reducción de la calidad inaceptable para el cliente	El resultado final del proyecto ni es realmente útil

8. Matriz de probabilidad e impacto

Tabla 15. Matriz de probabilidad e impacto

Probabilidad	**Muy Alta**	***0,90***	0,05	0,09	0,18	0,36	0,72
	Alta	***0,70***	0,04	0,07	0,14	0,28	0,56
	Media	***0,50***	0,03	0,05	0,10	0,20	0,40
	Baja	***0,30***	0,02	0,03	0,06	0,12	0,24
	Muy Baja	***0,10***	0,01	0,01	0,02	0,04	0,08
			0,05	***0,10***	***0,20***	***0,40***	***0,80***
			Muy Bajo	**Bajo**	**Medio**	**Alto**	**Crítico**
			Impacto				

9. Niveles de tolerancia

La tolerancia está en 0,17.

10. Formatos de la documentación

- Documentos en PDF

Anexo 2. Análisis inicial de los riesgos

Tabla 16. Riesgos iniciales del proyecto

ID	Nombre	Responsable	Probabilidad	Impacto				Impacto	Respuesta
				Presup.	Planific.	Alcance	Calidad		
1	Requerir más tiempo del planificado en la búsqueda de literatura relacionada	Autor del proyecto	Media	Bajo	Alto	Medio	Alto	0,20	Disponer de reservas de tiempo para poder analizar la literatura necesaria
2	Encontrarse con una literatura más extensa de lo estimado	Autor del proyecto	Media	Bajo	Alto	Medio	Alto	0,20	Priorizar la literatura de más a menos relevante y empezar a analizar por los artículos más relevantes
3	Dificultad para definir un objetivo adecuado dentro de la línea de investigación seleccionada	Autor del proyecto	Media	Bajo	Alto	Medio	Crítico	0,20	Analizar varias alternativas y consultar con el director del proyecto
4	Surgimiento de un trabajo de igual contribución	Autor del proyecto	Baja	Bajo	Bajo	Medio	Crítico	0,24	Al ser un proyecto de investigación se asume este riesgo
5	Defecto en el PC de trabajo	Autor del proyecto	Baja	Alto	Crítico	Alto	Bajo	0,24	Disponer de otro equipo que permita seguir avanzando en el proyecto
6	Requerir significativamente más tiempo del planificado para el aprendizaje de alguna de las tecnologías necesarias para poder diseñar una solución	Autor del proyecto	Media	Medio	Crítico	Medio	Medio	0,40	Disponer de reservas de tiempo para aprender las tecnologías necesarias
7	Diseño de una solución que resulta muy poco apropiada para el cumplimiento de los objetivos	Autor del proyecto	Media	Bajo	Alto	Alto	Alto	0,20	Consultar el diseño de la solución con el director del proyecto
8	Problemas de implementación del prototipo de la propuesta diseñada	Autor del proyecto	Media	Bajo	Crítico	Alto	Alto	0,40	Disponer de una reserva de tiempo para resolver posibles problemas de implementación
9	Prototipado: Defecto en el hardware de la placa	Autor del proyecto	Baja	Medio	Alto	Alto	Bajo	0,12	Disponer de reserva de dinero para en caso de producirse un defecto en la placa poder adquirir una nueva

10	Prototipado: Defecto en sensores o periféricos	Autor del proyecto	Baja	Bajo	Alto	Alto	Bajo	0,12	Disponer de reserva de dinero para en caso de producirse un defecto en la placa poder adquirir un nuevo
11	Prototipado: Problemas de conectividad con los dispositivos IoT	Autor del proyecto	Media	Bajo	Alto	Medio	Medio	0,20	Realizar pruebas de conectividad antes de empezar el proyecto
12	Evaluación: Errores en el diseño de los experimentos	Autor del proyecto	Baja	Bajo	Medio	Alto	Medio	0,12	Consultar el diseño de los experimentos con el director antes de recolectar datos
13	Evaluación: Errores en la medición de los datos	Autor del proyecto	Baja	Bajo	Alto	Alto	Medio	0,12	Revisar detenidamente la ejecución de cada medición de los experimentos
14	Asignatura suspensa	Autor del proyecto	Baja	Bajo	Alto	Bajo	Bajo	0,12	Tener reserva de horas para dedicar a recuperar las posibles a asignaturas suspensas
15	Falta de servicio de internet	Autor del proyecto	Muy Baja	Bajo	Alto	Bajo	Medio	0,04	Se asume este riesgo

Anexo 3. Datos recogidos en los experimentos

Complejidad de implementación

Tabla 17. Complejidad de implementación al cambiar de servidor de destino

	Líneas	Caracteres	Funciones no implementadas	Módulos incluidos	Módulos externos	
Pesos	2	1	3	4	5	
						Total
Actualización OTA	44	1224	10	1	1	1347
OTA Parcial	46	1284	10	1	2	1416
Software Parcialmente Actualizable	46	1284	10	1	2	1416
Proxy	43	1180	10	1	1	1301
Firebase	93	2584	16	3	2	2828
Flex-Request	27	652	3	0	1	720

Tabla 18. Complejidad de implementación al reenviar peticiones a múltiples destinos

	Líneas	Caracteres	Funciones no implementadas	Módulos incluidos	Módulos externos	
Pesos	2	1	3	4	5	
						Total
Actualización OTA	57	1435	12	1	1	1590
OTA Parcial	49	1394	11	1	2	1535
Software Parcialmente Actualizable	49	1394	11	1	2	1535
Proxy	43	1180	10	1	1	1301
Firebase	98	2699	17	3	2	2956
Flex-Request	27	652	3	0	1	720

Tabla 19. Complejidad de implementación al modificar la frecuencia de envío

	Líneas	Caracteres	Funciones no implementadas	Módulos incluidos	Módulos externos	
Pesos	2	1	3	4	5	
						Total
Actualización OTA	44	1204	10	1	1	1327
OTA Parcial	47	1408	12	1	2	1548
Software Parcialmente Actualizable	48	1408	12	1	2	1550
Firebase	101	2889	18	3	2	3155
Flex-Request	27	652	3	0	1	720

Complejidad de realizar un cambio de configuración

Tabla 20. Complejidad de realizar un cambio al modificar de servidor de destino

	Pulsaciones de teclas	Botón Izquierdo del Ratón	Botón Derecho del Ratón	Doble Clic	Rueda del Ratón	Movimiento del Ratón	
Pesos	1	2	1	1	1	0,5	
							Total
Actualización OTA	27	2	0	0	0	57	59,5
OTA Parcial	35	2	0	0	0	54	66
Software Parcialmente Actualizable	135	2	0	0	0	59	168,5
Proxy	2	2	0	0	0	55	33,5
Firebase	29	9	0	0	10	84	99
Flex-Request	34	2	0	0	0	51	63,5

Tabla 21. Complejidad de realizar un cambio al reenviar peticiones a múltiples destinos

	Pulsaciones de teclas	Botón Izquierdo del Ratón	Botón Derecho del Ratón	Doble Clic	Rueda del Ratón	Movimiento del Ratón	
Pesos	1	2	1	1	1	0,5	
							Total
Actualización OTA	236	2	0	0	0	54	267
OTA Parcial	248	2	0	0	0	50	277
Software Parcialmente Actualizable	338	2	0	0	0	61	372,5
Proxy	37	2	0	0	0	56	69
Firebase	55	9	0	0	8	82	122
Flex-Request	55	3	0	0	0	57	89,5

Tabla 22. Complejidad de implementación al modificar la frecuencia de envío

	Pulsaciones de teclas	Botón Izquierdo del Ratón	Botón Derecho del Ratón	Doble Clic	Rueda del Ratón	Movimiento del Ratón	
Pesos	1	2	1	1	1	0,5	
							Total
Actualización OTA	29	2	0	0	0	51	58,5
OTA Parcial	37	2	0	0	0	58	70
Software Parcialmente Actualizable	138	2	0	0	0	62	173
Firebase	31	9	0	0	9	83	99,5
Flex-Request	32	2	0	0	0	56	64

Tiempo de inactividad del dispositivo

Tabla 23. Tiempo de inactividad del dispositivo al cambiar el destino del envío

	Actualización OTA	OTA Parcial	Software Parcialmente Actualizable	Proxy	Firebase	Flex-Request
1	1278	381	166	352	733	374
2	1360	460	595	359	1300	579
3	1341	123	793	462	1200	674
4	1350	289	339	373	1288	559
5	1352	848	636	776	374	334
6	1357	384	485	224	1226	860
7	1346	77	951	324	952	217
8	1361	945	955	533	770	421
9	1389	748	205	236	577	187
10	1380	693	409	790	709	200
Media	1351,40	494,80	553,40	442,90	912,90	440,50
Desviación estándar	29,73	300,45	284,47	201,37	328,32	224,08

Tabla 24. Tiempo de inactividad del dispositivo al reenviar peticiones a múltiples destinos

	Actualización OTA	OTA Parcial	Software Parcialmente Actualizable	Proxy	Firebase	Flex-Request
1	1327	789	454	588	2022	345
2	1345	221	510	787	933	637
3	1343	56	376	533	1303	254
4	1332	150	382	627	1161	868
5	1357	912	417	803	957	626
6	1330	954	494	284	580	454
7	1332	694	399	262	1025	419

8	1342	642	457	599	969	366
9	1341	239	641	345	561	284
10	1352	204	469	614	1021	772
Media	1340,10	486,10	459,90	544,20	1053,20	502,50
Desviación estándar	9,83	344,55	78,35	191,41	409,80	211,30

Tabla 25. Tiempo de inactividad al modificar la frecuencia de envío

	Actualización OTA	OTA Parcial	Software Parcialmente Actualizable	Firebase	Flex-Request
1	792	409	503	146	251
2	781	506	503	526	261
3	586	510	502	234	264
4	585	417	511	784	265
5	608	515	502	391	264
6	648	610	502	530	262
7	768	317	503	417	263
8	635	700	502	986	259
9	579	512	503	405	267
10	586	413	503	1433	261
Media	656,80	490,90	503,40	585,20	261,70
Desviación estándar	88,34	109,55	2,72	385,89	4,40

Tráfico de red

Tabla 26. Tráfico de red sin cambios de configuración

	Actualización OTA	OTA Parcial	Software Parcialmente Actualizable	Proxy	Firebase	Flex-Request
1	374217	346077	346638	371802	345299	383137
2	359157	351942	350673	367510	336194	387209
3	366120	349614	348543	379003	346307	384815
4	366849	341379	344490	364335	346118	383244
5	372215	349493	345252	360586	348107	383102
Media	367711,60	347701,00	347119,20	368647,20	344405,00	384301,40
Desviación estándar	5896,17	4106,76	2514,33	7107,51	4703,10	1776,96

Tabla 27. Tráfico de red concambios de configuración

	Actualización OTA	OTA Parcial	Software Parcialmente Actualizable	Proxy	Firebase	Flex-Request
1	395600	349109	364746	364746	392728	391967
2	361961	352112	357741	357741	374012	383990
3	368182	361714	370770	370884	387338	384064
4	374123	348911	395686	395878	389398	382415
5	374891	366005	358648	358648	373069	382327
Media	374951,40	355570,20	369518,20	369579,40	383309,00	384952,60
Desviación estándar	12665,21	7821,17	15539,94	15623,09	9128,50	4007,84

Anexo 4. Artículo científico

Flex-request: library to make remote changes in the communication of IoT devices

Karol Mateusz Ciok (a) Jordán Pascual Espada (a)

a, Department of Computer Science, University of Oviedo, Asturias, Spain

Abstract

In recent years, Internet of Things (IoT) systems have changed the way we live, work and do businesses in many areas, even those that until recently seemed unlikely. Some areas that can benefit from their application are, among others, healthcare, smart cities, industrial automation, smart agriculture, intelligent transportation systems, smart logistics, and emergency response. This research work proposes a novel alternative that allows the creation of IoT systems capable of making remote changes in devices' communication in a fast and agile manner. Our proposal gives way to some of the most common changes in communication made during the development and maintenance phase in IoT systems, like changing the destination of data transmission, sending the data to multiple destinations, and changing the frequency of sending data. Our solution, which is used in the programs, is loaded on the device. When the device starts, it connects to a configuration server in the background and listens for changes. The changes are sent to the configuration server using specified commands. When a change is detected, the command is processed, and the change in communication is applied without stopping the running program. We designed experiments to evaluate the complexity of the programs developed using our proposal and the actions needed to make a change.

Keywords: Internet of things, Sensors, Remote configuration, Software update, MQTT protocol.

1. Introduction

In recent years, Internet of Things (IoT) systems have changed the way we live, work and do business in many areas, even those that until recently seemed unlikely. These IoT systems are defined as the interconnection of devices sensing and actuating with the real world, providing the ability to share information and coordinate themselves to carry out tasks [1]. Some areas that can benefit from their application are, among others, healthcare, smart cities, industrial automation, smart agriculture, intelligent transportation systems, smart logistics, and emergency response [2][3]. However, the number of fields that can accommodate an IoT solution is vast and continues expanding as more devices achieve internet connection capabilities. By 2020, more than 50 billion devices are expected to be connected to the Internet of Things [4].

IoT Devices usually obtain data from their near environment using the sensors. Devices also can perform actions using actuators like relays. In IoT systems, devices are distributed into networks to exchange data and perform tasks together. Communication between devices and services is one of the key points of IoT systems. However, when the number of devices and services increases, so does the difficulty of making changes in communications.

In many cases, IoT systems are made up of many small scattered devices. Because of that, some modifications may involve many updates. For example, if it is required to change the IP of the centralized service, developers might have to update the software of devices.

Despite the potential benefits of using IoT devices, the uptake by the industry of innovative large-scale sensor/actuator applications is slow. One of the main reasons is the lack of built-in support for the maintenance of such devices. Although the standards provide many options to configure network protocols, it is difficult to change the configuration settings at run-time; this implies that the entire network needs to be reconfigured offline if changes are required [5].

Due to this, it is interesting for developers to make changes in communication in an efficient manner from the start of the software's development for those devices. Those changes could bring advantages to the developers in different phases of the design and implementation of the IoT system, such as the ability to get rapid prototyping and deploy of alternatives to find the most suitable one for the system's needs. IoT systems can be based on many communication topologies, like client-server, trees, meshes, P2P, etc. In many cases, IoT systems are innovative solutions, but the creation of the system carries a high degree of uncertainty and restrictions, so it is often necessary to try different approaches and architectures to find an appropriate way to create the IoT system.

The complexity of the software in IoT devices could be very low as these devices usually do not require doing complex tasks. For example, some devices are limited to getting data through a sensor and sending it to a server. However, the complexity of the device software could be conditioned by a few factors, like the programing language, the code length, and the structure and particularities of the framework used to access the device's hardware and communications [6][7]. IoT tends to have some uncertainty, and writing less line of code enables programmers to make more effective changes, which allows them to test more alternatives and choose the one that's most suitable for their needs [8][9].

Another critical aspect of making changes when the system is already running is the system downtime. Applying the changes could require stopping the current program and executing the new one. This process is slower when the new software must be sent remotely to a device, such as an Over-the-Air (OTA) update. Because of this, there is some time where the device is neither collecting nor sending the sensor data [10]. For example, for a power plant, collecting and analyzing this data is critical to safely manage the plant and avoid potential important data loss.

Nowadays, there are certain alternatives to resolve the problem of remotely reconfiguring some aspects of IoT devices. Some of the most common alternatives can be grouped in the following categories:

- *Devices consulting its communication parameters from external services*. In this category, we can include services like Firebase Remote Config[*]. This involves developing the software in a specific and slightly more complex way so that it integrates with the remote configuration service. The main disadvantage of these services is that they enable little changes in the configuration parameters.
- *Devices which use modular software*. Some examples could be *node-supervisor*[†] or *hotswap*[‡] node module. These solutions also have some downsides. *node-supervisor* restarts the program when one of the files it is watching changes. On the other hand, *hotswap* reloads the modules when it detects a change without stopping the program. However, in this case, the programs must be done in modular way, forcing the developer to foresee all possible future changes and maintaining the modules small because reloading bigger modules can cause some performance issues.
- *Devices managed by an IoT platform that uses a proxy or a gateway server*. Some examples of these platforms could be [5] and [11]. These kinds of platforms tend to need a big architecture setup.
- *Full or partial software update based on OTA (Over the Air update).* Although there are some solutions that try to minimize the size of the update [8] or [9], many OTA updates still

[*] Firebase Remote Config - https://firebase.google.com/docs/remote-config

[†] node-supervisor - https://github.com/petruisfan/node-supervisor

[‡] hotswap - https://github.com/rlidwka/node-hotswap

need to restart the device in order to apply the update.

The main objective of this research work is to provide an alternative that leads to the creation of IoT systems capable of making remote changes to devices' communication in a fast and agile way. This will allow some of the most common changes in communication to be made during the development and maintenance phase in IoT systems, like changing the destination of data transmission, sending the data to multiple destinations, and changing the frequency of sending data. To be considered appropriate, the solution should comply with the following aspects:

- The complexity of implementing software capable of remote configuration must be reduced, not involving highly additional costs for developers compared to developing an IoT software without this feature.
- The cost and complexity of doing a change in a device communication configuration must be reduced. Developers should be able to make changes quickly and efficiently compared to current alternatives for making changes in IoT devices software.
- Finally, the proposal should minimize the time that the IoT device is off or not working correctly after making a change in communication. In most IoT systems, it is common that after a change or update in the device software, it remains inactive for a while. This inactivity time must be short. It could be critical for many IoT systems because they could lose relevant data during this time.

The remainder of this paper is structured as follows: Section 2 presents the background and related work on software update problems on IoT devices in literature. Section 3 describes the proposed solution. Section 4 describes some use cases of the proposal. Section 5 describes the evaluation process and the obtained results. Section 5 concludes the study and proposes future works.

2. Background

A microcontroller is a single integrated circuit that, as a minimum, contains the necessary elements of a complete computer system and can contain additional peripheral modules, such as serial and timer units. A microcontroller must have more than just a CPU, a program, and a data memory to be able to solve real-world problems. In addition to that, it must contain hardware allowing access to information from the outside world. Once it gathers information and processes the data, it must also be able to affect change on some portion of the outside world [12].

There are hundreds of different microcontrollers with a variety of features. Some examples of those are: Siemens S, Mitsubishi FX, Kunbus PR, Arduino, Raspberry Pi and Intel Galileo [13][14]. They usually fashion an array of GPIO (General Purpose Input/Output) pins to allow them to connect peripherals.

These microcontrollers are the fundamental basis for the next step in the ladder: smart objects. Smart objects are autonomous physical/digital objects augmented with sensing, processing, and network capabilities. They carry chunks of application logic that let them make sense of their local situation and interact with human users. They sense, log, and interpret what is occurring within themselves and the world, act on their own, intercommunicate with each other, and exchange information with people [15].

Smart objects can be very different and can be classified in many different ways, based on various features like computing capacity, communication skills, intelligence level, and purpose [3].

Managing IoT devices, especially if its number is high, entails some challenges like heterogeneity, scalability, interoperability, security, and privacy [16]. Also, there is the lack of build-in possibility to reconfigure the device dynamically without shutting down the device [5]. There are many commercial IoT management platforms like: Balena.io[*], Thethings.io[†], Cloudino[‡], Google Cloud IoT Core[§], Microsoft Azure IoT Central[**] and AWS IoT Device Management[††]. These

[*] https://www.balena.io/
[†] https://thethings.io/
[‡] http://www.cloudino.io/
[§] https://cloud.google.com/iot-core
[**] https://azure.microsoft.com/es-es/services/iot-central/
[††] https://aws.amazon.com/es/iot-device-management/

platforms allow adding or removing devices and have configured OTA updates. There are also other services like Firebase Remote Config [17] that enable setting some specific application parameters on the cloud and offers an API to update those parameters in the application if they change on the cloud. Some research works focus on changing some specific configuration parameters in medical devices [18][19][20], in home devices [21], or in remote controls [22]. There are also AT commands for configuring WiFi routers and Bluetooth devices.

There are many solutions based on OTA updates that try to eliminate or minimize some negative aspects of these updates. Some propose updates based on diff [8][9] to minimize the size of the update that get sent. Others reduce the time offline [10] or energy consumption [23]. Some solutions focus on heterogeneity [11] and open standards [24]. Some proposed solutions use proxy to reinforce security [25] or to analyze the network traffic [26].

There are different alternatives in terms of topology and communication patterns of IoT devices. Some research works use a central gateway or proxy to manage the IoT devices network. These solutions enable dynamic adding and removing of IoT devices and also modification of some communication parameters [5][24]. The most common update mechanism using that approach consists of installing the update on all the devices registered in the gateway. Other researchers present a more decentralized approach. Some enable the simultaneous updates of multiple distributed nodes involved in a running service to prevent mismatching communication [27]. Others focus on node failure tolerance, basing their solution on moving the aspect of the update process to the devices and leaving the central server only responsible for indicating when the updates are available [28][29].

As IoT systems bring many challenges [16], one important aspect when working with IoT Systems is fast and efficient prototyping. There are some research works that try to reduce development time by unifying the access to the most common IoT features [30], abstracting the business logic so that it can be written only once for different devices [31], using containers [32] using a macro programming framework [33] and graphic programming languages [34][35].

3. Proposed platform

This research work proposes a novel library, which can be used in the device software. This library allows modifying dynamically the communication characteristic of the software executed on the device without altering the software code. This library can be used as a standard library to send requests over the Internet and allows developers to change dynamically and easily some of the main aspects of the communication, such as:

- Changing the destination IP of the requests
- Allowing resending request to other devices or servers, in addition to the main destination
- Changing the frequency of requests being sent

These changes do not imply changes in the software implementation, nor running a new program or restarting the device.

The proposed platform has two parts. The main part is the communication library, which is used in the device's program, and the other part is the configuration server. This server is a simple MQTT broker that exposes a topic where the configuration updates can be published for the IoT devices to capture and apply the change. The configuration changes are sent using the MQTT protocol that is a Publish-Subscribe protocol.

The communication library is used in programs loaded on IoT devices. When the program starts, the library connects to the configuration server in a background process. This process is automatic and imperceptible to the developer. The connection listens to the communication configuration changes, and when it detects a new configuration change, it updates the local internal configuration of the library and applies the necessary changes.

The communication scheme between the library and the MQTT broker is depicted in figure 1.

1) IoT devices using the library have specified in the program code the services where they send the recorded data.
2) A developer can make a change in communication configuration by sending a command to the Configuration Server.

3) The library inside the IoT devices detects the configuration change that had been sent by the developer to the Configuration Server through a MQTT Broker, processes the command and applies the communication change.

4) Once the received command is processed by the device, its communication changes for example, it starts sending the recorder data to a new service.

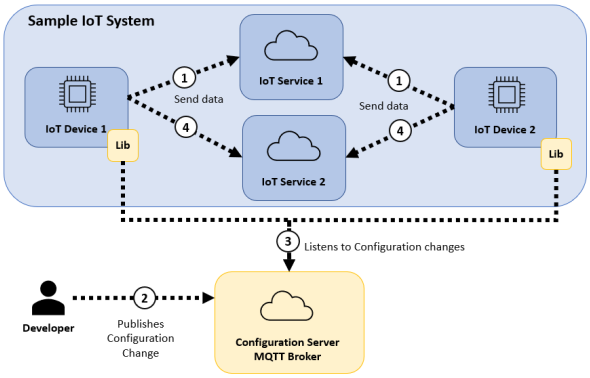

Figure 1. Proposal conceptual scheme

The proposed library defines a *request* method which accepts the following arguments:

url *[optional]* – url or IP of the destination server. Must include the protocol. When passed to the call, it is parsed with URL of WebAPI and overwritten by properties explicitly set in the *options* object.

options *[required]* – object with the following request options:

- **id** *[required]* – request id. Used to differentiate the different uses of the library inside the same program.
- **data** *[required]* – a function that returns a *Promise* with the data to be sent in the request.
- **host** *[optional, defaults to "localhost"]* – host of the destination server.
- **protocol** *[optional, defaults to "http:"]* – protocol used.
- **method** *[optional, defaults to "GET"]* – http method.
- **port** *[optional, defaults to 80]* – destination port.
- **requestFrecuency** *[optional, defaults to 1000]* – frequency (in milliseconds) of sending request.
- **forwardUrls** *[optional, defaults to empty array]* – array of URLs to forward the request sending.

callback – the function to be executed when the response is received.

In order to use the library, the developers have to include the library inside the program. The library implementation has to be compatible with the used programming language. Then, developers have to configure the communication with the configuration server for the entire program. After that, we can use the method offered by the library to send requests over the Internet. Each use of the request method must be identified with an id to allow independently modifying each use. Figure 2 represents an example of a program using the proposed library.

```
01: const sl = require("node-dht-sensor");
02: const FlexRequest = require("flex-request");
03: const factory = FlexRequest({
04:   masterServerUrl: "mqtt://192.168.0.27:5000",
05: });
06:
07: // Sending the temperature
08: factory.request(
09:   {
10:     id: 1,
11:     protocol: "http:",
12:     host: "192.168.0.27",
13:     port: 6001,
14:     data: new Promise((resolve, reject) => {
15:       resolve({
16:         temperature: sl.read(11, 4).temperature
,
17:       });
18:     }),
19:     method: "POST",
20:   },
21:   (err, res, body) => {
22:     if (err) {
```

```
23:       return console.log(`Error: ${err.message}
`);
24:     }
25:     console.log(`${body}`);
26:   }
27: );
28:
29: // Sending the humidity to another server
30: factory.request(
31:   {
32:     id: 2,
33:     protocol: "http:",
34:     host: "158.54.0.100",
35:     port: 7000,
36:     data: new Promise((resolve, reject) => {
37:       resolve({
38:         humidity: sl.read(11, 4).humidity,
39:       });
40:     }),
41:     method: "POST",
42:   },
43:   (err, res, body) => {
44:     if (err) {
45:       return console.log(`Error: ${err.message}
`);
46:     }
47:     console.log(`${body}`);
48:   }
49: );
```

Figure 2. Example implementation for a device software

When this program starts, it will do two things. In the foreground, it will read the sensor's temperature and send it to http://192.168.0.27:6001 every second and read the sensor's humidity and send it to http://158.54.0.100:7000 every second as well. When a response is received, it will be logged to the console. In the background, as explained earlier, it will connect to the MQTT broker and listen for configuration changes. When a configuration change is detected, it will parse the command received and update the local internal state. Then each request method whose id is equal to the one received in the command will change its communication configuration. That way, we may have many devices where the request method has the same id, and when the configuration change is made to that id, all devices will change their configuration.

In order to change a communication parameter, the administrator has to send a MQTT request to the configuration server with a specific command that enables the required change. The configuration server publishes the command to all the devices that are listening. In order to cover the communication changes, the following commands have been designed:

- set *<id>* *<destinationURL>* - changes the URL of the destination server of the request with *<id>*. That command can be useful to change the server to which the data is sent in a Client-Server scheme because it has been decided to change the current server to a new one. Another use may be to change the device with which the communication is done
    - *master-request* – is a reserved id for referring to the MQTT broker URL
- forward *<id>* *<forwardURL1>* *<forwardURL2>* - forwards the send request with *<id>* to the URLs form *<forwardURL>* arguments. It can have as many forward URLs as one wants, separated by spaces. This command can be used if there is a need to separate the data processing onto different servers, for instance.
- set *<id>* frequency *<frequencyValue>* - changes the frequency of the request sending to the *<frequencyValue>* (in milliseconds). This can be used to reduce the impact on network traffic as there might not be a need to send the data every second all the time.

The current commands were designed to be used in what may seem like the common cases where there are a set of IoT devices that send the captured data to a central service. However, the proposal may be extended to support other communication changes. In many cases, IoT systems have other topologies - a mesh, for instance, where an IoT device can communicate with one or more other IoT devices.

Our proposal could be used to help modify those interactions remotely. For example, if we have one detector device that is communicating with one actuator, we could change the actuator for another

one or make the detector interact with two actuators.

Another case where our proposal may be useful is to add new devices to the IoT systems. We could make the devices that we already have to start interacting with the ones we want to add to our mesh. For example, if we want to add a new light controller and we already have a presence detector running, we could make the detector send the data to that light controller in addition to the destination where it is currently sending the data.

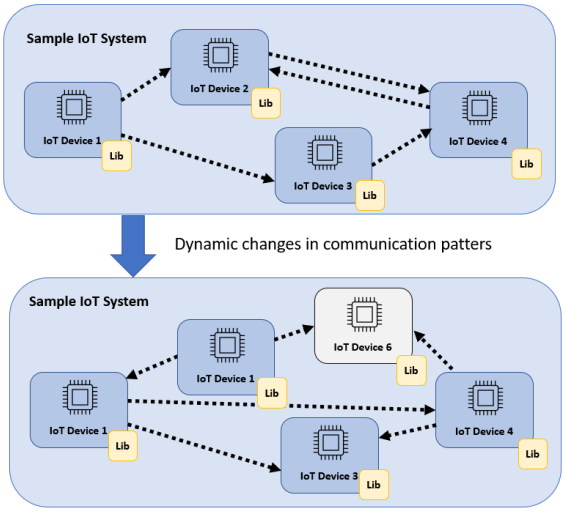

Figure 3 Dynamic changes in an IoT System with mesh topology.

4. Use cases

Below we present some use cases of the main functionalities of our proposal: changing the destination server IP, forwarding a request to another server, and changing the sending frequency.

In order to evaluate the use cases, we designed a small example with limited functionality, just enough to show the usage of the library. This example simulates the recording of the temperature and humidity inside a warehouse. The designed system has various devices with internet connectivity that are equipped with temperature and humidity sensors. Those devices capture the temperature and humidity in multiple locations of the warehouse and send the recorded data to a centralized service where it is analyzed and stored. Those generated analyses are used by the warehouse managers to assess changes in infrastructure, the position of machinery and doors, the placement of refrigeration equipment, etc.

That use case is one possible use of our proposal. We believe our proposal may be applied in many different environments. For example, it may be used in distributed systems without centralized topologies where IoT devices communicate with each other.

Use case 1 Change the destination server of the clients

The company that owns the warehouse decides to change its centralized service for another cloud computing provider. That change involves reconfiguring all the IoT devices that record the temperature and the humidity to send the data to the new service. The company wants to try various cloud computing providers in order to choose the one that best covers their needs.

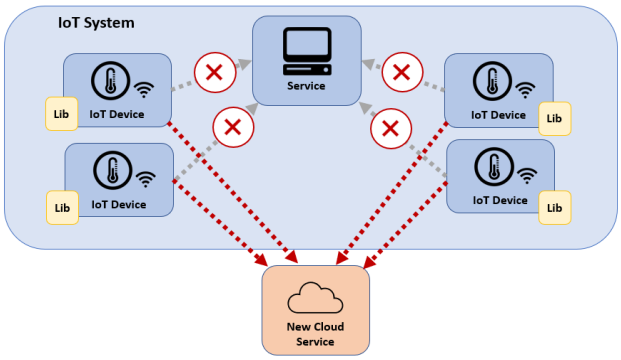

Figure 4. Changing the clients' destination server scheme

Once the program is loaded onto the IoT device and is running, it starts sending requests to the destination server http:192.168.0.27:6001. In order to change the destination IP, we have to send a MQTT request to the MQTT broker with the following command: *set 1 http://192.168.0.100:7071*. When the library that is listening for configuration changes detects a new change, it updates the destination server IP and starts sending data to the new destination.

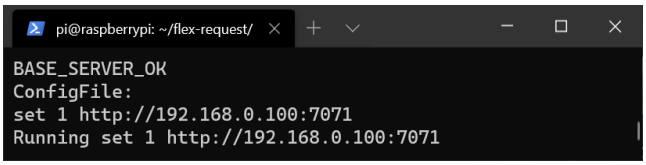

Figure 5. Configuration change received and applied by the IoT device

Use case 2 Forward the requests to another server

In another use case, the company detects a point inside the warehouse where the temperature management is critical. In order to overcome that problem, the company acquires a new device that can manage the air conditioning according to temperature. It is necessary that the devices that are close to that critical point record and send the temperature to that new device in addition to sending the data to the centralized service.

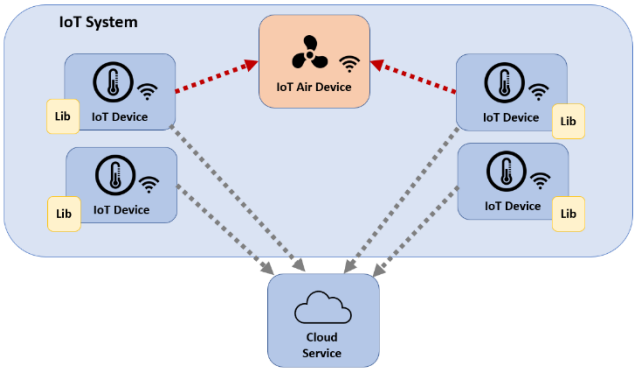

Figure 6. Forwarding the data sending to another device scheme

In this case, to make that change, we have to send the following command: *forward 1 http://192.168.0.200:4044*. Like in the previous use case, when the library detects a new configuration, it saves the forward address locally and starts forwarding each request sent to this new address.

Use case 3 Change the sending frequency

With the recent increase in the work force and the new machinery, it has been detected that temperature changes are much more pronounced and faster than before. Those fast temperature changes require increasing the frequency of the temperature being sent to the centralized service. The company determines that this sending frequency increase is necessary even though it means higher processing costs for the server and higher consumption for the devices. In order to change the frequency using our solution, the responsible system administrators have to send the following command: *set 1 frequency 500*. In this command, "500" is the number of milliseconds between each data sent.

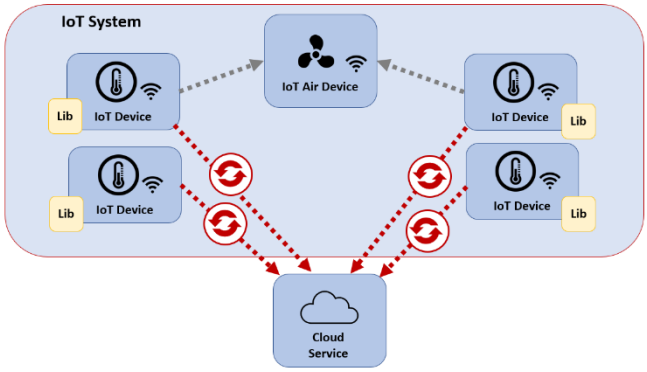

Figure 7. Changing the frequency of data sending scheme

5. Evaluation

In the evaluation section, three changes will be made in the communication of devices. Those changes correspond to those made in the use cases.

- **Change the destination server of the clients.** The initial version of the system has an IoT device that sends data to a service. After the change, the device must send the data to another different service.
- **Forward the requests to another server.** The initial version of the system has an IoT device that sends data to a service. After the change, the device must send the data to another new additional service as well.
- **Change the sending frequency.** The initial version of the system has an IoT device that sends data to a service every second. After the change, the device must send the data every 0.5 seconds.

Each one of the three use cases will be evaluated with the following alternatives.

- OTA Update. Over the Air update is one of the most common ways to update an IoT device software. It consists of remotely sending a new version of the software. This new software must replace the current version. Most IoT devices with some computing capacity can use

operating systems that allow this type of update. Many OTA updates require restarting the device to apply the update.

- Partial OTA. An optimized version of the previous OTA. This process allows generating a package with a partial update, which is small and can be done more quickly. This process is supported only for devices that run complex operating systems. It often requires restarting the device to apply the update.
- Partially updateable software. This is software that has been particularly designed since its inception to support partial updates of some of its components. In this case, the software update depends on itself, not on the device's operating system. This solution is highly dependent on the software implementation strategy, but always requires a more complex implementation than software without updates periodically. This type of update may or may not require restarting the device.
- Proxy. An intermediate proxy makes all devices communicate with the proxy, therefore only by modifying the proxy implementation could a developer modify some communication aspects of the entire IoT system.
- Firebase. Services like Firebase allow the inclusion of remote configurations, which could be obtained by the device's software. Developers usually store configuration settings in Firebase (like connection strings, configuration values, etc.). The device's software must know how to get and process those configuration values. Like "Partially updateable software" this kind of solution requires a more complex implementation of the device software.
- The proposal of this paper. As explained earlier, it uses a specialized command to change some aspects of the communication of the devices.

To evaluate our solution, we used one Raspberry Pi 3 Model B as an IoT device and a PC acting as a server. On the Raspberry Pi, we installed Raspbian GNU/Linux 10 (buster) operating system and Node.js version 12.16.1 as an execution platform. The PC has Windows 10 Pro x64 version 1909 operating system with an Intel Core i5-9600KF 3.70GHz CPU and 16 GB of RAM. In order to access the Raspberry device and easily transfer the programs used in the evaluation of the experiments, we used WinSCP.

Based on the research objectives, the proposal should allow the following:

1) Changes in the communication of the devices, implying a low implementation cost. For this reason, during the evaluation process, we will estimate the implementation complexity in each case.
2) Changes in communications must be made quickly and easily. For this reason, during the evaluation process, we will measure how complex it is for developers to make changes in communication.
3) Minimize the time that the device is not working correctly after a change. For this reason, during the evaluation process, we will measure how long it takes for a device to apply the change and start sending with the new configuration.

Additionally, we should analyze the network traffic in the IoT system. The evaluated alternatives could have different impacts on network traffic. The objective of this research is not to optimize network traffic, but this could be an important factor in some IoT systems.

Implementation complexity

This is the first measurement of all. It tries to estimate the initial implementation complexity of the IoT system before applying the configuration change, which will modify the initial communication scheme. In this evaluation process, we will estimate the implementation complexity of the initial IoT system for the three experiments using the six alternatives.

Measuring the complexity of program implementation is a complicated task. The complexity depends on the number of aspects considered. The simplest estimates of complexity are based solely on the size of the code;

in this case, we have started from a complexity estimation system used in previous research works [36][7]. This system considers the following:

- C – Size: number of the program characters, spaces included
- L – Number of lines of code in the program
- F – Number of functions used but not implemented by the programmer
- NM – Number of standard modules included in Node.js distribution
- NE – Number of modules not included in Node.js distribution and that had to be installed with npm

Each aspect measured was assigned with the following weights: C=1, L=2, F=3, NM=4, and NE=5. The global score for each evaluated alternative is the sum of the weights for each aspect.

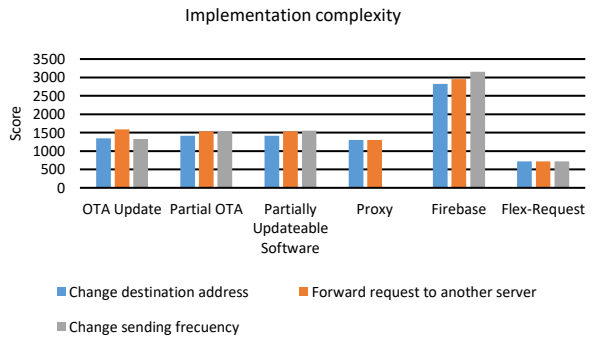

Figure 8. Implementation complexity of each alternative in each experiment

Figure 8 shows the results of the implementation complexity for each alternative in the three experiments. Firebase has the highest complexity in all experiments because additional code must be added to connect to the Firebase service. OTA Update, Partial OTA, and Partially Updateable Software have close implementation complexity with score differences no larger than 100 points between them in each experiment. Proxy has the same complexity in the first two experiments. Because the IoT device is the one that sends the request and the Proxy only redirects or forwards the request to another server, we determined that we cannot use Proxy to implement the sending frequency change. Our proposal has the lowest complexity in all experiments.

Configuration change complexity

This is the second measurement. It measures how complex it is to make a change over the initial version of the system.

Unlike the implementation of the software, the application of changes over the communication scheme could not only involve changes in the code, but sometimes it requires using configuration tools, commands, etc. Due to these particularities, we have included in this complexity estimation all actions performed by the developer with a mouse and keyboard.

We define configuration change complexity as the actions that must be done in order to apply a configuration change. In order to measure this complexity, the following actions were measured:

- K – Number of keystrokes
- MLB – Number of mouse's left buttons clicks
- MRB – Number of mouse's right button clicks
- MDC – Number of double clicks
- MWS – Number of mouse's wheel scroll
- MM – Mouse movement in centimeters

All these aspects were measured with Mousotron software. For each aspect, we associated the following weights: K=1, MLB=2, MRB=1, MDC=1, MWS=1, and MM=0,5. The global score for each evaluated alternative is the sum of the weights for each aspect. For every change that involves sending a request over the Internet to change the configuration, we made a script. Therefore, the action of sending the request with the change translates to writing the command to execute the script.

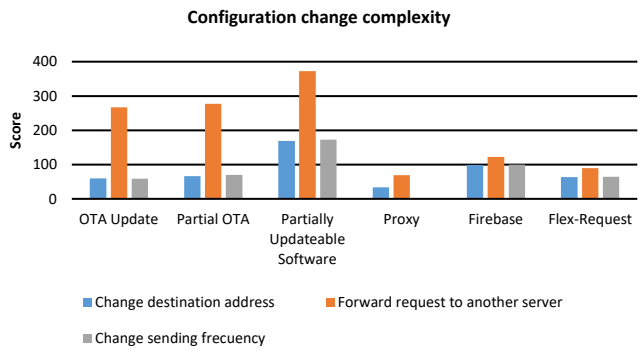

Figure 9. Configuration change complexity for each alternative in each experiment

The configuration change measurements from figure 9 show that Proxy has the lowest complexity for changing the destination address and forwarding requests. That is because the Proxy was on the same PC used to run the experiments. Therefore, those changes only imply changing and saving local files without connecting to other machines over the Internet. We determined that Proxy cannot be used in the third experiment as the sending frequency have to be modified on the IoT devices that send the requests. The lower change complexity for the third experiment is the OTA Update as it only needs to change the value of the property that stores the sending frequency and execute the script that sends the update. OTA Update, Partial OTA, and Partially Updateable Software have significantly bigger change complexity in the second experiment that the other ones. This is due to all the code that needs to be added in order to forward the request to another server. Our proposal has the second-lowest change complexity for the second experiment and the third-lowest change complexity for the other two.

Time the device is inactive

This is the third measurement, and it evaluates how long it takes until the device starts sending data with the new configuration after the configuration change had happened.

Applying an update on a running software means that during the period when the device update is being applied, the device may be in different incorrect states. It can be stopped, running but not sending data or running but sending the data using the old configuration. Because of that, we measure the time it takes for the device to apply the new configuration and start sending data using it.

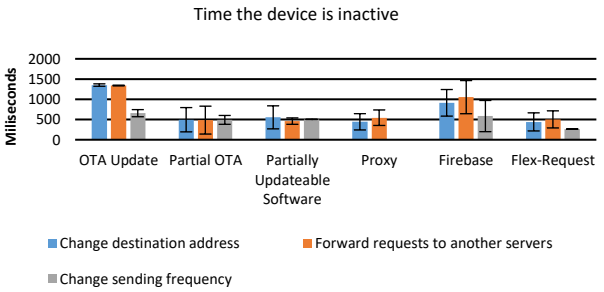

Figure 10.Time it takes for the device to apply the change

The results of measuring the time it takes for the device to apply the change are shown in figure 10. The only alternative that required entirely stopping the running program was the OTA Update. The times the program was not running for each experiment are: 346.30 ± 8.49 ms, 321.30 ± 7.85 ms, and 356.10 ± 20.59. Other alternatives did not require the program to be stopped. Even if the change was not yet applied, it could enable, for example, storing the data and sending it after the update was done. OTA Update has the longest change apply time in each experiment. In the third experiment, the time is significantly lower because, as explained, we measure the time until the device starts sending data using the new configuration. Therefore, in the case of the OTA Update and Firebase, where the change times are the longest, the frequency of sending the data has an impact on the obtained results. In the third experiment, we change the frequency from 1 second to 0.5 seconds, and because of that, the first data sent with a new configuration occurs earlier. As explained earlier, in the third experiment, we determined that the proxy solution cannot be used. Our proposal had no significant difference in experiments 1 and 2 with Partial OTA, Partially Updateable Software, and Proxy, however in the third experiment, our proposal had the lowest change time.

Network traffic

This is the last measurement. It evaluates how much network traffic consumes an IoT system developed with the different alternatives in a time interval. First, analyzing a time period of operation without doing configuration changes. Second, an equivalent time

period, but this time doing a configuration change. This will allow us to see the usual network traffic for IoT systems developed with different technologies and the cost of doing a change in the communication scheme.

To measure the network traffic, we used Wireshark software. We measured the total size (in bytes) of the packets sent between the IoT device and the main PC in a 5-minute time interval. In evaluations when the configuration change was performed, the change was always done one minute from the start of the measurement, because adding request forwarding to other servers or changing the sending frequency implies the modification of the total request sent and therefore the total size of the packets sent.

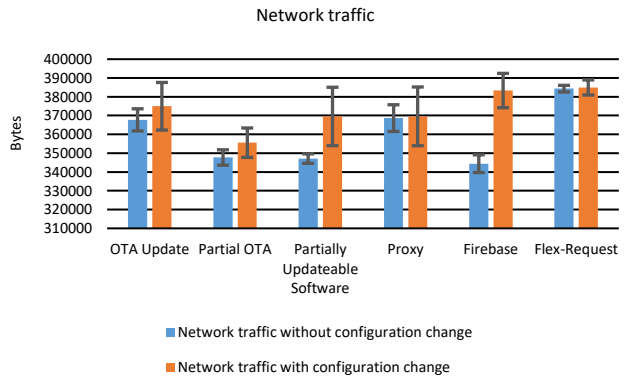

Figure 11. Traffic network with and without configuration change

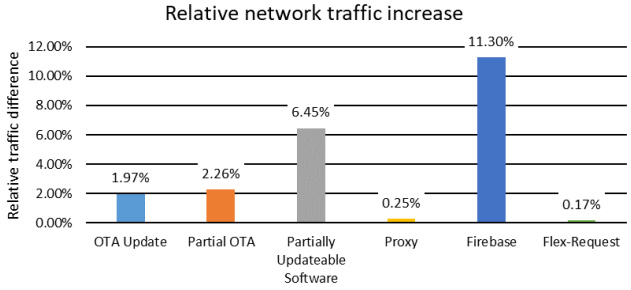

Figure 12. Relative network traffic increase

Figure 11 shows network traffic measurements. We measured the traffic usage of each alternative in a 5-minute period without doing configuration changes, and then we measured again the same period making a configuration change. In the first case, without changes, our proposal has the highest traffic usage and is followed by Proxy and OTA Update and then the rest of the alternatives. Even though our alternative has the highest traffic usage, the relative difference with other alternatives is only between 4% - 12%.

In the case where configuration change was performed, our proposal still has the highest traffic usage. However, it is closely followed by Firebase and Partially Updateable Software, which have significantly higher network traffic relative to the case without configuration changes. The relative difference in network traffic to our proposal is between 0.4% - 9%.

Figure 12 shows the relative increase in network traffic for each alternative between measurements without changes and with changes. Firebase has the lowest traffic usage without changes with just 344405 bytes; however, the relative increase in traffic for the case with changes is 11.30%, the highest one. Partially Updateable Software has a relative increase of 6.45%, it may be because in order to make the change, we have to connect directly to the device through ssh. Although our proposal has the highest network traffic: 384301,40 without configuration change and 384952,60 with changes, the relative traffic increase is the lowest one, with just 0.17 %.

6. Conclusions and Future Work

This research work focuses on providing an alternative that allows the creation of IoT systems capable of remotely modifying devices' communication configuration in a fast and agile way. This feature will facilitate prototyping and attempt different architectures during the development phase of an IoT system. Enabling agile changes is especially useful in systems with uncertainty. During the use and exploitation phase of IoT systems, changes in execution can also be frequent, due to the introduction of new devices, services, changes in the physical environment, etc.

The proposal to address this problem was the specification of a library to be used in the software of the devices. A specific implementation of this library has been developed in Node.js. This library allows the developer to do configuration changes remotely using commands; these changes do not require updating the device software.

The proposal should not increase the implementation complexity of IoT devices software. Based on the use cases evaluated, a software developed using the proposal could have significantly less implementation complexity than software developed using other alternatives. In the cases analyzed, implementation complexity was reduced by around 40% - 50%.

Once an IoT system was running, the cost of making changes to the device communication was measured. Among the analyzed alternatives, the one which led to changes in the device communication with less cost was the Proxy. Still, it did not have enough functionality to make complex changes, as changes specified in use case 3. The cost of making changes for the proposal had no significant difference in comparison to three of the alternatives, and this cost was much smaller than the last of the alternatives.

The proposal should minimize the time that the IoT device is off or not working properly after doing a change in communication and the time it takes to make the change. Only one of the analyzed alternatives requires to restart the device after the update (OTA). The time that our proposal took to apply the change had no significant differences with other alternatives in the use case 1 and 2. However, in the third use case, the time to apply the change of our proposal was between 45% - 60% shorter.

The proposal got an increase in network traffic compared to the other alternatives in the cases evaluated. This increase was between 4% - 12% during a period of normal use, without changes in communication. When these changes were done, the increase of the traffic network for the proposal was reduced to 0.4% - 9%. According to the results, the proposal always has more consumption of network traffic than other alternatives, but the relative increase in traffic network for each change made is just 0.17%, much less than in other alternatives.

Results show that this proposal could be very suitable for many IoT systems which require to apply changes in their devices' communication.

Future research lines will be related to include the possibility of changing data protocols (CoAP, AMQP, MQTT, etc.), communication schemes, and reducing the impact of the proposal over the traffic network.

References

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Futur. Gener. Comput. Syst. 29 (2013) 1645–1660. https://doi.org/10.1016/j.future.2013.01.010.

[2] D. Zhang, J. Wan, C.-H. (Robert) Hsu, A. Rayes, Industrial technologies and applications for the Internet of Things, Comput. Networks. 101 (2016) 1–4. https://doi.org/10.1016/j.comnet.2016.02.019.

[3] M.A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, Middleware for Internet of Things: A Survey, IEEE Internet Things J. 3 (2016) 70–95. https://doi.org/10.1109/JIOT.2015.2498900.

[4] L.M. Vaquero, L. Rodero-Merino, Finding your Way in the Fog, ACM SIGCOMM Comput. Commun. Rev. 44 (2014) 27–32. https://doi.org/10.1145/2677046.2677052.

[5] P. Ruckebusch, J. Van Damme, E. De Poorter, I. Moerman, Dynamic Reconfiguration of Network Protocols for Constrained Internet-of-Things Devices, in: Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST, 2016: pp. 269–281. https://doi.org/10.1007/978-3-319-47075-7_31.

[6] V. Antinyan, M. Staron, A. Sandberg, Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time, Empir. Softw. Eng. 22 (2017) 3057–3087. https://doi.org/10.1007/s10664-017-9508-2.

[7] J.P. Espada, V.G. Díaz, R.G. Crespo, O.S. Martínez, B.C.P. G-Bustelo, J.M.C. Lovelle, Using extended web technologies to develop Bluetooth multi-platform mobile applications for interact with smart things, Inf. Fusion. 21 (2015) 30–41. https://doi.org/10.1016/j.inffus.2013.04.008.

[8] H. Mukhtar, B.W. Kim, B.S. Kim, S.S. Joo, An efficient remote code update mechanism for wireless sensor networks, Proc. - IEEE Mil. Commun. Conf. MILCOM. (2009). https://doi.org/10.1109/MILCOM.2009.5379862.

[9] M.L. Chiang, T.L. Lu, Two-Stage Diff: An efficient dynamic software update mechanism for wireless sensor networks, Proc. - 2011 IFIP 9th Int. Conf. Embed. Ubiquitous Comput. EUC 2011. (2011) 294–299. https://doi.org/10.1109/EUC.2011.74.

[10] Y.C. Chang, T.Y. Chi, W.C. Wang, S.Y. Kuo, Dynamic software update model for remote entity management of machine-to-machine service capability, IET Commun. 7 (2013) 32–39. https://doi.org/10.1049/iet-com.2012.0459.

[11] W. Jin, D.H. Kim, IoT device management architecture based on proxy, Proc. 2017 6th Int. Conf. Comput. Sci. Netw. Technol. ICCSNT 2017. 2018-Janua (2018) 84–87. https://doi.org/10.1109/ICCSNT.2017.8343663.

[12] R. Bannatyne, G. Viot, Introduction to microcontrollers. I, in: Northcon/98. Conf. Proc. (Cat. No.98CH36264), IEEE, n.d.: pp. 238–248. https://doi.org/10.1109/NORTHC.1998.731542.

[13] K.J. Singh, D.S. Kapoor, Create Your Own Internet of Things: A survey of IoT platforms., IEEE Consum. Electron. Mag. 6 (2017) 57–68. https://doi.org/10.1109/MCE.2016.2640718.

[14] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, IEEE Commun. Surv. Tutorials. 17 (2015) 2347–2376.

https://doi.org/10.1109/COMST.2015.2444095.

[15] G. Kortuem, F. Kawsar, V. Sundramoorthy, D. Fitton, Smart objects as building blocks for the Internet of things, IEEE Internet Comput. 14 (2010) 44–51. https://doi.org/10.1109/MIC.2009.143.

[16] C. Sarkar, A.U. Nambi S. N., R.V. Prasad, A. Rahim, R. Neisse, G. Baldini, DIAT: A Scalable Distributed Architecture for IoT, IEEE Internet Things J. 2 (2015) 230–239. https://doi.org/10.1109/JIOT.2014.2387155.

[17] A. Alsalemi, Y. Al Homsi, M. Al Disi, I. Ahmed, F. Bensaali, A. Amira, G. Alinier, Real-Time Communication Network Using Firebase Cloud IoT Platform for ECMO Simulation, in: 2017 IEEE Int. Conf. Internet Things IEEE Green Comput. Commun. IEEE Cyber, Phys. Soc. Comput. IEEE Smart Data, IEEE, 2017: pp. 178–182. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.31.

[18] H.G. Barron-Gonzalez, M. Martinez-Espronceda, S. Led, L. Serrano, C. Fischer, M. Clarke, New use cases for remote control and configuration of interoperable medical devices, in: 2013 35th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc., IEEE, 2013: pp. 4787–4790. https://doi.org/10.1109/EMBC.2013.6610618.

[19] D. Kumper, R. Tonjes, Remote configuration and deployment of sensor drivers for a medical bluetooth sensor gateway, in: 2011 IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks, IEEE, 2011: pp. 1–6. https://doi.org/10.1109/WoWMoM.2011.5986194.

[20] H.G. Barron-Gonzalez, M. Martinez-Espronceda, J.D. Trigo, S. Led, L. Serrano, Proposal of a novel remote command and control configuration extension for interoperable Personal Health Devices (PHD) based on ISO/IEEE11073 standard, in: 2014 36th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc., IEEE, 2014: pp. 6312–6315. https://doi.org/10.1109/EMBC.2014.6945072.

[21] A.E. Nikolaidis, S.S. Papastefanos, G.I. Stassinopoulos, M.-P.K. Drakos, G.A. Doumenis, Automating Remote Configuration Mechanisms for Home Devices, IEEE Trans. Consum. Electron. 52 (2006) 407–413. https://doi.org/10.1109/TCE.2006.1649657.

[22] B. Seo, S.H. Kim, H. Choi, The remote command and control system for outdoor robot, in: 2013 10th Int. Conf. Ubiquitous Robot. Ambient Intell., IEEE, 2013: pp. 303–304. https://doi.org/10.1109/URAI.2013.6677371.

[23] G. Jurković, V. Sruk, Remote firmware update for constrained embedded systems, 2014 37th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2014 - Proc. (2014) 1019–1023. https://doi.org/10.1109/MIPRO.2014.6859718.

[24] E. Dalipi, F. Van Den Abeele, I. Ishaq, I. Moerman, J. Hoebeke, EC-IoT: An easy configuration framework for constrained IoT devices, 2016 IEEE 3rd World Forum Internet Things, WF-IoT 2016. (2017) 159–164. https://doi.org/10.1109/WF-IoT.2016.7845483.

[25] M. Su, B. Zhou, A. Fu, Y. Yu, G. Zhang, PRTA: A Proxy Re-encryption based Trusted Authorization scheme for nodes on CloudIoT, Inf. Sci. (Ny). 527 (2020) 533–547. https://doi.org/10.1016/j.ins.2019.01.051.

[26] O. Kayode, A.S. Tosun, Analysis of IoT Traffic using HTTP Proxy, in: ICC 2019 - 2019 IEEE Int. Conf. Commun., IEEE, 2019: pp. 1–7. https://doi.org/10.1109/ICC.2019.8761601.

[27] M. Weisbach, N. Taing, M. Wutzler, T. Springer, A. Schill, S. Clarke, Decentralized coordination of dynamic software updates in the Internet of Things, 2016 IEEE 3rd World Forum Internet Things, WF-IoT 2016. (2017) 171–176. https://doi.org/10.1109/WF-IoT.2016.7845450.

[28] K. Kolomvatsos, An intelligent, uncertainty driven management scheme for software updates in pervasive IoT applications, Futur. Gener. Comput. Syst. 83 (2018) 116–131. https://doi.org/10.1016/j.future.2018.01.036.

[29] T.P. Raptis, A. Passarella, M. Conti, Distributed Path Reconfiguration and Data Forwarding in Industrial IoT Networks, in: K.R. Chowdhury, M. Di Felice, I. Matta, B. Sheng (Eds.), Springer International Publishing, Cham, 2018: pp. 29–41. https://doi.org/10.1007/978-3-030-02931-9_3.

[30] S.K. Datta, C. Bonnet, Easing IoT application development through DataTweet framework, in: 2016 IEEE 3rd World Forum Internet Things, IEEE, 2016: pp. 430–435. https://doi.org/10.1109/WF-IoT.2016.7845390.

[31] G. Lee, S. Heo, B. Kim, J. Kim, H. Kim, Rapid prototyping of IoT applications with Esperanto compiler, in: Proc. 28th Int. Symp. Rapid Syst. Prototyp. Shortening Path from Specif. to Prototype - RSP '17, ACM Press, New York, New York, USA, 2017: pp. 85–91. https://doi.org/10.1145/3130265.3138857.

[32] D. Mazzei, G. Baldi, G. Montelisciani, G. Fantoni, A full stack for quick prototyping of IoT solutions, in: 2016 Cloudification Internet Things, IEEE, 2016: pp. 1–5. https://doi.org/10.1109/CIOT.2016.7872915.

[33] P. Patel, D. Cassou, Enabling high-level application development for the Internet of Things, J. Syst. Softw. 103 (2015) 62–84. https://doi.org/10.1016/j.jss.2015.01.027.

[34] T. Nepomuceno, T. Carneiro, T. Carneiro, C. Korn, A. Martin, A GUI-based Platform for Quickly Prototyping Server-side IoT Applications, in: Smart SysTech 2018; Eur. Conf. Smart Objects, Syst. Technol., 2018: pp. 1–9.

[35] S. Kikuchi, I. Thomas, O. Jallouli, J. Doerr, A. Morgenstern, E. Baccelli, K. Schleiser, Orchestration of IoT Device and Business Workflow Engine on Cloud, in: 2018 3rd Cloudification Internet Things, IEEE, 2018: pp. 1–2. https://doi.org/10.1109/CIOT.2018.8627118.

[36] G. Cueva-Fernandez, J.P. Espada, V. García-Díaz, C.G. García, N. Garcia-Fernandez, Vitruvius: An expert system for vehicle sensor tracking and managing application generation, J. Netw. Comput. Appl. 42 (2014) 178–188. https://doi.org/10.1016/j.jnca.2014.02.013.