UNIVERSIDAD DEL AZUAY

FACULTAD DE CIENCIA Y TECNOLOGÍA

ESCUELA DE INGENIERÍA ELECTRÓNICA

Implementación de arquitecturas de despliegue remoto de firmware para optimizar la operatividad de redes de sensores inalámbricos

Trabajo de graduación previo a la obtención del título de:

INGENIERO ELECTRÓNICO

Autor:

Mateo Francisco Estévez Vélez

Director:

Ing. Jaime Sebastián Burbano Villavicencio, Mgt.

CUENCA, ECUADOR

2025

DEDICATORIA

A mis padres Catalina y Fernando, quienes me inculcaron los valores fundamentales que me han permitido ser quien soy. Ustedes me enseñaron la importancia de la responsabilidad y cultivaron en mí un profundo sentido de la curiosidad.

A mi querida hermana Andrea, por su apoyo constante, cariño incondicional y por ser siempre un ejemplo a seguir. A mis abuelitos Rodrigo y Laura, Fausto y Leonor, a quienes llevo siempre en mi corazón, por su cariño inigualable y por sus enseñanzas que me han marcado profundamente.

A mis amigos Nicolás Bárcenes, Paula Abril, Paula Orellana, Diego Aguirre, Evelyn Ochoa, Francisco Gálvez, William Goercke, Berenice Guerrero y Daniela Mendoza, con quienes compartí risas, desafíos y momentos inolvidables que hicieron de esta etapa de mi vida algo realmente especial.

AGRADECIMIENTOS

Agradezco sinceramente a la Universidad del Azuay por la excelente formación que me ha otorgado y al IERSE por darme la oportunidad de contribuir al desarrollo tecnológico con este trabajo. Mi agradecimiento también a los docentes de la Escuela de Ingeniería Electrónica, por impulsar mi aprendizaje y crecimiento profesional.

Quiero expresar un especial agradecimiento al ingeniero Jaime Burbano, quien aceptó la dirección de este trabajo. Su retroalimentación y guía constante fueron cruciales para la elaboración del mismo. Además, agradezco a los ingenieros Carlos Zeas y Bryan Fajardo por su valioso apoyo, especialmente durante la fase experimental de la investigación.

IMPLEMENTACIÓN DE ARQUITECTURAS DE DESPLIEGUE REMOTO DE FIRMWARE PARA OPTIMIZAR LA OPERATIVIDAD DE REDES DE SENSORES INALÁMBRICOS

Las actualizaciones remotas de firmware (*FOTA*) en redes de sensores inalámbricos (*WSN*) favorecen la adaptabilidad y mantenimiento de nodos distribuidos; sin embargo, incrementan el consumo energético, introducen nuevos vectores de ataque y plantean desafíos de escalabilidad. En este trabajo se diseñan e implementan dos arquitecturas *FOTA*: una basada en soluciones comerciales y otra independiente que integra múltiples tecnologías abiertas. Ambas soluciones son puestas a prueba a partir de experimentos que cuantifican su consumo energético y latencias en la comunicación. Con base en los resultados experimentales y tomando en cuenta sus aptitudes en ciberseguridad, escalabilidad y flexibilidad tecnológica, se propone una metodología comparativa fundamentada en el Proceso Analítico Jerárquico (*AHP*). Como resultado, se obtiene un puntaje que cuantifica el impacto de cada arquitectura propuesta sobre la operatividad de una *WSN*, demostrando que la arquitectura independiente supera a la comercial, al presentar menores latencias en la comunicación y un menor consumo energético.

Palabras clave: *FOTA*, *WSN*, *IoT*, *AHP*, Cloudflare, AWS

IMPLEMENTATION OF FOTA ARCHITECTURES TO OPTIMIZE OPERABILITY IN WIRELESS SENSOR NETWORKS

Firmware Over The Air (*FOTA*) updates enhance the adaptability and maintenance of distributed nodes in Wireless Sensor Networks (*WSNs*); however, they also increase energy overheads, introduce new vulnerabilities and present scalability challenges. This study designs and implements two *FOTA* architectures: one based on commercial solutions, and an independent architecture that integrates various open technologies. Both architectures are evaluated through experiments measuring energy consumption and communication latency. Taking the experimental results and each architecture's capabilities in cybersecurity, scalability and technological flexibility into consideration, a comparison methodology using the Analytic Hierarchy Process (*AHP*) is proposed. As a result, a score quantifying the impact of each architecture on a *WSN*'s operability is obtained, demonstrating the superiority of the independent solution over the commercial one by presenting lower energy consumption and reduced communication latency.

Keywords: *FOTA*, *WSN*, *IoT*, *AHP*, Cloudflare, AWS

ÍNDICE DE CONTENIDOS

Dedicatoria i

Agradecimientos ii

Resumen iii

Abstract iv

Índice de Contenidos v

Índice de Figuras vi

Índice de Tablas vii

I. Introducción 1

II. Estado del arte 2
- A. Trabajos relacionados . . . 2
- B. Servicios comerciales para *FOTA* . . . 3

III. Metodología 4
- A. Diseño y características de la arquitectura comercial . . . 4
- B. Diseño y características de la arquitectura independiente . . . 6
- C. Integración de las arquitecturas en el nodo sensor . . . 7
- D. Métricas para la evaluación de las arquitecturas . . . 7
- E. Diseño experimental . . . 8
- F. Proceso analítico jerárquico . . . 10

IV. Resultados 12
- A. Sobrecoste energético . . . 12
- B. Tiempo de descarga . . . 12
- C. Tiempo de propagación . . . 13
- D. Comparación analítica entre las arquitecturas propuestas . . . 14

V. Conclusiones 16

Referencias 16

ÍNDICE DE FIGURAS

1. Arquitectura comercial basada en los servicios de AWS. 5
2. Arquitectura independiente. 6
3. Secuencia de actualización en la arquitectura comercial. 7
4. Secuencia de actualización en la arquitectura independiente. 8
5. Curvas de consumo energético acumulado. 12
6. Sobrecoste energético de cada arquitectura. 12
7. Tiempos de descarga individuales para distintos tamaños de firmware. 13
8. Tiempos de descarga durante actualizaciones de un grupo de nodos. 13
9. Distribución de los tiempos de propagación de todos los nodos del grupo para cada arquitectura. 14
10. Jerarquía *AHP* para la selección de la arquitectura *FOTA* con mayor puntaje de operatividad. 14

ÍNDICE DE TABLAS

I. Características de servicios *FOTA* comerciales 3
II. Esquema de particiones modificado para experimentación en los nodos empleados 9
III. Sobrecostes energéticos de cada arquitectura 12
IV. Estadísticas para los tiempos de descarga individuales 12
V. Prueba post-hoc de Dunn en la arquitectura comercial 13
VI. Tasas de consistencia por matriz de prioridades 15
VII. Matriz de rendimiento normalizada 15
VIII. Matriz de decisión ponderada y puntajes de operatividad 15

Implementación de arquitecturas de despliegue remoto de firmware para optimizar la operatividad de redes de sensores inalámbricos

Mateo Francisco Estévez Vélez
Escuela de Ingeniería Electrónica
Universidad del Azuay
Cuenca, Ecuador
mateo.estevez@es.uazuay.edu.ec

***Resumen*—Las actualizaciones remotas de firmware (*FOTA*) en redes de sensores inalámbricos (*WSN*) favorecen la adaptabilidad y mantenimiento de nodos distribuidos; sin embargo, incrementan el consumo energético, introducen nuevos vectores de ataque y plantean desafíos de escalabilidad. En este trabajo se diseñan e implementan dos arquitecturas *FOTA*: una basada en soluciones comerciales y otra independiente que integra múltiples tecnologías abiertas. Ambas soluciones son puestas a prueba a partir de experimentos que cuantifican su consumo energético y latencias en la comunicación. Con base en los resultados experimentales y tomando en cuenta sus aptitudes en ciberseguridad, escalabilidad y flexibilidad tecnológica, se propone una metodología comparativa fundamentada en el Proceso Analítico Jerárquico (*AHP*). Como resultado, se obtiene un puntaje que cuantifica el impacto de cada arquitectura propuesta sobre la operatividad de una *WSN*, demostrando que la arquitectura independiente supera a la comercial, al presentar menores latencias en la comunicación y un menor consumo energético.**

***Palabras clave*—*FOTA*, *WSN*, *IoT*, *AHP*, Cloudflare, AWS**

I. Introducción

La creciente demanda de sistemas inteligentes de monitoreo y control remoto ha impulsado la evolución de tecnologías capaces de conectar dispositivos más allá de sus entornos inmediatos, permitiendo el acceso a recursos distribuidos y facilitando la colaboración en red. Reportes recientes sobre la adopción de *IoT* (*Internet of Things*) apuntan a un incremento sustancial en la cantidad de dispositivos conectados para fines de la década en curso, esperando que se superen las 40 mil millones de unidades [1]. En este contexto, las redes de sensores inalámbricos (*WSN*, por sus siglas en inglés) han emergido como una solución para la caracterización de diversos ambientes a través de la recolección de datos espacial y temporalmente distribuidos provenientes de los nodos que las conforman [2]. El despliegue de las *WSN* se ha mostrado favorable en múltiples campos de aplicación, tales como el monitoreo ambiental, la agricultura de precisión, la gestión de desastres naturales, la vigilancia de seguridad en áreas urbanas, la caracterización de la polución en zonas de interés y las aplicaciones de ciudades inteligentes [3].

Sin embargo, garantizar la continuidad operativa de las *WSN* plantea varios desafíos importantes. Entre estos, destacan las necesidades continuas de corregir errores, implementar nuevas funcionalidades, reducir vulnerabilidades en la red y optimizar el consumo energético del sistema. Una de las prácticas comunes que permiten abordar estos problemas consiste en reemplazar el firmware de los nodos a través de actualizaciones. No obstante, conforme aumenta la densidad de una *WSN*, o en cuanto crece su área de distribución, los tiempos requeridos para cumplir con estas tareas se ven exponencialmente extendidos, lo cual dificulta su mantenimiento [4]. Para dar una posible solución a esta problemática, se ha propuesto integrar sistemas compuestos por múltiples tecnologías que trabajan en conjunto para permitir el despliegue remoto del firmware a través del canal de comunicación inalámbrico empleado por los nodos. Estos sistemas se conocen como *FOTA* (*Firmware Over The Air*).

Pese a las ventajas que los sistemas *FOTA* ofrecen, la comunicación inalámbrica para la transferencia de firmware puede implicar un consumo energético significativo. Este factor es particularmente crítico, dado que los nodos de las *WSN* suelen operar bajo estrictas limitaciones de capacidad computacional y energía disponible [5]. Además, para garantizar su viabilidad a largo plazo, una arquitectura *FOTA* debe incorporar tecnologías y mecanismos que favorezcan la escalabilidad de la red [4]. En este sentido, es necesario que el sistema cuente con la capacidad de integrar nuevos nodos de manera eficiente y gestionar el aumento en la densidad de dispositivos, sin comprometer el rendimiento, la seguridad o la fiabilidad de la red. Adicionalmente, la habilitación de canales para modificar el comportamiento de los dispositivos en la red incrementa las vulnerabilidades del sistema, exponiendo nuevos vectores de ataque que pueden ser explotados por actores malintencionados [6], [7].

Si bien existen varias soluciones comerciales que permiten la incorporación de *FOTA* en las *WSN*, el uso de tecnologías propietarias presenta importantes limitaciones. Entre ellas destaca la necesidad de adaptar el sistema a requisitos impuestos por el proveedor, lo que reduce la flexibilidad tecnológica de la red de sensores y genera preocupaciones en torno a la confidencialidad de las comunicaciones y los datos al depender de terceros [8]. Además, la adaptación de una *WSN* consolidada para integrar estas soluciones puede

resultar compleja y no siempre posible, especialmente debido a la heterogeneidad de los nodos en términos de capacidades computacionales, tecnologías de comunicación y accesibilidad a la red.

Un enfoque alternativo que podría ofrecer mayores beneficios consiste en diseñar una arquitectura *FOTA* basada en tecnologías independientes y abiertas, capaz de adaptarse a las características de la red existente. Este enfoque permite abordar de manera flexible la diversidad de recursos y requerimientos en una *WSN*, además de ofrecer mayor control sobre la información recolectada. Por otro lado, dicha implementación puede también conllevar otras limitaciones como, por ejemplo, la escalabilidad.

Ante la falta de estudios que comparan la operatividad de las soluciones comerciales frente a sistemas desarrollados de forma independiente en el estado del arte, la elección de una solución *FOTA* para una *WSN* se convierte en una decisión no trivial. Por ello, este trabajo aborda esta brecha mediante el diseño e implementación de dos arquitecturas *FOTA* distintas: una basada en soluciones comerciales y otra independiente que integra múltiples tecnologías abiertas. Además, se realiza un diseño experimental que posibilita la comparación tanto cuantitativa como cualitativa de las arquitecturas en términos de consumo energético, latencias de comunicación, flexibilidad tecnológica, escalabilidad y ciberseguridad. Este análisis integral permite demostrar su aplicabilidad en escenarios reales.

El resto del documento se organiza de la siguiente manera: la sección II expone las características fundamentales y factores a considerar en el desarrollo de arquitecturas *FOTA*, a partir de una revisión del estado del arte, además de un análisis de las soluciones comerciales disponibles en el mercado. En la sección III, se describen las arquitecturas *FOTA* propuestas y se detalla la metodología experimental utilizada para su evaluación comparativa. La sección IV presenta y analiza los resultados obtenidos a partir de la metodología propuesta basada en el proceso analítico jerárquico *AHP*. Finalmente, la sección V describe las conclusiones y limitaciones identificadas en este estudio, destacando además posibles direcciones para trabajos futuros.

II. Estado del arte

A. Trabajos relacionados

Las actualizaciones *FOTA* son fundamentales para la gestión del ciclo de vida de las *WSN*, ya que facilitan el mantenimiento remoto, la mejora de funcionalidades y la implementación de parches de seguridad sin requerir intervención física. La rápida proliferación de *WSN* ha conducido a un incremento en las implementaciones de *FOTA*, tanto de diseño independiente como basadas en soluciones comerciales. Aunque cada aplicación presenta requerimientos específicos, las principales dificultades para la integración de *FOTA* han sido reconocidas por múltiples autores, de modo que la atención de investigadores y proveedores comerciales se ha dirigido hacia la optimización del consumo de energía, los tiempos de descarga del firmware, la robustez frente a fluctuaciones en la red, la adopción de protocolos ligeros y las medidas de ciberseguridad [9].

Los sistemas *FOTA* en las *WSN* se clasifican generalmente en modelos centralizados, distribuidos e híbridos. Las arquitecturas centralizadas, en las que un único servidor orquesta la distribución del firmware, son relativamente sencillas, pero pueden enfrentar desafíos de escalabilidad, particularmente en implementaciones a gran escala [10]. En contraste, las arquitecturas distribuidas aprovechan protocolos de comunicación *peer-to-peer* o de múltiples saltos para difundir las actualizaciones, mejorando la tolerancia a fallos y la resiliencia de la red [11]–[13]. Los enfoques híbridos combinan estos paradigmas, buscando un equilibrio entre el control centralizado y la escalabilidad distribuida [14]. Las soluciones comerciales de *FOTA* suelen adherirse a modelos centralizados bien establecidos, mientras que las arquitecturas independientes pueden adaptarse para aprovechar elementos distribuidos, tales como *bootloaders* modificados, nodos dedicados únicamente a la actualización o difusión del firmware, circuitos programadores externos al sistema principal del nodo, entre otros. El uso de un modelo u otro dependerá de las limitaciones de hardware de la *WSN*, la cantidad de datos transmitidos durante las actualizaciones y la robustez requerida por la red [4], [15]–[17].

La selección de tecnologías de comunicación influye de manera significativa en la operatividad de las arquitecturas *FOTA*. Las tecnologías tradicionales de alto ancho de banda, tales como Wi-Fi y GPRS, se emplean frecuentemente para garantizar una rápida difusión del firmware [7], [18]. Sin embargo, estas soluciones a menudo incurren en un mayor consumo de energía, lo cual puede ser una restricción crítica en nodos de *WSN* alimentados por pequeñas baterías o que dependen de la recolección de energía. Por el contrario, se han investigado tecnologías alternativas como NB-IoT, LoRa y Zigbee debido a sus menores requerimientos energéticos y su mayor idoneidad para redes geográficamente dispersas [19]–[21]. Sin embargo, dichas tecnologías presentan limitaciones cuando el tamaño del firmware del nodo es relativamente extenso. Estudios recientes que comparan soluciones WLAN y LPWAN han subrayado la compensación entre la tasa de datos y la eficiencia energética para mitigar un consumo excesivo de energía mientras se mantienen tiempos de descarga aceptables [22], [23].

Los sobrecostes adicionales de comunicación y la carga computacional asociada con las actualizaciones de firmware incrementan inherentemente el consumo de energía. Por ello, algunas metodologías han explorado los beneficios de las actualizaciones incrementales —transmitiendo solo la diferencia (*delta*) entre versiones de firmware— para reducir el gasto energético [22], [24], [25]. Las arquitecturas *FOTA* independientes pueden optimizarse para implementar actualizaciones parciales del firmware y escrituras diferidas en memoria flash, reduciendo así tanto el tiempo total de descarga como la energía consumida durante el proceso de actualización. En contraste, muchos sistemas comerciales, si bien son más robustos, a menudo muestran menor flexibilidad para adaptar las estrategias de actualización a los perfiles energéticos heterogéneos de los nodos *WSN*.

La robustez frente a las fluctuaciones en la red a través de la cual se comunican los nodos de la *WSN* es esencial para asegurar una actualización exitosa del firmware y la operatividad de la red. Algunas arquitecturas *FOTA* frecuentemente incorporan mecanismos como estrategias de reversión (*rollback*) y protocolos de retransmisión adaptativos para mitigar el impacto de conexiones inestables [24], [26]. Los protocolos distribuidos han demostrado un rendimiento superior bajo condiciones de red fluctuantes al aprovechar la comunicación *multi-hop* y rutas de actualización redundantes [11]–[13]. No obstante, la designación de nodos especializados para la distribución de firmware impone la necesidad de crear por lo menos dos grupos de actualización separados.

La seguridad en los sistemas *FOTA* se logra típicamente mediante la encriptación, autenticación y control de acceso [8], [16]. Estos conceptos contribuyen a la mitigación de ataques comunes como la suplantación de identidad, manipulación de datos en tránsito, robo de información confidencial, entre otros [27]. El uso de certificados TLS (*Transport Layer Security*) y protocolos de comunicación seguros, como MQTT sobre TLS o HTTP sobre TLS (HTTPS), mejora la integridad y confidencialidad de los datos durante la transmisión del firmware [7]. La tecnología blockchain también se ha integrado en las arquitecturas *FOTA* para proporcionar un marco descentralizado e inalterable para la distribución y verificación de archivos de actualización [28], [29]. Además, las firmas digitales y las comprobaciones de actualidad del firmware aseguran que solo se desplieguen en la red versiones autorizadas y actualizadas del mismo [15]. No obstante, la inclusión de mecanismos de ciberseguridad se contrapone ante el resto de requerimientos de las arquitecturas *FOTA*, especialmente debido a sus costos computacionales y energéticos elevados, por lo que la elección de protocolos y librerías adaptadas a las restricciones de los nodos resulta crítica [30].

B. Servicios comerciales para FOTA

Ante la creciente popularidad de *FOTA*, varios proveedores de servicios de computación en la nube han desarrollado sus propias soluciones comerciales que permiten su implementación en *WSN*. Generalmente, las arquitecturas comerciales dependen de la conexión de los nodos a un servicio web establecido por el proveedor, quien se encarga de aspectos clave como la distribución de firmware y la autenticación u otras formas de seguridad en caso de disponer de ellas. No obstante, la cantidad de servicios existentes dificulta la selección de un único proveedor capaz de cubrir todos los requerimientos reconocidos en la literatura. Por este motivo, en este trabajo se analizan algunos de los servicios *FOTA* comerciales más relevantes, al tratarse de soluciones ofrecidas por proveedores establecidos en el mercado o por tratarse de servicios enfocados principalmente en tecnologías *IoT* y actualizaciones *FOTA*.

La Tabla I resume las características fundamentales de los principales servicios en la nube identificados que soportan actualizaciones *FOTA*. Estos servicios incluyen plataformas a gran escala como Amazon Web Services (AWS) con su servicio IoT Core [31] y Microsoft Azure con IoT Hub [32], así como soluciones de menor magnitud más enfocadas en servicios específicos para *IoT* como Arduino Cloud [33],

TABLA I
CARACTERÍSTICAS DE SERVICIOS *FOTA* COMERCIALES

Parámetro	AWS	Azure	Arduino Cloud	Blynk IoT	OTAdrive
Manejo de archivos de firmware	Carga de archivos de firmware al servicio S3 y creación de un "*job*" de actualización en IoT Core	Carga de archivos de firmware en contenedores de IoT Hub y despliegue a través del servicio ADU	Programación del nuevo firmware en el editor en línea de Arduino Cloud o envío de archivos a través de la herramienta CLI	Carga de archivos de firmware a través de la consola de Blynk y notificación de actualización utilizando consola web	Carga de archivos de firmware a través de la consola de OTAdrive y notificación de actualización utilizando consola web
Protocolos de comunicación para la actualización	MQTT para notificación, HTTP para descarga	MQTT para notificación, HTTP para descarga	MQTT para notificación, descarga no especificada	No especificados	No especificados
Mecanismos de seguridad	Protocolo TLS en capa de transporte, certificados de cliente generados por AWS, generación de firmas digitales para archivos de firmware	Protocolo TLS en capa de transporte, clave de autenticación o certificados de cliente generados por Azure	Protocolo TLS en capa de transporte, clave de autenticación generada por Arduino Cloud	Protocolos no especificados, clave de autenticación generada por Blynk	Protocolos no especificados, clave de API y autorización única por medio de la consola web
Actualizaciones por grupos	Función nativa a través de grupos estáticos o dinámicos altamente flexibles	Función nativa a través de grupos de actualización con límites	Posible a través de "etiquetas" y herramienta CLI	Función nativa a través de "plantillas" de dispositivos y consola web con limitaciones	Función nativa a través de grupos de dispositivos con limitaciones
Número de dispositivos que se pueden registrar	Virtualmente ilimitado	Un millón de dispositivos por instancia de IoT Hub, máximo 50 instancias por usuario suscrito	Dependiente del nivel de suscripción, máximo de 100 en planes individuales, no especificado para planes empresariales	Dependiente del nivel de suscripción, máximo de 500	Dependiente del nivel de suscripción, máximo de 2 000

Blynk IoT [34] y OTAdrive [35]. Los parámetros comparados corresponden a las características identificadas en la literatura. Todos los servicios evaluados, a excepción de Azure, permiten únicamente actualizaciones integrales del firmware, lo que exige a los nodos descargar el archivo completo para sustituir la versión anterior en cada ciclo de actualización. Dicho enfoque puede resultar en mayores consumos energéticos en comparación con el sistema de actualizaciones *delta* ofrecido por Azure.

En cuanto a la ejecución del proceso de actualización, plataformas como Blynk y OTAdrive requieren que el usuario interactúe mediante una interfaz web para seleccionar los nodos de destino y cargar el archivo de firmware. En contraste, Arduino Cloud ofrece una alternativa mediante una interfaz de línea de comandos (CLI), aunque el mecanismo principal para la ejecución de operaciones *FOTA* es asimismo un panel de control en línea. Por su parte, las plataformas más avanzadas como AWS y Azure incorporan diversas opciones para iniciar la actualización, incluyendo consolas en línea, herramientas CLI y kits de desarrollo de software (SDK) para la integración de sistemas individualizados.

Respecto a los protocolos de comunicación para la descarga de firmware, las plataformas a gran escala como AWS y Azure especifican el uso de HTTP, MQTT o una combinación de los mismos. La carencia de especificaciones detalladas sobre estos mecanismos en otros servicios complica la implementación de bibliotecas adicionales y promueve una dependencia hacia los módulos de software preestablecidos, lo cual limita la flexibilidad y aplicabilidad de estas soluciones en nodos existentes donde se busca integrar un sistema *FOTA*. Por ejemplo, Arduino Cloud emplea módulos propios para comunicación MQTT, mientras que el resto de soluciones analizadas no revelan los protocolos de comunicación empleados.

Desde la perspectiva de la ciberseguridad, Azure y AWS ofrecen las características más completas, proporcionando mecanismos como la generación de certificados TLS para cada dispositivo registrado. En contraste, los demás servicios no clarifican del todo los mecanismos de seguridad empleados. Por lo general, la ciberseguridad en estos servicios de menor escala se basa en la generación de claves de autorización o autenticación para verificar los nodos, sin garantizar de forma explícita la encriptación durante la transmisión de datos.

Para realizar operaciones de actualización en grupo, AWS se destaca por ofrecer avanzadas funcionalidades de gestión de grupos. Su sistema permite a los administradores de la red crear grupos estáticos de forma manual o generar grupos dinámicos de manera automática, basándose en propiedades adicionales definidas durante el registro de los nodos. Además, el sistema admite que un mismo nodo forme parte de hasta 10 grupos distintos a la vez. En comparación, las soluciones ofrecidas por Azure, Blynk y OTAdrive presentan funcionalidades de actualización en grupo más limitadas, restringiendo cada nodo a un único grupo. A su vez, Arduino Cloud permite actualizar múltiples nodos a través de la herramienta CLI, asignándoles etiquetas, aunque la documentación disponible para estas funciones es escasa.

En términos de escalabilidad, las plataformas ofrecidas por Arduino Cloud, Blynk y OTAdrive limitan seriamente la cantidad de dispositivos que pueden registrarse. En contraposición, Azure permite un registro de hasta cincuenta millones de dispositivos y AWS no establece limitaciones en este aspecto.

III. Metodología

A. Diseño y características de la arquitectura comercial

A partir de la comparación de servicios presentada anteriormente, se eligió AWS para el desarrollo de la arquitectura comercial. Los principales motivos para elegir este servicio frente al resto de opciones incluyen la posibilidad de administrar y actualizar grupos de nodos de forma nativa, la escalabilidad técnicamente ilimitada de la *WSN* en cuanto al número de nodos que se pueden registrar y las mejoras en términos de ciberseguridad. La arquitectura diseñada incorpora múltiples servicios de AWS, que incluyen: IoT Core, Lambda, DynamoDB, S3 y la administración de roles IAM.

El servicio IoT Core de AWS es una colección de herramientas de nube para la administración, registro y control de dispositivos *IoT*. Entre algunas de sus características más importantes, se incluye la capacidad de conectar y administrar grandes cantidades de nodos (llamados *Things* en la nube de AWS); la centralización, análisis y presentación de información recopilada por sensores remotos, así como también la generación de alarmas y detección de eventos o anomalías; el control y ejecución de acciones remotas en nodos de redes *IoT* y *WSN*, incluyendo reinicios, secuencias de código, actualizaciones de firmware; entre otros servicios.

Por su parte, Lambda es el servicio de computación en la nube de AWS, caracterizado por su versatilidad y la posibilidad de ejecutar código en varios lenguajes de programación. Lambda permite combinar, conectar e integrar la mayoría de herramientas de AWS, compartiendo información entre servicios, además de responder de forma programática ante diversos eventos.

A diferencia de la mayoría de arquitecturas descritas en trabajos de investigación previos, esta propuesta elimina la necesidad de mantener un servidor local gracias a los servicios de aplicación *server-less* de AWS. Esto implica que, en lugar de mantener la información de la red de forma local en bases de datos relacionales, dicha información se almacena en una tabla de DynamoDB para facilitar la interconexión entre componentes de la arquitectura. DynamoDB es un servicio de AWS que permite la creación de bases de datos no relacionales o de estructura flexible (NoSQL), donde cada entrada se compone de una clave principal y múltiples claves asociadas que a su vez pueden contener subclaves. La flexibilidad de este servicio admite el crecimiento horizontal de las bases de datos creadas, lo que lo vuelve atractivo para sistemas escalables a pesar de la posible reducción de rendimiento frente a esquemas clásicos basados en SQL.

Ahora bien, aunque DynamoDB sea capaz de almacenar información clave con respecto a los nodos de una *WSN*, como parámetros de funcionamiento o versiones de firmware instaladas, toda arquitectura *FOTA* requiere asimismo de otra

clase de almacenamiento al cual los nodos puedan acceder para descargar archivos de actualización. Para este propósito se adopta el servicio de almacenamiento *Simple Storage Service* o S3 de AWS. Este servicio admite el almacenamiento de datos tanto estructurados como no estructurados en formato de objetos. Además, empleando las características nativas del servicio de almacenamiento S3, es posible generar enlaces únicos de descarga o puntos de acceso, simplificando la descarga de actualizaciones en nodos *WSN*.

Finalmente, el servicio de administración de acceso e identidad (IAM) de AWS es una herramienta que permite la creación de roles de acceso a la consola de AWS. Cada rol creado se asocia a un grupo de "políticas" o permisos específicos dentro de la consola de AWS. De esta manera, con el servicio IAM de AWS es posible detallar a qué recursos específicos dentro de la nube de Amazon puede acceder cada usuario para efectuar cambios. Estas características pueden ser especialmente útiles en *WSN* a gran escala, donde grupos de personas trabajan en la administración de distintos aspectos de la red. Asimismo, la asignación de permisos específicos protege la arquitectura frente a cambios accidentales en la información de la *WSN* y otras vulnerabilidades.

La Fig. 1 muestra los componentes principales de esta arquitectura, así como también la secuencia básica de actualización. Este sistema no incluye funcionalidades para el almacenamiento ni recepción de datos recopilados por los nodos, de modo que separa el proceso de actualización de la recopilación de datos del entorno de la *WSN*. La comunicación se realiza a través de dos protocolos distintos para notificar y descargar los archivos de actualización (MQTT y HTTPS, respectivamente). Esto implica que la notificación de nuevas actualizaciones inicia con una comunicación activa por parte del sistema hacia los nodos. En términos de seguridad, el protocolo TLS es el estándar utilizado tanto para MQTT como para HTTPS, siendo AWS el encargado de proveer certificados de confianza para la infraestructura de nube y para los nodos que actúan como clientes del sistema. Cada nodo registrado en el servicio *IoT* de AWS recibe un certificado de cliente único, diferente al del resto de nodos.

La secuencia de actualización en la arquitectura comercial inicia cuando el administrador de red carga un archivo de firmware en un *bucket* del servicio de almacenamiento S3 de AWS a través de la consola en línea del servicio de nube (1). Seguido de esto, una herramienta CLI diseñada para comunicarse con la nube de AWS autentica al administrador para gestionar las operaciones *FOTA* (2). Una vez autenticado, el usuario accede al listado de nodos y grupos registrados (3), a partir del cual se genera una lista de nodos objetivo por actualizar en formato JSON. Posteriormente, la herramienta CLI carga automáticamente este listado al mismo *bucket* S3 (4). Una función Lambda escrita en Python se encarga de actualizar la versión objetivo de los nodos en la tabla de DynamoDB, así como de enviar un mensaje de notificación a cada uno de los nodos objetivo a través de un tópico MQTT (5). Dicha función es invocada cada vez que un listado se carga al *bucket*. Tras recibir la notificación, los nodos objetivo descargan el archivo de actualización del *bucket* utilizando el protocolo HTTPS (6). Una vez finalizada la actualización, los nodos publican un mensaje de MQTT que contiene la versión actual del firmware instalado (7). Por último, una regla de mensajería *IoT* que escucha todos los mensajes de actualización remota redirige los mensajes de los nodos hacia una segunda función Lambda cuyo objetivo es actualizar la entrada de versión actual de los nodos en la tabla de DynamoDB (8).

Otra de las funciones de la tabla de DynamoDB es proporcionar un mecanismo de visualización de información para el administrador de la *WSN*, permitiendo así comprobar el estado actual de los nodos. A pesar de que la tabla no se relaciona directamente con el disparo de eventos, este mecanismo se emplea con el fin de presentar un medio para la verificación

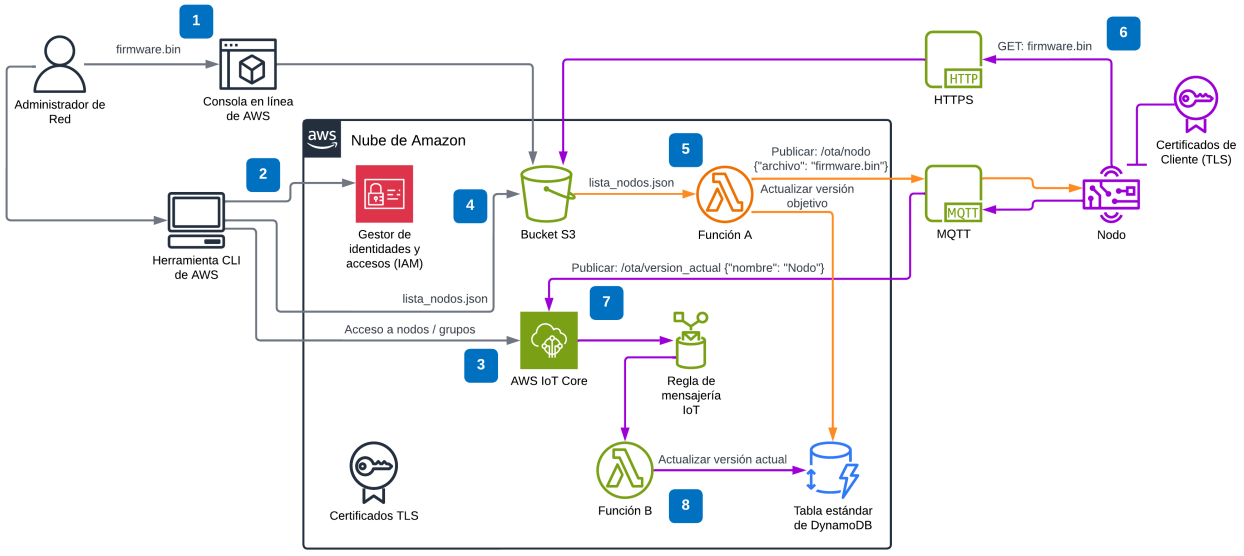

Fig. 1. Arquitectura comercial basada en los servicios de AWS.

de procesos de actualización *FOTA*.

B. Diseño y características de la arquitectura independiente

La arquitectura independiente desarrollada en este trabajo basa su estructura en una síntesis de requerimientos descritos en varios trabajos del estado del arte, tomando en consideración criterios de eficiencia energética y computacional, escalabilidad, robustez y seguridad necesarios para el correcto desempeño de las actualizaciones *FOTA*. Los componentes principales de esta arquitectura son un proxy inverso y un servidor local, interconectados entre sí a través de un túnel de comunicaciones seguro y encriptado. Adicionalmente, el servidor local mantiene información sobre los usuarios con permiso de administración y los nodos de la *WSN*, además de ejecutar servicios web y APIs accesibles para los actores descritos.

Un proxy inverso es un servidor especializado capaz de recolectar y redireccionar las solicitudes provenientes de distintas partes del Internet hacia un servidor web de *backend*. La inclusión de este intermediario en la comunicación permite proteger a los servidores que procesan las solicitudes de clientes, evitando la exposición innecesaria de puertos y direcciones IP. Además, los proxys inversos permiten agregar características de balance de cargas para evitar ataques distribuidos de denegación de servicios. Por último, un proxy inverso también puede aliviar la carga computacional de los procesos de encriptación y desencriptación TLS, permitiendo que los servidores *backend* dediquen sus recursos exclusivamente a los servicios que ofrecen. En el diseño de arquitectura propuesto, cualquier servicio de proxy inverso puede ser integrado. Sin embargo, para la implementación de la misma se ha seleccionado el servicio de Cloudflare debido a su amplio uso a nivel comercial y las capacidades de ciberseguridad y configuraciones adicionales que ofrece. El servicio de Cloudflare incluye la emisión de los certificados requeridos para la comunicación segura entre nodos y servidor local, así como el procesamiento de cualquier solicitud antes de ser enviada a este último.

Por su parte, el servidor local ejecuta la API que utilizan los nodos para reportar datos recolectados por sus sensores y ejecutar actualizaciones *FOTA*. Para el diseño de la API, se eligió el framework de desarrollo web FastAPI, basado en Python. Este framework moderno de alto rendimiento permite la implementación de APIs complejas utilizando una sintaxis simplificada. Luego de definir los *endpoints* específicos para la interacción de los nodos con el servidor local, se integró una base de datos SQL para administrar la *WSN*. Esta base de datos contiene información estructurada sobre los nodos que conforman la red, los grupos a los que pertenece cada nodo, su versión de firmware y las actualizaciones pendientes. Adicionalmente, el servidor local cuenta con un segundo servicio web a través del cual los administradores de la *WSN* pueden iniciar procesos de actualización o administrar los grupos de nodos existentes. Toda comunicación iniciada por los nodos o por los administradores con el servidor local utiliza el protocolo HTTPS, además de ser direccionada al *endpoint* correcto para su procesamiento, evitando que esta comunicación sea fácil de replicar para actores externos.

A más de representar gráficamente la estructura de la arquitectura independiente, la Fig. 2 incluye el flujo básico requerido para completar una actualización remota. En primer lugar, el administrador de la *WSN* carga un archivo de firmware a través de la GUI Web, que el servidor guardará en su almacenamiento interno (1) y utiliza la misma interfaz para modificar la versión objetivo de un nodo o un grupo de nodos en la base de datos del servidor (2). Por su parte, el nodo envía la información recopilada del entorno al servidor periódicamente para ser procesada y almacenada (3). Aprovechando esta comunicación iniciada por cada nodo, el servidor analiza la base de datos comparando la versión actual del firmware del nodo contra la versión objetivo establecida por el administrador (4). En caso de existir diferencias, el servidor incluye un comando de actualización en respuesta a la información compartida por el nodo. Una vez notificado de las actualizaciones pendientes, el nodo descarga el archivo de firmware correspondiente desde el almacenamiento del

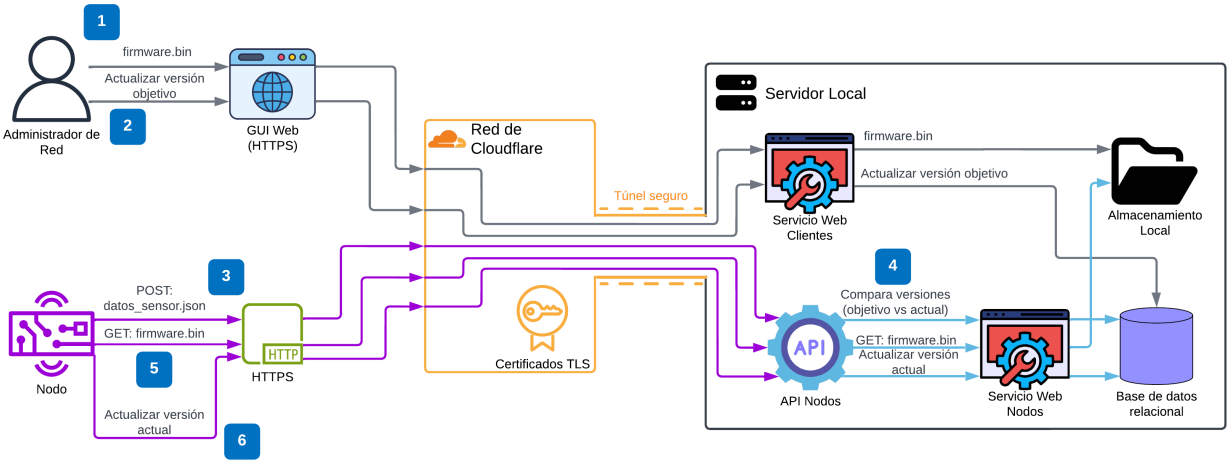

Fig. 2. Arquitectura independiente.

servidor (5), y tras finalizar el proceso de sobrescritura del firmware, notifica al servidor su nueva versión de firmware a fin de actualizar la base de datos (6).

C. *Integración de las arquitecturas en el nodo sensor*

Con el fin de evaluar el desempeño de cada arquitectura en nodos de una *WSN* real, se desarrolló un conjunto de librerías de código en C++ para cada una de ellas. Las librerías incluyen los métodos necesarios para ajustar el comportamiento de los nodos a las secuencias básicas de actualización detalladas anteriormente, y aunque fueron implementadas siguiendo el framework de Arduino, es posible generalizar su diseño para cualquier otro framework de desarrollo. Estas librerías deben incluirse en el firmware de los nodos para entablar las comunicaciones requeridas en cada arquitectura *FOTA* diseñada.

Las librerías requieren que los nodos dispongan de un sistema de almacenamiento externo (ej. microSD) tanto para alojar temporalmente los archivos de actualización como para almacenar información persistente ante reinicios. Esto incluye certificados de confianza, identificadores únicos, la versión de firmware actualmente instalada y otros parámetros.

Las operaciones efectuadas por la librería diseñada durante un ciclo de actualización en la arquitectura comercial se detallan en la Fig. 3. La librería incluye métodos para leer certificados de confianza desde la memoria externa e iniciar un canal de comunicación seguro basado en los protocolos TLS y MQTT. Al incluir certificados tanto para el cliente como para el servidor, el inicio de sesión MQTT espera a la autenticación cruzada entre ambas partes antes del intercambio de datos. Una vez iniciada la comunicación, el nodo notifica a la nube de AWS sobre su versión actual de firmware. Posteriormente, la infraestructura de AWS notificará al nodo sobre cualquier actualización disponible a través de tópicos MQTT, lo que a su vez dispara una secuencia de descarga y sobrescritura de la memoria flash en el nodo. Luego de culminar el proceso de actualización, el nodo se reinicia y vuelve a arrancar con la nueva versión de firmware instalada.

De manera similar, la Fig. 4 presenta la secuencia implementada para la librería de *FOTA* en la arquitectura independiente. A diferencia de la arquitectura implementada con servicios de AWS, este sistema no cuenta con un *broker* MQTT para la comunicación, por lo que al inicializar la librería, el nodo notifica su versión de firmware actual a través de HTTPS. Además, se utiliza un único certificado de confianza que permitirá al nodo determinar si el servidor con el cual ha establecido la comunicación es efectivamente el adecuado. En este caso, el proceso de verificación es efectuado únicamente por el cliente, quien notifica al servidor sobre su versión actual de firmware luego de haber autenticado la comunicación. Otra de las diferencias considerables entre la arquitectura independiente y la comercial consiste en el proceso de notificación de actualizaciones. En este caso, el servidor carece de capacidades para iniciar una comunicación activa con el nodo, por lo que aprovecha las comunicaciones iniciadas por el nodo para incluir datos de actualización en el cuerpo de la respuesta HTTPS. Nuevamente, al recibir la

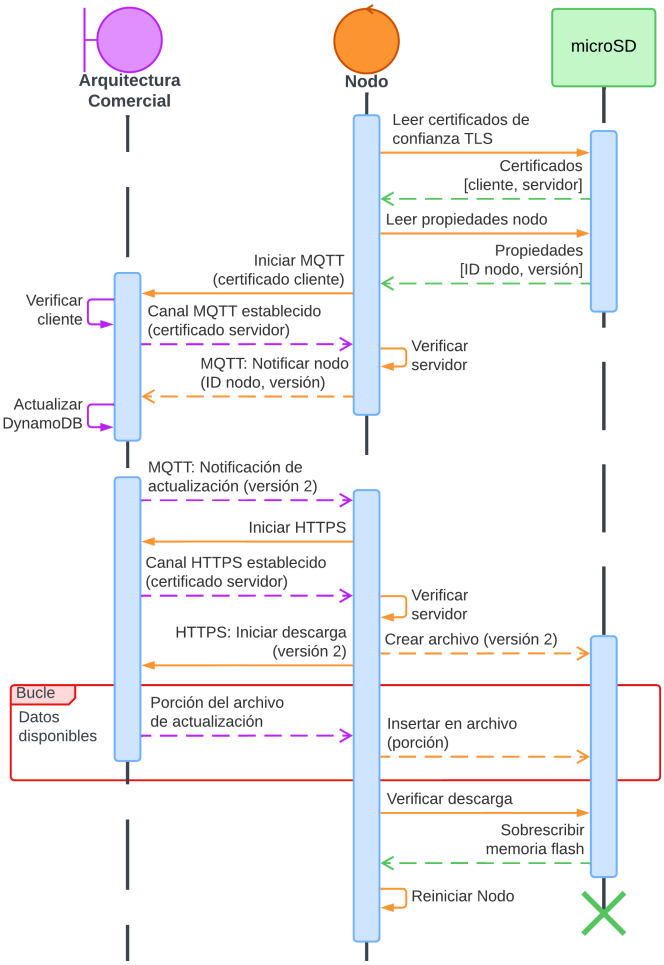

Fig. 3. Secuencia de actualización en la arquitectura comercial.

notificación de actualizaciones disponibles, el nodo inicia una solicitud de descarga del nuevo firmware que se almacenará en la memoria microSD externa. Al finalizar el proceso de descarga, la memoria flash del nodo es sobrescrita con el nuevo código para finalmente reiniciar y cargar la última actualización.

D. *Métricas para la evaluación de las arquitecturas*

Con base en los requerimientos descritos en la sección II, se eligieron métricas apropiadas para evaluar y comparar las arquitecturas implementadas y su impacto en la operatividad de una *WSN*. En términos de eficiencia energética, se compara el consumo adjudicado al proceso de actualización de cada arquitectura. Para distinguir la cantidad de energía consumida al realizar procesos de actualización del consumo base de un nodo, se calcula el sobrecoste energético ΔE, definido como:

$$\Delta E = \int_{t_0}^{t_f} [P_A(t) - P_B(t)]\, dt \tag{1}$$

Donde $P_B(t)$ representa la curva base de potencia instantánea en el nodo, medida para un período de tiempo en el que no se realiza ningún proceso de actualización, y $P_A(t)$ la

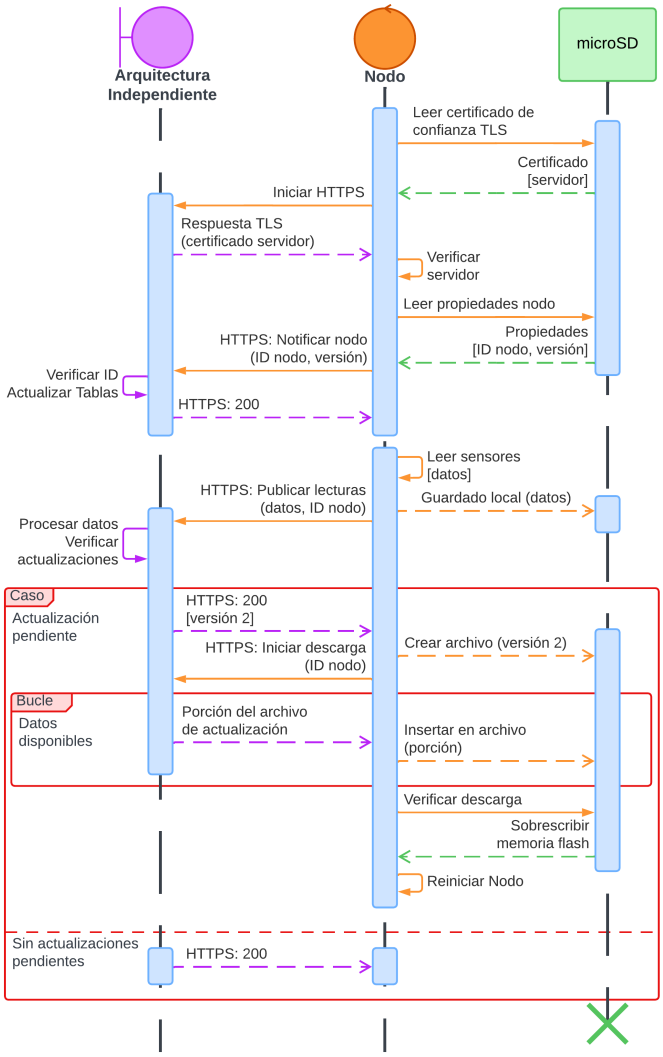

Fig. 4. Secuencia de actualización en la arquitectura independiente.

curva de potencia instantánea en el nodo para un período de tiempo durante el cual se realizan actualizaciones periódicas a intervalos controlados en la arquitectura evaluada.

Por otro lado, parámetros como la latencia en la comunicación pueden afectar la robustez y confiabilidad de un sistema *FOTA*. Idealmente, las actualizaciones deben ocurrir tan pronto como se despliega una nueva versión de firmware. Sin embargo, los tiempos de descarga del firmware pueden verse afectados por parámetros como el tamaño del mismo, la complejidad de los protocolos de comunicación seleccionados para su difusión, el tráfico general en la red, la arquitectura del sistema *FOTA*, entre otros. Por este motivo, se considera el tiempo de descarga (τ_d) como una métrica a evaluar. Esta se define como:

$$\tau_d = t_r - t_n \tag{2}$$

Donde t_n define el instante de tiempo en el que el nodo recibe la notificación de actualización para cada arquitectura. En la arquitectura comercial, este instante corresponde al momento en que el mensaje MQTT de notificación es procesado por el nodo, mientras que en la arquitectura independiente es el instante en el que el nodo recibe la respuesta del servidor ante el POST de datos por HTTPS. Finalmente, t_r corresponde al instante de tiempo en el que el nodo ha terminado la secuencia de reinicio tras sobrescribir su memoria flash con la nueva actualización en ambas arquitecturas.

En una *WSN*, dependiendo del caso de uso, la cantidad de nodos desplegados puede incrementar sustancialmente, introduciendo mayores cargas computacionales al sistema centralizado encargado de manejar las actualizaciones. Un mayor número de nodos podría incrementar los tiempos de despliegue o incluso impedir la correcta difusión de firmware a todos los nodos, lo que podría resultar crítico, especialmente en aplicaciones con requerimientos estrictos de resolución temporal o de seguridad. Por ello, la tercera métrica a considerar es el tiempo de propagación durante actualizaciones de grupo. Sea N un grupo de nodos perteneciente a la *WSN*, t_{n_k} el tiempo en el que el k-ésimo nodo del grupo ha sido notificado sobre la actualización pendiente, $t_m = \min\{t_{n_k} \,|\, k \in N\}$, t_{r_k} el tiempo en el que el k-ésimo nodo del grupo ha finalizado la secuencia de reinicio tras el proceso de actualización y $t_M = \max\{t_{r_k} \,|\, k \in N\}$. Entonces, el tiempo de propagación (τ_p) para actualizar el grupo de nodos N se define como:

$$\tau_p = t_M - t_m \tag{3}$$

En cuanto a los requerimientos de seguridad, la validación de estos atributos suele ser compleja y difícil de cuantificar. Por dicho motivo, se parte de una verificación de los vectores de ataque y vulnerabilidades más comunes en *WSN*, que permitirán evaluar de forma cualitativa las arquitecturas implementadas. Posteriormente, se analizan los mecanismos incluidos por cada arquitectura para mitigar estas vulnerabilidades y se asigna una calificación que permita determinar cuál de las dos arquitecturas presenta un mejor desempeño de forma comparativa. Por otro lado, para evaluar la escalabilidad de cada arquitectura se tomará en cuenta la cantidad de nodos que cada una de ellas sería capaz de integrar sin sufrir reducciones significativas en su rendimiento. Finalmente, la flexibilidad tecnológica de las arquitecturas se evalúa de forma cualitativa para ambas arquitecturas, según la posibilidad de implementar reglas de comunicación personalizadas, pasos adicionales de encriptación y mantener la confidencialidad de la información recopilada por la *WSN*.

E. Diseño experimental

Para obtener las características cuantitativas de comparación de las arquitecturas, se implementaron las librerías diseñadas en nodos sensores de prueba que fueron posteriormente sometidos a pruebas experimentales. El hardware que compone cada nodo incluye un microcontrolador ESP32, una memoria microSD externa con interfaz SPI, un reloj de tiempo real (RTC) con interfaz I2C, un sensor de corriente y voltaje integrado en el sistema INA3221 y baterías de polímero de litio de $8{,}4$ voltios para alimentar al sistema completo. El

microcontrolador cuenta con una memoria flash de 4MB, memoria RAM integrada y un conector para antena externa compatible con Wi-Fi y BLE. La comunicación inalámbrica para los procesos de actualización se realiza a través de Wi-Fi y la red móvil de comunicaciones.

El primer experimento cuantifica el sobrecoste energético neto debido al proceso de actualización *FOTA* (ΔE) en las dos arquitecturas diseñadas. Para ello, se parte de una medición de la potencia base del nodo ($P_B(t)$) sin disparar ningún proceso de actualización. Aprovechando el hardware propio del nodo, se realizan mediciones de corriente y voltaje ininterrumpidamente por 15 horas, con una resolución temporal de 6 muestras por minuto. Una vez obtenida la curva $P_B(t)$ del nodo, se repite el mismo procedimiento para obtener curvas de potencia con cada arquitectura, pero esta vez disparando procesos de actualización a intervalos controlados de 3 minutos ($P_A(t)$). Este tiempo se selecciona debido a que es el intervalo mínimo necesario, común a ambas arquitecturas, entre dos actualizaciones FOTA consecutivas para garantizar su correcto despliegue. Las mediciones de corriente y voltaje, que poseen la misma resolución temporal indicada anteriormente, se almacenan en formato CSV en la tarjeta microSD del nodo para su posterior análisis.

Debido a la naturaleza discreta de las mediciones capturadas por el sensor de corriente y voltaje incorporado en el nodo, se emplea la ley del trapecio descrita en (4) para aproximar la diferencia de consumo energético provocado por los procesos de actualización, donde D_i y D_{i+1} representan las diferencias entre las potencias instantáneas con y sin actualizaciones para los tiempos t_i y t_{i+1} respectivamente, y n el número de muestras disponibles.

$$D_i = P_A(t_i) - P_B(t_i) \quad D_{i+1} = P_A(t_{i+1}) - P_B(t_{i+1})$$

$$\Delta E \approx \sum_{i=1}^{n-1} \left(\frac{[D_i] + [D_{i+1}]}{2} \right) (t_{i+1} - t_i) \tag{4}$$

El segundo experimento determina el tiempo de descarga (τ_d) para cada arquitectura de firmware, además de analizar la influencia del tamaño de firmware sobre esta métrica. Para ello, se implementaron las librerías diseñadas en un único nodo sensor conectado a la red Wi-Fi generada por un router 4G BluCastle que emplea la red de comunicaciones móviles GPRS para establecer la conexión con la nube de AWS o el servidor de Cloudflare en cada caso. Esta configuración busca semejar la conexión de un nodo perteneciente a una *WSN* de gran extensión geográfica tanto como sea posible. Una vez generada la conexión, se realizan varias actualizaciones consecutivas disparadas por una RaspberryPi 3 Model B en intervalos predeterminados para cada tamaño de firmware. Este dispositivo busca simular la acción del administrador de la *WSN*, quien se encargaría de disparar procesos de actualización, sean individuales o por grupo. Finalmente, en el firmware de los nodos se incluye la funcionalidad necesaria para generar pulsos digitales cada vez que se procesa un mensaje o termina una actualización *FOTA*, que, al ser capturados por la RaspberryPi, permiten medir los tiempos t_n y t_r.

Con el fin de determinar la influencia del tamaño del firmware desplegado sobre el tiempo de descarga, se diseñaron tres versiones del firmware de tamaños pequeño, mediano y grande. Estos tamaños fueron determinados en función de la cantidad mínima de memoria requerida para implementar un código funcional y el esquema de particiones de memoria de los microcontroladores de los nodos. Por defecto, al programar el ESP32, la memoria flash se divide en 6 particiones dedicadas a distintos propósitos: *NVS*, *coredump*, *app0*, *app1*, *otadata* y *SPIFFS*. La partición de almacenamiento no volátil, o *NVS* por sus siglas en inglés, permite al microcontrolador guardar información de variables específicas que persisten en la memoria incluso tras ocurrir fallas de alimentación o ciclos de reinicio. La partición de *coredump* guarda automáticamente información referente a fallas del microcontrolador en tiempo de ejecución. En cuanto a las particiones de aplicación *app0* y *app1*, se trata de las secciones de la memoria flash donde es permitido instalar código fuente para la ejecución de programas en el microcontrolador. Dichas particiones son de extrema importancia, pues, a más de determinar el comportamiento del microcontrolador, son las únicas susceptibles a cambios durante las actualizaciones *FOTA*. La partición *otadata* se encarga de almacenar datos referentes a las actualizaciones remotas, permitiendo al *bootloader* arrancar desde la partición donde se completó la última actualización válida. Finalmente, la partición más grande incluida por defecto es aquella dedicada al sistema de archivos en flash (*SPIFFS*), cuyo propósito es el de admitir la lectura y escritura de archivos arbitrarios en la memoria del ESP32.

TABLA II
ESQUEMA DE PARTICIONES MODIFICADO PARA EXPERIMENTACIÓN EN LOS NODOS EMPLEADOS

Partición	**Tipo**	**Tamaño (KB)**
nvs	Datos	20
otadata	Datos	8
app0	Aplicación	1 984
app1	Aplicación	1 984
coredump	Datos	64
Total		4 060

El tamaño mínimo de firmware para la implementación de las arquitecturas en los nodos es de 971KB. Por lo tanto, dicho valor constituye el tamaño de firmware pequeño empleado en los experimentos. Sin embargo, dado que esta versión mínima del firmware cubre más del 90 % de las particiones de aplicación por defecto, fue necesario definir un esquema de particiones modificado para implementar los tamaños restantes. Para ello, se eliminó la partición del *SPIFFS* y se distribuyó este espacio adicional entre las particiones de aplicación, tal como se detalla en la Tabla II. Ahora bien, aunque la modificación de las particiones permite extender el tamaño de las aplicaciones hasta los 1 984KB, es recomendable reservar parte de la

memoria para operaciones de asignación en runtime, por lo que las pruebas con firmware de tamaño grande emplean un firmware modificado para alcanzar los 1 979KB. En consecuencia, el firmware de tamaño medio ocupa aproximadamente 1 475KB de las particiones de aplicación, representando justamente el punto medio entre los tamaños pequeño y grande. Durante el proceso de actualización, el archivo de firmware se descarga inicialmente en la memoria microSD externa. Una vez finalizada la descarga, el microcontrolador sobrescribe la partición de aplicación inactiva para luego arrancar desde dicha partición. Esto implica que, con cada actualización, el microcontrolador alterna la partición activa entre *app0* y *app1*.

Finalmente, para la medición del tiempo de propagación (τ_p) se emplean cuatro nodos que representan un grupo N desplegado. Las cuatro unidades se conectan a la red Wi-Fi generada por el router BluCastle. Al igual que en el experimento anterior, se emplea una RaspberryPi tanto para disparar procesos seguidos de actualización como para la detección de flancos digitales que permitan medir los tiempos t_{n_k} de cada nodo conectado al sistema de adquisición. En un análisis posterior se determinan los valores de t_m y t_M.

F. Proceso analítico jerárquico

Una vez obtenidas las métricas para la comparación de las arquitecturas, se ponderan las características tanto cuantitativas como cualitativas de cada sistema, empleando como caso de uso una *WSN* para la medición de variables ambientales y contaminantes atmosféricos. Para determinar cuál de las arquitecturas presenta una mejor operatividad, se utiliza una metodología basada en el proceso analítico jerárquico *AHP*.

La decisión entre m opciones ($\mathcal{A} = \{A_1, \ldots, A_m\}$), en este caso las arquitecturas *FOTA* propuestas, se determina por un conjunto de n criterios de evaluación, definido como $\mathcal{C} = \{c_1, \ldots, c_n\}$. Dada la naturaleza mixta del análisis que se pretende efectuar para la evaluación operativa de las arquitecturas *FOTA*, donde se incluyen parámetros tanto cuantitativos como cualitativos, el proceso analítico jerárquico se construye como sigue:

1) Descomposición jerárquica del problema: El proceso analítico jerárquico permite descomponer el problema de selección de una u otra arquitectura *FOTA* en múltiples niveles jerárquicos. El primer nivel en la jerarquía es siempre el objetivo de la comparación efectuada, que en este estudio es la selección de la arquitectura *FOTA* con mayor puntaje de operatividad. El segundo nivel describe los criterios globales de comparación o evaluación que se analizan para cada una de las opciones. En el tercer nivel de la jerarquía, cada criterio global se divide en subcriterios que se evalúan independientemente uno del otro. Aunque la descomposición de los criterios globales es opcional, esta facilita la comparación de las alternativas consideradas cuando existen muchos parámetros de evaluación. Finalmente, el último nivel de la jerarquía incluye todas las alternativas consideradas para el proceso *AHP* [36].

En función de los requerimientos identificados en este estudio, los criterios globales contemplan: consumo energético, latencias en la comunicación, ciberseguridad, escalabilidad y flexibilidad tecnológica. Los dos primeros criterios se consideran de tipo *costo*, donde valores menores son preferidos, mientras que el resto de criterios son del tipo *beneficio*, donde valores mayores representan una ventaja [37].

2) Construcción de las matrices de prioridades: Por cada par de criterios globales, se asigna un puntaje de importancia del criterio i sobre el criterio j en una escala del 1 al 9; donde 1 implica que ambos criterios son igual de importantes, mientras que 9 indica que el criterio i es significativamente más importante que el criterio j en el caso de uso evaluado [38]. En caso de existir subcriterios en la jerarquía, este procedimiento se repite para cada conjunto de subcriterios, de modo que se obtiene una matriz de prioridades por cada grupo de criterios en un mismo nivel de la jerarquía. Todas estas matrices son recíprocas y siguen la forma:

$$\mathbf{P} = \left[p_{ij}\right]_{d\times d}, \quad p_{ij} > 0,\, p_{ji} = {p_{ij}}^{-1},\, p_{ii} = 1$$

$$\mathbf{P} = \begin{pmatrix} 1 & p_{12} & p_{13} & \cdots & p_{1d} \\ 1/p_{12} & 1 & p_{23} & \cdots & p_{2d} \\ 1/p_{13} & 1/p_{23} & 1 & \cdots & p_{3d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1/p_{1d} & 1/p_{2d} & 1/p_{3d} & \cdots & 1 \end{pmatrix} \tag{5}$$

Donde d representa el número de criterios globales o subcriterios dentro de una misma rama de la jerarquía (n en el caso de los criterios globales).

3) Cálculo de los vectores de pesos: Para cada matriz de prioridades $\mathbf{P}$ existe un conjunto de eigenvalores $\sigma(\mathbf{P})$ con sus eigenvectores $\mathbf{v}$ asociados, que satisfacen la ecuación (6). Esta propiedad corresponde a la definición misma de los eigenvectores y eigenvalores, que indica que el resultado del producto entre la matriz $\mathbf{P}$ y cualquiera de sus eigenvectores $\mathbf{v}$ es igual al mismo vector escalado por su eigenvalor λ correspondiente.

$$\forall \lambda \in \sigma(\mathbf{P}),\, \exists\, \mathbf{v} \neq \mathbf{0} \;:\; \mathbf{P}\mathbf{v} = \lambda \mathbf{v} \tag{6}$$

Luego, con base en esta definición, se busca obtener un único vector $\mathbf{w}$ conformado por los pesos relativos de cada criterio (w_i) [38]. Este vector no es más que el eigenvector correspondiente al mayor eigenvalor de la matriz:

$$\lambda_{\max} = \max\{\mathrm{Re}(\lambda) \mid \lambda \in \sigma(\mathbf{P})\} \tag{7}$$

Finalmente, este vector se normaliza dividiendo cada elemento por la suma de todos sus elementos:

$$\forall w \in \mathbf{w}, \quad w_i \leftarrow \frac{w_i}{\sum_{k=1}^{d} w_k}, \quad i = 1, \ldots, d \tag{8}$$

Esto garantiza que la suma de los pesos representados en el vector $\mathbf{w}$ sea 1, de modo que todos los criterios en la matriz $\mathbf{P}$ contribuyen en cierta medida porcentual para cumplir con el objetivo de la comparación.

4) Combinación jerárquica de pesos de criterios y subcriterios: En aquellos casos donde uno o varios criterios globales se descomponen en subcriterios, el cálculo de los vectores de peso sigue una estructura jerárquica. Inicialmente, se calcula un vector de pesos $\mathbf{w}^{(C)}$ a partir de la matriz de prioridades de criterios globales. Posteriormente, para cada criterio compuesto c_i, se construye su respectiva matriz de prioridades de subcriterios, de la cual se obtiene un vector $\mathbf{w}^{(S_i)}$ que representa la importancia relativa de sus subcriterios locales.

El peso global de cada subcriterio se obtiene como el producto del peso del criterio padre y el peso relativo del subcriterio, es decir:

$$\mathbf{w}_G = [w_{ij}], \quad w_{ij} = w_i \cdot s_{ij} \tag{9}$$

Donde w_i es el peso asignado al criterio global c_i en el vector $\mathbf{w}^{(C)}$, s_{ij} es el peso relativo del subcriterio j dentro de la matriz de prioridades local del criterio c_i, y w_{ij} es el peso final del subcriterio en el contexto global del problema.

De este modo, todos los criterios simples y subcriterios derivados de criterios complejos se incluyen en un único vector global de pesos $\mathbf{w}_G$ de tamaño p. Este proceso asegura que los criterios y subcriterios contribuyan proporcionalmente a su importancia en la jerarquía general.

5) Verificación de la consistencia: Otra de las ventajas de la metodología *AHP* es la inclusión de pasos para verificar la consistencia lógica de las comparaciones efectuadas en las matrices de prioridades. Por ejemplo, si el criterio c_1 tiene un peso de 2 por sobre el c_2 y este último tiene un peso igualmente de 2 frente a c_3, es de esperarse que el criterio c_1 tenga una importancia de 4 por sobre c_3. Para verificar esta coherencia en el modelo de comparación, se parte del cálculo del índice de consistencia CI para cada matriz de prioridades $(\mathbf{P})$:

$$\mathrm{CI} = \frac{\lambda_{\max} - d}{d - 1} \tag{10}$$

Entiéndase que en una matriz de prioridades totalmente consistente, se cumple la condición $\lambda_{\max} = d$, por lo que el parámetro $(\lambda_{\max} - d)$ representa una medida de cuánto se aleja la matriz de prioridades de ser totalmente consistente [37]. Dada la cantidad de criterios que pueden llegar a constituir un solo proceso de comparación en *AHP*, no se espera que las matrices de prioridades lleguen a cumplir dicha condición a la perfección. Por este motivo, se calcula la tasa de consistencia CR, que permite comparar el índice de consistencia de las matrices de prioridades con el índice aleatorio de Saaty (RI). Este último representa el valor medio que tomaría el índice de consistencia CI para matrices recíprocas de prioridades llenadas con valores aleatorios, y puede aproximarse en función del número de criterios incluidos en cada matriz de prioridades (d):

$$\mathrm{CR} = \frac{\mathrm{CI}}{\mathrm{RI}(d)}, \quad \mathrm{RI}(d) \approx \frac{1{,}98(d-2)}{2} \tag{11}$$

Si todas las matrices de prioridades cumplen la condición $\mathrm{CR} < 0{,}1$, se puede asumir que las prioridades asignadas son coherentes y continuar con el análisis. Caso contrario, se deben reevaluar las prioridades para garantizar que el análisis sea consistente.

6) Construcción de la matriz de rendimiento normalizada: Una vez definidas las relaciones entre los distintos criterios en la jerarquía (resumidas en las matrices de prioridades), se procede a evaluar cada alternativa bajo dichos criterios. Bajo cada criterio c_i y para cada alternativa A_j, se asigna una puntuación x_{ij}, construyendo una matriz de rendimiento. En el caso de los criterios cuantitativos, la puntuación corresponderá exactamente al valor de la métrica asociada para cada arquitectura, mientras que la puntuación para criterios cualitativos se asigna tras un análisis de características, en una escala del 1 al 10, donde 10 corresponde al mejor desempeño posible. Finalmente, las puntuaciones iniciales son transformadas en valores adimensionales r_{ij} que componen la matriz de rendimiento normalizada $\mathbf{R} = [r_{ij}]_{p\times m}$, en la que cada fila representa un criterio y cada columna una arquitectura. El elemento r_{ij} representa la calificación normalizada de la alternativa j respecto al criterio i:

$$r_{ij} = \begin{cases} \dfrac{\min_j x_{ij}}{x_{ij}}, & \text{si } c_i \text{ es un } \textit{costo} \\[2ex] \dfrac{x_{ij}}{\max_j x_{ij}}, & \text{si } c_i \text{ es un } \textit{beneficio} \end{cases} \tag{12}$$

Esta normalización garantiza que la alternativa con mejor rendimiento en cada criterio de comparación alcance una puntuación de 1, mientras que el resto de puntuaciones caerán en el intervalo $(0, 1]$. Además, este enfoque permite comparar métricas con unidades heterogéneas (como joules y segundos) de forma directa y justa de acuerdo a los valores de importancia asignados a cada una de ellas en la matriz de prioridades.

7) Calificaciones de desempeño: A continuación, se pondera cada una de las columnas de la matriz $\mathbf{R}$ utilizando los pesos w_i de cada criterio o subcriterio, contenidos en el vector global de pesos $\mathbf{w}_G$. Este proceso genera la matriz de decisión ponderada $\mathbf{D}$, obtenida como el producto elemento a elemento entre cada fila de $\mathbf{R}$ y el peso correspondiente del criterio i, es decir:

$$\mathbf{D} = [w_i \cdot r_{ij}]_{p\times m} \tag{13}$$

Esta matriz permite observar con detalle cómo cada alternativa puntúa en cada criterio individualmente, considerando la ponderación relativa asignada a ese criterio en el análisis [36]. Finalmente, el puntaje general de cada alternativa en la metodología *AHP* —que en este caso representa un puntaje de operatividad— se calcula como el producto escalar entre el vector de pesos ponderados $\mathbf{w}_G$ y la columna j de la matriz de rendimiento $\mathbf{R}$:

$$\vartheta_j = \sum_{i=1}^{p} w_i \cdot r_{ij}, \qquad j = 1, \ldots, m \tag{14}$$

El resultado es un vector de puntuaciones finales $\boldsymbol{\vartheta} = [\vartheta_1, \ldots, \vartheta_m]$, donde cada ϑ_j representa el puntaje de operatividad de la alternativa A_j en relación con todos los criterios jerarquizados. Este enfoque no solo permite comparar las arquitecturas *FOTA* implementadas de manera global, sino también identificar en qué criterios específicos cada opción es más o menos competitiva, gracias a la visibilidad de la matriz $\mathbf{D}$. Cabe destacar que, al emplear los valores normalizados tanto de los pesos globales como de la matriz de rendimiento para el cálculo, los valores en la matriz de decisión ponderada se pueden entender como valores porcentuales correspondientes a cada criterio en las dos arquitecturas evaluadas.

IV. Resultados

A. *Sobrecoste energético*

A partir de los experimentos de energía se obtuvieron las series temporales discretas que describen la potencia instantánea del nodo para caracterizar su consumo energético base y mientras se realizan las actualizaciones repetidas en cada arquitectura ($P_B(t)$ y $P_A(t)$ respectivamente). Las curvas de consumo energético acumulado a lo largo de 15 horas se ilustran en la Fig. 5, donde se observa que el consumo base del nodo supera los $7\,845\,J$. Por otro lado, al realizar actualizaciones repetidas a intervalos de 3 minutos, la arquitectura independiente presenta un consumo de $8\,935\,J$, mientras que la arquitectura comercial supera los $9\,328\,J$.

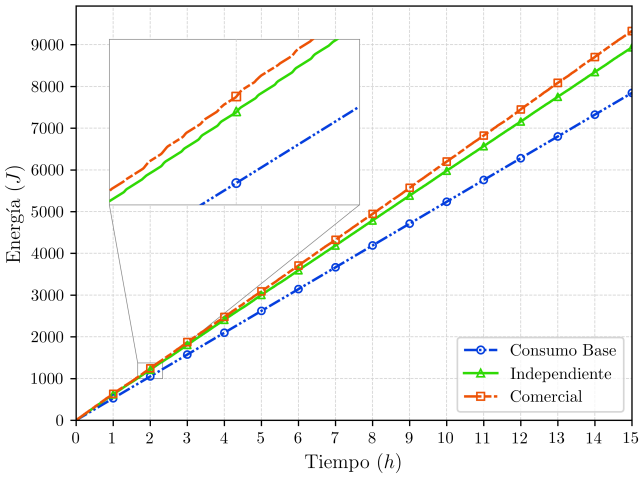

Fig. 5. Curvas de consumo energético acumulado.

A pesar de que las curvas aparentan tener una pendiente constante en la gráfica, al realizar un acercamiento a escala para un intervalo de 40 minutos, las variaciones en la pendiente del consumo energético se vuelven evidentes. Estos incrementos en la pendiente de cada curva de consumo energético representan un incremento en la potencia del nodo durante el proceso de descarga del firmware.

Por otro lado, los valores de sobrecoste energético (ΔE) de cada arquitectura se detallan en la Fig. 6. Además, al considerar el número de ciclos de actualización completados durante el experimento, se obtiene un sobrecoste promedio por ciclo de actualización. Los resultados de este experimento se resumen en la Tabla III.

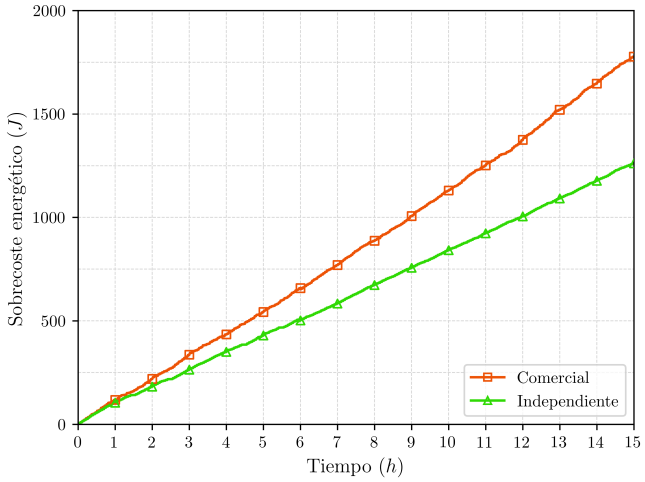

Fig. 6. Sobrecoste energético de cada arquitectura.

TABLA III
Sobrecostes energéticos de cada arquitectura

Arquitectura	Sobrecoste total ΔE	Sobrecoste medio por ciclo de actualización
Independiente	$1\,268{,}43\,J$	$4\,200\,mJ$
Comercial	$1\,789{,}05\,J$	$5\,924\,mJ$

B. *Tiempo de descarga*

Partiendo de las mediciones de los tiempos de descarga τ_d durante las actualizaciones *FOTA* de cada arquitectura, se obtuvo un análisis estadístico basado en la desviación estándar y la media de los tiempos τ_d (Tabla IV). Además, la Fig. 7 muestra un diagrama de caja y bigotes agrupado para cada tamaño de firmware en las dos arquitecturas analizadas. Como se demuestra tanto en la tabla como en la figura, los tiempos τ_d incrementan paulatinamente conforme se aumenta el tamaño del firmware. Además, τ_d tiende a tomar valores más altos en promedio en la arquitectura comercial frente a la arquitectura independiente, mientras que las desviaciones estándar σ de la misma variable son generalmente mayores para la arquitectura independiente. Esto implica que la arquitectura independiente permite descargas más rápidas que la arquitectura comercial,

TABLA IV
Estadísticas para los tiempos de descarga individuales

Arquitectura	Comercial			Independiente		
Firmware	Media	Mediana	σ	Media	Mediana	σ
Pequeño	75,471	75,397	1,351	62,452	62,874	2,843
Mediano	116,541	116,828	3,408	98,856	99,034	3,121
Grande	151,325	150,921	3,078	132,425	134,318	6,030

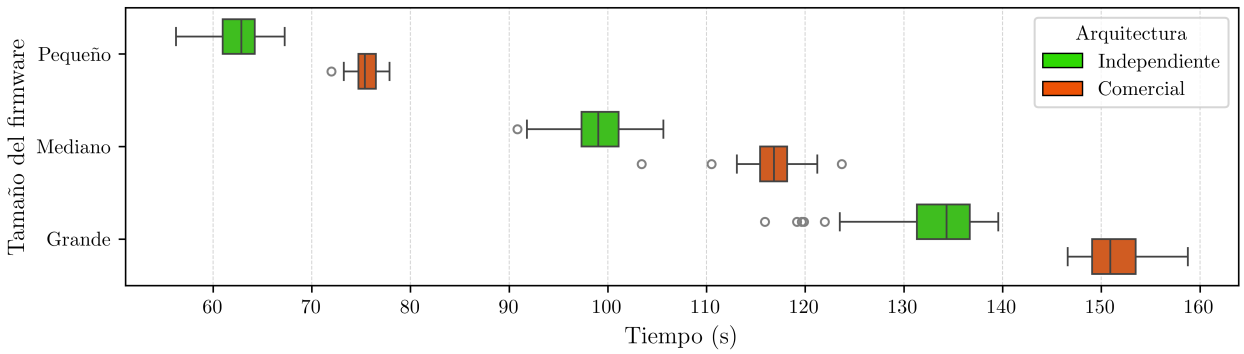

Fig. 7. Tiempos de descarga individuales para distintos tamaños de firmware.

aunque el tiempo de descarga sea menos variable en esta última.

Con el fin de comprobar si este comportamiento es consistente incluso frente a procesos de actualización por grupos de nodos en cada arquitectura, se obtuvieron nuevamente mediciones de los tiempos τ_d, calculados a partir de las mediciones de t_n y t_r durante varios procesos de actualización para un grupo de cuatro nodos. La media de τ_d fue de $70{,}09\,s$ en la arquitectura independiente y de $81{,}11\,s$ en la comercial, valores mayores a los promedios calculados para las actualizaciones individuales. Esto implica que el crecimiento de la *WSN* impacta negativamente los tiempos de descarga en ambas arquitecturas. La distribución de los tiempos de actualización para cada nodo en ambas arquitecturas se muestra en la Fig. 8, donde se observa nuevamente que la arquitectura individual presenta tiempos τ_d generalmente menores, pero con mayor variabilidad frente a la arquitectura comercial.

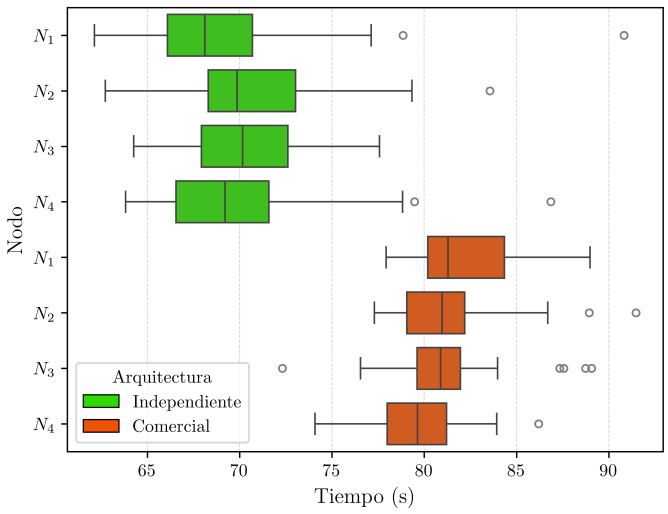

Fig. 8. Tiempos de descarga durante actualizaciones de un grupo de nodos.

Para evaluar la consistencia de los tiempos de descarga τ_d en el grupo de nodos empleado durante los experimentos, se analizan las diferencias estadísticas entre las observaciones reportadas para cada arquitectura. Dado que la mayoría de conjuntos de datos registrados para los nodos en ambas arquitecturas siguen distribuciones no normales de acuerdo con el test de Shapiro-Wilk ($p < 0{,}05$), se utiliza el test de Kruskal-Wallis para la comparación entre los nodos.

Para la arquitectura independiente, el test de Kruskal-Wallis confirmó la hipótesis nula con un valor $p = 0{,}0767$, por lo que se asume que los nodos del grupo no presentaron diferencias significativas en cuanto a los tiempos de descarga. Por otro lado, en la arquitectura comercial se descarta la hipótesis nula por presentar un valor $p < 0{,}01$. En consecuencia, para determinar qué nodos presentan diferencias estadísticas en el grupo, se realizó la prueba de Dunn con corrección de Bonferroni, obteniendo los resultados detallados en la Tabla V. Esta prueba permitió determinar que el único nodo que presenta una diferencia significativa en la media de su tiempo de descarga τ_d es el nodo N_4, mientras que el resto mantienen un valor p suficiente para aceptar la hipótesis nula. Esto implica que las diferencias de τ_d en N_4 no pueden ser directamente atribuidas al funcionamiento de la arquitectura comercial.

TABLA V
PRUEBA POST-HOC DE DUNN EN LA ARQUITECTURA COMERCIAL

	N_1	N_2	N_3	N_4
N_1	1,00	0,836	1,00	$p < 0{,}01$
N_2		1,00	1,00	0,119
N_3			1,00	0,073
N_4				1,00

C. *Tiempo de propagación*

A partir de los tiempos t_m y t_M registrados durante las actualizaciones del grupo de nodos, se calcularon los tiempos de propagación τ_p durante cada ciclo de actualización en las dos arquitecturas. El análisis estadístico de esta métrica reveló que los tiempos de propagación son mayores en la arquitectura comercial. La media de τ_p para el grupo de nodos empleado fue de $77{,}07\,s$ en la arquitectura independiente y $83{,}57\,s$ en la arquitectura comercial. Además, la variabilidad de esta métrica es mayor en comparación con los tiempos de descarga τ_d. Esto es evidente al calcular las desviaciones estándar del tiempo de propagación: $4{,}91\,s$ en la arquitectura independiente

y 3,24 s en la comercial. Por último, las distribuciones de los tiempos de propagación registrados durante el experimento se demuestran en la Fig. 9.

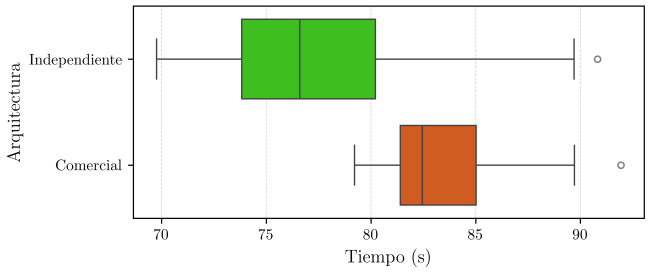

Fig. 9. Distribución de los tiempos de propagación de todos los nodos del grupo para cada arquitectura.

D. *Comparación analítica entre las arquitecturas propuestas*

Tomando en cuenta el caso de uso para el cual se realiza esta comparación y partiendo por los requerimientos descritos en secciones anteriores, se construye la jerarquía representada en la Fig. 10, que incluye los pesos ponderados de cada criterio de evaluación. Los criterios globales se eligieron para representar grupos de métricas o características generales de cada arquitectura *FOTA*, facilitando el proceso de asignación de prioridades. Por ejemplo, el criterio global de latencias en la comunicación agrupa las métricas τ_d y τ_p para evitar tener que comparar la prioridad de ambas opciones individualmente con criterios cualitativos como la flexibilidad tecnológica.

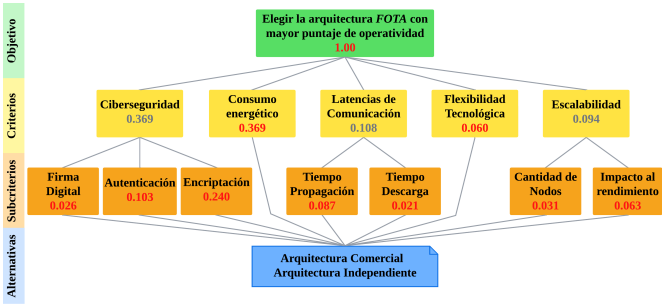

Fig. 10. Jerarquía *AHP* para la selección de la arquitectura *FOTA* con mayor puntaje de operatividad.

Una vez definida la jerarquía, se construye la primera matriz de prioridades, comparando la importancia de los criterios globales por pares. Los valores de importancia asignados se eligieron de modo tal que la eficiencia energética y ciberseguridad sean considerados como máxima prioridad frente al resto de criterios debido a su amplio reconocimiento en la literatura. Ambos criterios comparten un mismo nivel de prioridad para evitar favorecer injustamente a cualquiera de ellos. Por otro lado, se considera la escalabilidad como un criterio ligeramente más importante que la latencia de comunicación y a esta última como prioridad frente a la flexibilidad tecnológica. La matriz de prioridades entre criterios globales $\mathbf{P}_G$ se detalla a continuación, presentando tanto filas como columnas en el mismo orden detallado en la jerarquía construida anteriormente de izquierda a derecha:

$$\mathbf{P}_G = \begin{pmatrix} 1 & 1 & 7 & 4 & 3 \\ 1 & 1 & 7 & 4 & 3 \\ 1/7 & 1/7 & 1 & 3 & 2 \\ 1/4 & 1/4 & 1/3 & 1 & 1/2 \\ 1/3 & 1/3 & 1/2 & 2 & 1 \end{pmatrix} \tag{15}$$

La inclusión de subcriterios que componen la ciberseguridad permite valorar las medidas que cada arquitectura *FOTA* ofrece frente a potenciales vectores de ataque que podrían impactar su funcionamiento. En particular, se evalúan las medidas de seguridad implementadas en la comunicación de los nodos con el sistema centralizado de cada arquitectura (encriptación de la comunicación y los datos intercambiados), el uso de firmas digitales para proteger los archivos de actualización y los mecanismos de autenticación implementados para verificar y proteger el acceso a la plataforma de actualizaciones *FOTA*.

El criterio global referente a las latencias en la comunicación no es más que la combinación ponderada de las mediciones de τ_d y τ_p. Se favorece el tiempo de propagación frente al de descarga por tratarse de una métrica obtenida por grupos de nodos, por lo que se acerca más al comportamiento real de una *WSN*.

Por último, para evaluar la escalabilidad de cada arquitectura se consideran dos subcriterios: la cantidad de nodos que se pueden registrar en cada arquitectura y el impacto que tiene el crecimiento de la *WSN* sobre los tiempos de descarga.

El proceso de asignación de prioridades se repite para cada grupo de subcriterios derivados de la ciberseguridad, las latencias de comunicación y la escalabilidad, resultando en las matrices $\mathbf{P}_C$, $\mathbf{P}_L$ y $\mathbf{P}_E$ respectivamente:

$$\mathbf{P}_C = \begin{pmatrix} 1 & 1/5 & 1/7 \\ 5 & 1 & 1/3 \\ 7 & 3 & 1 \end{pmatrix}, \quad \mathbf{P}_L = \begin{pmatrix} 1 & 4 \\ 1/4 & 1 \end{pmatrix},$$
$$\mathbf{P}_E = \begin{pmatrix} 1 & 1/2 \\ 2 & 1 \end{pmatrix} \tag{16}$$

El vector global de pesos $\mathbf{w}_G$ (de tamaño $p = 9$) se obtiene a partir de la combinación jerárquica de los eigenvectores principales normalizados $\mathbf{w}$ de cada una de las matrices de prioridades construidas, como se describe en la ecuación (9). Los resultados de esta combinación se resumen asimismo en la Fig. 10, de modo que el vector global de pesos incluye los pesos ponderados de los criterios globales simples y los pesos ponderados de todos los subcriterios de la jerarquía. Es decir, se incluyen los valores de todos los criterios o subcriterios directamente conectados al cuarto nivel de la jerarquía en la figura, representados con color rojo. La suma de los elementos de este vector da por resultado la unidad, de modo que los pesos ponderados representan el porcentaje de aporte de cada

criterio o subcriterio para cumplir el objetivo planteado al iniciar la comparación.

La verificación de consistencia de las matrices de prioridades se resume en la Tabla VI, donde se demuestra que la condición $\mathrm{CR} < 0{,}1$ se cumple para todas las matrices de prioridades, por lo que la jerarquía construida mantiene relaciones lógicas entre los distintos criterios elegidos.

TABLA VI
TASAS DE CONSISTENCIA POR MATRIZ DE PRIORIDADES

Grupo de criterios	Matriz	CR
Criterios globales	$\mathbf{P}_G$	0,099
Subcriterios ciberseguridad	$\mathbf{P}_C$	0,056
Subcriterios latencia de comunicación	$\mathbf{P}_L$	0
Subcriterios escalabilidad	$\mathbf{P}_E$	0

Tras definir los pesos ponderados de cada criterio y verificar la consistencia del análisis, se asignan calificaciones a cada arquitectura *FOTA* para construir la matriz de rendimiento normalizada $\mathbf{R}$, cuyos elementos se describen en la ecuación (12). La matriz de rendimiento se presenta en la Tabla VII, con los criterios listados por fila y las arquitecturas por columna. Las calificaciones para el criterio de consumo energético corresponden a los valores normalizados de sobrecoste total en cada arquitectura, tomando en cuenta que se trata de un criterio del tipo *costo*. Para las calificaciones de tiempo de propagación y de descarga, se asignan los valores medios de τ_d y τ_p normalizados bajo la misma consideración. Por su parte, los subcriterios de ciberseguridad, escalabilidad y la flexibilidad tecnológica se evalúan de forma cualitativa a partir de un análisis del diseño de cada arquitectura.

TABLA VII
MATRIZ DE RENDIMIENTO NORMALIZADA

Criterio	Arquitectura comercial	Arquitectura independiente
Consumo energético	0,709	1
Encriptación	1	0,714
Autenticación	1	0,6
Tiempo de propagación	0,864	1
Impacto al rendimiento	1	0,738
Flexibilidad tecnológica	0,5	1
Cantidad de nodos	1	1
Firma digital	1	1
Tiempo de descarga	0,827	1

La arquitectura comercial presenta ventajas claras en cuanto a ciberseguridad frente a la arquitectura independiente. Los mecanismos de encriptación disponibles para los nodos son inherentemente más complejos al incluir certificados bidireccionales durante el intercambio de datos por HTTPS (certificados de confianza tanto para el servidor como para el cliente). De forma similar, los requerimientos de autenticación multifactor y el uso del servicio IAM de AWS presentan ventajas significativas frente a la autenticación únicamente por credenciales de usuario en la arquitectura comercial. Por último, si bien AWS permite el uso de firmas digitales para proteger porciones de código, esta funcionalidad no fue implementada en la arquitectura comercial debido al uso de librerías de código fuera del estándar de AWS para los nodos sensores. En consecuencia, la calificación asignada a ambas arquitecturas en este subcriterio es idéntica.

Tanto en la arquitectura comercial como en la independiente, la cantidad de nodos que pueden registrarse es suficiente para el caso de uso considerado, asumiendo incluso cientos de nodos que conforman la red (garantías de servicio en la primera, funcionamiento de SQL en la segunda), por lo que ninguna se ve favorecida frente a la otra en este aspecto. Por otro lado, el impacto del crecimiento de la red se cuantifica a partir de la diferencia entre las medias de los tiempos de descarga correspondientes a los dos primeros experimentos reportados en esta sección.

La puntuación de flexibilidad tecnológica se asigna en función de la facilidad con la cual se puede alterar el funcionamiento de cada arquitectura. Estos cambios pueden ir desde el incremento en las capas de encriptación hasta la adopción de otros protocolos de comunicación para la difusión del firmware. En este sentido, si bien la calidad de la documentación de los servicios de AWS es innegable y la cantidad de ejemplos oficiales disponibles para cada uno de ellos es considerable, la construcción de una arquitectura independiente desde cero favorece la adopción de múltiples tecnologías sin restricción alguna. De esta manera, la posibilidad de ajustar las características y tecnologías que componen la arquitectura independiente "a medida" le otorgan una clara ventaja frente a la solución comercial.

TABLA VIII
MATRIZ DE DECISIÓN PONDERADA Y PUNTAJES DE OPERATIVIDAD

Criterio	Arquitectura comercial	Arquitectura independiente
Consumo energético	26,18	36,92
Encriptación	23,97	17,12
Autenticación	10,30	6,18
Tiempo de propagación	7,39	8,55
Impacto al rendimiento	6,30	4,64
Flexibilidad tecnológica	3,01	6,02
Cantidad de nodos	3,15	3,15
Firma digital	2,66	2,66
Tiempo de descarga	1,76	2,14
Puntaje de operatividad	84,72	87,38

Concluyendo el proceso de jerarquía analítica, se obtiene la matriz de decisión ponderada $\mathbf{D}$, calculada según la ecuación (13). Asimismo, la sumatoria de los elementos correspondientes a cada arquitectura en esta matriz resulta en el puntaje de operatividad ϑ_j de las mismas. Los valores de la matriz $\mathbf{D}$ se resumen en la Tabla VIII, mientras que las calificaciones generales ϑ_j corresponden a los valores presentados en la última fila de la misma. Finalmente, la metodología sistemática de comparación presentada revela que, aunque la arquitectura

comercial es superior a la independiente en algunos criterios (especialmente aquellos relacionados con la ciberseguridad), la operatividad de la arquitectura independiente es mejor.

V. Conclusiones

El análisis de requerimientos y características esenciales de las arquitecturas *FOTA* presentado a lo largo de este trabajo permitió diseñar, implementar y evaluar dos arquitecturas distintas: una basada en soluciones comerciales y otra a partir de tecnologías independientes. Además, ante la falta de estudios comparativos entre arquitecturas de esta índole, este estudio propone una metodología basada en *AHP* donde se comparó el desempeño general de cada arquitectura. Los criterios considerados incluyeron el consumo energético, la ciberseguridad, las latencias de comunicación, la escalabilidad y la flexibilidad tecnológica. Para evaluar cada uno de ellos se propusieron métricas acordes y se realizaron experimentos exhaustivos.

La revisión de documentación y características de servicios comerciales desarrollada previo a la implementación de las arquitecturas reveló que los servicios de AWS permiten abordar la mayoría de requerimientos básicos de toda arquitectura *FOTA*. Por este motivo, la arquitectura comercial presentada en este trabajo se construyó a partir de la combinación de varios servicios de AWS. Por su parte, el diseño de la arquitectura independiente tomó en cuenta los requerimientos identificados para la selección de tecnologías abiertas que la integraron. Además, se diseñaron librerías de software embebido para la integración de ambas arquitecturas en un grupo de nodos *WSN*, con el fin de evaluar su desempeño. Se demuestra la factibilidad de la implementación de ambas arquitecturas, permitiendo realizar actualizaciones tanto individuales como por grupo en la red *WSN* de prueba.

El diseño experimental seguido en este trabajo permitió valorar el consumo energético de cada arquitectura, los tiempos de descarga y propagación del firmware y el impacto que tiene el crecimiento de la red sobre ellos. Los resultados revelaron que el consumo energético tiende a ser mayor en la arquitectura comercial. Esta particularidad podría relacionarse con la necesidad de mantener constantemente abierto el canal de comunicaciones requerido para el intercambio de mensajes MQTT, aunque sería necesario realizar más experimentos para confirmar este efecto. Asimismo, la arquitectura independiente superó a la comercial en cuanto a sus tiempos de descarga a pesar de presentar mayor variabilidad en estas métricas. Este desempeño superior es atribuible a la menor cantidad de saltos requeridos para el envío de mensajes hacia el servidor central, o al funcionamiento interno de los servicios de AWS. Por otro lado, el impacto del crecimiento de la red fue mayor en la arquitectura independiente, cuyos tiempos de descarga sufrieron mayores incrementos al aumentar la cantidad de nodos por actualizar.

Además de las métricas cuantitativas, el proceso analítico jerárquico permitió valorar parámetros cualitativos como la ciberseguridad y la flexibilidad tecnológica de cada arquitectura. Aunque la arquitectura comercial presenta mejores características de ciberseguridad frente a la independiente, esta otra presenta una flexibilidad tecnológica característica que permitiría adoptar tecnologías adicionales para contrarrestar estas diferencias. En general, el desempeño de la arquitectura independiente fue superior al de la arquitectura comercial de acuerdo con los resultados de la comparación analítica de ambas arquitecturas, de modo que la selección de esta arquitectura presentaría mayores ventajas en el caso de uso estudiado. No obstante, es importante reconocer que la arquitectura independiente presenta falencias frente a la solución comercial, particularmente en términos de ciberseguridad. Para abordar estos problemas, en trabajos futuros se deberán incrementar los mecanismos de seguridad del sistema, integrando tecnologías más modernas para la autenticación y algoritmos de firma digital para verificar la autenticidad de las actualizaciones desplegadas.

Por otra parte, la cantidad de nodos empleada para la ejecución de los experimentos en este estudio representa una posible limitación, aunque se espera que las tendencias demostradas en este trabajo se mantengan similares incluso para redes *WSN* extensas. En este sentido, en trabajos futuros se pretende incrementar la cantidad de nodos en la red *WSN* de prueba para mejorar los resultados de valoración de la escalabilidad de los sistemas implementados y extender los experimentos realizados para comprender la relación entre los protocolos de red empleados y su consumo energético asociado.

Referencias

[1] S. Sinha, "State of IoT 2024: Number of connected IoT devices growing 13 % to 18.8 billion globally," sep. 2024. [En línea]. Disponible en: https://iot-analytics.com/number-connected-iot-devices/

[2] N. Correal y N. Patwari, "Wireless Sensor Networks: Challenges and Opportunities," Motorola Labs, Plantation, FL 33322, Inf. Téc., mar. 2002. [En línea]. Disponible en: https://www.researchgate.net/profile/Neiyer-Correal-2/publication/2539977_Wireless_Sensor_Networks_Challenges_and_Opportunities/links/0deec5225f927b43ab000000/Wireless-Sensor-Networks-Challenges-and-Opportunities.pdf

[3] I. A. Ejimofor, O. P. Okechukwu, y U. J. Ebih, "APPLICATIONS AND CHALLENGES OF WIRELESS SENSOR NETWORK (WSN) IN REMOTE SENSING AND MONITORING," *International Journal of Computing, Science and New Technologies (IJCSNT)*, vol. 2, n. 1, pp. 1–10, mar. 2024. [En línea]. Disponible en: https://www.ijcsnt.com/index.php/IJCSNT/article/view/8

[4] K. Kerliu, A. Ross, G. Tao, Z. Yun, Z. Shi, S. Han, y S. Zhou, "Secure Over-The-Air Firmware Updates for Sensor Networks," en *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW)*, Monterey, CA, USA, nov. 2019, pp. 97–100. doi: 10.1109/MASSW.2019.00026

[5] K. Arakadakis, P. Charalampidis, A. Makrogiannakis, y A. Fragkiadakis, "Firmware Over-the-air Programming Techniques for IoT Networks - A Survey," *ACM Comput. Surv.*, vol. 54, n. 9, pp. 1–36, oct. 2021. doi: 10.1145/3472292

[6] S. El Jaouhari, "Toward a Secure Firmware OTA Updates for constrained IoT devices," en *2022 IEEE International Smart Cities Conference (ISC2)*, Pafos, Cyprus, sep. 2022, pp. 1–6. doi: 10.1109/ISC255366.2022.9922087

[7] N. Nikolov y D. Gotseva, "A Secure Firmware Update over the Air of ESP32 using MQTT Protocol from Cloud," en *2023 XXXII International Scientific Conference Electronics (ET)*, Sozopol, Bulgaria, sep. 2023, pp. 1–4. doi: 10.1109/ET59121.2023.10278597

[8] N.-Z. Wang y H.-Y. Chien, "Design and Implementation of MQTT-Based Over-the-Air Updating Against Curious Brokers," *IEEE Internet of Things Journal*, vol. 11, n. 6, pp. 10 768–10 777, mar. 2024. doi: 10.1109/JIOT.2023.3327447

[9] A. Bhattacharjee, H. Mahmood, S. Lu, N. Ammar, A. Ganlath, y W. Shi, "Poster: Edge-Assisted Over-the-Air Software Updates," en *2023 IEEE/ACM Symposium on Edge Computing (SEC)*, Wilmington, DE, USA, dic. 2023, pp. 285–286. doi: 10.1145/3583740.3628425

[10] A. I. Ahmed, E. I. Shahin, L. A. Said, y A. H. Madian, "A Scalable Firmware-Over-The-Air Architecture suitable for Industrial IoT Applications," en *2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, Egypt, oct. 2021, pp. 228–231. doi: 10.1109/NILES53778.2021.9600506

[11] T. Laukkarinen, L. Määttä, J. Suhonen, T. D. Hämäläinen, y M. Hännikäinen, "Design and Implementation of a Firmware Update Protocol for Resource Constrained Wireless Sensor Networks," *Int. J. Embed. Real-Time Commun. Syst.*, vol. 2, n. 3, pp. 50–68, jul. 2011. doi: 10.4018/jertcs.2011070103

[12] T. Laukkarinen, L. Määttä, J. Suhonen, y M. Hännikäinen, "A Multi-Hop Software Update Method for Resource Constrained Wireless Sensor Networks," en *Advancing Embedded Systems and Real-Time Communications with Emerging Technologies*. IGI Global Scientific Publishing, 2014, pp. 85–106. ISBN: 978-1-4666-6034-2

[13] J. Gu, S.-S. Lee, y H. Kang, "DSME-FOTA: Firmware over-the-air update framework for IEEE 802.15.4 DSME MAC to enable large-scale multi-hop industrial IoT networks," *Internet of Things*, vol. 27, p. 101239, oct. 2024. doi: 10.1016/j.iot.2024.101239

[14] M. Pule y A. M. Abu-Mahfouz, "Firmware Updates Over the Air Mechanisms for Low Power Wide Area Networks: A Review," en *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, Vanderbijlpark, South Africa, nov. 2019, pp. 1–7. doi: 10.1109/IMITEC45504.2019.9015851

[15] L. Carnevale, A. Ficara, G. Catalfamo, A. Galletta, M. Fazio, y M. Villari, "Secure and Energy Efficient Filtered Over-the-Air Internet of Things Setup in a Wireless Mesh Network for Firmware Freshness," en *2023 IEEE International Conference on Big Data (BigData)*, Sorrento, Italy, dic. 2023, pp. 3904–3913. doi: 10.1109/BigData59044.2023.10386600

[16] A. I. Ahmed, L. A. Said, y A. H. Madian, "Over-The-Air Firmware Updating Model suitable for Industrial IoT based on Microchip AVR MCU," en *2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA)*, Arad, Romania, dic. 2021, pp. 492–496. doi: 10.1109/ICCCA52192.2021.9666279

[17] A. Thantharate, C. Beard, y P. Kankariya, "CoAP and MQTT Based Models to Deliver Software and Security Updates to IoT Devices over the Air," en *2019 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)*, Atlanta, GA, USA, jul. 2019, pp. 1065–1070. doi: 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00183

[18] F. Hassan, A. Roy, y N. Saxena, "Convergence of WSN and cognitive cellular network using maximum frequency reuse," *IET Communications*, vol. 11, n. 5, pp. 664–672, mar. 2017. doi: 10.1049/iet-com.2016.0966

[19] G. Valecce, P. Petruzzi, S. Strazzella, y L. A. Grieco, "NB-IoT for Smart Agriculture: Experiments from the Field," en *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, vol. 1, Prague, Czech Republic, jun. 2020, pp. 71–75. doi: 10.1109/CoDIT49905.2020.9263860

[20] S. H. Sharf, R. A. Elhamied, M. K. Habib, y A. H. Madian, "An Efficient OTA firmware updating Architecture based on LoRa suitable for agricultural IoT Applications," en *2021 International Conference on Microelectronics (ICM)*, New Cairo City, Egypt, dic. 2021, pp. 262–265. doi: 10.1109/ICM52667.2021.9664942

[21] O. N. Samijayani, R. Darwis, S. Rahmatia, A. Mujadin, y D. Astharini, "Hybrid ZigBee and WiFi Wireless Sensor Networks for Hydroponic Monitoring," en *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, Istanbul, Turkey, jun. 2020, pp. 1–4. doi: 10.1109/ICECCE49384.2020.9179342

[22] V. Nikic, D. Bortnik, M. Lukic, D. Danilovic, y I. Mezei, "Comparisons of firmware delta updates over the air using WLAN and LPWAN technologies," en *2022 30th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, nov. 2022, pp. 1–4. doi: 10.1109/TELFOR56187.2022.9983677

[23] B. P. Neves, A. Valente, y V. D. N. Santos, "Efficient Runtime Firmware Update Mechanism for LoRaWAN Class A Devices," *Eng*, vol. 5, n. 4, pp. 2610–2632, oct. 2024. doi: 10.3390/eng5040137

[24] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, y E. D. Poorter, "Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles," *IEEE Communications Magazine*, vol. 58, n. 2, pp. 35–41, feb. 2020. doi: 10.1109/MCOM.001.1900125

[25] W. Wei, J. Banerjee, S. Islam, C. Pan, y M. Xie, "Energy-aware Incremental OTA Update for Flash-based Batteryless IoT Devices," en *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Knoxville, TN, USA, jul. 2024, pp. 51–56. doi: 10.1109/ISVLSI61997.2024.00021

[26] S. E. Jaouhari y E. Bouvet, "Toward a generic and secure bootloader for IoT device firmware OTA update," en *2022 International Conference on Information Networking (ICOIN)*, Jeju Island, Korea, ene. 2022, pp. 90–95. doi: 10.1109/ICOIN53446.2022.9687242

[27] K. G. Crowther, R. Upadrashta, y G. Ramachandra, "Securing Over-the-Air Firmware Updates (FOTA) for Industrial Internet of Things (IIOT) Devices," en *2022 IEEE International Symposium on Technologies for Homeland Security (HST)*, Boston, MA, USA, nov. 2022, pp. 1–8. doi: 10.1109/HST56032.2022.10025441

[28] H. Joshi, A. S. Anand, y J. Kokila, "Secure Firmware Update Architecture for IoT Devices using Blockchain and PUF," en *2023 International Conference on Quantum Technologies, Communications, Computing, Hardware and Embedded Systems Security (iQ-CCHESS)*, KOTTAYAM, India, sep. 2023, pp. 1–7. doi: 10.1109/iQ-CCHESS56596.2023.10391484

[29] I.-C. Lin y L. Bo, "A Pull Firmware Update Mechanism for Industrial Control Based on IOTA Streams," en *2024 19th Asia Joint Conference on Information Security (AsiaJCIS)*, Tainan, Taiwan, ago. 2024, pp. 56–61. doi: 10.1109/AsiaJCIS64263.2024.00019

[30] K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig, y E. Baccelli, "Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check," *IEEE Access*, vol. 7, pp. 71 907–71 920, may. 2019. doi: 10.1109/ACCESS.2019.2919760

[31] "AWS IoT Core." [En línea]. Disponible en: https://aws.amazon.com/es/iot-core/

[32] "IoT Hub | Microsoft Azure." [En línea]. Disponible en: https://azure.microsoft.com/es-es/products/iot-hub

[33] "Arduino Cloud | Build, Control, Monitor Your IoT Projects." [En línea]. Disponible en: https://cloud.arduino.cc/

[34] "Blynk IoT Software platform." [En línea]. Disponible en: https://blynk.io/blynk-iot-low-code-software-platform

[35] "OTAdrive." [En línea]. Disponible en: https://otadrive.com/

[36] T. L. Saaty, "Decision making with the analytic hierarchy process," *International Journal of Services Sciences*, vol. 1, n. 1, pp. 83–98, 2008. doi: 10.1504/IJSSCI.2008.017590

[37] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *European Journal of Operational Research*, vol. 48, n. 1, pp. 9–26, sep. 1990. doi: 10.1016/0377-2217(90)90057-I

[38] T. L. Saaty, "Fundamentals of the Analytic Hierarchy Process," en *The Analytic Hierarchy Process in Natural Resource and Environmental Decision Making*, K. Von Gadow, T. Pukkala, M. Tomé, D. L. Schmoldt, J. Kangas, G. A. Mendoza, y M. Pesonen, Eds. Dordrecht: Springer Netherlands, 2001, vol. 3, pp. 15–35. ISBN: 978-90-481-5735-8