Infraestructura autoconfigurable de red de sensores inalámbricos para computación en el borde

Daniel Rodríguez[1], Andrés Rodríguez[1] y Oscar Plata[1]

Resumen— **Presentamos el diseño de una infraestructura de red autoconfigurable para el despliegue rápido de aplicaciones IoT que permitan la comunicación entre nodos vecinos. Esta infraestructura nos permitirá montar con facilidad bancos de prueba (testbeds) para evaluar nuevos algoritmos de procesamiento en el borde donde los nodos sensor realicen un procesamiento local descentralizado y a la vez cooperativo, teniendo en cuenta la información compartida por sus vecinos. El hardware seleccionado es el popular módulo ESP32, un SoC de bajo consumo y bajo coste, que integra una variedad de periféricos y comunicación inalámbrica (WiFi y BLE 2.4GHz). Concretamente utilizamos módulos ESP32-C3 que implementan un procesador RISC-V de 32 bits de bajo consumo. La programación se realiza en C++ con soporte freeRTOS y las comunicaciones las realizamos sobre el protocolo ESP-NOW, un protocolo inalámbrico desarrollado por del fabricante que usa el nivel físico y de enlace WiFi, pero que no necesita de punto de acceso inalámbrico y permite la comunicación directa entre nodos con menor latencia, mayor eficiencia energética y mejor rango de alcance que si se utilizase una conexión WiFi tradicional. Nuestra implementación ofrece un nodo coordinador ya programado, que permite el autoregistro de nuevos nodos en la red, la comunicación con Internet a través del coordinador usando MQTT y la comunicación asíncrona entre nodos vecinos en la misma red de área personal (PAN). También hemos implementado una clase C++ que permite la programación de los nodos sensores mediante un API sencillo, ocultando los detalles del protocolo de comunicación desarrollado.**

Palabras clave— **Red de sensores, Infraestructura de comunicación IoT, ESP-NOW, RTOS, RISC-V**

I. Introducción

LOS avances que se están produciendo en el campo de la tecnología de la comunicación, así como en la electrónica y los paradigmas de programación cada vez más sofisticados, dan lugar a una nueva era en la que la presencia de la tecnología es prácticamente necesaria. Esta tecnología dota a los dispositivos de la capacidad de percibir y actuar en función de algún estímulo, conectarse a Internet o incluso tomar decisiones con cierto grado de autonomía. Por mencionar algunas aplicaciones como la automatización de viviendas o edificios, la automatización en el transporte o la adquisición de variables ambientales. Ante el escenario actual, donde nos enfrentamos a la creación de sistemas más interconectados y variados, surge como paradigma el Internet de las cosas ("Internet of Things", IoT)[1] para conectar dispositivos. Tradicionalmente, los sistemas IoT suelen ubicarse en un continuo computacional [2], en donde los datos detectados en el entorno (mediante los sensores incluidos) son pre-procesados localmente de forma sencilla debido a su capacidad limitada de procesamiento y almacenamiento. En consecuencia, los resultados pre-procesados son enviados a un centro de datos (típicamente en la nube) para un procesamiento más avanzado. Sin embargo, en un contexto más general, los dispositivos IoT pueden estar en movimiento y, por tanto, no pueden depender de elementos de infraestructura de comunicación fijos, sino que se conectan en red sobre la marcha, por lo que con frecuencia se unen a enjambres temporales y los abandonan. Además, la comunicación con un nodo central suele implicar un coste de comunicación muy elevado. Por ello, surge la necesidad de establecer una red de sensores inalámbricos ("Wireless Sensor network", WSNs) [3], [4], [5], [6], para permitir que los sistemas basados en IoT puedan modelarse como un enjambre de dispositivos simples, ofreciendo la posibilidad de integrarlos para alcanzar ciertos objetivos globales. De este modo, partiendo de reglas sencillas para los comportamientos individuales y las interacciones entre objetos, se puede alcanzar un óptimo global a nivel de sistema. Esta capacidad de autoorganización es necesaria para adaptar los sistemas a condiciones ambientales variables, para escalar eficientemente la plataforma y proporcionar un funcionamiento resistente para el sistema.

Las WSN, o redes de sensores y actuadores ("Wireless Sensor and Actuator network", WSAN) permiten transmitir datos de forma cooperativa a través de la red a otras ubicaciones, siendo una comunicación bidireccional que permite controlar la actividad de los sensores. Son redes pequeñas y versátiles, a menudo sin interconexión, en las que los nodos son móviles. Deben caracterizarse por la facilidad de despliegue de esta tecnología y por ser auto-configurables, pudiendo convertirse en cualquier momento en emisor o receptor y ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos relativos a los sensores locales de cada nodo. Otra de sus características es su eficiente gestión de la energía, lo que les permite obtener una gran autonomía. En la actualidad, las WSN han propiciado la miniaturización de los ordenadores, desarrollando equipos cada vez más pequeños y baratos que se comunican de forma inalámbrica y autónoma. Con el desarrollo y maduración de diferentes tecnologías de comunicación (WiFi, 5G, etc.) que han permitido una mayor diversificación de las tipologías de red existentes, el foco de atención se está desplazando

[1]Departamento de Arquitectura de Computadores, Universidad de Málaga, e-mail: {danrodcar,andres,oplata}@uma.es.

hacia las aplicaciones. Sin embargo, estas plantean algunas cuestiones complejas [7, p. 44] relacionadas con la escala prevista de los despliegues de red, el coste energético de alimentar estos nodos instalados a menudo en lugares remotos o el análisis de grandes cantidades de datos que habría que almacenar y analizar para tomar posteriormente cualquier decisión. Por ello, gracias a la naturaleza descentralizada de las WSN, nos permiten obtener medidas de diferentes puntos espaciales en el tiempo para cada lectura realizada por los nodos en el lugar donde está desplegado. Además, el coste reducido de las alternativas de hardware abierto permite desplegar un mayor número de nodos, obteniendo así una mayor calidad en la adquisición de datos.

En este trabajo presentamos una infraestructura para interconectar de forma eficiente y automática una red de sensores inalámbricos capaces de intercambiar datos para su análisis local y descentralizado. El objetivo es establecer un marco que mejore la eficiencia energética, la velocidad computacional y de transmisión y, sobre todo, dotar a este análisis local de cierto grado de inteligencia para implementar algoritmos de inteligencia colectiva (IC) (*swarm intelligence*) [8], [9], [10] que ofrece robustez y flexibilidad, sirviendo como paradigma de diseño satisfactorio para algoritmos que se enfrentan a problemas cada vez más complejos, como los sistemas basados en IoT. Para ello, se ha diseñado un sistema de auto-emparejamiento basado en el protocolo de radio ESP-NOW, desarrollado por Espressif, que permite un intercambio de datos muy rápido, eficiente y a larga distancia de hasta 500 metros. Con esta aplicación, permitimos a los nodos compartir información de estado y lecturas de sensores para que sus vecinos realicen cálculos, tomen decisiones, realicen calibraciones o detecten anomalías en sus propias lecturas. Evaluamos esta aplicación en una amplia gama de experimentos en los que observamos la velocidad de envío, el consumo energético del envío y la fiabilidad de la transmisión.

II. Estado del arte

El entorno de desarrollo IoT de Espressif ("Espressif IoT Development Framework", ESP-IDF) proporciona un gran conjunto de librerías, herramientas e interfaces de programación de aplicaciones ("Application Programming Interfaces", APIs) para el desarrollo de aplicaciones en los microcontroladores ESP32. Ofrece una amplia variedad de características que permiten el desarrollo de aplicaciones IoT, amén de proporcionar una plataforma flexible para su diseño. Una de las principales ventajas del *framework* ESP-IDF es que soporta de manera nativa C/C++ como lenguajes de programación, lo cual permite una programación más eficiente y código de alto rendimiento. Además, ESP-IDF proporciona un conjunto de características y capacidades para crear aplicaciones de IoT, como:

- Soporte para conectividad WiFi y Bluetooth
- Amplio conjunto de controladores para varios sensores, periféricos y protocolos de comunicación
- Compatibilidad con actualizaciones inalámbricas ("Over the air", OTA) y arranque seguro
- Compatibilidad con FreeRTOS, un sistema operativo en tiempo real para microcontroladores

Como alternativa al *framework* de desarrollo del fabricante (ESP-IDF), el IDE de Arduino proporciona una alternativa completa para el desarrollo de aplicaciones en microcontroladores y sistemas empotrados diseñada para simplificar el proceso de programación, siendo popular por su facilidad de uso y su interfaz simplificada. Afortunadamente el fabricante proporciona el soporte de sus dispositivos en este entorno de desarrollo simplificado. Arduino utiliza el sencillo, pero efectivo modelo de programación conocido como *super-loop*[11]. Este modelo de programación se basa en el ejecución de una primera función de inicialización ("*setup*()"), para justamente después, ejecutar en un bucle sin fin las instrucciones programadas en la función que aloja el bucle principal de control ("*loop*()"). Afortunadamente, programando desde el IDE de Arduino podemos acceder a todas las API de FreeRTOS y del resto de componentes del entorno de desarrollo del fabricante, por lo que podemos optar por programar nuestros dispositivo ESP32 con arquitectura RISC-V con el soporte de un sistema operativo en tiempo real, usando tareas, colas, semáforos, etc. Así podemos dejar de lado las limitaciones del modelo de programación Arduino, que usa un simple bucle de control, pero aprovechándonos de la facilidad de instalación y uso del IDE de Arduino.

En el campo de los protocolos de comunicación, los dispositivos de la familia Espressif permiten conectividad WiFi, BLE 2.4GHz y ESP-NOW sin necesidad de elementos añadidos que supongan más gasto energético como LoRa/LoRaWAN. El protocolo ESP-NOW es un protocolo radio que trasmite en la banda de los 2.4GHz y posee elementos del protocolo WiFi, con una reducción del número de capas del modelo OSI, lo que lo convierte en un protocolo liviano y eficiente. Es conocido por soportar distancias mayores al WiFi o Bluetooth con muy baja latencia [12] y que permite que los dispositivos envíen o reciban datos siempre que estén conectados al mismo canal de radio. Junto con el paquete de datos que se trasmite, se envía la dirección MAC que servirá de identificación inequívoca y los datos enviados o recibidos. Es por esto que se suele hablar de dos roles definidos en este protocolo según el flujo de datos, el dispositivo que envía los datos y el que los recibe, permitiendo además que el mismo dispositivo pueda trabajar con esos dos roles de manera simultánea. Además, se puede expandir la red de área personal en donde los nodos se están enviando mensajes a una red de un nivel superior local o amplia (LAN o WAN), utilizando protocolos de comunicación como MQTT ("Message Queuing Telemetry Transport") [13] para poder publicar mensajes de esos nodos en algún servidor dedicado al estudio o publicación de los datos leídos, así como la posibilidad de actuar sobre

los nodos. Esto posibilita trabajar en topologías de red variadas y evaluar los efectos de los diferentes tipos de red para las WSN y sus posibles aplicaciones utilizando ESP-NOW [14].

Es por esto que el objetivo de este trabajo es establecer marco para desplegar de forma sencilla una red de dispositivos de bajo consumo interconectados de forma inalámbrica. Los dispositivos se programan con el soporte de un sistema operativo en tiempo real (RTOS), la conexión a la red es auto-configurable y con una API sencilla de programar para los usuarios, utilizando protocolos de comunicación como MQTT para publicar datos en Internet y ESP-NOW que nos permitan que los dispositivos cercanos que pertenezcan un mismo grupo o red de área personal (PAN), intercambien datos para realizar cálculos de forma local que típicamente requerirían de comunicación con un servidor externo centralizado que realizara los cálculos agregados. La escalabilidad es una característica importante en este tipo de redes, por lo que se da la posibilidad de ampliarla a LAN o WAN utilizando un nodo central que actúe como pasarela y *router* de mensajes a un servidor externo con MQTT.

III. Metodología

El marco diseñado para intercambiar mensajes parte de una estructura en la que varios nodos pertenecientes a una misma red de área personal son capaces de comunicarse entre ellos utilizando ESP-NOW. El nodo central, además de comunicarse con el resto de nodos de la red por ESP-NOW, utiliza el mismo canal de radio para mantener una conexión WiFi con un punto de acceso inalámbrico (WAP) y conectar con un *broker* MQTT para enviar y recibir mensajes en Internet. De esta forma y a través del nodo central, cualquier nodo de la infraestructura puede publicar y recibir mensajes en Internet. El nodo central también permite la comunicación asíncrona dentro de un grupo de dispositivos, manteniendo en una cola los mensajes destinados a nodos que estén en modos de ahorro de energía o *Deep-Sleep* y por tanto no pueden recibir los mensajes hasta que despierten y soliciten una actualización al nodo central.

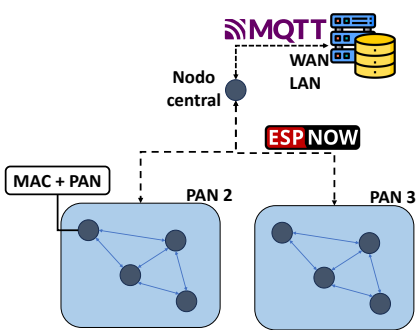

Fig. 1: Modelo de la infraestructura. Los nodos que se desean agregar deben indicar la dirección MAC y el identificador de red de área personal PAN_{ID} a la que desean pertenecer al nodo central. La comunicación con el nodo central y la red personal se realiza vía ESP-NOW mientras que el envío de datos al servidor se realiza vía MQTT.

El nodo central, según el modelo de la figura 1 es idéntico a los nodos desplegados en la red y juega el papel de *router* o pasarela. Los datos de emparejamiento se almacenarán en la pasarela para poder gestionar los mensajes que vayan tanto a otros nodos de la misma red como para el servidor. Una vez emparejado, ya se puede enviar mensajes al nodo central desde los nodos o desde el servidor así como enviar información desde el servidor a los diferentes nodos que formen la red personal o al propio nodo central.

A. Protocolo de comunicación ESP-NOW

El protocolo de ESP-NOW permite comunicarnos mediante mensajes cuya capacidad no debe de exceder los 250 bytes útiles, por lo que se debe de optimizar el envío del mensaje. Al inicio de cada mensaje enviado se añade un byte de control que posee la estructura que se ve en la figura 2 y sirve para saber el tipo de mensaje que se ha recibido.

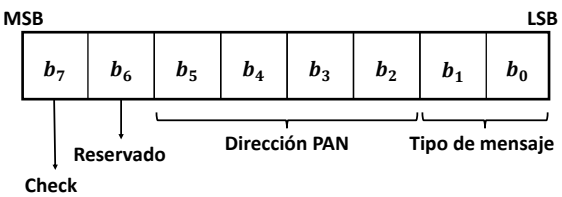

Fig. 2: Campos del byte de control

El primer bit es de comprobación y se usa para saber si hay mensajes para el dispositivo. El segundo bit queda reservado para futuras aplicaciones. Los cuatro siguientes bits corresponden a la dirección de la red personal a la que pertenece el nodo y los dos bits restantes al tipo de mensaje que se pueden recibir o enviar, que pueden ser los que se muestran en la figura 3.

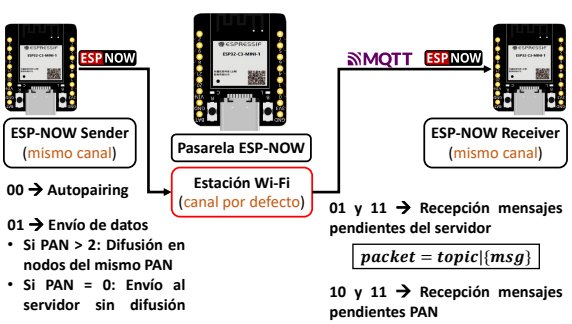

Fig. 3: Diferentes tipos de mensajes que se pueden enviar desde el nodo a la pasarela o recibir desde el servidor o área personal a un nodo concreto.

Durante el emparejamiento automático o *auto-pairing*, el dispositivo intenta conectarse a la pasarela para poder enviar y recibir datos de la misma. Tiene la ventaja de que no requiere actividad por parte del usuario ya que una vez se pierde conectividad, es el dispositivo quien intenta realizar de nuevo la conexión. Es una funcionalidad rápida, puesto que, tras un primer emparejamiento, la información

se queda almacenada en la memoria no volátil ("Non-volatile storage", NVS) y se reutiliza para futuras conexiones. Para los envíos de datos a la pasarela se establecen dos supuestos en función de la identificación que se le haya dado a la red de área personal. Además, en el envío de datos se puede aprovechar el bit "CHECK" para solicitar los mensajes que estén pendientes para el dispositivo. Se ha diferenciado el tipo de datos que se envía utilizando el identificador de área personal para el supuesto en el que solo queramos enviar datos a la pasarela o compartirlo con otros nodos. Se ha dejado reservado el caso de que el identificador de área personal valga uno para usos futuros.

Dado que se reciben mensajes que pueden venir del servidor vía MQTT o de otros nodos vía ESP-NOW, hay que gestionar el formato en el que se envían para poder procesarlos. Para el primer caso, el mensaje está compuesto por el *topic* del mismo y el contenido útil o *payload*, separados entre ellos por una barra vertical. En el caso de que recibamos un mensaje de dispositivos que pertenezcan a la misma red de área personal, el formato del mensaje que es recibido tiene el formato de la figura 4.

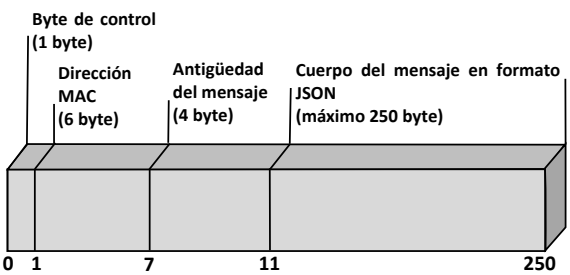

Fig. 4: Campos de un mensaje enviado de un nodo que pertenece a la misma red de área personal.

B. Emparejamiento automático ESP-NOW

El proceso de emparejamiento al nodo central se realiza de forma automática una vez se despliega el dispositivo. Este proceso está formado por una tarea que se ejecuta al inicio de la aplicación y se suspende en función del estado en el que se encuentre la conexión con la pasarela. Dicha tarea cuenta con varios estados que determinan la rutina de emparejamiento con el servidor, ilustrados en la figura 5.

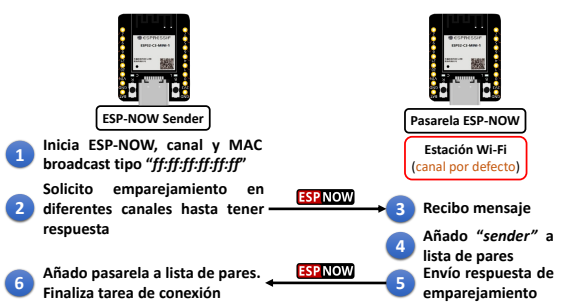

Fig. 5: Mecanismo de emparejamiento de un nodo al servidor o pasarela.

Durante la petición de conexión se debe de iniciar el protocolo ESP-NOW, así como un canal por defecto para luego dar comienzo al protocolo de emparejamiento. Mientras se está solicitando el emparejamiento, se va iterando en los diferentes canales de comunicación disponibles, generalmente del 1 al 13, hasta que la petición de conexión sea respondida por parte de la pasarela con un mensaje de confirmación en el que se incluye la MAC de la pasarela y quedando emparejado con el nodo. En este punto, suspendemos la tarea de conexión y damos lugar a la lógica de nuestro programa.

A lo largo de la conexión se está pendiente del tiempo de conexión t_{conn}, el cual si excede de un tiempo determinado por el usuario t_{out} el dispositivo entra en un modo de sueño profundo de ultra bajo consumo y permanecerá en este modo un tiempo t_{sleep} determinado también por el usuario hasta que se despierte y reintente la conexión.

IV. Resultados experimentales

Para evaluar la infraestructura diseñada con el fin de enviar y recibir datos de los nodos vecinos, primero analizaremos la aplicación desde el punto de vista del usuario, ya que como objetivo principal se pretende que fuese una interfaz fácil e intuitiva desde el punto de vista de la programación. Otro aspecto que pretendemos analizar es el tamaño del binario final, así como la eficiencia energética de nuestra aplicación ejecutándose en dispositivos de la familia de microcontroladores Espressif y con otros entornos de programación. En otras palabras, tratamos de responder a la pregunta: ¿La infraestructura diseñada sobre microcontroladores mejorará su rendimiento? Por último, analizamos algunas funcionalidades como la posibilidad de realizar compartición de datos de nodos vecinos.

A. Plataforma hardware

En nuestras pruebas hemos utilizado la placa de desarrollo *Beetle ESP32-C3* del fabricante DFROBOT, una placa con un tamaño de 20 x 20,5 milímetros y unos diez euros de coste. La placa se basa en el SoC **ESP32-C3** de *Espressif Systems* que integra un microcontrolador RISC-V de 32 bits con una frecuencia máxima de reloj de 160MHz. Posee conectividad WiFi y Bluetooth 5 (LE), 22 GPIOs configurables, 400KB de memoria RAM y varios modos de funcionamiento de bajo consumo. El SoC se alimenta a 3.3V y sus consumos van desde los 2μA en modo de sueño profundo (μC apagado, sólo el sistema de RTC en marcha), hasta los 335mA máximo cuando se transmite por radio a máxima potencia. El consumo típico con la WiFi desactivada oscila entre los 23 y 28mA.

Hemos comparado esta nueva placa, cuya novedad consiste principalmente en usar la arquitectura RISC-V, con una predecesora: la *ESP32 devkit v1*, que se basa en un SoC ESP32 del mismo fabricante (Espressif Sys.) e incorpora dos cores RISC de 32-bit Xtensa LX6, con una frecuencia máxima de reloj

```
1   #include "AUTOpairing.h"
2   AUTOpairing_t apClient
3   void process_msg_mqtt(String topic, String payload);
4   void process_msg_pan(String topic, String payload);
5   void app_main()
6   {
7     apClient.set pan(2);
8     apClient.set deepSleep(300);
9     apClient.set_mqtt_msg_cb(process_msg_mqtt);
10    apClient.set_pan_msg_cb(process_msg_pan);
11    apClient.begin();
12    // User program logic begins
13    struct_data_t data = readSensor();
14    String msg = mountMessage(data);
15  }
```

Fig. 6: Código de ejemplo para la ejecución de la infraestructura

de 240MHz. Posee conectividad WiFi y Bluetooth 4.2 (LE), 34 GPIOs configurables, 520KB de memoria RAM y también varios modos de funcionamiento de bajo consumo. El SoC se alimenta a 3.3V y sus consumos van desde los 10μA en modo de sueño profundo, hasta los 240mA máximo cuando se transmite por radio. El consumo típico con la WiFi desactivada oscila entre los 20 y 40mA.

B. Programación del dispositivo

Para facilitar la programación de los dispositivos sensores que utilizan nuestra infraestructura de comunicaciones y otras funcionalidades, hemos optado por organizar el código de todas estas funcionalidades comunes en una clase C++. La idea es ocultar al programador los detalles del uso de los protocolos de comunicación que hemos diseñado y, si en el futuro es necesario introducir mejoras o nuevas funcionalidades, no es necesario cambiar el código programado para el dispositivo más allá de sustituir la implementación de nuestra librería auto-contenida (clase) en un par de ficheros. Básicamente la aplicación tiene su código principal en el fichero de cabecera que será el que comience a ejecutarse cuando se encienda el dispositivo, como puede verse en el código de ejemplo de la figura 6. Este código necesita incluir el fichero de cabecera de nuestra clase (línea 1) y a partir de ahí, el dispositivo estará listo para ejecutarse.

Para recibir datos provenientes de una red de área personal específica o del servidor vía MQTT, el dispositivo sensor simplemente llama a una función de nuestra clase indicando el *topic* asignado al mensaje y el contenido del mismo, siendo estas funciones de retorno o *callback* las encargadas de procesar estos mensajes (líneas 3-4). Si el dispositivo se ha unido a una red de área personal, ese mensaje también llegará a los demás dispositivos vecinos de la misma red personal. Para unirse a la red el dispositivo simplemente tiene que utilizar una función de nuestra clase indicando el identificador de red de área personal como un número entero que desea utilizar. Además, el usuario también puede especificar el tiempo en segundos que el dispositivo permanezca en el régimen de sueño profundo (lineas 7-8).

Con sólo declarar un objeto de nuestra clase e invocar la función "*begin*()" (línea 11) de la misma, comienza el emparejamiento con el *router* o pasarela que controla las comunicaciones. La primera vez que se ejecuta el dispositivo, nuestra clase realiza un escaneo para encontrar la dirección y el canal al que está conectado la pasarela. A partir de ahí almacena esos datos en memoria no volátil para que las siguientes ejecuciones no necesiten realizar este proceso.

Se le puede proporcionar dos funciones propias para recibir mensajes, las configura simplemente utilizando dos llamadas a nuestra clase (líneas 9-10). A partir de ese momento las funciones proporcionadas serán invocadas respectivamente, cada vez que se reciba un mensaje desde Internet redirigido desde MQTT por el *router*/pasarela hacia el nodo al que vaya destinado el mensaje, y cada vez que se reciba un mensaje desde un vecino de la red personal a la que pertenece. De esta forma, el programador que quiera utilizar nuestra infraestructura sólo tiene que programar la lógica que le permite obtener localmente los datos de sus sensores y las funciones que procesan la información que llega de los vecinos o de Internet. Toda la gestión de estas comunicaciones bidireccionales es proporcionada de forma totalmente transparente por nuestra clase como resultado de este trabajo.

C. Análisis del rendimiento

Una vez realizada la aplicación utilizando el entorno de desarrollo proporcionado por Espressif (ESP-IDF), para comprobar si existe alguna mejora en el rendimiento general se nos ocurrió realizar una rutina similar compilando el dispositivo en Arduino, conocido por su gran comunidad y facilidad de uso para personas iniciadas en el mundo de la programación. También hemos comparado el rendimiento de la infraestructura en otro dispositivo de la misma familia de microcontroladores, el ESP32 DevKit que trae montado un procesador Tensilica XL6 que permite frecuencias de hasta 240MHz. Pretendemos analizar la infraestructura en los siguientes aspectos: tamaño de los binarios creados al depurar el código, tiempos de ejecución y eficiencia energética para las arquitecturas propuestas.

En primer lugar se analiza el tamaño de los ficheros binarios, porque el tamaño es fundamental para conseguir una aplicación ligera y optimizar en aspectos de almacenamiento. Esto está directamente relacionado con el coste de la memoria, que en algunos casos no puede ser superior a unos pocos *megabytes* (incluso *kilobytes* o unos pocos bytes, dependiendo

Tabla I: Comparación de tiempos de ejecución y consumos de energía

MCU	SDK	Tiempo emparejamiento (ms)	Tiempo Ejecución (ms)	Corriente (mA)	Potencia (mW)	Energía (mWh)
ESP32-C3 (RISC-V@160)	Arduino	36	47	91,7	385,14	0,0024
ESP32 DevKit (Tensilica XL6@240)	Arduino	32	44	116,59	1049,31	0,0078
ESP32-C3 (RISC-V@160)	ESP-IDF	22,52	35,81	80,1	336,42	0,0017
ESP32 DevKit (Tensilica XL6@240)	ESP-IDF	13,5	36,84	110,2	991,8	0,0053

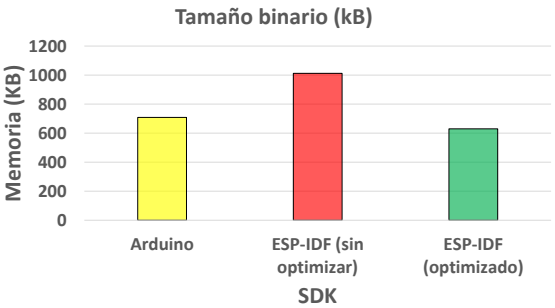

Fig. 7: Comparativa del tamaño de binarios compilados

de la aplicación). En este sentido, se podría tratar de encontrar una aplicación que genere un archivo que ocupe poco espacio, permitiendo al dispositivo utilizar el resto de la memoria para otras funcionalidades. Además, el ESP32-C3 dispone de unos 4MB, por lo que limitamos las dimensiones del fichero a este espacio. Se ha comprobado que Arduino trae por defecto una configuración de entorno que, juntos con sus ficheros están optimizados sin necesidad de modificar nada. Como ESP-IDF permite personalizar las características del programa generado, se ha comprobado que con una configuración mejorada a la que trae por defecto obtenemos un binario más pequeño que el producido por Arduino, como se muestra en el gráfico 7, lo que nos reduce un 12,5 % el tamaño del binario compilado y obteniendo un menor consumo de memoria y una mayor velocidad de acceso.

Se ha estudiado el consumo energético donde comparamos el rendimiento utilizando el modelo *superloop* de Arduino y el paradigma de programación en tiempo real utilizando FreeRTOS en dos placas de la misma familia. Las pruebas se han realizado en los diferentes dispositivos y sin sensores conectados al ADC. Para obtener el consumo, se han probado los ESP32 y ESP32-C3 en vacío conectándolos a un sensor de corriente de alta precisión INA219 y a una fuente de alimentación de 9V para el ESP32 y por USB (5V) en el caso del ESP32-C3, ya que por USB el ESP32 no funcionaba junto con el INA219 (es recomendable, de hecho, alimentarlo con una tensión entre 6 o 7 voltios para evitar pérdidas de potencia en el regulador de voltaje que viene instalado). Los resultados se ilustran en la tabla I. Por un lado, el núcleo del ESP32 es un doble núcleo Tensilica Xtensa LX6 de 32 bits que funciona entre 160 o 240MHz mientras que el núcleo de ESP32-C3 es un procesador con arquitectura RISC-V mononúcleo de 32 bits con una frecuencia máxima de 160 MHz. Este procesador se utiliza como procesador *Ulta-Low-Power* (ULP), lo que significa que puede utilizarse aunque esté completamente apagado, lo que permite al chip absorber energía y seguir trabajando con un coste energético muy bajo. Esto implica que el procesador no tiene una gran velocidad de procesamiento, ya que, al ser de un solo núcleo, no puede someterse a una carga de trabajo excesiva. En la tabla I podemos ver como el consumo entre un dispositivo y otro es notablemente diferente. El ESP32 funcionando a una frecuencia superior muestra un coste energético elevado comparado con el ESP32-C3, sin perder este dispositivo velocidad de ejecución y siendo funcional incluso con un coste energético mínimo.

En la tabla I mostramos el tiempo total de ejecución de un ciclo del programa, desde el arranque del dispositivo hasta el envío de datos y la entrada en modo de ahorro de energía (sueño profundo). No hay una variación notable en cuanto a tiempos de ejecución. Así que podemos concluir que el dispositivo ESP32-C3 con micro-arquitectura RISC-V aun contando con un sólo núcleo, frente a los dos del ESP32 y con menor frecuencia de reloj, consigue resultados de rendimiento comparables en este tipo de aplicaciones. El tiempo de emparejamiento mostrado también en la tabla, es un tiempo añadido que consume el dispositivo la primera vez que se conecta, ya que debe escanear los canales de radio disponibles hasta localizar nuestra plataforma. Esta configuración se almacena en memoria FLASH (no volátil) y en subsiguientes ciclos de ejecución se recuperará de esta memoria consumiendo un tiempo prácticamente despreciable.

D. Proceso de comunicación entre nodos

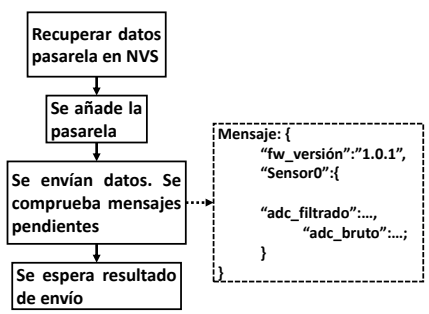

Fig. 8: Rutina de recuperación de datos de la NVS, envío de mensaje y solicitud de mensajes entrantes. Una vez se añade la pasarela, se debe de liberar el recurso que bloquea el proceso de emparejamiento para proceder al envío de mensajes con formato JSON.

En la figura 8 se ilustra el proceso de conexión y envío de un mensaje determinado. Con el mensaje, se envía el byte de control con el bit de "CHECK" activo, indicándole a la pasarela que nos devuelva posibles mensajes pendientes para el nodo que lo solicite. Al enviar nuestro mensaje, se espera un acuse de recibo tras el cual el dispositivo entra en el modo de sueño profundo. Para un mensaje recibido de un nodo perteneciente a una área específica se procesaría de forma similar a si llega desde el servidor vía MQTT, como se observa en la figura 9. Al enviar la trama de comprobación de mensajes, indicamos que también queremos saber si hay mensajes que nos pertenecen. Estos mensajes serán personales y tendrán la antigüedad del mensaje (desde que fue recibido en la pasarela por el otro nodo), la MAC de origen y el contenido explícito del mensaje en cuestión.

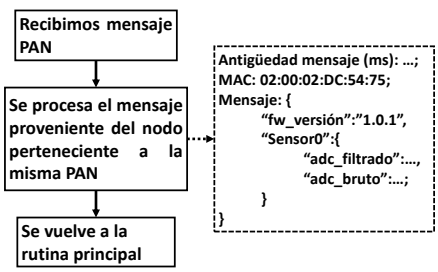

Fig. 9: Rutina de recepción de datos proveniente de otro dispositivo perteneciente a la misma red de área personal. Al igual que con la rutina de envío de datos, se debe de liberar el recurso que bloquea el proceso de recepción de datos.

En definitiva, con esta información compartida en los nodos, podremos obtener nuevas medidas, realizar cálculos, ajustes, calibraciones, algoritmos de control o cualquier otro proceso para optimizar los datos ya obtenidos y almacenados localmente. Estos dispositivos tienen la ventaja de que tienen una potencia de procesamiento elevada con un coste energético mínimo pero con una memoria reducida. Implementar un modelo para una red neuronal [15] podría suponer un coste elevado y hacerlo en una memoria externa limitaría la velocidad de acceso a los datos.

Un primer ejemplo de uso que se está estudiando actualmente es la aplicación de algoritmos de IC [16], puesto que el paralelismo potencial y las características distribuidas de los algoritmos IC permiten la posibilidad de resolver problemas no lineales complejos con capacidades avanzadas en términos de auto-adaptibilidad, robustez y capacidad de búsqueda. Finalmente, como objetivo principal se explorarán soluciones que se alejen del paradigma centralizado clásico descrito anteriormente [17]. Esto es, resolver las aplicaciones de una manera completamente descentralizada (sin servidor externo en la nube) y utilizando métodos de inteligencia colectiva no supervisada. Las comunicaciones entre los dispositivos serán ad-hoc, dinámicas y locales, minimizando el consumo energético. Un caso de uso en el que aplicaremos las soluciones que desarrollemos será el análisis automático de patrones en las señales detectadas por los sensores (por ejemplo, para la detección de eventos específicos).

V. Conclusiones y trabajos futuros

A. Conclusiones

Hemos conseguido cumplir con los objetivos planteados al comienzo de este trabajo. El objetivo principal era presentar una infraestructura de red autoconfigurable para el despliegue rápido de aplicaciones IoT que permitiera la comunicación entre nodos vecinos. Esta infraestructura permitirá implementar algoritmos de inteligencia colectiva en dispositivos conectados IoT de muy bajo consumo para la implementación de redes de sensores auto-configurables y capaces de compartir información en el borde. Escogimos el nuevo módulo ESP32-C3 con microarquitectura RISC-V para evaluar sus características comparándolas con versiones anteriores de esta familia de dispositivos basadas en otras microarquitecturas (RISC, Tensilica Xtensa). También hemos comparado dos modelos de programación bien diferenciados, el *super-loop* clásico de Arduino y la programación de tareas en tiempo real que ofrece un sistema operativo de tiempo real como el que proporciona ESP-IDF, *framework* de programación basado en FreeRTOS. Esta infraestructura que hemos desarrollado, servirá para implementar fácilmente aplicaciones que permitan explotar la capacidad de cálculo de los dispositivos IoT desplegados en el borde para procesar e integrar la información en el mismo lugar donde se produce: la red de sensores. El diseño de la infraestructura utilizando clases y programada en C++ permite modularidad, versatilidad, robustez y eficiencia. Era importante que la aplicación tuviera una interfaz sencilla de programar y desplegar, más allá de la complejidad intrínseca de la lógica de la propia aplicación, y se ha logrado, como también se ha logrado el envío y recepción de mensajes de forma fiable.

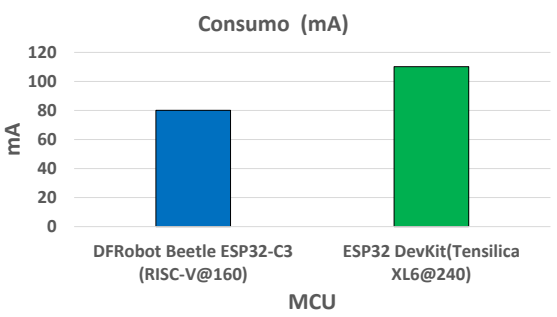

Fig. 10: Diferencia de consumo eléctrico entre ESP32-C3 (RISC-V) y ESP32 (Tensilica Xtensa LX6)

En cuanto al consumo, podemos apreciar en la figura 10 que el consumo de corriente eléctrica de la placa ESP32-C3 con RISC-V es inferior a la placa ESP32. Es lo esperable puesto que esta última posee dos *cores* en vez de uno y mayor frecuencia de CPU. Con respecto a los tiempos de ejecución no hay una diferencia significativa si comparamos los

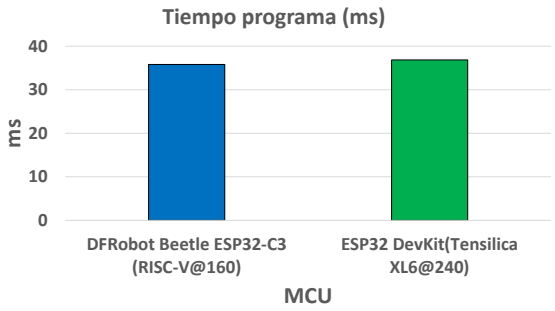

Fig. 11: Diferencia de tiempos de programa entre ESP32-C3 (RISC-V) y ESP32 (Tensilica Xtensa LX6)

dos SDK estudiados puesto que están haciendo operaciones similares. Si comparamos las dos placas con micro-arquitecturas diferentes en sus microcontroladores (ver la figura 11) sí se aprecia alguna diferencia aunque no es significativa (2.8 %). A pesar de que la CPU del ESP32 está funcionando a 240MHz frente a los 160MHz del ESP32-C3 (es decir, 80MHz más rápido), no se ve una diferencia notable a favor del dispositivo con mayor frecuencia de reloj como *a priori* cabría de esperar. Seguramente en aplicaciones que requieran de una mayor carga computacional el dispositivo con mejores prestaciones sería más adecuado, pero creemos que habría que seguir apostando por la arquitectura RISC-V. De hecho, el mismo fabricante ha anunciado el lanzamiento de nuevos desarrollos como el módulo ESP32-P4, orientado a aplicaciones de alto rendimiento que incorpora en el SoC dos *cores* RISC-V de hasta 400MHz y con FPU de simple precisión, más un *cores* RISC-V de muy bajo consumo a 40MHz. En resumen, el módulo que hemos probado que incorpora la arquitectura RISC-V puede realizar las mismas tareas, en tiempos similares y con un menor consumo que sus antecesores, lo que parece dar la razón al fabricante que últimamente está incorporando la arquitectura RISC-V [18], [19] en su catalogo de módulos IoT inalámbricos de bajo consumo.

Por último, mencionar que estas pequeñas placas de desarrollo basadas en RISC-V serían una buena plataforma para la docencia de temas relacionados con la programación de micro-arquitecturas RISC-V, permitiendo el uso de elementos como la entrada-salida a través de GPIOs, interrupciones y temporizadores, etc. [20]. También podrían plantearse prácticas de programación de microcontroladores en tiempo real usando FreeRTOS (tareas, planificación de prioridades, colas de mensajes, semáforos, mutex, eventos de sincronización, ...). Todo ello se puede realizar desde el entorno de programación de Arduino que facilita mucho la instalación de la plataforma de desarrollo y el uso y programación de propia placa a través de USB. Arduino nos permitirá acceder a todo el API de FreeRTOS desde nuestro programa C/C++ y si es necesario trabajar con ensamblador RISC-V podremos incluir código ensamblador de forma sencilla, que puede ser enlazado con el mismo programa C/C++ que ejecutaremos en la placa.

B. *Trabajos futuros*

El protocolo ESP-NOW limita en número de dispositivos con el que se puede comunicar cada nodo. Se necesita emparejar previamente con el nodo con el que se quiere establecer comunicación y el sistema limita a veinte pares la capacidad de comunicación de cada nodo. Para sobrepasar esta limitación, actualmente estamos adaptando nuestro código para utilizar siempre mensajes de *broadcast*, que contempla el protocolo, en vez los actuales mensajes *peer to peer* que requieren de emparejamiento previo. Añadiremos la dirección destino de la comunicación en el propio mensaje y será en la capa de aplicación donde filtraremos los mensajes que están dirigidos a cada nodo. También pensamos añadir un mensaje de *acknowledgment* del que carece la comunicación *broadcast* para confirmar la recepción de los mensajes por parte del destinatario concreto de la difusión.

También estamos trabajando en superar la limitación de tamaño máximo de mensaje en el protocolo ESP-NOW. En este protocolo los mensajes ocupan una trama y sólo hay espacio para transmitir 250 bytes. En nuestra propuesta un mensaje de mayor tamaño de trocea en paquetes de 250 bytes, que nuestra plataforma gestiona para enviarlos individualmente y reconstruir el mensaje completo en el destino de la comunicación.

Por otro lado, en la implementación que tenemos hasta ahora no hemos usado cifrado (porque el cifrado limita más aún el número de pares diferentes con el que se puede establecer la comunicación ESP-NOW). Cuando pasemos a basar toda la comunicación en mensajes de tipo *broadcast* podremos activar el cifrado de la comunicación que ya soporta ESP-NOW y está basado en el método de cifrado CCM-P/AES incluido en el estándar WiFi IEEE 802.11-2012.

Agradecimientos

El trabajo descrito se ha desarrollado con financiación de los proyectos MICIN PID2019-105396RB-I00 y PID2022-136575OB-I00.

Referencias

[1] Giancarlo Fortino and Paolo Trunfio, *Internet of Things Based on Smart Objects, Technology, Middleware and Applications*, Springer, 01 2014.

[2] Ana Maria Juan Ferrer, Soeren Becker, Florian Schmidt, Lauritz Thamsen, and Odej Kao, "Towards a cognitive compute continuum: An architecture for ad-hoc self-managed swarms," *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 634–641, 2021.

[3] Diego Tibaduiza Burgos, Nayibe Chio, Laura Aparicio, and Luis Caro, "Redes de sensores inalámbricos," in *Congreso Internacional de Ingeniería Mecatrónica-UNAB*, 2011.

[4] José Caicedo-Ortiz, Melisa Acosta-Coll, and Alejandro Cama-Pinto, "Modelo de despliegue de una wsn para la medición de las variables climáticas que causan fuertes precipitaciones," *Prospectiva*, vol. 13, pp. 106, 06 2015.

[5] Diana Castro, Luis Chamorro, and Carlos Viteri-Mera, "Una red de sensores inalámbricos para la automatización y control del riego localizado," *Revista de Ciencias Agrícolas*, vol. 33, pp. 106–116, 12 2016.

[6] Sana Sultan, Tehmina Khan, and Sairah Khatoon, "Implementation of hvac system through wireless sensor net-

work," *Communication Software and Networks, International Conference on*, vol. 0, pp. 52–56, 01 2010.

[7] José Carlos Reyes Guerrero and Miguel Villagrán Marín, *Localización en interiores usando Redes Inalámbricas de Sensores: Un sistema para la monitorización remota de personas mayores y discapacitados*, Editorial académica Española, 03 2012.

[8] Ouarda Zedadra, Antonio Guerrieri, Nicolas Jouandeau, Giandomenico Spezzano, Hamid Seridi, and Giancarlo Fortino, "Swarm intelligence-based algorithms within iot-based systems: A review," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 173–187, 2018.

[9] Weifeng Sun, Min Tang, Lijun Zhang, Zhiqiang Huo, and Lei Shu, "A survey of using swarm intelligence algorithms in iot," *Sensors*, vol. 20, pp. 1420, 03 2020.

[10] Malvinder Singh Bali and Kamali Gupta, "Use of swarm intelligence algorithms in internet of things-based systems: A comprehensive review," in *2023 International Conference on Advancement in Computation and Computer Technologies (InCACCT)*, 2023, pp. 767–772.

[11] Corrado De Sio, Sarah Azimi, and L. Sterpone, "Evaluating reliability against see of embedded systems: A comparison of rtos and bare-metal approaches," *Microelectronics Reliability*, vol. 150, pp. 115124, 10 2023.

[12] Dania Eridani, Adian F. Rochim, and Faiz N. Cesara, "Comparative performance study of esp-now, wi-fi, bluetooth protocols based on range, transmission speed, latency, energy usage and barrier resistance," in *2021 Int. Seminar on Application for Tech. of Information and Communication (iSemantic)*, 2021, pp. 322–328.

[13] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark, "Mqtt-s — a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, 2008, pp. 791–798.

[14] Ghalwash A Labib MI, ElGazzar M and AbdulKader, "An efficient networking solution for extending and controlling wireless sensor networks using low-energy technologies," *PeerJ Computer Science*, pp. 17–39, 11 2021.

[15] Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi, and Sergio Saponara, "A lightweight posit processing unit for risc-v processors in deep neural network applications," *IEEE Transactions on Emerging Topics in Computing*, 2021.

[16] James Kennedy, *Swarm Intelligence*, Morgan Kaufmann, USA, 2001.

[17] HH Bosman, G Iacca, A Tejada, JH Wörtche, and Antonio Liotta, "Spatial anomaly detection in sensor networks using neighborhood information," *Information Fusion*, vol. 33, pp. 41 – 56, 2017.

[18] Enfang Cui, Tianzheng Li, and Qian Wei, "Risc-v instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24696–24711, 2023.

[19] *A comparative survey of open-source application-class RISC-V processor implementations*. ACM, 2021.

[20] Espressif Systems, "ESP32-C3 Technical Reference Manual," `https://www.espressif.com/sites/default/files/documentation/esp32-c3_technical_reference_manual_en.pdf`, 2024, [Online; accessed 10/03/2024].