Segundo Informe de Avance 2025 - Investigador en Formación

En el marco del:

PROYECTO PINV01-24

Pruebas Básicas de Interoperabilidad Entre Dispositivos Embebidos

Junio, 2025

Fabian Palacios-Pereira, MSc. Ing. [1]

[1] fpalacios@fiuna.edu.py Laboratorio de Sistemas Distribuidos (LSD), Dpto. de Ing. Electrónica y Mecatrónica (DIEM), Centro de Innovación Tecnológica (CITEC), Facultad de Ingeniería de la UNA (FIUNA), Isla Bogado, Luque, Paraguay.

Resumen

Este informe presenta el desarrollo y las pruebas de un sistema básico de interoperabilidad en red local, utilizando dispositivos ESP32 y una Raspberry Pi como servidor central. Los nodos se anuncian automáticamente mediante mensajes JSON, permitiendo su detección, registro y monitoreo en tiempo real. Se documentan las pruebas de conexión, desconexión y verificación de estado, así como las métricas de rendimiento obtenidas.

Palabras clave: dispositivos embebidos; conectividad; redes locales; Raspberry, ESP32.

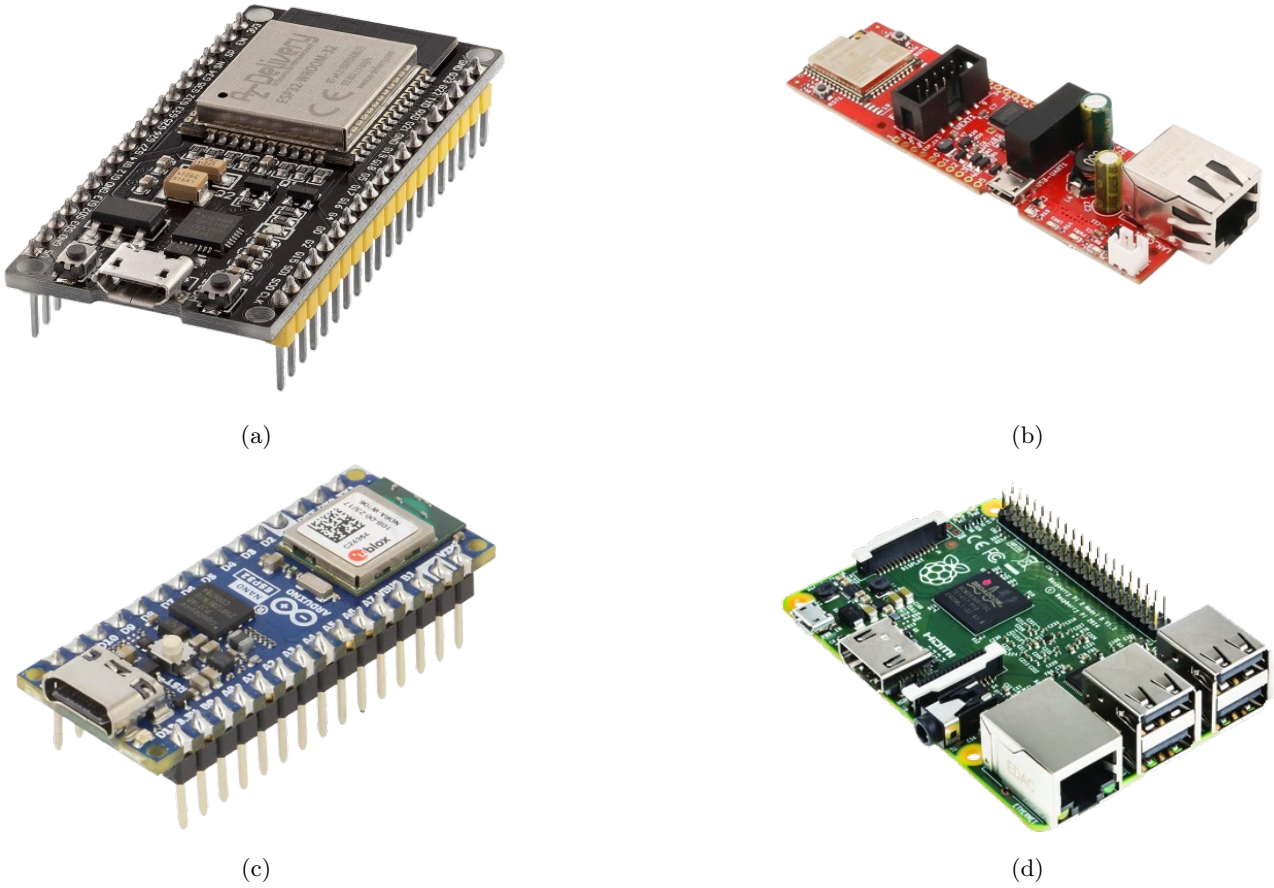
1. Introducción

En el desarrollo de ciudades inteligentes, los sistemas de transporte inteligentes (ITS, por sus siglas en inglés) desempeñan un papel fundamental al integrar tecnologías digitales con infraestructura urbana. Una característica clave para el funcionamiento eficiente de estos sistemas es la interoperabilidad entre dispositivos, es decir, la capacidad de que diferentes equipos —de distintos modelos, fabricantes o tecnologías— se comuniquen e interactúen de manera transparente dentro de una misma red [1][2][3]. Esta interoperabilidad se vuelve aún más robusta cuando se complementa con capacidades de tipo plug and play, permitiendo que un dispositivo pueda incorporarse automáticamente al sistema sin necesidad de configuraciones manuales complejas [4][5].

Este informe presenta una implementación básica de un sistema de interoperabilidad local utilizando dispositivos ESP32 y una Raspberry Pi como nodo central de gestión. En este sistema, cada ESP32 se conecta a la red local y se anuncia automáticamente mediante una solicitud JSON enviada a un servidor Flask alojado en la Raspberry Pi. A partir de ese momento, el servidor registra al dispositivo, mantiene su estado actualizado (activo o inactivo) y almacena su información en una cadena JSON estructurada. Este mecanismo permite que el sistema detecte nuevos dispositivos, gestione su presencia y monitoree su conectividad de manera automática.

El objetivo de este informe es documentar las pruebas realizadas para validar el correcto funcionamiento de esta arquitectura, con foco en la interoperabilidad entre dispositivos y el rendimiento del sistema de comunicación y gestión. Se realizarán pruebas de conexión, detección, desconexión y registro dinámico, así como la medición de indicadores clave como latencia de respuesta y consumo de recursos en la Raspberry Pi.

Cabe destacar que, si bien el sistema está pensado con una estructura que permite extenderlo hacia un funcionamiento plug and play completo, en esta etapa no se implementa asignación automática de tareas ni actualización remota de firmware en los dispositivos. Estas funcionalidades se consideran para futuras fases del proyecto, donde se utilizarán nodos más potentes para permitir una configuración y control más avanzados o también la descentralizacion del sistema a fin de hacerlo mas robusto. En esta versión, el enfoque está centrado exclusivamente en la comunicación, detección y registro de dispositivos en un entorno de red local controlado.



(a) (b)

(c) (d)

Figura 1: (a) ESP32-WROOM32. (b) ESP32 Olimex PoE. (c) Nano ESP32. (d) Raspberry Pi 2B.

2. Descripción de Hardware y Arquitectura

2.1 Dispositivos Utilizados

Para implementar la arquitectura propuesta se utilizan diferentes microcontroladores y plataformas de propósito general que reflejan la diversidad de dispositivos que podrían coexistir en un sistema distribuido dentro de un entorno urbano. La selección incluye modelos con conectividad Wi-Fi y Ethernet, baja demanda energética y compatibilidad con entornos de desarrollo ampliamente utilizados.

2.1.1 ESP32-WROOM32

Este microcontrolador, ESP32-WROOM32 (Fig. 1a) es ampliamente utilizado en sistemas IoT debido a su balance entre potencia de procesamiento, consumo y facilidad de desarrollo. Se utilizó para simular nodos inalámbricos capaces de conectarse automáticamente al servidor, enviar su información y mantener una comunicación estable a través de Wi-Fi.

- Fabricante: Espressif Systems
- Procesador: Dual-core Tensilica Xtensa LX6, hasta 240 MHz
- Memoria: 520 KB SRAM interna
- Conectividad: Wi-Fi 802.11 b/g/n, Bluetooth v4.2 BR/EDR y BLE
- Entradas/salidas: hasta 34 GPIO, ADC, DAC, PWM, UART, SPI, I^2C
- Alimentación: 3.0V - 3.6V

2.1.2 ESP32 Olimex-PoE

A diferencia del modelo Wroom32, el ESP32 PoE (Fig. 1b) incluye interfaz Ethernet nativa, lo que permite evaluaciones de interoperabilidad en redes cableadas. Es útil para entornos donde la comunicación inalámbrica puede presentar interferencias o donde se requiere mayor estabilidad en la conexión. Además, al contar con soporte para Power over Ethernet (PoE), facilita instalaciones simplificadas en campo.

- Fabricante: Olimex
- Procesador: Dual-core Tensilica LX6, 240 MHz
- Memoria: 4 MB Flash, 520 KB SRAM
- Conectividad: Ethernet 10/100 Mbps (PHY LAN8720), PoE opcional
- Entradas/salidas: 30+ GPIO, ADC, UART, SPI, I^2C
- Alimentación: 5V vía conector o PoE

2.1.3 Nano ESP32 (Arduino-compatible)

El Nano ESP32 (Fig. 1c) es compatible con el entorno de desarrollo Arduino IDE, este modelo fue utilizado para probar la facilidad de integración en un flujo de trabajo accesible. Aunque posee características similares al Wroom32, su perfil fue considerado como nodo genérico de prueba, representando posibles sensores o actuadores conectados sin demasiada lógica local.

- Fabricante: Varios (basado en referencia de Espressif)
- Procesador: Dual-core Xtensa LX6
- Memoria: 4 MB Flash (común), 520 KB SRAM
- Conectividad: Wi-Fi 802.11 b/g/n, Bluetooth
- Alimentación: 5V vía microUSB

2.1.4 Raspberry Pi 2B V1.1

En este proyecto, la Raspberry Pi 2B (Fig. 1d) cumple el rol de servidor local, ejecutando un servidor Flask encargado de recibir y procesar solicitudes JSON de los dispositivos ESP32. Su capacidad de procesamiento y sistema operativo Linux permiten desarrollar y mantener fácilmente scripts en Python para la gestión de red, registro de dispositivos y monitoreo del sistema. Aunque no posee Wi-Fi integrado, la conexión Ethernet le otorga estabilidad y bajo retardo en la comunicación con los demás nodos.

- Fabricante: Raspberry Pi Foundation
- Procesador: Broadcom BCM2836, Quad-core ARM Cortex-A7 @ 900 MHz
- Memoria RAM: 1 GB LPDDR2
- Conectividad: Ethernet 10/100 Mbps, 4 × USB 2.0, GPIO de propósito general
- Almacenamiento: MicroSD (utilizado con Raspberry Pi OS Lite)
- Alimentación: 5V/2A vía microUSB
- Listado y descripción de los dispositivos usados: ESP32 Wroom32, ESP32 Olimex con Ethernet, ESP32 Arduino, Raspberry Pi (modelo).
- Arquitectura de la red local: Raspberry Pi como servidor gestor, ESP32 como nodos clientes.
- Descripción de la comunicación: protocolo HTTP/REST con mensajes JSON para anuncio y estado.

3. Estructura del Sistema

El sistema desarrollado implementa un esquema de red local orientado a la interoperabilidad entre dispositivos embebidos con capacidades de conexión tanto por Wi-Fi como por Ethernet. La arquitectura está centrada en una Raspberry Pi 2B que cumple el rol de gestor de red y servidor local, encargándose de recibir, registrar y monitorear automáticamente a cada uno de los dispositivos ESP32 conectados.

En la Fig. 2, se presenta la arquitectura completa del sistema: la Raspberry Pi está conectada a un switch Ethernet, al cual también se conecta un Access Point Wi-Fi que proporciona conectividad a los nodos inalámbricos. Los dispositivos cableados (como el ESP32 Olimex con puerto Ethernet) también están conectados directamente al switch. De este modo, todos los nodos pueden operar simultáneamente en un entorno controlado de red local, sin necesidad de acceso a Internet.

3.1 Firmware base para ESP32: anuncio con JSON

Todos los dispositivos ESP32 utilizan un firmware base unificado [6], cuyo propósito principal es permitir que, al iniciar, el dispositivo se anuncie automáticamente al servidor local ejecutado en la Raspberry Pi. Este anuncio se realiza mediante una solicitud HTTP POST al servidor Flask, con un mensaje JSON que contiene información clave sobre el dispositivo. Este mensaje permite que la Raspberry Pi identifique de forma única a cada dispositivo, sin necesidad de configuración manual. Esta capacidad es clave en la construcción de un sistema plug and play básico, donde los dispositivos se integran automáticamente a la red.

3.2 Formato del JSON enviado por los dispositivos

El JSON que envía cada ESP32 contiene los siguientes campos:

```
{
  "id": "esp32-wifi-01",
  "ip": "192.168.1.101",
  "hardware": "ESP32-WROOM32"
}
```

Donde:

- "id": identificador único del dispositivo, definido en el firmware.
- "ip": dirección IP asignada en la red local (configuración estática).
- "hardware": tipo de hardware utilizado o variante del ESP32.

Este mensaje permite al servidor registrar y organizar a los dispositivos conectados.

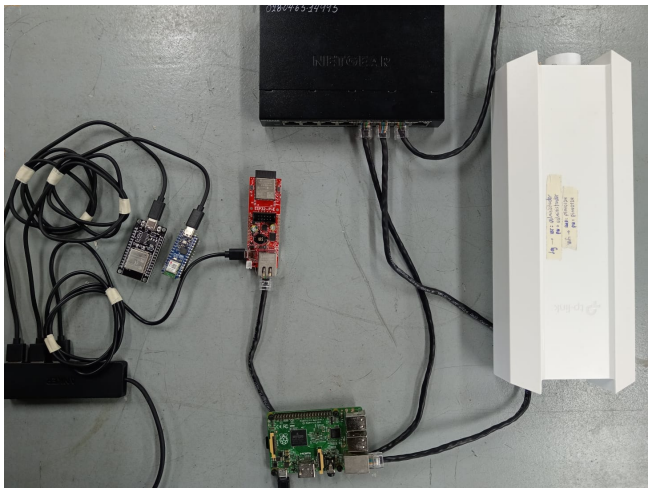


Figura 2: Sistema implementado.

3.3 Registro de dispositivos en la Raspberry Pi

La Raspberry Pi, actuando como servidor, mantiene una estructura en formato JSON que representa el estado actual de todos los dispositivos detectados en la red. Este registro es actualizado en tiempo real cada vez que un dispositivo se anuncia o cuando el sistema detecta cambios en su disponibilidad. Este registro en la Raspberry Pi agrega:

- "state": estado actual del dispositivo (active/inactive).
- "last-seen": registro en tiempo de última conexión/desconexión.
- "registered-at": momento en el que el dispositivo fue registrado por primera vez.

3.4 Gestión del estado y desconexión

Además del anuncio inicial, la Raspberry Pi ejecuta un mecanismo periódico de verificación del estado de los dispositivos. Esto se realiza mediante comandos de ping programados en intervalos regulares. Si un dispositivo no responde después de varios intentos consecutivos, el servidor actualiza su estado a ïnactive"dentro del registro JSON.

De esta manera, el sistema es capaz de:

- Detectar cuándo un dispositivo se desconecta o deja de responder.
- Mantener un registro histórico del estado y tiempo de actividad de cada nodo.
- Simular una gestión de red más compleja, pero basada en herramientas ligeras y fácilmente adaptables.

Esta estructura, simple pero robusta, permite validar los principios de interoperabilidad en redes embebidas, simulando un sistema de detección y monitoreo que puede escalarse y enriquecerse en etapas posteriores del proyecto.

4. Pruebas Realizadas y Resultados

A fin de validar el funcionamiento del sistema de interoperabilidad desarrollado, se llevaron a cabo pruebas en un entorno de red local sin acceso a Internet, simulando un escenario realista de despliegue en campo. El objetivo fue comprobar el comportamiento de los dispositivos al conectarse, anunciarse, ser registrados por el servidor, mantenerse activos y gestionar adecuadamente su estado en caso de desconexión.

Durante estas pruebas, se recolectaron tanto datos técnicos como evidencias gráficas del funcionamiento del sistema, incluyendo imágenes del montaje físico con los dispositivos reales, una captura del archivo JSON generado por la Raspberry Pi y una vista del log del servidor Flask que muestra en tiempo real los eventos de conexión, registro y verificación periódica de los nodos.

4.1 Pasos para conectar dispositivos y validar detección

Cada ESP32 fue conectado individualmente (por Wi-Fi o Ethernet según el caso), siguiendo este procedimiento:

- Conexión física para el caso de los dispositivos cableados.
- Encendido del dispositivo con firmware base preinstalado.
- Conexión automática a la red local y obtención de una IP válida.
- Envío automático del mensaje JSON de anuncio al servidor Flask.
- Verificación del registro del dispositivo en la estructura mantenida por la Raspberry Pi.
- Confirmación visual desde la terminal y mediante la inspección del archivo de registro.

```
{
  "esp32-olimex-01": {
    "id": "esp32-olimex-01",
    "ip": "192.168.1.102",
    "hardware": "ESP32-OLIMEX-PoE",
    "state": "active",
    "last-seen": "2025-06-24T11:17:49.020226",
    "registered-at": "2025-06-24T11:17:49.020355"
  },
  "esp32-wifi-01": {
    "id": "esp32-wifi-01",
    "ip": "192.168.1.101",
    "hardware": "ESP32-WROOM32",
    "state": "active",
    "last-seen": "2025-06-24T11:17:24.336668",
    "registered-at": "2025-06-24T11:17:24.336803"
  },
  "esp32-nano-01": {
    "id": "esp32-nano-01",
    "ip": "192.168.1.103",
    "hardware": "NANO-ESP32",
    "state": "inactive",
    "last-seen": "2025-06-24T11:22:14.730696",
    "registered-at": "2025-06-24T11:21:07.378457"
  }
}
```


Figura 3: Cadena de JSON generada por la Raspberry.

4.2 Pruebas de anuncio y registro en cadena JSON

En todos los casos, el servidor recibió correctamente los mensajes enviados por los nodos y generó entradas completas en su archivo de registro. Los datos fueron organizados con los campos id, ip, hardware, state, last-seen y registered-at, tal como se observa en la Fig. 3, que muestra una sección del JSON generado durante las pruebas reales.

Este mecanismo demostró ser estable y repetible, permitiendo incorporar automáticamente nuevos dispositivos sin intervención manual, validando así una funcionalidad básica de tipo plug and play e interoperabilidad.

4.3 Pruebas de desconexión y actualización

Se realizaron desconexiones manuales (apagando nodos) y desconexiones simuladas (retirándolos del alcance del AP Wi-Fi). El servidor, mediante su sistema de verificación periódica con ping, detectó estas ausencias y actualizó correctamente el campo state a “inactive” en el archivo JSON, en la Fig. 3 es posibole visualizar el estado “active” de los primeros dos dispositivos, mientras que el dispositivo con ID: esp32-nano-01, se encuentra “inactive”.

La actualización del estado también se refleja en la salida de la terminal, como se aprecia en la Fig. 4, donde se muestra el proceso continuo de verificación y registro de dispositivos.

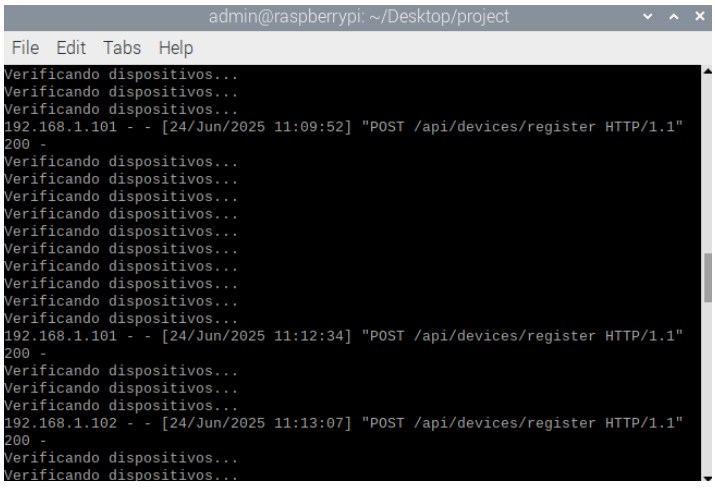


Figura 4: Debug en terminal de la Raspberry Pi.

4.4 Medición de tiempos de respuesta y actualización

Durante las pruebas se tomaron los siguientes tiempos aproximados:

- Tiempo medio de respuesta al recibir el anuncio de un dispositivo: aprox. 1 s.
- Tiempo de detección de desconexión: entre 10 y 15 segundos, según el intervalo configurado de verificación.
- Tiempo de actualización del archivo JSON tras una detección: inmediato.

Estos valores son adecuados para un sistema local simple y permiten mantener un monitoreo constante del estado de la red de dispositivos.

4.5 Repetición con varios dispositivos simultáneos

Se conectaron de manera simultánea tres dispositivos ESP32:

- ESP32 Olimex (vía Ethernet)
- ESP32 Wroom32 (vía Wi-Fi)
- ESP32 Nano (vía Wi-Fi)

El sistema registró exitosamente a cada uno sin conflicto. Las actualizaciones al JSON y las verificaciones periódicas funcionaron de forma paralela y sin interferencia entre dispositivos. Esta prueba reafirma que el sistema puede escalar en cantidad de nodos manteniendo una respuesta consistente.

5. Conclusión

Las pruebas realizadas durante esta etapa del proyecto permitieron validar con éxito un sistema básico de interoperabilidad entre dispositivos embebidos, utilizando tanto conexiones Wi-Fi como Ethernet en una red local gestionada por una Raspberry Pi. El sistema implementado permitió que los dispositivos ESP32 se anunciaran automáticamente al servidor, fueran registrados sin intervención manual y monitoreados en tiempo real mediante el mantenimiento de una estructura JSON centralizada.

Este resultado confirma la viabilidad de un esquema plug and play elemental, en el cual nuevos nodos pueden incorporarse al sistema sin necesidad de configuración específica, cumpliendo así con uno de los pilares de los sistemas inteligentes distribuidos. Se logró documentar satisfactoriamente la detección, el registro y el seguimiento del estado de los dispositivos conectados a la red.

Sin embargo, es importante destacar las limitaciones actuales del sistema. Si bien se logra una interoperabilidad funcional y automática, aún no se implementó la asignación dinámica de tareas ni la actualización remota de firmware, debido a las restricciones propias del hardware utilizado (ESP32). Esto limita temporalmente la capacidad de autoconfiguración completa, aspecto clave del plug and play avanzado.

Se plantea como siguiente etapa de trabajo establecer la funcionalidad de asignación de tareas, de forma que los dispositivos conectados no solo se registren, sino que también reciban automáticamente un rol o función según su ubicación o contexto. Con esto se buscará incorporar plenamente la capacidad plug and play al sistema interoperable básico actual.

Los resultados obtenidos en esta fase representan un avance significativo hacia la implementación de soluciones escalables en el ámbito de los sistemas de transporte inteligentes. La arquitectura demostrada podría adaptarse fácilmente para aplicaciones como monitoreo de infraestructura, control de semáforos, sensores ambientales, cámaras de vigilancia o estaciones de información vehicular. La interoperabilidad demostrada facilita además la integración futura con middleware semántico y ontologías, elementos que forman parte de la visión integral del proyecto.

Agradecimientos

El autor desea agradecer al Consejo Nacional de Ciencia y Tecnologia y al programa PROCIENCIA por el apoyo financiero mediante el Proyecto PINV01-24 “Interoperabilidad de Dispositivos con Capacidad Plug and Play Aplicado Sistemas de Transporte Inteligentes: Un Paso Hacia las Smart Cities”.

Referencias

[1] J. Vogt, H. D. Schotten y H. Wieker, «Intelligent Transportation System Protocol Interoperability Evaluation,» *IEEE Open Journal of Intelligent Transportation Systems*, 2025.

[2] F. Alanazi y M. Alenezi, «Interoperability for intelligent traffic management systems in smart cities,» *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 14, n.º 2, págs. 1864-1874, 2024.

[3] P. Agbaje, A. Anjum, A. Mitra, E. Oseghale, G. Bloom y H. Olufowobi, «Survey of interoperability challenges in the internet of vehicles,» *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, n.º 12, págs. 22 838-22 861, 2022.

[4] G. Baldoni, M. Melita, S. Micalizzi, C. Rametta, G. Schembra y A. Vassallo, «A dynamic, plug-and-play and efficient video surveillance platform for smart cities,» en *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2017, págs. 611-612.

[5] I. Al Ridhawi, S. Otoum, M. Aloqaily y A. Boukerche, «Generalizing AI: Challenges and opportunities for plug and play AI solutions,» *IEEE Network*, vol. 35, n.º 1, págs. 372-379, 2020.

[6] fpalacios09, *FPALACIOS09/interoperability*. dirección: `https://github.com/fpalacios09/interoperability`.