CIENTIFICA
TechNoLogica
ISSN:3119-7906, Vol.1Núm.1(2026): Enero-Marzo
https://doi.org/10.17268/tech1.1.2026.06

Sistema de Smart Parking basado en IoT con comunicación MQTT y visualización en tiempo real

IoT-based Smart Parking System with MQTT Communication and Real-Time Visualization

Ruiz Cruzado Claudia Valentina del Pilar, Diaz Diaz Franklin Alexis, Saldaña Amaya Jeanpool Fabricio, Mostacero Bazán Kevin Alejandro, Quesquén Rodas Carlos Miguel

[1] Escuela de Ingeniería de Sistemas, Universidad Nacional de Trujillo, La Libertad, Peru.

*Autor correspondiente: fadiaz@unitru.edu.pe (F-DiazDiaz)

RESUMEN

Esta investigación tuvo como objetivo desarrollar un sistema de gestión de estacionamiento inteligente (Smart Parking) basado en el Internet de las cosas (IoT) para optimizar el monitoreo de plazas disponibles en entornos urbanos. El alcance del estudio abarcó desde el diseño del hardware hasta la implementación de interfaces de usuario para la gestión de datos en tiempo real. La metodología se fundamentó en una arquitectura de cuatro capas: una capa física con sensores ultrasónicos HC-SR04 y microcontroladores ESP32; una capa de integración mediante Node-RED y protocolo MQTT; una capa de datos con MySQL; y una capa de usuario compuesta por un dashboard en Streamlit y un bot de Telegram. Los resultados principales demostraron un flujo de datos continuo y estable, logrando la detección precisa de vehículos con un filtro antirrebote de 2000 ms y la notificación automática de espacios liberados a los usuarios. Se concluye que la arquitectura propuesta es una solución escalable, de bajo costo y eficiente para la infraestructura de ciudades modernas, destacando su capacidad de resguardar la privacidad del usuario y mejorar la toma de decisiones.

Palabras clave: Internet de las cosas (IoT), Smart Parking, MQTT, ESP32, Gestión Urbana.

ABSTRACT

This research aimed to develop an intelligent parking management system (Smart Parking) based on the Internet of Things (IoT) to optimize the monitoring of available spaces in urban environments. The scope of the study ranged from hardware design to the implementation of user interfaces for real-time data management. The methodology was based on a four-layer architecture: a physical layer with HC-SR04 ultrasonic sensors and ESP32 microcontrollers; an integration layer using Node-RED and the MQTT protocol; a data layer with MySQL; and a user layer composed of a Streamlit dashboard and a Telegram bot. The main results demonstrated a continuous and stable data flow, achieving accurate vehicle detection with a 2000 ms debounce filter and automatic notification of freed spaces to users. It is concluded that the proposed architecture is a scalable, low-cost and efficient solution for the infrastructure of modern

cities, highlighting its ability to protect user privacy and improve operational decision-making through the clear separation of responsibilities between hardware and software.

Keywords: *Internet of things (Iot), Smart Parking, MQTT, ESP32, Urban Management.*

1. INTRODUCCIÓN

La regeneración urbana ha evolucionado hacia enfoques que integran innovación tecnológica y conciencia ecológica para abordar los desafíos de congestión y uso del espacio en las ciudades actuales [13]. La urbanización y el crecimiento del parque automotor han puesto en evidencia la ineficiencia de los sistemas tradicionales de estacionamiento, generando grandes pérdidas económicas, de t iempo y combustible, además de mayor congestión, incremento de emisiones de CO2 y riesgo de inseguridad vehicular [1]. Esta asignación ineficiente, junto con la alta demanda, genera una escasez crítica de espacios, lo que provoca que los conductores circulen innecesariamente buscando aparcamiento o utilicen lugares no permitidos en áreas públicas y privadas [1]. Ante esto, la integración del IoT facilita automatizar la detección de espacios y controlar el acceso mediante sensores y plataformas en la nube, optimizando la operación y satisfacción del usuario [2][3][4]. Los sistemas de estacionamiento inteligente disminuyen la congestión al ofrecer información en tiempo real sobre la disponibilidad de las plazas, optimizando los recursos urbanos. [5][6]. El IoT potencia los sistemas al recopilar datos masivos de múltiples sensores y combinarlos con tecnologías como el aprendizaje automático e integración de datos, favoreciendo a una toma de decisiones más eficiente y una gestión del tráfico urbana más ágil ante el crecimiento vehicular constante [10][11][12]. En este sentido, Reyes Salazar et al. demuestran que la integración de cámaras y Big Data, apoyada por IA, permite adaptar los sistemas de estacionamiento a las necesidades específicas de cada ciudad [14]. Por su parte, Al Mamun et al. proponen arquitecturas basadas en Arduino y Raspberry Pi con comunicación MQTT para la gestión en tiempo real [15], mientras que Luque Vega et al. ofrecen soluciones escalables mediante visión por computador y reconocimiento de matrículas para entornos inteligentes [16]. Complementariamente, Alaa Ala'anzy et al. investigan el uso de fog computing para procesar datos cerca de la fuente, demostrando que este enfoque reduce la latencia y la congestión de red de manera más eficiente que los modelos basados puramente en la nube [17]. La implementación de estas soluciones hace posible reservar espacios con anticipación y optimizar la movilidad en eventos masivos [8]. Además, la llegada de redes como 6G potenciará estas herramientas gracias a su menor latencia y mayor fiabilidad [9]. No obstante, la literatura también señala desafíos críticos que aún deben resolverse: los altos costos de implementación, la complejidad técnica y los problemas de privacidad y estabilidad que afectan la confiabilidad total de los sistemas IoT [7]. A pesar de estos avances, persiste una brecha entre la recopilación de datos a nivel de sensor y su integración efectiva en sistemas de gestión urbana accesibles. Esta problemática impulsa la necesidad de proponer arquitecturas que utilicen el prototipado virtual para validar el flujo de información en entornos inteligentes. Por consiguiente, el presente artículo tiene como objetivo general desarrollar un sistema de parqueo inteligente, que centralice el monitoreo de plazas disponibles. Para lograrlo, este estudio define los objetivos específicos: diseñar un circuito inteligente; aplicar un modelo de 3 o 5 capas de IoT e implementar una aplicación web para la gestión del parqueo. A través de este enfoque, se busca validar la interoperabilidad entre sensores y protocolos de comunicación, utilizando simuladores virtuales y herramientas de orquestación que ofrecen construir soluciones eficientes y escalables para la infraestructura de las ciudades modernas.

2. MATERIALES Y METODOS

2.A. Arquitectura del Sistema

El sistema de estacionamiento inteligente desarrollado en este trabajo se fundamenta en una arquitectura de cuatro capas que integra dispositivos IoT, middleware de flujos, persistencia de datos e interfaz de usuario, La Figura 1 ilustra el diagrama de bloques funcional del sistema completo.

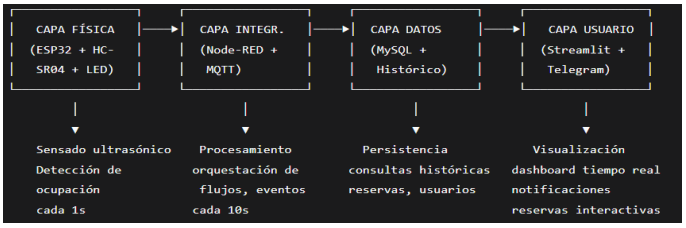

Fig.1. *Arquitectura general del sistema Smart Parking*

La capa física emplea seis módulos de ultrasonido HC-SR04 conectados a un microcontrolador ESP32 (Espressif Systems, Shanghai, China) que publica los estados mediante MQTT hacia un broker público. La capa de integración, implementada en Node-RED (OpenJS Foundation), suscribe los tópicos, procesa los cambios de estado, actualiza la base de datos y gestiona la lógica de reservas y notificaciones. La capa de datos utiliza MySQL Community Server 8.0 (Oracle Corporation, Austin, TX, EE.UU.) con un esquema normalizado que soporta multipropietario (chat_id como

identificador de usuario). Finalmente, la capa de usuario comprende dos interfaces complementarias: un dashboard analítico desarrollado en Streamlit 1.31.0 (Streamlit Inc., San Francisco, CA, EE.UU.) y un bot conversacional en Telegram integrado mediante la API de Node-RED.

2.B. Materiales y Componentes

2.B1 Hardware

Microcontrolador. Se utilizó un módulo ESP32-WROOM-32 (Espressif Systems) con CPU dual-core Xtensa® LX6 a 240 MHz, 520 KB SRAM y conectividad WiFi 802.11 b/g/n integrada. La selección obedece a su bajo costo, amplia disponibilidad y soporte nativo para protocolos TCP/IP y MQTT.

Sensores de proximidad. Se emplearon seis módulos ultrasónicos HC-SR04 (Cytron Technologies, Selangor, Malasia) con las siguientes especificaciones: voltaje de operación 5 VCC, corriente estática <2 mA, rango de medición 2–400 cm, resolución 0.3 cm y ángulo de detección efectivo 15°. Cada sensor integra un transmisor, receptor y circuito de control en una placa de 45×20×15 mm.

Actuadores. Un LED de 5 mm (Everlight Electronics, Taipei, Taiwán) con resistencia limitadora de 220 Ω se configuró como indicador visual de actividad del sistema (parpadeo cada 500 ms).

2.B2 Software y Librerías

Entorno de desarrollo embebido. El firmware del ESP32 se desarrolló en Arduino IDE 2.3.2 (Arduino SA) sobre el framework ESP32 Arduino Core v2.0.17. Las librerías empleadas fueron:

- WiFi.h (estándar) para conectividad inalámbrica.
- PubSubClient v2.8.0 de Nick O'Leary para comunicación MQTT [18].
- ArduinoJson v6.21.3 de Benoit Blanchon para serialización de datos [19].

Plataforma de integración. Node-RED v3.1.0 se ejecutó sobre Node.js v18.19.0 (OpenJS Foundation) en un contenedor Docker (Docker Inc., Palo Alto, CA, EE.UU.). Los nodos adicionales instalados fueron:

- node-red-contrib-telegrambot v17.0.5 para integración con Telegram.
- node-red-node-mysql v3.0.0 para conectividad con MySQL.
- node-red-dashboard v3.6.6 para visualización embebida.

Base de datos. MySQL Community Server 8.0.36 (Oracle Corporation) con motor InnoDB. La configuración de caracteres se estableció en UTF-8.

Dashboard analítico. Se implementó en Streamlit 1.31.0 (Streamlit Inc.) con las siguientes dependencias Python:

- pandas 2.1.4 para manipulación de datos.
- plotly 5.18.0 para visualizaciones interactivas.
- mysql-connector-python 8.2.0 para conexión a base de datos.

Simulación y prototipado. La validación preliminar del hardware se realizó en Wokwi Online Simulator v1.0 (Wokwi, Tel Aviv, Israel) utilizando diagramas JSON personalizados que replicaban el conexionado físico.

2.B3 Servicios Externos

Broker MQTT. Se seleccionó el broker público HiveMQ Cloud (HiveMQ GmbH, Múnich, Alemania) en el endpoint broker.hivemq.com:1883 por su disponibilidad gratuita y estabilidad.

Bot de Telegram. Se creó un bot mediante @BotFather con el nombre @smartpkgbot, asignando un token de acceso único.

2.C. Métodos

2.C1 Diseño e Implementación del Firmware Embebido

El firmware del ESP32 se estructuró siguiendo un patrón de máquina de estados para la lectura de sensores y publicación MQTT. El algoritmo principal (Algoritmo 1) opera en tres fases: inicialización de periféricos, conexión a red WiFi y broker, y ciclo principal de sensado-publicación.

Conexión WiFi y MQTT. Se implementó una rutina de reconexión automática con reintentos exponenciales. El identificador de cliente MQTT se generó dinámicamente mediante la concatenación del proyecto ID (smartparking_demo) y un sufijo hexadecimal aleatorio, garantizando unicidad en el broker compartido.

Medición de distancia. La función readDistance(trigPin, echoPin) implementa el protocolo estándar del HC-SR04 [20]: generación de un pulso TTL de 10 µs en el pin TRIG, medición del ancho del pulso de retorno en ECHO mediante pulseln(). Se configuró un tiempo máximo de espera de 30 ms, equivalente a una distancia máxima detectable de 510 cm, excediendo el rango útil del sensor. La conversión a distancia lineal se realizó mediante la ecuación:

distancia_cm = (duración_µs × 0.034) / 2

Los valores fuera de rango (>400 cm o <2 cm) se descartaron automáticamente, manteniendo la última lectura válida.

Filtro antirrebote. Para evitar transiciones espurias causadas por reflexiones o detecciones marginales, se incorporó un filtro temporal de 2000 ms (DEBOUNCE_TIME). Un cambio de estado solo se confirma si la nueva medición persiste durante al menos dicho intervalo. Este valor se determinó experimentalmente en pruebas preliminares con vehículos en movimiento lento (<5 km/h).

Publicación MQTT. Se definieron tres tópicos bajo el espacio nominal smartparking_demo/:

- /status: publica el estado completo de los seis espacios cada 10 segundos (PUBLISH_INTERVAL). El payload JSON incluye un arreglo con objetos conteniendo space, status y distance.
- /changes: publica eventos inmediatos ante una transición confirmada de estado, incluyendo space, status, previous y distance.
- /system: mensaje de presencia al iniciar la conexión, indicando estado "online" y número total de espacios.

El buffer MQTT se incrementó a 1024 bytes para permitir payloads completos sin fragmentación.

2.C2 Flujos de Node-RED

La capa de integración se implementó mediante dos pestañas de flujo independientes pero interconectadas: MQTT a Dashboard y Telegram Bot.

Flujo principal (MQTT-Dashboard). Este flujo suscribe los tópicos /status y /changes del broker HiveMQ. Al recibir /status:

El nodo función Procesar Estado extrae el arreglo spaces y calcula métricas agregadas (total, disponibles, ocupados, tasa de ocupación).

Las métricas se envían a:

- Nodo ui_gauge para visualización en dashboard Node-RED.
- Nodo ui_text para resumen textual.
- Nodo ui_template que renderiza HTML dinámico con tarjetas para cada espacio.

Simultáneamente, se ejecutan consultas parametrizadas a MySQL mediante mysql node para actualizar parking_spaces.

Al recibir /changes:

- Se valida la transición: si el estado anterior era "reservado" y el nuevo es "disponible", el evento se descarta (lógica de integridad).
- Se ejecuta CALL update_parking_status con los parámetros correspondientes, si el nuevo estado es "disponible", se consulta la tabla subscribers para obtener todos los chat_id con notificaciones activas. Por cada suscriptor, se envía un mensaje al bot de Telegram mediante el nodo telegram sender.

Flujo de Telegram. El bot atiende comandos mediante un switch router basado en msg.payload.content. Cada comando implementa un patrón pipeline: consulta a MySQL → procesamiento → respuesta. Destacan:

- /status: consulta parking_spaces, formatea respuesta con emojis y parse_mode Markdown.
- /subscribe: realiza INSERT ... ON DUPLICATE KEY UPDATE, activando notificaciones para el chat_id.

La comunicación con el broker MQTT es bidireccional: el comando /status fuerza una publicación en el tópico /command del broker, lo que desencadena una respuesta inmediata del ESP32.

3.3 Dashboard Analítico en Streamlit

La interfaz de visualización avanzada se implementó en Streamlit, ejecutándose en un entorno Python virtualizado (Python 3.11.9). La arquitectura del dashboard sigue el patrón Model-View-Controller implícito:

Modelo. Clase DatabaseManager que encapsula la conexión MySQL mediante mysql.connector. Implementa métodos parametrizados para consultas predefinidas (get_parking_status, get_statistics, get_hourly_activity, get_active_reservations). Todas las consultas utilizan prepared statements para prevenir inyección SQL.

Vista. Se compone de componentes renderizados mediante st.markdown con CSS personalizado embebido. Los estilos definen:

- Tarjetas dinámicas para espacios (space-card) con gradientes y efectos hover.
- Métricas principales con tipografía de gran formato.
- Contenedores para gráficas Plotly.

Controlador. La función main() orquesta la lógica de negocio: obtiene filtros desde la barra lateral, consulta datos según el tipo de análisis seleccionado, y despliega visualizaciones apropiadas. Se implementó un mecanismo de auto-refresh configurable (5–60 s) mediante st.rerun().

Visualizaciones. Se empleó Plotly Graph Objects para:

- Gráfico de indicador tipo gauge para tasa de ocupación
- Gráfico de pastel con distribución disponible/ocupado.

3. RESULTADOS

Smart Parking fue validado mediante simulaciones y pruebas de integración, logrando un flujo de datos continuo. Dando asi los siguientes resultados:

III-A. Operación del Hardware y Conectividad

El microcontrolador ESP32 gestionó con éxito la lectura de los seis sensores ultrasónicos HC-SR04, manteniendo una estabilidad de conexión con el broker HiveMQ Cloud.

- Las mediciones de distancia se estabilizaron mediante el filtro de software, descartando valores fuera del rango de 2 a 400cm.
- El filtro antirrebote de 2000ms eliminó las transiciones espurias, confirmando el cambio de estado solo cuando la presencia del vehículo fue persistente.

- La latencia de publicación en el tópico /changes fue inmediata tras la confirmación del estado, mientras que el reporte general en /status se cumplió en intervalos de 10 segundos.

III-B. Gestión de Datos y Orquestación

La capa de integracion en Node-RED procesó las payloads JSON de hasta 1024 bytes sin fragmentación.

- Se ejecutaron los procedimientos almacenados en MySQL para actualizar la tabla parking_spaces en tiempo real.
- La lógica de integrad validó las transiciones, evitando que un espacio marcado como “reservado” cambiara a “disponible” por errores de lectura del sensor.
- El bot de Telegram envió notificaciones automáticas a los usuarios registrados en la tabla suscribers cada vez que se detectó la liberación de una plaza.

III-C. Visualizacion e Interaccion

Se implementaron dos plataformas funcionales para el monitoreo y control:

1. Dashboard en Streamlit: Mostró métricas agregadas y la tasa de ocupación mediante gráficos de tipo gauge y pastel de la librería Plotly. El mecanismo de auto-actualización refrescó los datos satisfactoriamente en el intervalo configurado.
2. Bot de Telegram: Respondió a los comandos /status y /suscribe con un formato de mensajes tipo Markdown y emojis, facilitando la lectura para el usuario final

Componente	Función Validada	Protocolo/Herramienta
ESP32	Detección y publicación de eventos	MQTT
Node-RED	Orquestación y lógica de negocio	JavaScript/Nodos
MySQL	Persistencia de estados y usuarios	SQL
Streamlit	Visualización analítica	Python

Tabla 1. Resumen de la validacion funcional de los componentes y protocolos integrados en la arquitectura IoT

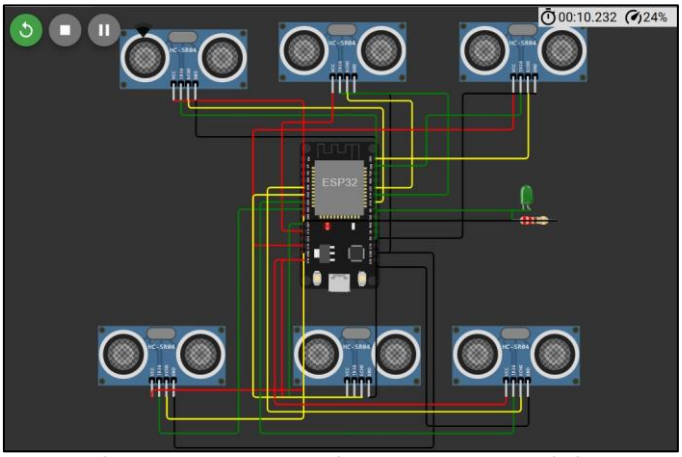

Fig. 2. *Simulación del circuito ESP32 con sensores HC-SR04 en Wokwi*

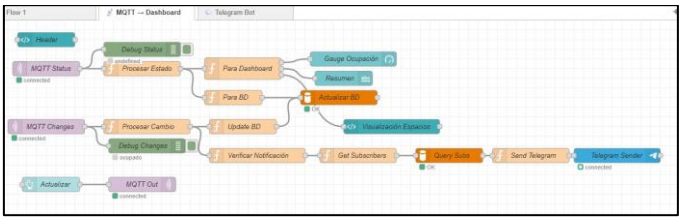

Fig. 3. *Flujo MQTT–Dashboard en Node-RED para procesamiento y notificaciones*

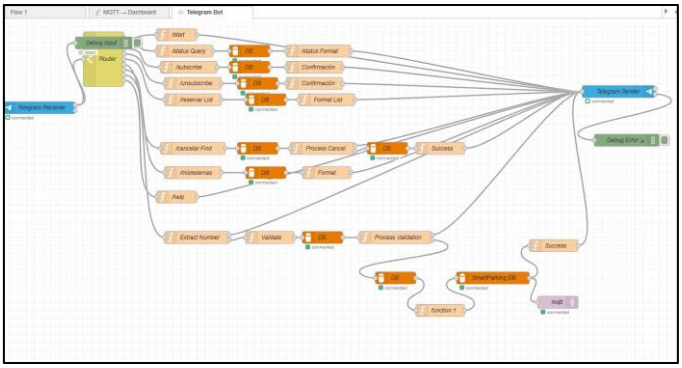

Fig. 4. *Flujo del bot de Telegram en Node-RED con enrutamiento de comandos*

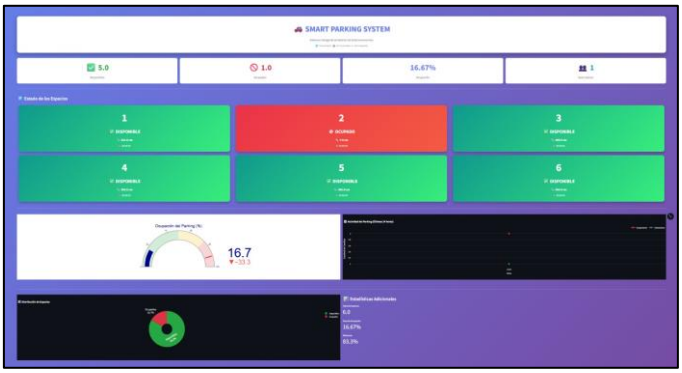

Fig. 5. *Dashboard analítico del Smart Parking System en Streamlit*

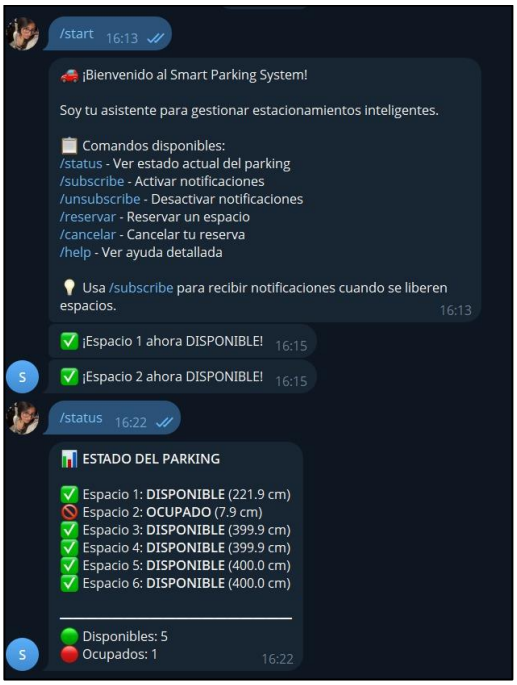

Fig. 6. *Interfaz del bot de Telegram con notificaciones y estado del parking*

4. DISCUSIÓN

Los resultados obtenidos en la implementación del sistema Smart Parking demuestran que es posible lograr un monitoreo estable y en tiempo real superando varios de los obstáculos identificados en la literatura reciente. Mientras que autores como Reyes Salazar et al. [14] y Luque-Vega et al. [16] apuestan por la integración de cámaras, visión por computador e Inteligencia Artificial para la gestión de estacionamientos, estas aproximaciones suelen enfrentarse a los desafíos críticos advertidos por Ahmed et al. [7]: altos costos de implementación y vulnerabilidades en la privacidad. Nuestro sistema, al basarse en una capa física de sensores ultrasónicos (HC-SR04) y microcontroladores ESP32, ofrece una alternativa que mitiga estos problemas, garantizando la anonimización de los datos vehiculares (privacidad) y reduciendo drásticamente la complejidad y el costo de hardware.

En cuanto a la arquitectura de comunicación, Al Mamun et al. [15] evidenciaron la eficacia del protocolo MQTT utilizando combinaciones de Arduino y Raspberry Pi. En nuestro estudio, la centralización de la conectividad y el procesamiento inicial directamente en el ESP32 simplificó la topología de la red. Además, Alaa Ala'anzy et al. [17] subrayan la necesidad del fog computing para reducir la latencia que generan los sistemas basados puramente en la nube. Aunque nuestra arquitectura utiliza un broker MQTT en la nube (HiveMQ), la implementación de un filtro antirrebote de 2000 ms a nivel de firmware actuó como una forma de procesamiento en el borde (edge processing). Esto permitió que las decisiones de cambio de estado se confirmaran localmente, logrando una latencia de publicación inmediata hacia la nube sin saturar el ancho de banda con falsos positivos.

Finalmente, la interacción con el usuario final es un componente vital para optimizar la movilidad urbana [8]. En consonancia con las tendencias recientes que buscan soluciones accesibles [7], la integración de un bot de Telegram operado mediante Node-RED demostró ser altamente efectiva. La notificación automática y bidireccional de espacios liberados, complementada con el dashboard analítico en Streamlit, cierra la brecha entre la recolección de datos crudos y la toma de decisiones del conductor, demostrando que una orquestación adecuada de herramientas de código abierto puede igualar o superar la eficiencia de sistemas comerciales más robustos, incluso antes de la masificación de tecnologías de ultra baja latencia como el 6G [9].

5. CONCLUSION

Se logró desarrollar exitosamente un sistema de parqueo inteligente que facilita una movilidad más ágil, reduciendo el tiempo de búsqueda de estacionamiento y las emisiones asociadas.

Se diseñó un filtro anti rebote en el circuito que fue determinante en reducir los falsos positivos antes de la publicación al broker MQTT en la nube.

Se aplicó una arquitectura de cuatro capas que permitió una separación clara de responsabilidades entre el hardware y el software, facilitando la interoperabilidad entre sensores y protocolos de comunicación.

Se orquestó Node-RED, combinada con el bot de Telegram y el dashboard en Streamlit, confirmando que herramientas de código abierto permiten cerrar la brecha entre la captura de datos y la toma de decisiones del usuario de forma efectiva y accesible.

Como trabajos futuros, se propone integrar modelos predictivos de ocupación, incorporar reconocimiento de matrículas para un control de acceso y validar el sistema en un entorno físico real con tráfico urbano variable.

I. RECONOCIMIENTOS

Expresamos nuestro profundo agradecimiento a la Universidad Nacional de Trujillo, siendo esta publicación un reflejo de la excelencia académica que la universidad promueve, reafirmando nuestro compromiso de contribuir al

avance del conocimiento en beneficio de nuestra sociedad.

6. REFERENCIAS

[1] A. O. Elfaki, W. Messoudi, A. Bushnag, S. Abuzneid, and T. Alhmiedat, "A smart real-time parking control and monitoring system," Sensors, vol. 23, no. 24, p. 9741, Dec. 2023, doi: 10.3390/s23249741.

[2] N. Sakib, A. S. M. Bakibillah, S. Susilawati, Md. A. S. Kamal, and K. Yamada, "Eco-friendly smart car parking management system with enhanced sustainability," Sustainability, vol. 16, no. 10, p. 4145, May 2024, doi: 10.3390/su16104145.

[3] M. Laaouafy, F. Lakrami, and O. Labouidya, "A smart parking system combining IoT and AI to address improper parking," Int. J. Inf. Technol. Secur., vol. 16, no. 2, pp. 39–50, Jun. 2024, doi: 10.59035/zmry7124.

[4] Z. S. Attarbashi, T. Thamodharan, A. Abuzaraida, M. A. A. M. Ali, S. Madanian, and T. Nguyen, "Using IoT-based mobile application to build smart parking system," in 2023 IEEE 9th Int. Conf. Comput. Eng. Design (ICCED), 2023, pp. 1–6, doi: 10.1109/ICCED60214.2023.10425326.

[5] A. Tembhurne, Y. Sadawarti, Y. Meshram, D. Wadgaonkar, N. Panchbhute, and P. Kshirsagar, "IoT based smart parking system," Int. J. Res. Appl. Sci. Eng. Technol., vol. 12, no. 4, pp. 3821–3825, Apr. 2024, doi: 10.22214/ijraset.2024.60763.

[6] C. Liang and Z. Zhang, "Intelligent parking system based on Internet of Things," Acad. J. Sci. Technol., vol. 11, no. 3, pp. 73–74, Jul. 2024, doi: 10.54097/dw2peg62.

[7] M. M. Ahmed et al., "Integrating IoT technologies for smart parking management: A cost-effective system with Telegram-based real-time alerts," in Proc. 2024 IEEE 22nd Student Conf. Res. Develop. (SCOReD), Shah Alam, Malaysia, Dec. 2024, doi: 10.1109/SCOReD63787.2024.10872700.

[8] M. Abu Shroud, M. Eame, E. Elsaghayer, A. Almabrouk, and Y. F. Nassar, "Challenges and opportunities in smart parking sensor technologies," Int. J. Electr. Eng. Sustain., vol. 1, no. 3, pp. 44–59, 2023.

[9] A. A. Anvigh, Y. Khavan, and B. Pourghebleh, "Transforming vehicular networks: How 6G can revolutionize intelligent transportation?," Science, Engineering and Technology, vol. 4, no. 1, pp. 80–93, 2024, doi: 10.54327/set2023/v4.i1.127.

[10] Z. Din, D. I. Jambari, M. M. Yusof, and J. Yahaya, "Critical success factors for managing information systems security in smart city enabled by internet of things," Int. J. Adv. Res. Eng. Technol., vol. 11, no. 12, pp. 1108–1120, 2020, doi: 10.34218/IJARET.11.12.2020.108.

[11] R. Hassan, F. Qamar, M. K. Hasan, A. H. M. Aman, and A. S. Ahmed, "Internet of things and its applications: A comprehensive survey," Symmetry, vol. 12, no. 10, Art. no. 1674, 2020, doi: 10.3390/sym12101674.

[12] X. Mu and M. F. Antwi-Afari, "The applications of Internet of Things (IoT) in industrial management: A science mapping review," Int. J. Prod. Res., vol. 62, no. 7, pp. 1928–1952, 2024, doi: 10.1080/00207543.2023.2290229.

[13] G. F. G. Catania, F. Sortino, and A. Petralia, "Review and evaluation of the evolution of urban regeneration in the era of technological and ecological transition," AIP Conf. Proc., vol. 3269, Art. no. 080002, 2025, doi: 10.1063/5.0248176.

[14] F. d. R. Reyes Salazar et al., "IoT and new technologies to improve Smart Parking efficiency: Systematic review," in Proc. 22nd LACCEI Int. Multi-Conf. Eng., Educ., Technol., San Jose, Costa Rica, Jul. 2024, pp. 1–11, doi: 10.18687/LACCEI2024.1.1.999.

[15] A. A. Mamun, A. Hasib, A. S. M. Mussa, R. Hossen, and A. Rahman, "IoT-Enabled Smart Car Parking System through Integrated Sensors and Mobile Applications," arXiv preprint arXiv:2412.10774, Dec. 2024.

[16] L. F. Luque-Vega, D. A. Michel-Torres, E. Lopez-Neri, M. A. Carlos-Mancilla, and L. E. González-Jiménez, "IoT smart parking system based on the visual-aided smart vehicle presence sensor: SPIN-V," Sensors, vol. 20, no. 5, Art. no. 1476, Mar. 2020, doi: 10.3390/s20051476.

[17] M. AlaaAla'anzy, A. Abilakim, R. Zhanuzak, and L. Li, "Real time smart parking system based on IoT and fog computing evaluated through a practical case study," Sci. Rep., vol. 15, Art. no. 33483, 2025, doi: 10.1038/s41598-025-15507-6.

[18] N. O'Leary, PubSubClient MQTT library for Arduino, GitHub repository, 2024. [En línea]. Disponible: https://github.com/knolleary/pubsubclient

[19] B. Blanchon, ArduinoJson: Efficient JSON serialization for embedded C++, GitHub repository, 2024. [En línea]. Disponible: https://github.com/bblanchon/ArduinoJson

[20] Cytron Technologies, HC-SR04 Ultrasonic Sensor User Guide, Rev. 1.0, 2018.

[21] L. F. Luque-Vega, D. A. Michel-Torres, E. Lopez-Neri, M. A. Carlos-Mancilla, and L. E. González-Jiménez, "IoT

smart parking system based on the visual-aided smart vehicle presence sensor: SPIN-V," Sensors, vol. 20, no. 5, Art. no. 1476, Mar. 2020. [En línea]. Disponible: https://www.mdpi.com/1424-8220/20/5/1476

[22] F. d. R. Reyes Salazar, J. J. Valdivia Rios, H. T. Ráez Martinez, and G. H. Pachas Quispe, "IoT and new technologies to improve Smart Parking efficiency: Systematic review," in Proc. 22nd LACCEI Int. Multi-Conf. Eng., Educ., Technol., Jul. 2024. [En línea]. Disponible: https://laccei.org/LACCEI2024-CostaRica/papers/Contribution_999_final_a.pdf

[23] M. AlaaAla'anzy, A. Abilakim, R. Zhanuzak, and L. Li, "Real time smart parking system based on IoT and fog computing evaluated through a practical case study," Scientific Reports, vol. 15, Art. no. 33483, 2025. [En línea]. Disponible: https://www.nature.com/articles/s41598-025-15507-6

[24] A. A. Mamun, A. Hasib, A. S. M. Mussa, R. Hossen, and A. Rahman, "IoT-Enabled Smart Car Parking System through Integrated Sensors and Mobile Applications," arXiv preprint arXiv:2412.10774, Dec. 2024. [En línea]. Disponible: https://arxiv.org/html/2412.10774v1