

ANÁLISIS COMPARATIVO DE LATENCIA EN PROTOCOLOS DE COMUNICACIÓN IOT UTILIZANDO ESP8266 Y NODE-RED

Guzmán-González Wendy Janett*, Guzmán-González Cynthia Liliana*, Pallares-Rubio Sonia Lisbeth*, Gómez-Gaytán Jaime Fernando*, Valenzuela-Segura Lucrecia Guadalupe*, Guzmán-González Brenda Eunice**

*Tecnológico Nacional de México / Instituto Tecnológico de Chihuahua. Av. Tecnológico No. 2909, Chihuahua, Chih, México. C.P. 31310. Tel. +52 (614) 2-01-2000

** Colegio de Estudios Científicos y Tecnológicos del Estado de Durango plantel 09. Avenida del Tecnológico s/n, Lerdo Durango, México. C.P. 35150. Tel. +52 (871) 9-43-1523

wendy.gg@chihuahua.tecnm.mx, cynthia.gg@chihuahua.tecnm.mx, sonia.pr@chihuahua.tecnm.mx,
jaime.gg@chihuahua.tecnm.mx, lucrecia.vs@chihuahua.tecnm.mx, brenda.guzman.cecyled@cecyled.edu.mx

RESUMEN.

La presente investigación tiene como objetivo analizar y comparar el tiempo de latencia de diferentes protocolos de comunicación utilizados en aplicaciones de IoT (Internet of Things): HTTP GET y HTTP POST (Hypertext Transfer Protocol), TCP (Transmission Control Protocol), UDP (User Datagram Protocol) y MQTT (Message Queuing Telemetry Transport). A través de un entorno experimental controlado, se midieron los tiempos de respuesta en milisegundos bajo condiciones idénticas de red. Los resultados evidencian que los protocolos UDP y MQTT ofrecen una latencia significativamente menor (1ms), lo que los posiciona como alternativas eficientes para aplicaciones sensibles al tiempo, tales como monitoreo médico en tiempo real, control de robots industriales, etc. El estudio también incluye una revisión de literatura que valida estos hallazgos y permite contrastar el desempeño de los protocolos evaluados. Este análisis, complementado con la revisión de literatura, proporciona información clave para la selección de tecnologías de comunicación en sistemas IoT. En particular, la comparación con estudios previos permite identificar qué protocolos resultan más adecuados en distintos contextos.

Palabras clave: internet de las Cosas, latencia, protocolos de comunicación

ABSTRACT.

The present research aims to analyze and compare the latency of different communication protocols used in Internet of Things (IoT) applications: HTTP GET and HTTP POST (Hypertext Transfer Protocol), TCP (Transmission Control Protocol), UDP (User Datagram Protocol), and MQTT (Message Queuing Telemetry Transport). Through a controlled experimental environment, response times were measured in milliseconds under identical network conditions. The results show that UDP and MQTT protocols offer significantly lower latency (1 ms), positioning them as efficient alternatives for time-sensitive applications such as real-time medical monitoring and industrial robot control, among others. The study also includes a literature review that validates these findings and enables a comparison of the performance of the evaluated protocols. This analysis, complemented by the literature review, provides key information for the selection of communication technologies in IoT systems. In particular, the

comparison with previous studies makes it possible to identify which protocols are most suitable in different contexts.

Keywords: Internet of Things, latency, communication protocols

INTRODUCCIÓN

Las arquitecturas y protocolos de comunicación en el Internet de las Cosas han evolucionado para afrontar nuevos retos derivados de entornos con gran cantidad de dispositivos heterogéneos y con recursos limitados.

Algunos de estos desafíos incluyen el procesamiento de grandes volúmenes de datos, filtrado, minería de datos, clasificación, y el soporte a una alta heterogeneidad tanto en hardware como en software, así como en los tipos de tráfico. Además, los servicios de IoT de extremo a extremo son soportados por el protocolo TCP/IP (Transmission Control Protocol / Internet Protocol), que, aunque permite interoperabilidad, presenta limitaciones en escenarios IoT debido a requisitos como: sensibilidad al tiempo, restricciones de energía o la necesidad de baja latencia. Por lo tanto, los escenarios IoT se apoyan en protocolos de mensajería basados en IP o en marcos protocolares como OPC UA (Open Platform Communications Unified Architecture) para satisfacer estos requisitos, proporcionando soporte a la comunicación asincrónica mediante un servidor o entidad intermediaria (broker). [1]

TRABAJOS RELACIONADOS

A fin de contextualizar los resultados obtenidos en este estudio, se llevó a cabo una revisión de literatura enfocada en investigaciones previas que evalúan el rendimiento de distintos protocolos de comunicación utilizados en entornos IoT, particularmente en relación con el tiempo de latencia. La Tabla

1 resume los resultados más relevantes de estos trabajos, donde se realiza una comparación cualitativa de protocolos como HTTP, MQTT, CoAP (Constrained Application Protocol), TCP y UDP en función de su eficiencia, estabilidad y adaptabilidad en escenarios con restricciones de red o procesamiento. Estos estudios permiten establecer un marco comparativo con la experimentación desarrollada en el presente artículo.

Artículo	Protocolos comparados	Conclusión principal
Amirkhanov et al. (2025) [2]	HTTP, MQTT/TCP, MQTT/WebSocket	MQTT/TCP mostró menor latencia de 290 ms
Azzouzi et al. (2021) [3]	MQTT, CoAP, OPC UA	CoAP tuvo mejor desempeño en latencia
Seoane & Garcia (2021) [4]	CoAP, MQTT (con seguridad)	CoAP mantiene menor latencia con TLS/DTLS
Gündoğan et al. (2018) [5]	NDN, CoAP, MQTT	NDN mejor en multihop; CoAP/MQTT rápidos en 1 salto
Kumar & Dezfouli (2018) [6]	MQTT/TCP vs MQTT/QUIC	QUIC reduce la latencia en ~55%
Abid & Kaur (2024) [7]	MQTT + TOR	Seguridad, añade latencia considerable

Tabla 1. Comparación cualitativa de estudios relacionados sobre latencia en protocolos de comunicación para IoT

En general, se observa que los estudios coinciden en que MQTT y CoAP tienden a ofrecer mejores resultados en términos de latencia en comparación con protocolos como HTTP o incluso TCP. Por ejemplo, Amirkhanov [2] reporta que MQTT sobre TCP presenta menor latencia que HTTP, mientras que Seoane & Garcia [4] destacan que CoAP supera a MQTT en ciertos escenarios, especialmente cuando se aplican mecanismos de seguridad como TLS/DTLS.

Asimismo, otros trabajos como el de Gündoğan [5] analiza el desempeño de NDN (Named Data Networking), encontrando que, aunque CoAP y MQTT son rápidos en topologías simples, NDN puede ser más eficiente en redes multihop. Por su parte, Kumar y Dezfouli [6] proponen el uso de QUIC (Quick UDP Internet Connections) como alternativa a TCP en implementaciones MQTT, logrando reducir significativamente la latencia (~55%). Finalmente, Abid y Kaur [7] advierten que el uso de capas adicionales de seguridad como TOR (The Onion Router), aunque útiles para anonimato, incrementa notablemente los tiempos de respuesta.

METODOLOGÍA

Para evaluar la latencia de los protocolos de comunicación, se diseñó un entorno de pruebas controlado utilizando la plataforma Node-RED como servidor de recepción de datos, y un microcontrolador ESP8266 configurado como cliente. La comunicación se desarrolló en una red WiFi local con direccionamiento IP estático para asegurar consistencia en las pruebas.

Se eligió Node-RED como servidor debido a su flexibilidad para la integración de múltiples protocolos de comunicación IoT y su facilidad de visualización de datos en tiempo real. Asimismo, se seleccionó el microcontrolador ESP8266 como cliente por su bajo costo, amplio uso en aplicaciones IoT y soporte nativo para los protocolos evaluados. Esta combinación permitió diseñar un entorno experimental representativo de escenarios IoT reales, pero controlado y replicable.

Las Condiciones del experimento fueron:

1. Todos los ensayos se realizaron con el mismo microcontrolador ESP8266.
2. Se utilizó el mismo router, sin tráfico adicional en la red durante las pruebas.
3. La latencia fue registrada directamente desde el monitor serial del IDE de Visual Studio Code.
4. Se tomaron valores bajo condiciones idénticas para asegurar un análisis comparativo justo.

Lo anterior permitió evaluar de forma controlada y replicable el desempeño de cada protocolo, midiendo su latencia real en un entorno de red local doméstico típico de aplicaciones IoT.

CONFIGURACIÓN DEL SERVIDOR (NODE-RED)

Se configuraron cinco flujos en Node-RED, uno para cada protocolo: HTTP GET, HTTP POST, TCP, UDP y MQTT. Cada flujo estaba compuesto por:

- Un nodo de entrada, específico para el protocolo (por ejemplo, HTTP in, TCP in, UDP in, MQTT in).
- Un nodo de función que almacenaba un sello de tiempo (timestamp) en el momento de recepción.
- Un nodo de depuración que mostraba los datos recibidos y el tiempo en la consola de Node-RED.

El servidor Node-RED se ejecutó en una máquina con dirección IP local 192.168.1.142. Los puertos utilizados para cada protocolo fueron los siguientes:

- HTTP: puerto 1880, ruta /timestampMeasure.
- TCP: puerto 4567.
- UDP: puerto 3456.
- MQTT: puerto 1883, tópico esp8266/test.

CONFIGURACIÓN DEL CLIENTE ESP8266

Para cada protocolo se desarrolló un programa individual en PlatformIO sobre Visual Studio Code. Todos los scripts seguían el mismo procedimiento general:

1. Conexión WiFi: conexión a la red especificada mediante `WiFi.begin()`.
2. Inicio de la prueba: registro del tiempo inicial con `millis()`.
3. Envío del mensaje al servidor usando el protocolo correspondiente.
4. Recepción de respuesta (cuando aplica, como en TCP o MQTT).
5. Cálculo de latencia: tiempo final menos tiempo inicial.
6. Visualización del resultado en el monitor serie.

PROTOCOLO HTTP

Para llevar a cabo el proceso de experimentación, se implementó una estructura básica en el entorno de desarrollo que consta de un nodo de entrada y un nodo de respuesta, los cuales se interconectaron estratégicamente con un nodo de depuración. El nodo de entrada cumple la función de recibir las solicitudes o datos enviados desde el dispositivo cliente, mientras que el nodo de respuesta se encarga de emitir una contestación correspondiente, permitiendo así completar el ciclo de comunicación.

El nodo de depuración fue incorporado con el propósito de monitorear en tiempo real la información recibida y enviada a través de la consola de Node-RED. Esta configuración permite observar detalladamente los mensajes transmitidos, facilitando la verificación del flujo de datos y la detección de posibles errores o anomalías durante las pruebas experimentales. En la Figura 1 se presenta el flujo diseñado para los protocolos HTTP GET y HTTP POST.

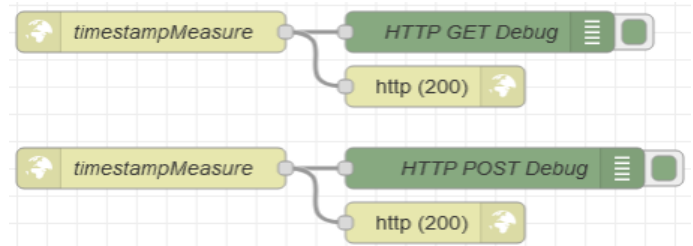


Figura 1. Flujo de Node-RED para la medición de latencia con los métodos HTTP GET y POST

En la Figura 2 se muestra el código implementado en el microcontrolador para el protocolo HTTP POST, el cual lo primero que realiza es conectarse a una red WiFi utilizando las credenciales especificadas, una vez establecida la conexión, se inicia una comunicación HTTP mediante un cliente. Se registra el tiempo justo antes de enviar la solicitud POST vacía al endpoint `http://192.168.1.142:1880/timestampMeasure`, y posteriormente se registra el tiempo en que se recibe la respuesta.

La diferencia entre ambos tiempos se interpreta como la latencia de la comunicación, la cual es mostrada en milisegundos a través del monitor serial. Esta rutina se ejecuta únicamente una vez durante el arranque del sistema, ya que la función `loop()` permanece vacía.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>

const char* ssid = "*****";
const char* password = "*****";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }
  Serial.println("Conectado a la red WiFi");

  if (WiFi.status() == WL_CONNECTED) {
    WiFiClient client;
    HTTPClient http;

    unsigned long startTime = millis();

    http.begin(client, "http://192.168.1.142:1880/timestampMeasure");
    int httpCode = http.POST("");
    unsigned long endTime;
    if (httpCode > 0) {
```

```

        endTime = millis();    }

    unsigned long latency = endTime - startTime;
    Serial.print("Latencia HTTP POST: ");
    Serial.print(latency);
    Serial.println(" ms");
    http.end();  }}

void loop() {}

```

Figura 2. Código para la medición de latencia con el método HTTP POST

Por otra parte, para la implementación del código HTTP GET el dispositivo establece una conexión WiFi utilizando el nombre de red y la contraseña especificados, y una vez conectada, imprime un mensaje de confirmación en el monitor serial. Si la conexión es exitosa, se crea un cliente HTTP y se registra el tiempo justo antes de enviar una solicitud GET al endpoint `http://192.168.1.142:1880/timestampMeasure`. Una vez que el servidor responde, se registra el tiempo final y se calcula la diferencia entre ambos tiempos, lo que representa la latencia total de la comunicación. El valor resultante se muestra en milisegundos a través del puerto serial. Al igual que en el ejemplo anterior, este proceso se realiza una única vez durante la ejecución de la función `setup()`, ya que la función `loop()` no contiene instrucciones. Ver Figura 3.

```

#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>

const char* ssid = "*****";
const char* password = "*****";

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Conectando a WiFi...");
    }
    Serial.println("Conectado a la red WiFi");

    if (WiFi.status() == WL_CONNECTED) {
        WiFiClient client;
        HTTPClient http;
        unsigned long startTime = millis();

        http.begin(client, "http://192.168.1.142:1880/timestampMeasure");
        int httpCode = http.GET();
        unsigned long endTime;
        if (httpCode > 0) {

```

```

        endTime = millis();    }

    unsigned long latency = endTime - startTime;
    Serial.print("Latencia HTTP GET: ");
    Serial.print(latency);
    Serial.println(" ms");
    http.end();  }}

void loop() {}

```

Figura 3. Código para la medición de latencia con el método HTTP GET

PROTOCOLO TCP

El diagrama implementado en Node-RED para el protocolo TCP, se presenta en la Figura 4. En dicho flujo, se emplea un nodo TCP de entrada configurado como servidor, el cual permanece a la escucha en el puerto 4567 a la espera de conexiones entrantes. Una vez que el cliente establece la conexión y transmite el mensaje, este es capturado y visualizado en tiempo real mediante un nodo de depuración, lo que permite monitorear la información recibida directamente en la consola de Node-RED. Esta arquitectura permite verificar la recepción efectiva de los paquetes y, en conjunto con el código embebido en el dispositivo emisor, posibilita el cálculo preciso del retardo de ida y vuelta (RTT). El flujo desarrollado, aunque sencillo, resulta funcional y adecuado para los fines de análisis del rendimiento del protocolo TCP en entornos de comunicación local.



Figura 4. Flujo de Node-RED para la medición de latencia con el método TCP

En la Figura 5 se muestra el código implementado en el microcontrolador para establecer una conexión mediante el protocolo TCP con un servidor ubicado en una red local. El programa inicia conectándose a una red WiFi utilizando las credenciales previamente definidas. Una vez establecida la conexión inalámbrica, se intenta abrir una conexión TCP con el servidor alojado en la dirección IP 192.168.1.142, utilizando el puerto 4567. Antes de iniciar dicha conexión, se registra el tiempo de inicio, y si esta se establece exitosamente, se registra también el tiempo de finalización, lo cual permite calcular la latencia de conexión.

Posteriormente, el microcontrolador envía un mensaje al servidor y permanece a la espera de una respuesta. En caso de

recibirla, el mensaje es mostrado a través del monitor serial, lo que permite validar la correcta recepción por parte del servidor. Finalmente, la conexión se cierra de forma controlada. Esta rutina se ejecuta una sola vez en la función setup(), ya que la función loop() permanece vacía.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

const char* ssid = "INFINITUM24AD";
const char* password = "HFnyEZ3y4E";
const char* serverAddress = "192.168.1.142"; //Dirección IP del
servidor TCP
const int serverPort = 4567;          // Puerto del servidor TCP
WiFiClient client;

void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password); // Conectar a la red WiFi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }
  Serial.println("Conectado a la red WiFi");

  // Intentar conectar al servidor TCP
  unsigned long startTime = millis();
  if (client.connect(serverAddress, serverPort)) {
    unsigned long endTime = millis();
    unsigned long latency = endTime - startTime;
    Serial.print("Conectado al servidor TCP. Latencia de
conexión: ");
    Serial.print(latency);
    Serial.println(" ms");

    // Enviar un mensaje al servidor TCP
    client.print("Mensaje TCP desde ESP8266");

    // Esperar respuesta del servidor
    while (client.connected() && !client.available()) {
      delay(10);
    }

    if (client.available()) {
      String response = client.readString();
      Serial.print("Respuesta del servidor: ");
      Serial.println(response);
      // Cerrar la conexión
      client.stop();
      Serial.println("Conexión cerrada");
    } else {
      Serial.println("No se pudo conectar al servidor TCP");
    }
  }
  void loop() { }
```

Figura 5. Código para la medición de latencia con el método TCP

PROTOCOLO UDP

El diagrama implementado para el protocolo UDP está diseñado para recibir y visualizar paquetes de datos enviados desde un microcontrolador a través de una comunicación no orientada a conexión. En este flujo, se configura un nodo de entrada UDP, el cual permanece a la escucha en un puerto específico, en particular, el puerto 3456, capturando los mensajes que llegan desde el dispositivo emisor.

Una vez que el paquete es recibido, su contenido se muestra en tiempo real mediante un nodo de depuración, lo que permite observar directamente en la consola de Node-RED la información transmitida. La configuración presentada en la Figura 6 al no requerir el establecimiento formal de una conexión como en los protocolos TCP o HTTP, permite evaluar con mayor precisión la velocidad y eficiencia del envío de datos, siendo particularmente útil para la medición del retardo de ida (one-way delay). Ver Figura 6.

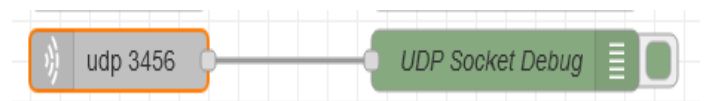


Figura 6. Flujo de Node-RED para la medición de latencia con el método UDP

El programa implementado se presenta en la Figura 7 para evaluar la latencia con el protocolo UDP inicia estableciendo una conexión WiFi con los datos de acceso previamente definidos. Una vez conectado a la red, se inicializa el objeto WiFiUDP, que permite enviar paquetes de datos a través del protocolo UDP. A continuación, se registra el tiempo de inicio justo antes del envío del mensaje y se prepara el paquete con la función beginPacket(), indicando la dirección IP 192.168.1.142 y el puerto 3456 del servidor receptor. El mensaje es enviado mediante print() y el paquete se finaliza con endPacket(). Posteriormente, se registra el tiempo final y se calcula la latencia de envío como la diferencia entre ambos registros temporales, mostrando el resultado en milisegundos a través del monitor serial. Dado que UDP es un protocolo no orientado a conexión, no se contempla una respuesta del servidor, por lo que la medición se enfoca exclusivamente en el tiempo que toma el microcontrolador en generar y despachar el paquete. La función loop() queda vacía, lo que indica que la prueba se realiza una única vez al inicio del programa.

```

#include <WiFiUdp.h>
#include <ESP8266WiFi.h>

const char* ssid = "INFINITUM24AD";
const char* password = "HFnYEZ3y4E";

const char* udpAddress = "192.168.1.142"; // Dirección IP del
servidor UDP
const int udpPort = 3456;           // Puerto del servidor UDP

WiFiUDP udp;

void setup() {
  Serial.begin(115200);

  // Conectar a la red WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }
  Serial.println("Conectado a la red WiFi");

  // Iniciar conexión UDP
  udp.begin(udpPort);

  // Medir latencia de envío UDP
  unsigned long startTime = millis();

  // Enviar mensaje al servidor UDP
  udp.beginPacket(udpAddress, udpPort);
  udp.print("Mensaje UDP desde ESP8266");
  udp.endPacket();

  unsigned long endTime = millis();
  unsigned long latency = endTime - startTime;

  Serial.print("Latencia de envío UDP: ");
  Serial.print(latency);
  Serial.println(" ms");
}

void loop() {
}

```

Figura 7. Código para la medición de latencia con el método UDP

PROTOCOLO MQTT

La Figura 8 muestra el diagrama de flujo para la recepción de mensajes a través del protocolo MQTT. En el flujo se observa un nodo de tipo MQTT de entrada, configurado para suscribirse al tópico esp8266/test, el cual recibe los mensajes publicados por el dispositivo. Este nodo está conectado a un nodo de depuración, identificado como *Local MQTT Broker Debug*, que permite visualizar en tiempo real los mensajes entrantes directamente en la consola de Node-RED. La etiqueta *connected* indica que la conexión con el broker MQTT se encuentra activa, lo cual garantiza que los mensajes transmitidos serán correctamente capturados.



Figura 8. Flujo de Node-RED para la medición de latencia con el método MQTT

En la Figura 9 se presenta el código implementado en un microcontrolador para establecer comunicación mediante el protocolo MQTT. El programa inicia conectando el dispositivo a una red WiFi utilizando las credenciales proporcionadas. Una vez establecida la conexión, se configura un cliente MQTT mediante la biblioteca PubSubClient, indicando la dirección IP “192.168.1.142” y el puerto “883” del broker MQTT, así como el tema “esp8266/test” sobre el cual se publicarán y recibirán mensajes. Durante la inicialización, se intenta establecer conexión con el broker y, si es exitosa, se publica un mensaje en el tema definido. Se mide el tiempo transcurrido desde el inicio hasta el final del proceso de publicación, permitiendo así calcular la latencia de envío. Este valor se muestra en milisegundos por el monitor serial. Adicionalmente, se define una función de “callback” que se ejecuta cada vez que se recibe un mensaje en el tema suscrito, imprimiendo el contenido del mensaje recibido. En la función “loop()”, se mantiene activa la conexión con el broker y se procesan los mensajes entrantes. En caso de pérdida de conexión, el sistema intenta reconectarse de forma automática.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "INFINITUM24AD";
const char* password = "HFnYEZ3y4E";

const char* mqttServer = "192.168.1.142"; // Dirección IP del broker MQTT
const int mqttPort = 1883; // Puerto del broker MQTT
const char* mqttTopic = "esp8266/test"; // Tema para publicar y suscribirse

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    Serial.begin(115200);

    // Conectar a la red WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Conectando a WiFi...");
    }
    Serial.println("Conectado a la red WiFi");

    // Configurar servidor MQTT y función de callback
    client.setServer(mqttServer, mqttPort);
    client.setCallback(callback);

    // Conectar al broker MQTT
    connectToMQTT();

    // Publicar un mensaje en el tema
    if (client.connected()) {
        unsigned long startTime = millis();
        client.publish(mqttTopic, "Mensaje MQTT desde ESP8266");
        unsigned long endTime = millis();
        unsigned long latency = endTime - startTime;

        Serial.print("Mensaje publicado. Latencia de publicación: ");
        Serial.print(latency);
        Serial.println(" ms");
    }
}

void loop() {
    // Mantener la conexión al broker y procesar mensajes
    if (!client.connected()) {
        connectToMQTT();
    }
    client.loop();
}
```

```
// Función de callback para manejar mensajes recibidos
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Mensaje recibido en el tema ");
    Serial.print(topic);
    Serial.print(": ");

    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println();
}

// Función para conectar al broker MQTT con reintentos
void connectToMQTT() {
    while (!client.connected()) {
        Serial.println("Conectando al broker MQTT...");
        if (client.connect("ESP8266Client")) {
            Serial.println("Conectado al broker MQTT");

            // Suscribirse al tema
            client.subscribe(mqttTopic);
        } else {
            Serial.print("Falló la conexión. Código de error: ");
            Serial.print(client.state());
            delay(2000); // Espera 2 segundos antes de reintentar
        }
    }
}
```

Figura 9. Código para la medición de latencia con el método MQTT

RESULTADOS

Los resultados obtenidos muestran una diferencia significativa en la latencia entre los distintos protocolos de comunicación evaluados. Tanto HTTP GET como POST presentaron una latencia promedio de 21 milisegundos, lo cual es coherente con la naturaleza de este protocolo, que requiere el establecimiento de una conexión TCP y el manejo de encabezados adicionales, lo que incrementa el tiempo de respuesta. Por otro lado, el protocolo TCP registró una latencia ligeramente menor, de 19 milisegundos, debido a que, aunque también es orientado a conexión, evita el sobrecosto de procesamiento propio de HTTP. En contraste, los protocolos UDP y MQTT demostraron un desempeño superior, alcanzando latencias de tan solo 1 ms, en comparación con los 19 ms registrados para TCP y los 21 ms de HTTP. Este resultado es consistente con sus características: UDP, al ser un protocolo sin conexión y sin mecanismos de confirmación, permite una transmisión casi instantánea, mientras que MQTT, aunque se basa en TCP, está optimizado para entornos con restricciones de ancho de banda y latencia, lo que

lo hace altamente eficiente en aplicaciones IoT sensibles al tiempo. En la Figura 10 se presentan los resultados obtenidos.

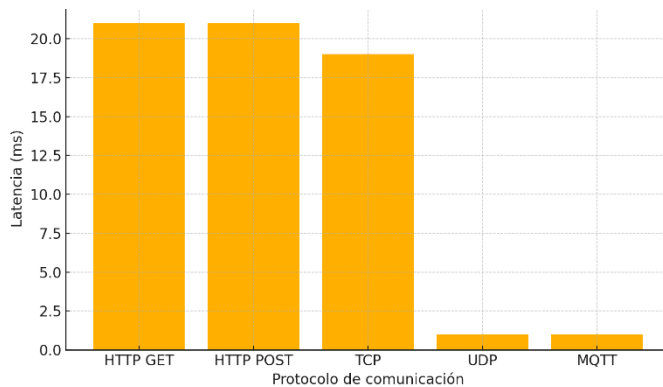


Figura 10. Comparación de la latencia promedio por protocolo de comunicación evaluado

CONCLUSIONES

Los resultados obtenidos en este estudio, donde se midieron los tiempos de latencia de los protocolos HTTP GET, HTTP POST, TCP, UDP y MQTT utilizando microcontroladores ESP8266 y Node-RED, coinciden en gran medida con los hallazgos reportados en la literatura especializada.

En la presente investigación, UDP y MQTT presentaron las latencias más bajas (1 ms), seguidos de TCP (19 ms) y HTTP (21 ms). Estos valores concuerdan con lo reportado por Amirkhanov et al. [2], quienes encontraron que MQTT sobre TCP presentó menor latencia (~290 ms en su entorno experimental), mientras que HTTP resultó más lento debido a la sobrecarga de encabezados y establecimiento de conexión.

Por su parte, CoAP, aunque no fue evaluado experimentalmente en este trabajo, se menciona como el protocolo con mejor desempeño en múltiples estudios [3][4], lo que sugiere que su inclusión en futuras pruebas podría enriquecer el análisis comparativo. Además, se observó que la incorporación de mecanismos de seguridad, como TLS/DTLS, tiende a aumentar la latencia, especialmente en protocolos como MQTT, según lo reportado por Abuarqoub y Jararweh [4].

El estudio de Kumar y Dezfouli [6] demuestra cómo la implementación de QUIC en lugar de TCP para MQTT puede reducir significativamente la latencia (~55 %), lo cual abre una línea de investigación adicional para mejorar aún más el

desempeño de soluciones IoT en redes con alta exigencia temporal.

Finalmente, el estudio de Abid y Kaur [7] reafirma que la incorporación de capas de anonimato y cifrado como TOR puede penalizar el tiempo de respuesta, lo cual debe ser considerado en aplicaciones donde la latencia es crítica.

En conjunto, estos estudios respaldan los resultados experimentales obtenidos en esta investigación, confirmando que para aplicaciones IoT que requieren mínima latencia, los protocolos UDP y MQTT son las opciones más adecuadas, en contextos donde incluso milisegundos de retraso pueden afectar la operación, como en telemedicina, sistemas de control industrial en tiempo real y aplicaciones de seguridad. Mientras que HTTP y TCP podrían reservarse para contextos donde la fiabilidad de la transmisión o la compatibilidad con servicios web tradicionales sea prioritaria.

REFERENCIAS

- [1] S. Daniel, C. Liliana y S. José, «A Performance Analysis of Internet of Things Networking,» *Applied sciences*, vol. 11, pp. 1-20, 2021.
- [2] A. Sergey, E. Boris y B. Dmitry, «Evaluating HTTP, MQTT over TCP and MQTT over WebSocket for digital twin applications: A comparative analysis on latency, stability, and integration.,» *International Journal of Innovative Research and Scientific Studies*, n° 4, pp. 679-694, 2025.
- [3] A. Adeel y K. Amandeep, «A performance analysis of Internet of Things networking protocols,» *Applied Sciences*, vol. 11, n° 21, 2021.
- [4] V. Seoane, C. Garcia-Rubio, F. Almenares Mendoza y C. Campo, «Performance evaluation of CoAP and MQTT with security support for IoT environments,» *Computer networks*, 2021.
- [5] G. Cenik, K. Peter, L. Martine y P. Hauke, «NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT,» *arXiv preprint*, 2018.
- [6] K. Puneet y D. Behnam, «Implementation and Analysis of QUIC for MQTT,» *arXiv preprint*, 2018.
- [7] A. A. a. A. Kaur, «A network performance analysis of MQTT security protocols with constrained hardware in the dark net for DMS,» *Applied Sciences*, 2024.
- [8] S. Victor, G. Carlos, A. Florina y C. Celeste, «Performance evaluation of CoAP and MQTT with security support for IoT environments,» *Computer networks*, 2021.