

IMPLEMENTACIÓN DE UNA SOLUCIÓN IoT

Sara Albarracín, Isabella Palacio, Joshua Prieto Zambrano
Pontificia Universidad Javeriana
Bogotá, Colombia

Abstract—This paper proposes a method for timely identification of forest fires using NodeMCU ESP8266 microcontrollers and DHT22 environmental sensors. The system consists of two nodes: a sensor node that detects changes in temperature and humidity, and an alert node that displays data on an LCD screen and activates an audible alarm. Essential components such as the doorbell are analyzed and improvements such as advanced detection algorithms, extended wireless communication, and solar power options are suggested. Additionally, the MQTT protocol for device communication is examined, covering its features, quality of service levels, security, and case studies. MQTT brokers such as Eclipse Mosquitto, Mosca and EMQ are analyzed, along with a network topology for the system implementation. Finally, a test protocol is provided to evaluate the fire detection system.

I. INTRODUCCIÓN

Los incendios forestales representan una de las mayores amenazas para el medio ambiente y las comunidades en todo el mundo. Su capacidad para causar devastación en la vida silvestre, el ecosistema y la infraestructura humana los convierte en un problema de gran relevancia. En este sentido, la detección temprana y la implementación de medidas preventivas y de respuesta son cruciales para mitigar el impacto negativo que estos fenómenos pueden causar. En respuesta a este problema, se propone un proyecto para desarrollar un sistema para detección y alerta de incendios forestales.

II. DEFINICIÓN DE PROBLEMÁTICA

Cuando hablamos de los incendios forestales podemos concluir que son una amenaza muy importante para el medio ambiente y las comunidades. Por eso la detección y medidas contra este tipo de fenómenos de manera oportuna son cruciales para minimizar el daño que puedan causar. No obstante, la prevención es difícil debido a las dificultades de acceso a los bosques y la falta de sistemas de monitoreo.

Possible Solución:

Como solución a esta problemática planteamos el desarrollo de un sistema, el cual se encargará de la detección y alerta de incendios forestales. Para su desarrollo utilizaremos microcontroladores NodeMCU ESP8266 y sensores ambientales; en específico contará con dos nodos principales (ESP8266):

1. Nodo Sensor:

- Ubicado en un punto estratégico dentro del bosque.
- Cuenta con un sensor DHT22 encargado de medir y analizar la temperatura y humedad, 24/7.
- Detecta cambios repentinos en la temperatura específicamente el aumento de ella y la disminución de la humedad, lo que podría ser un indicio de un inicio de un incendio forestal.

Al momento de detectar esta anomalía se enviará una señal al nodo alerta.

2. Nodo Alerta:

- Se mantiene conectado a una red Wi-Fi o red telefónica.
- Recibe la señal del nodo sensor y la interpreta como una señal de un posible incendio.
- Muestra la información de tanto la temperatura como la humedad en una pantalla LCD 16x2.
- Genera la activación de un buzzer pasivo para generar una alarma sonora.

Componentes necesarios:



Imagen 1: NodeMCU ESP8266 (2)
<https://www.sigmaelectronica.net/wp-content/uploads/2017/12/New-Wireless-Module-CH340-NodeMcu-V3.jpg>



Imagen 3: Buzzer Pasivo (1)

<https://www.steren.com.co/media/catalog/product/cache/b69086f136192bea7a4d681a8eaf533d/image/20313c1b1/buzzer-pasivo.jpg>

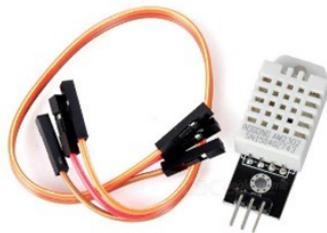


Imagen 2: Sensor DHT22 (Temperatura y humedad) (1)

<https://www.sigmaelectronica.net/wp-content/uploads/imported/DHT22.jpg>

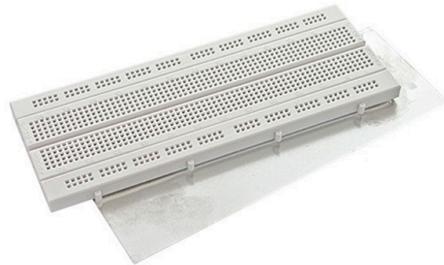


Imagen 4: Protoboard (1)
<https://www.avs.la/imagenes/productos/normales/b/BOARWB102-R/color/1.jpg>

Cosas que podrían complementar el sistema para su futuro desarrollo:

1. Para asegurarnos de que el funcionamiento del sistema sea el requerido implementamos algoritmos de detección de incendios. No obstante, se pueden incorporar algoritmos más sofisticados para analizar tanto los datos de temperatura como humedad, teniendo en cuenta valores como la variabilidad estacional y las condiciones climáticas de donde se implemente el sistema.

Lo mencionado anteriormente nos ayudaría a mejorar la precisión de la detección y reducir en la mayoría las falsas alarmas que se generen.

2. Comunicación inalámbrica de largo alcance, para mejorar la

comunicación se podría utilizar módulos de comunicación inalámbrica de largo alcance tales como LoRA o LoRaWAN, estos se encargan de ampliar la cobertura del sistema y permiten la implementación de esta en áreas remotas y sin acceso a redes telefónicas o Wi-Fi.

3. Integración con sistema de monitoreo existentes, para mejorar la compatibilidad del sistema, para que este pudiera integrarse con sistemas de monitoreo forestal existentes para generar una visión general de las condiciones ambientales y en específico la detección de incendios.

4. Alimentación solar, para hacerlo un sistema autosustentable y que en su mayoría no necesite de intervenciones en el sitio de instalación se puede implementar la energía solar para alimentar los nodos del sistema, haciendo de este sistema algo más amigable y sostenible en el ámbito ambiental.

5. Sensores adicionales, para mejorar el sistema puede darse la implementación de sensores de gases o cámaras para recopilar más datos sobre el territorio o sitio donde se encuentra implementado el sistema.

III. DIAGRAMAS

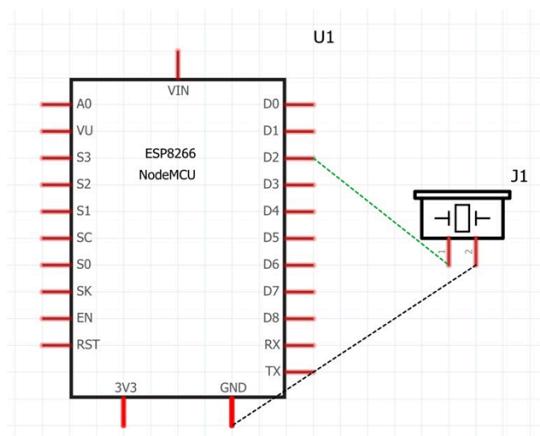


Imagen 5: Diagrama esquemático Buzzer-Nodo ESP8266

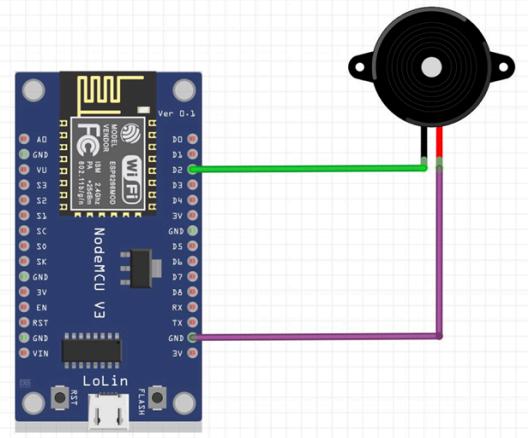


Imagen 6: Diagrama vista protoboard Buzzer-Nodo ESP8266

Funcionamiento del circuito No.1:

En las imágenes 5 y 6 el **buzzer** se activa mediante el pin D2 de la placa **NodeMCU**. Cuando este pin se establece en estado alto, el buzzer emite un sonido; cuando está en estado bajo, el buzzer está apagado. Esto dependerá del programa.

Conexiones esenciales del buzzer :

-Pino positivo: Se conecta al pin D2 de la placa NodeMCU para controlar el buzzer .

- Pino negativo: Se conecta al pin GND de la placa NodeMCU para completar el circuito y proporcionar una referencia de tierra.

Consideraciones adicionales:

- Podría llegar a ser necesaria una resistencia de 100, conectada entre el pin D2 y el pin positivo del buzzer . Aunque no es estrictamente necesaria, esta resistencia puede ayudar a proteger tanto el buzzer como la placa NodeMCU de picos de corriente.

- La variación en el tipo de buzzer utilizado puede afectar al sonido producido, así como la frecuencia y la duración del sonido programado en el código.

Estos casos si se decide cambiar el buzzer utilizado o la fuente de poder.

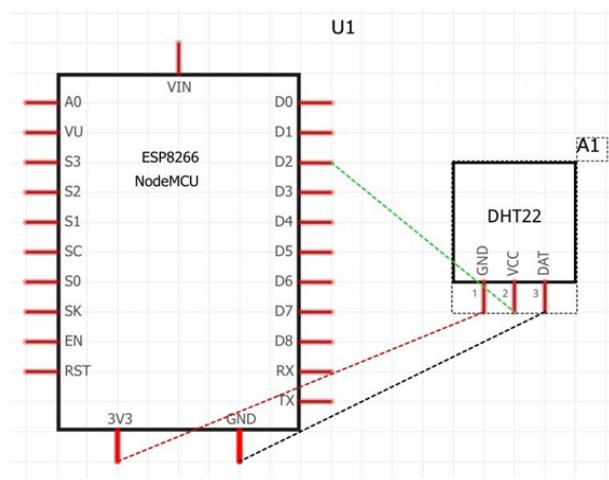


Imagen 7: Diagrama esquemático SensorTH-Nodo

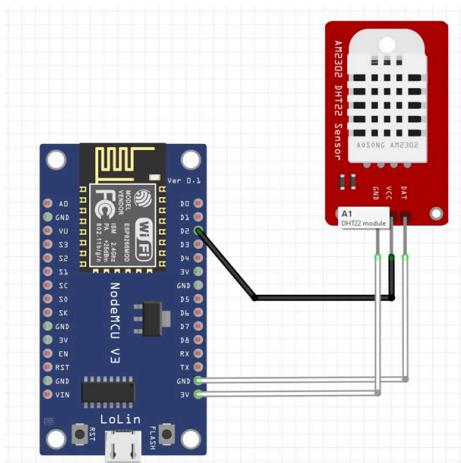


Imagen 8 :Diagrama vista protoboard SensorTH -Nodo ESP8266

Funcionamiento del circuito No.2:

En las *imágenes* 7 y 8 se muestra la conexión entre el **sensor DHT22** que mide la temperatura y humedad del aire, que está conectado y envía los datos al pin DAT de la **NodeMCU**. La placa NodeMCU procesa estos datos a través del microcontrolador **ESP8266**. Para que posteriormente, los datos serán enviados a un servidor web o una aplicación móvil para su visualización o análisis.

Conexiones esenciales del sensor DHT22:

DAT: Este pin transmite los datos de temperatura y humedad del sensor y se conecta al

pin D2 de la placa NodeMCU. Es crucial para la transmisión de datos entre el sensor y la placa.

VCC y GND: Proporcionan la alimentación necesaria para el sensor.

Conexiones utilizadas en las placas NodeMCU:

3V y GND: Estos pines suministran la energía necesaria para la NodeMCU y otros dispositivos conectados, asegurando su correcto funcionamiento.

Algunas de las conexiones no utilizadas en estos circuitos, en las placas NodeMCU fueron: D1 (A1), D3, D4 (S2), D5 (SC), D6 (SO), D7, RST: Estos pines están disponibles en la placa NodeMCU pero no se utilizan en esta configuración. Por ejemplo, D1 podría emplearse para conectar un sensor analógico, mientras que D4 podría usarse para un interruptor o un pulsador. Sin embargo, en esta aplicación particular, no son necesarios.

IV. PRESUPUESTO

Elementos	Cantidad	Precio
NodeMCU	2	33.320
Sensor	1	12.000
Buzzer	1	7.000
Protoboard	1	Gratis
TOTAL	5	52.200

Tabla 1. precios y cantidades de componentes

V. PROTOCOLO MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ligero ampliamente utilizado en el Internet de las Cosas (IoT). Fue ideado por IBM con el objetivo de proporcionar una solución rentable y confiable para la conexión de dispositivos. Se emplea principalmente para la comunicación de máquina a máquina (M2M).

Este protocolo, que opera en la capa de aplicación, se destaca por su eficiencia y versatilidad. Adopta una topología estrella, donde los mensajes se transmiten utilizando el modelo de publicación/suscripción. En esta estructura, el nodo

central, denominado broker, actúa como intermediario al que se conectan los clientes remotos. Estos clientes pueden ser dispositivos con sensores y actuadores, así como también centrales de monitoreo, interfaces gráficas, entre otros. [3].

- Características

- Transferencia de mensajes mediante colas de publicación/suscripción.
 - Comunicación de uno a muchos o muchos a muchos.
 - Distintos niveles de calidad de servicio.
 - Funcionalidad llamada LWT (Last will testament) que se utiliza para avisar si un nodo se desconecta inesperadamente
 - Conexión orientada a eventos asincrónicos. Un mensaje puede llegar en cualquier momento independientemente si se lo está esperando o no.
 - Protección opcional de los datos mediante SSL/TLS o bien con usuario-contraseña.
- Gran cantidad de brokers disponibles, muchos de ellos opensource.
- Clientes disponibles para la mayoría de los lenguajes de programación y para plataformas de hardware. [6]

- Arquitectura de MQTT

MQTT se ejecuta sobre TCP/IP, adoptando una topología de publicación/suscripción. Además, MQTT establece un componente central denominado bróker, responsable de administrar la red. El bróker, a su vez, funciona como un servidor que facilita la comunicación entre los clientes: recibe mensajes de unos y los transmite a otros. Los clientes no interactúan directamente entre sí, sino que se conectan al bróker. Cada cliente tiene la capacidad de ser tanto editor como suscriptor, o ambos, participando así en la publicación y recepción de mensajes según sus necesidades. [4]

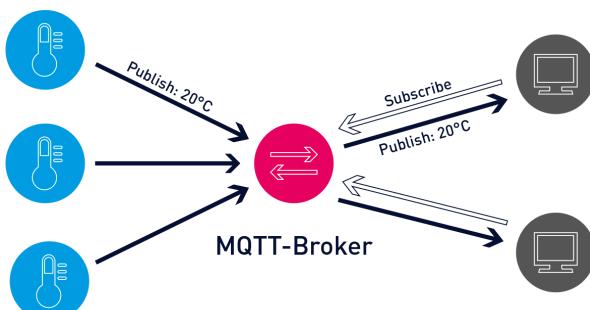


Imagen 9. Funcionamiento del MQTT publish/subscribe. Tomada de Paessler.

<https://www.paessler.com/es/it-explained/mqtt>

- Arquitectura de los mensajes o calidad de servicio (QoS)

MQTT, diseñado para ser simple y eficiente en el uso del ancho de banda, se presenta como una opción atractiva para diversas aplicaciones. Sin embargo, esta simplicidad implica que la interpretación de los mensajes recae completamente en el diseñador del sistema. Para abordar este desafío, MQTT ofrece distintos niveles de calidad de servicio (QoS), que determinan cómo se entrega cada mensaje. Es esencial especificar un nivel de QoS para cada tema MQTT, permitiendo a los diseñadores optimizar entre la eficiencia y la confiabilidad de la transmisión de datos.

Además, MQTT reduce sus transmisiones mediante un tamaño de mensaje pequeño y bien definido. Cada mensaje cuenta con un encabezado fijo de apenas 2 bytes, aunque se puede emplear un encabezado opcional, lo que incrementa el tamaño del mensaje. La carga útil del mensaje está limitada a únicamente 256 MB. Con tres niveles diferentes de calidad de servicio (QoS), los diseñadores de redes pueden elegir entre minimizar la transmisión de datos y maximizar la fiabilidad de acuerdo con las necesidades específicas de su aplicación.

- QoS 0 : ofrece la cantidad mínima de transmisión de datos y los mensajes se mandan sin tener en cuenta si llega o no, esto quiere decir que cada mensaje se entrega a un suscriptor una vez, sin confirmación, por lo cual no hay forma de saber si el mensaje le llegó a los subscriptores, a veces este mecanismo se llama lanzar y olvidar y puede sufrir pérdida de mensajes, por lo cual no se puede hacer retransmisión de mensajes[3].
- QoS 1: El procedimiento implica que el bróker inicialmente intenta entregar el mensaje al suscriptor y luego aguarda una confirmación de recepción. En caso de no recibir dicha confirmación dentro de un lapso de tiempo predeterminado, el mensaje se reenvía. Bajo esta metodología, el suscriptor podría llegar a recibir el mensaje en múltiples ocasiones si el bróker no recibe la confirmación de

recepción a tiempo. Este enfoque se conoce comúnmente como el método de "entrega al menos una vez".[6]

- QoS 2: se asegura que el mensaje llegue exactamente una vez. Eso incrementa la sobrecarga en la comunicación, pero es la mejor opción cuando la duplicación de un mensaje no es aceptable.[3]

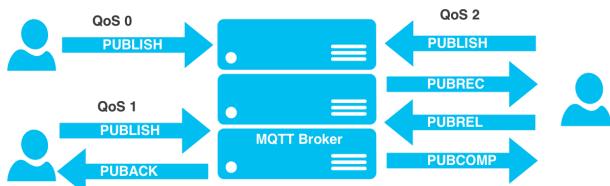


Imagen 10. Calidad de servicio QoS.<https://www.paessler.com/es/it-explained/mqtt>

- Temas/topics

Los mensajes en MQTT se publican como "temas", los cuales están organizados jerárquicamente utilizando el carácter "/" como delimitador. Esta estructura es similar a los directorios en un sistema de archivos de computadora, permitiendo una organización lógica y accesible. Por ejemplo, un tema podría ser "edificio1/planta5/sala1/raspberry2/temperatura", o "/edificio3/planta0/sala3/arduino4/ruido". Esta jerarquía facilita la creación de redes de clientes que intercambian datos de manera eficiente y ordenada. En la siguiente imagen se presenta un ejemplo de esta estructura de temas. [3]

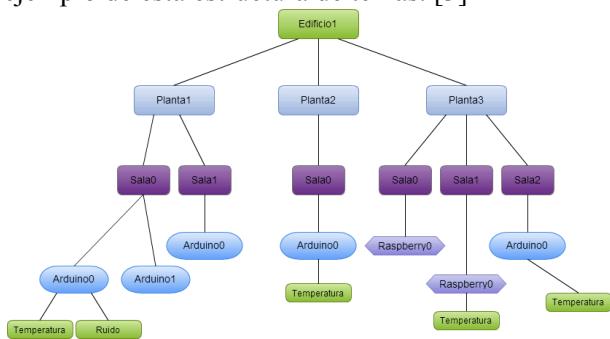


Imagen 11.Demostración de topic o de temas en MQTT.<https://www.paessler.com/es/it-explained/mqtt>

- Seguridad

Al diseñar e implementar MQTT, la seguridad fue una preocupación fundamental, especialmente

en el contexto de las aplicaciones de Internet de las Cosas (IoT). Se diseñó con el objetivo de garantizar una autenticación segura y ejecución, y ofrece soporte para cifrado mediante SSL. Durante este proceso, se consideraron cuidadosamente las siguientes consideraciones:

- Autenticación: MQTT ofrece opciones para autenticación mediante el uso de nombre de usuario y contraseña. Esto asegura que solo los clientes autorizados puedan conectarse y publicar o suscribirse a temas (topics).
- Nombre de usuario y contraseña: MQTT permite nombres de usuario y contraseñas para que un cliente establezca una conexión con un bróker. Lamentablemente, para mantener la claridad
- SSL/TLS : MQTT puede utilizar TLS (Transport Layer Security) o SSL (Secure Sockets Layer) para cifrar la comunicación entre clientes y el servidor MQTT. Esto garantiza la confidencialidad y la integridad de los datos durante la transmisión. [7]

MQTT cuenta con divisiones de los niveles de seguridad que son:

- Nivel de red: Para garantizar una conexión fiable, es fundamental contar con una red físicamente segura o utilizar una VPN para toda la comunicación entre clientes y agentes. Esta solución puede ser efectiva en aplicaciones de gateway, donde el gateway actúa como intermediario entre los dispositivos y la VPN.
- Nivel de Transporte: SSL/TLS se utiliza ampliamente para cifrar la comunicación en el nivel de transporte cuando se requiere confidencialidad. Gracias a este método seguro, los datos transmitidos no pueden ser interceptados ni leídos. Además, proporciona autenticación de certificados de cliente para verificar la identidad de ambas partes involucradas en la comunicación.
- Nivel de aplicación: El protocolo MQTT ofrece una variedad de mecanismos de seguridad en el nivel de aplicación. Estos incluyen identificación de cliente, identificador de cliente y credenciales de nombre de usuario/contraseña para autenticar los

dispositivos. Las propiedades de seguridad son proporcionadas por el propio protocolo. La autorización de los dispositivos para realizar ciertas acciones está definida por la implementación específica del agente. Además, en el nivel de aplicación, es posible utilizar la encriptación de la carga útil para garantizar la seguridad de la información transmitida sin depender exclusivamente de la encriptación de transporte. [7]

- Mensajes en MQTT

Para mantener el protocolo lo más liviano posible, se limita la comunicación a cuatro acciones principales:

- Publicar: Esta acción implica enviar un bloque de datos que contiene el mensaje que se desea transmitir. Los datos pueden variar según la implementación, desde indicaciones simples de encendido/apagado hasta lecturas de sensores como temperatura o presión. Si el tema al que se publica el mensaje no existe, el broker lo crea automáticamente.
- Suscribirse: Al suscribirse, un cliente se convierte en suscriptor de un tema específico. Puede suscribirse a temas individuales o mediante comodines, que permiten suscripciones a una rama completa de temas o parte de ella. El cliente envía un paquete SUBSCRIBE y recibe un paquete SUBACK como confirmación. Si hay mensajes retenidos para el tema al que se suscribe, el cliente los recibirá también.
- Ping: Un cliente puede enviar un ping al broker para mantener viva la conexión. El cliente envía un paquete PINGREQ y recibe un paquete PINGRESP como respuesta. Estos pings se utilizan para asegurarse de que la conexión permanezca activa y que la sesión TCP no se cierre inesperadamente debido a problemas de red.
- Desconectar: Cuando un suscriptor o editor ya no necesita comunicarse con el broker, envía un mensaje de desconexión (DISCONNECT). Esto

informa al broker que no se necesitan enviar más mensajes ni poner en cola mensajes para ese cliente. Al desconectarse correctamente, el cliente puede volver a conectarse en el futuro utilizando la misma identidad de cliente. Si un cliente se desconecta sin enviar un mensaje de desconexión, se envía su "última voluntad y testamento" a los suscriptores, que indica cómo deben manejar la desconexión. [4]

- Mensajes LWT

Cuando un cliente se conecta al broker puede definir un topic y un mensaje que se publicará automáticamente si se desconecta de forma inesperada. Cuando el temporizador keep alive del broker detecta que el cliente no ha enviado recientemente un topic o un mensaje PINGREQ (que se usa para mantener activa la conexión) se publica inmediatamente el mensaje LWT especificado por el cliente.

La función LWT se puede utilizar para crear notificaciones en aplicaciones que supervisan la actividad de los clientes. En la siguiente figura se observa un diagrama que demuestra cómo ocurre un mensaje LWT. [3]

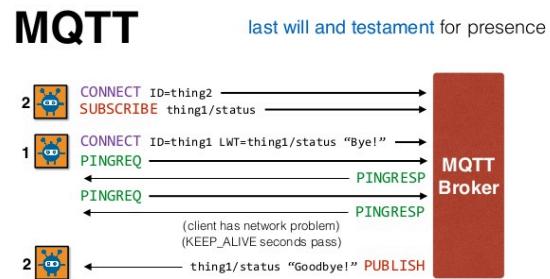


Imagen 12.Mensaje LWT.
<https://www.paessler.com/es/it-explained/mqtt>

- Ventajas

Desconexión entre clientes: Este modo de funcionamiento permite una mejor segmentación de las redes industriales, donde solo se requiere la conexión de los clientes con el broker, pero no entre ellos mismos.

Número y configuración de receptores transparentes para el productor de datos: En el ámbito industrial, la distribución de mensajes de

un productor a varios receptores de manera transparente es fundamental. Un mismo dato puede ser utilizado por varios sistemas receptores para diferentes funciones. La capacidad de cambiar los receptores de los mensajes de manera transparente para el productor del dato resulta ventajosa al integrar nuevos sistemas o actualizar los existentes. Esto permite realizar modificaciones sin intervenir en los dispositivos productores de los datos.

Datos jerarquizados: La posibilidad de realizar suscripciones a datos de manera jerarquizada facilita la introducción de nuevos productores de datos en un área determinada. Los datos pueden llegar a los receptores interesados en ellos sin necesidad de realizar modificaciones en los sistemas receptores existentes.[9]

- Caso de uso de MQTT

El transporte de mercancías mediante drones autónomos es una industria en rápido crecimiento que está transformando la forma en que las empresas mueven bienes valiosos. Por ejemplo, Matternet , una startup con sede en Menlo Park, California, está construyendo drones para reducir el tiempo que lleva transportar muestras médicas entre hospitales y laboratorios de pruebas.

Matternet emplea HiveMQ para asegurar un seguimiento confiable del estado de los vuelos en tiempo real. Durante los vuelos, los drones y las estaciones de aterrizaje envían mensajes MQTT a HiveMQ, que opera en la plataforma Matternet. Estos mensajes se transmiten en tiempo real, permitiendo utilizar la información de telemetría para proporcionar el estado del vuelo en tiempo real. La capacidad de HiveMQ para gestionar de manera fiable los niveles de calidad de servicio MQTT en múltiples nodos garantiza una entrega de mensajes confiable, especialmente crucial dado que los drones se comunican a través de una red LTE que puede ser poco confiable. El protocolo de publicación/suscripción de MQTT también habilita la comunicación bidireccional entre los drones y la plataforma en la nube, permitiendo que los operadores interactúen con los drones en vuelo.

Además del seguimiento de vuelos, Matternet aprovecha el marco de extensión de HiveMQ para integrar datos de control de tráfico aéreo en tiempo real en su plataforma. Esta integración directa facilita el intercambio de datos entre la flota de drones y las estaciones de aterrizaje. Matternet también ha desarrollado extensiones de HiveMQ para almacenar registros de auditoría de vuelos en una base de datos de series temporales. Esta

capacidad no solo ayuda a la empresa a cumplir con los requisitos de auditoría, sino que también respalda la capacidad de analizar datos históricos de vuelos para mejorar la operatividad y la eficiencia. [5].

- Broker MQTT
- Eclipse mosquitto

Es un bróker MQTT de código abierto que permite la implementación del protocolo MQTT. Además es un intermediario de mensajes liviano de fácil uso para dispositivos de baja potencia como Arduino, o Raspberry pi. [9]

Características:

- Admite MQTTv3.1, v3.1.1 y v5.0
- Utiliza pocos recursos aptos para sensores de baja potencia, dispositivos móviles, y microcontroladores.
- Posee un repositorio de vulnerabilidades detectadas a tomar en consideración
- De fácil uso e implementación disponible para multiplataformas.

En nuestro proyecto, implementaremos el protocolo Mosquitto debido a su capacidad para proporcionar mensajería ligera, especialmente diseñada para sistemas con recursos limitados. Su eficiente implementación del protocolo MQTT lo convierte en una elección ideal para aplicaciones que requieren comunicación entre dispositivos conectados a Internet.

El primer paso es la instalación de Mosquitto en un servidor o computadora, seguido de la configuración necesaria. En el caso de los ESP, se requiere configurarlos para publicar y suscribirse a temas específicos de MQTT.

Por ejemplo, el ESP1 actúa como publicador, encargado de medir la temperatura y la humedad. Se configura un cliente MQTT que publica estos datos en un tema específico. Mientras tanto, el segundo ESP se configura como suscriptor del mismo tema donde el ESP publica los datos, además de estar preparado para enviar notificaciones en caso de detectar un incendio forestal.

Se definen temas MQTT para los datos de temperatura y humedad, como "Sensores/humedad" y "Sensores/temperatura". Además, se establece un tema MQTT adicional, como "Peligro/sensación térmica", para las alertas

de incendios o la determinación de la sensación térmica.

Luego, se implementa la lógica de notificación y visualización en el segundo ESP, que actúa como suscriptor. Este dispositivo mostrará los datos de temperatura y humedad y, en caso de detectar un incendio, enviará una notificación y activará una alerta sonora.

Además, se debe tener en cuenta que el servidor MQTT (bróker) operará con la dirección IP 192.168.43.103. Java se utilizará tanto como publicador como suscriptor. El suscriptor estará suscrito al tema de temperatura y publicará en el tema de peligro. El tema de temperatura será de tipo float, enviado por el ESP publicador, mientras que el tema de peligro será un booleano enviado tanto por la aplicación Java al suscriptor ESP como al bot de Telegram.

- Mosca

Es un bróker MQTT utilizado para desarrollo de infraestructuras IoT basado en el protocolo MQTT. Mosca está basado en una librería de JavaScript de fácil uso e implementación

Sus características principales son:

- MQTTv 3.1 y v3.1.1
- QoS 0 y QoS 1
- Almacena paquetes fuera de línea para QoS1
- Rápido
- Se puede usar dentro de aplicaciones node.js

- EMQ

Es un agente para mensajería MQTT distribuido y altamente escalable. Está escrito en Erlang/OTP, de fácil uso e implementación en dispositivos x86/ARM ya sea con recursos limitados hasta en nubes privadas, públicas e híbridas..

Características:

- Soporta MQTTv3.1, v3.1.1 y v5.0
- Garantiza una integración fluida con clientes y herramientas MQTT
- Permite escribir complementos para admitir protocolos en la capa TCP y UDP
- Permite un millón de conexiones concurrentes.
- Garantiza una latencia en milisegundos
- Varios nodos pueden trabajar en conjunto garantizando confiabilidad y disponibilidad[9].

VI. TOPOLOGÍA DE LA RED

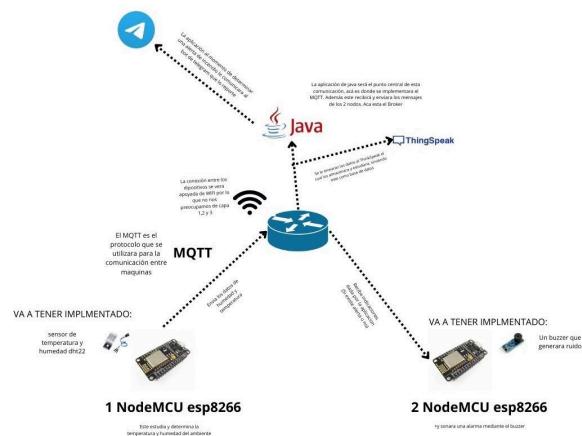


Imagen 13. Topología de la solución IoT implementada

Esta topología representa una arquitectura de sistema de Internet de las Cosas (IoT) que involucra diferentes componentes y protocolos de comunicación. El punto central es MQTT, que es el protocolo utilizado para la comunicación entre máquinas. Java actúa como el punto central de esta comunicación, donde su aplicación interactúa con el protocolo MQTT para enviar y recibir mensajes (acá se encuentra el broker). Hay dos NodeMCU esp8266, que son placas de desarrollo con sensores integrados. El primer NodeMCU tiene un sensor de temperatura y humedad (dht22), mientras que el segundo NodeMCU tiene un buzzer que genera ruido y una pantalla LCD para mostrar mensajes. La conexión entre los dispositivos se realiza a través de una red WiFi, utilizando el protocolo MQTT para la comunicación de datos. ThingSpeak es una plataforma de análisis de IoT que se menciona, lo que sugiere que los datos recopilados podrían enviarse a esta plataforma para su análisis y visualización

VII. MODIFICACIONES EN LA REALIZACIÓN E IMPLEMENTACIÓN DEL PROYECTO

- Principalmente hablando en términos de la configuración, se implementó un bot de Telegram que va a trabajar con Java.
- Se ha realizado una modificación en la topología de la red para optimizar el flujo de datos. En la nueva configuración, el módulo NodeMCU ESP8266 se encarga de recoger los datos de temperatura y transmitirlos directamente a la plataforma ThingSpeak. ThingSpeak almacena estos datos y permite su posterior análisis. Este proceso no solo facilita la monitorización en tiempo real de las condiciones

ambientales, sino que también proporciona una base de datos centralizada para nuestro proyecto. A diferencia de la configuración anterior, donde ThingSpeak iba a trabajar con Java

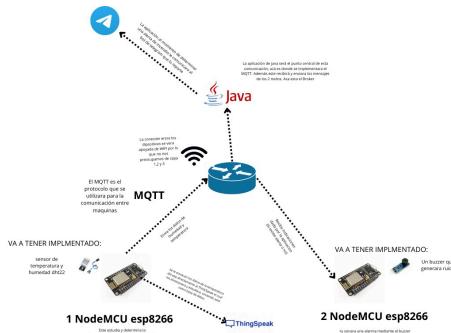


Imagen 14. Topología modificada de la solución IoT implementada

- Como se tenía propuesto inicialmente, ya no se tendrá en cuenta la sensación térmica. Además, para notificar a través del sonido del buzzer que la temperatura ha aumentado, este comenzará a sonar cuando la temperatura alcance los 35 grados, a diferencia de lo planeado anteriormente, donde iba a sonar cuando la temperatura llegará a los 40 grados.
- De acuerdo con lo planteado inicialmente, ya no se implementará una pantalla LCD para mostrar un mensaje de alerta sobre el aumento de temperatura y el riesgo de incendios forestales. En su lugar, se ha configurado para que, cuando esto ocurra, se active el sonido del buzzer y se envíe una notificación de alerta a través del bot de Telegram.

VIII. DOCUMENTACIÓN DEL PROTOTIPO



Imagen 15. primera parte del prototipo

En esta primera fase del prototipo, se ha utilizado una tarjeta NodeMCU ESP8266 conectada directamente con cables hembra-hembra a un sensor DHT, el cual mide la temperatura y la humedad. Esta configuración permite detectar las condiciones ambientales, fundamentales para nuestro proyecto. Además, el NodeMCU ESP8266 funciona como publicador gracias al protocolo MQTT, enviando el tópico de temperatura de manera eficiente.

El sensor DHT envía los datos de temperatura y humedad al NodeMCU ESP8266, el cual está programado para transmitir esta información a la plataforma ThingSpeak. ThingSpeak actúa como nuestra base de datos centralizada, encargándose de recopilar, almacenar y analizar los datos proporcionados por el sensor DHT.

Además de su función de almacenamiento, ThingSpeak ofrece herramientas avanzadas para la visualización y análisis de datos, lo que facilita el monitoreo en tiempo real de las condiciones ambientales. Esto es especialmente útil para detectar variaciones en la temperatura y humedad que puedan indicar condiciones peligrosas, como el riesgo de incendios forestales.

```

1 #include <ESP8266WiFi.h>
2 #include <DHT.h>
3 #include <PubSubClient.h>
4 #include <ThingSpeak.h>
5
6 #define DHTPIN 4
7 #define DHTTYPE DHT22
8
9
10 DHT dht(DHTPIN, DHTTYPE);
11 String ssid = "Joselito";
12 String password = "wachinchon";
13 const char* mqtt_server = "192.168.41.135";
14 const char* mqtt_topic_temp = "temperatura";
15 unsigned long channelID = 2545323;
16 const char* writeAPIkey = "RKBXX0Z1753WVH2";
17 WiFiClient espClient;
18 PubSubClient client(espClient);
19
20 byte cont = 0;
21 byte max_intents = 50;
22
23 void setup() {
24     // Inicia Serial
25     Serial.begin(115200);
26     Serial.println("\n");
27 }
28
29 // Conexion al servidor MQTT
30 client.setServer(mqtt_server, 1883);
31 ThingSpeak.begin(espClient);
32
33
34 void loop() {
35
36     // Intentamos reconectar al servidor MQTT si no estamos conectados
37     if (!client.connected()) {
38         reconnect();
39     }
40     client.loop();
41
42     // Leemos la temperatura y la sensación térmica
43     float hum = dht.readHumidity();
44     float temp = dht.readTemperature();
45     float hi = dht.computeHeatIndex(temp, hum);
46     String tempa= String(temp).c_str();
47     if (client.connected()) {
48         client.publish(mqtt_topic_temp, String(temp).c_str());
49     }
50
51     ThingSpeak.setField(1,temp);
52     Serial.print("Temperatura: ");
53     Serial.print(temp);
54     Serial.print(" °C, Humedad: ");
55     Serial.print(hum);
56     Serial.print(" %, Sensacion termica: ");
57     Serial.print(hi);
58 }
```

```

74 |     client.publish(mqtt_topic_temp, String(temp).c_str());
75 |
76 ThingSpeak.setField(1,temp);
77 Serial.print("Temperatura: ");
78 Serial.print(temp);
79 Serial.print(" °C, Humedad: ");
80 Serial.print(hume);
81 Serial.print(" %, Sensacion termica: ");
82 Serial.print(hi);
83 Serial.println(" °C");
84 ThingSpeak.writeFields(channelID,WriteAPIKey);
85 delay(5000);
86 }
87
88 // Función para reconnectar al servidor MQTT
89 void reconnect() {
90     while (!client.connected()) {
91         Serial.print("Intentando conexión MQTT...");
92         if (client.connect("ESP8266Client1")) { //
93             Serial.println("Conectado");
94         } else {
95             Serial.print("falló, rc=");
96             Serial.print(client.state());
97             Serial.println(" Intentaremos de nuevo en 5 segundos");
98             delay(5000);
99         }
100    }

```

Imagen 16. código del NodeMCU ESP8266 con el sensor DHT22.

En el código se evidencia el funcionamiento de un microcontrolador ESP8266. Su propósito es conectarse a una red WiFi, leer datos de un sensor DHT22 (que mide temperatura y humedad), publicar la temperatura en un servidor MQTT y enviar estos datos a ThingSpeak para su visualización y análisis. A continuación, se explica el funcionamiento del código en detalle:

Inclusión de Librerías:

- ESP8266WiFi.h: Maneja la conexión WiFi del ESP8266.
- DHT.h: Permite la interacción con el sensor DHT22.
- PubSubClient.h: Proporciona funciones para conectarse y comunicarse con un servidor MQTT.
- ThingSpeak.h: Facilita el envío de datos a la plataforma ThingSpeak.

Definición de Constantes y Variables:

- DHTPIN y DHTTYPE: Definen el pin al que está conectado el sensor DHT22 y el tipo de sensor.
- ssid y password: Credenciales de la red WiFi.
- mqtt_server y mqtt_topic_temp: Dirección IP del servidor MQTT y el tema en el que se publicará la temperatura.
- channelID y WriteAPIKey: Identificadores y claves para enviar datos a ThingSpeak.
- espClient: Cliente WiFi.
- client: Cliente MQTT.

- cont y max_intentos: Contadores para gestionar los intentos de conexión a la red WiFi.

Configuración Inicial en setup():

- Se intenta conectar a la red WiFi utilizando las credenciales proporcionadas. Si la conexión falla después de max_intentos intentos, se informa de un error.
- Se inicializa el sensor DHT22.
- Se configura el cliente MQTT para conectarse al servidor especificado.
- Se inicia la comunicación con ThingSpeak.

Bucle Principal loop():

- Se verifica y restablece la conexión MQTT si es necesario.
- Se leen los valores de temperatura y humedad del sensor DHT22.
- Se calcula la sensación térmica (índice de calor) basada en la temperatura y la humedad.
- Si está conectado al servidor MQTT, publica la temperatura en el tema especificado.
- Se actualizan los campos de ThingSpeak con los valores leídos.
- Se imprime en el monitor serial los valores de temperatura, humedad y sensación térmica.
- Se espera 5 segundos antes de repetir el proceso.

Función reconnect():

- Intenta conectar al servidor MQTT en caso de que la conexión se pierda. Sigue intentando hasta que la conexión se restablezca, con un retraso de 5 segundos entre intentos.

Función del Código:

- WiFi Connection: El código conecta el ESP8266 a una red WiFi usando las credenciales proporcionadas.
- Sensor Data Acquisition: Lee periódicamente la temperatura y la humedad del sensor DHT22.
- MQTT Communication: Publica los datos de temperatura en un servidor MQTT.
- ThingSpeak Communication: Envía los datos de temperatura a la plataforma

ThingSpeak para su almacenamiento y análisis.

- Serial Monitoring: Imprime los datos leídos y el estado de la conexión en el monitor serial.



Imagen 17. segunda parte del prototipo

En la segunda fase del prototipo, también se ha utilizado una tarjeta NodeMCU ESP8266, a la cual se ha conectado un buzzer pasivo mediante cables hembra-hembra. Este segundo ESP8266 actúa como suscriptor y se suscribe directamente al tópico de peligro mediante el protocolo MQTT.

La programación de este NodeMCU ESP8266 se ha realizado en Arduino. Su función principal es detectar si existe un peligro de incendio, basándose en la información publicada por el primer ESP8266. Cuando el primer ESP publica datos de temperatura que indican un riesgo potencial, el segundo ESP los recibe a través del tópico de peligro y activa el buzzer para emitir una alerta sonora.

Además, la suscripción al tópico de peligro permite al segundo ESP8266 reaccionar en tiempo real ante cualquier variación crítica en las condiciones ambientales detectadas por el sensor DHT del primer ESP. Esto asegura una respuesta inmediata y efectiva ante posibles riesgos de incendio, proporcionando una capa adicional de seguridad al sistema.

Esta integración entre los dos módulos NodeMCU ESP8266, utilizando el protocolo MQTT, optimiza la comunicación y coordinación entre los dispositivos, garantizando una monitorización constante y una reacción rápida ante situaciones de emergencia.

```

1 #include <PubSubClient.h>
2 #include <ESP8266WiFi.h>
3
4
5 String ssid = "Joshua";
6 String password = "wanchinchon";
7 const unsigned char pinSound = 4;
8
9 const char* mqtt_server = "192.168.41.135";
10
11 WiFiClient espClient;
12 PubSubClient client(espClient);
13
14 const char* mqtt_topic_bool = "peligro";
15
16 bool peligro;
17
18 byte cont = 0;
19 byte max_intentos = 50;
20 String messageString;
21
22 void callback(char* topic, byte* payload, unsigned int length) {
23     char message[length + 1];
24
25     memcpy(message, payload, length);
26     message[length] = '\0';
27
28     messageString = String(message);
29
30     if(strcmp(topic, mqtt_topic_bool) == 0){
31         if (messageString == "true") {
32             peligro = true;
33         } else if (messageString == "false") {
34             peligro = false;
35         }
36         if(peligro==true){
37             //suena misiquita de peligro
38             // c_dhail.c();
39             Serial.print("SE HA REPORTADO UN CAMBIO, HAY INCENDIO ");
40         }
41         else{
42             //suena misiquita normal
43             Serial.print("SE HA REPORTADO UN CAMBIO, TODO ESTA TRANQUILO ");
44             // c_starWars_c();
45         }
46     }
47
48 }
49
50 void reconnect() {
51
52     while (!client.connected()) {
53         Serial.print("Intentando conexión MQTT...");
54
55         if (client.connect("ESP8266Client2")) {
56
57             if (client.connect("Conectado")){
58
59                 client.subscribe(mqtt_topic_bool);
60             } else {
61                 Serial.print("Error, rcc");
62                 Serial.print(client.state());
63                 Serial.println(" Intentando de nuevo en 5 segundos");
64
65                 delay(5000);
66             }
67         }
68     }
69
70 void setup() {
71     Serial.begin(115200);
72
73     WiFi.begin(ssid, password);
74     while (WiFi.status() != WL_CONNECTED and cont < max_intentos) {
75         cont++;
76         delay(500);
77         Serial.println(".");
78     }
79     Serial.println("");
80     if (cont < max_intentos) {
81         Serial.println("*****");
82         Serial.print("Conectado a la red WiFi: ");
83
84         Serial.println(WiFi.SSID());
85         Serial.println(WiFi.localIP());
86         Serial.print("macAddress: ");
87         Serial.println(WiFi.macAddress());
88         Serial.println("*****");
89     } else {
90         Serial.println(".....");
91         Serial.println("Error de conexión");
92         Serial.println(".....");
93     }
94     // Configurar el callback para recibir mensajes
95     client.setServer(mqtt_server, 1883);
96     client.setCallback(callback);
97 }
98
99 void loop() {
100     if (!client.connected()) {
101         reconnect();
102     }
103     client.loop();
104     if(peligro==true){
105         tone(pinSound, 523);
106         Serial.println(" la temperatura esta alta, hay peligro de incendio");
107     } else{
108         noTone(pinSound);
109         Serial.println("No hay de que preocuparse, no hay peligro de incendio");
110     }
111 }
```

```

89 } else {
90     Serial.println("-----");
91     Serial.println("Error de conexión");
92     Serial.println("-----");
93 }
94 // Configurar el callback para recibir mensajes
95 client.setServer(mqtt_server, 1883);
96 client.setCallback(callback);
97 }
98
99 void loop() {
100 if (!client.connected()) {
101     reconnect();
102 }
103 client.loop();
104 if(peligro==true){
105     tone(pinSound, 523);
106     Serial.println(" la temperatura esta alta, hay peligro de incendio");
107 } else{
108     noTone(pinSound);
109     Serial.println("No hay de que preocuparse, no hay peligro de incendio");
110 }
111 delay(5000);
112 }
113 }
```

Imagen 18. código del NodeMCU ESP8266 conectado con el buzzer.

En este código se observa un programa para un microcontrolador ESP8266 que se conecta a una red WiFi y se suscribe a un tema en un servidor MQTT y realiza acciones basadas en los mensajes recibidos. En particular, el código está configurado para escuchar si hay peligro de incendio y activar una alarma sonora si se detecta peligro. Aquí se detalla el funcionamiento del código:

1. Inclusión de Librerías:

- PubSubClient.h: Proporciona funciones para conectarse y comunicarse con un servidor MQTT.
- ESP8266WiFi.h: Maneja la conexión WiFi del ESP8266.

2. Definición de Variables y Constantes:

- ssid y password: Credenciales de la red WiFi
- pinSound: Pin GPIO del ESP8266 al que está conectado el zumbador (buzzer).
- mqtt_server: Dirección IP del servidor MQTT.
- espClient: Cliente WiFi
- client: Cliente MQTT.
- mqtt_topic_bool: Tema MQTT al que el ESP8266 se suscribe para recibir mensajes.
- peligro: Variable booleana que indica si hay peligro (por ejemplo, peligro de incendio).
- cont y max_intentos: Contadores para gestionar los intentos de conexión a la red WiFi.
- messageString: Cadena que almacena el mensaje recibido del servidor MQTT.

3. Función `callback():`

- Esta función se llama cuando se recibe un mensaje en el tema al que está suscrito el ESP8266.

- Convierte el mensaje recibido (payload) en una cadena (`messageString`).
- Comprueba si el mensaje indica peligro (`true`) o no (`false`).
- Si hay peligro, imprime un mensaje de advertencia y activa la alarma sonora.
- Si no hay peligro, imprime un mensaje de tranquilidad y desactiva la alarma sonora.

4. Función `reconnect():`

- Intenta reconectar al servidor MQTT si la conexión se pierde.
- Suscribe al tema `mqtt_topic_bool` después de conectarse.
- Imprime mensajes de estado de conexión en el monitor serial.

5. Función `setup():`

- Conecta a la red WiFi usando las credenciales proporcionadas. Si la conexión falla después de `max_intentos` intentos, se informa de un error.
- Configura el cliente MQTT para conectarse al servidor y establece la función `callback` para manejar los mensajes entrantes.

6. Función `loop():`

- Comprueba si el cliente MQTT está conectado y, si no lo está, intenta reconnectar.
- Llama a `client.loop()` para mantener el cliente MQTT operativo.
- Activa o desactiva el zumbador según el valor de la variable `peligro`.
- Imprime el estado de peligro en el monitor serial.
- Espera 5 segundos antes de repetir el ciclo

7. Función del Código:

- WiFi Connection: El código conecta el ESP8266 a una red WiFi usando las credenciales proporcionadas.
- MQTT Communication: Suscribe al tema `peligro` en un servidor MQTT y maneja los mensajes recibidos.
- Alarm Activation: Activa un zumbador conectado al pin `pinSound` si se recibe un mensaje indicando peligro.

- SendMessageToTelegram(String messageText): Envía un mensaje a Telegram.

XIV. FLUJO DE TRABAJO

1. Inicio de la Aplicación: `App` es la clase principal que inicia la aplicación, configura el cliente MQTT y el bot de Telegram.
2. Conexión a MQTT: `App` crea una instancia de `MqttAsyncClient` y se conecta al broker MQTT.
3. Suscripción al Tópico: `App` se suscribe al tópico de temperatura.
4. Manejo de Mensajes: `MyCallback` recibe los mensajes de temperatura, analiza el valor y publica un mensaje de peligro en otro tópico si la temperatura es mayor a 35 grados.
5. **Notificación a Telegram: `MyCallback` utiliza `TelegramBot` para enviar mensajes de alerta a un chat de Telegram cuando se detecta una temperatura peligrosa.

XV. CONFIGURACIÓN Y EJECUCIÓN

1. Instalar dependencias:
 - Añadir las bibliotecas MQTT y Telegram Bots al proyecto.
2. Configurar el bot de Telegram:
 - Crear un bot de Telegram y obtener el token.
3. Ejecutar la Aplicación:
 - Compilar y ejecutar la clase `App`.

XVI. PROTOCOLOS DE PRUEBA

- Para evaluar el sistema de detección de incendios basado en el sensor DHT22 y el NodeMCU esp8266, se realizaron pruebas controladas de aumento gradual de temperatura. Inicialmente, se registrará la temperatura ambiente del área como línea base. Posteriormente, se acercará un mechero encendido al sensor DHT22, incrementando la temperatura de manera progresiva y controlada, sin exceder los 65°C, que es la máxima temperatura de prueba establecida. Durante este proceso, se monitorea y registra los valores de

temperatura reportados por el DHT22. Si la temperatura supera los 35°C, el sistema deberá activar una alerta de incendio mediante un buzzer. Una vez alcanzada la temperatura máxima o activada la alerta, se alejará el mechero y se permitirá que el sistema se enfrie hasta regresar a la temperatura ambiente inicial, registrando el tiempo de enfriamiento. Este ciclo se repetirá varias veces para obtener datos consistentes, analizando el tiempo de respuesta del sistema, la precisión de las lecturas del DHT22 y el comportamiento de las alertas.

XVII. DOCUMENTACIÓN DE PRUEBAS

En todo el proceso de pruebas se deben evidenciar diferentes cosas como la conexión al broker mqtt, la conexión a la red wifi, publicación y suscripción de mensajes, el bot de telegram, etc. Para esto todo se evidenciará mediante screenshots presentados por la terminal de java, los mensajes del bot de telegram y la terminal serial de arduino. Adicionalmente se mostrará registro de los datos tomados por la base de datos asignada la cual es thingspeak.

1. Conexión del nodo publicador a wifi:

Para esta prueba simplemente debemos prender el esp8266 conectándolo a un portatil mediante cable usb y tener habilitado la red wifi al cual se conectara, caso tal de que esta conexión salga bien se mostrará en pantalla la red wifi y la ip asignada al nodo. Caso tal que no se conecte se mostrará un error.

```
13:42:39.905 -> ....
13:42:46.602 -> ****
13:42:46.636 -> Conectado a la red WiFi: Joshua
13:42:46.636 -> IP: 192.168.41.238
13:42:46.636 -> macAddress: 48:3F:DA:6A:04:1F
13:42:46.636 -> ****
```

En este caso se conecto perfectamente por lo que ya tiene conexión a wifi y con esto también se puede dar paso a su conexión con el broker MQTT.

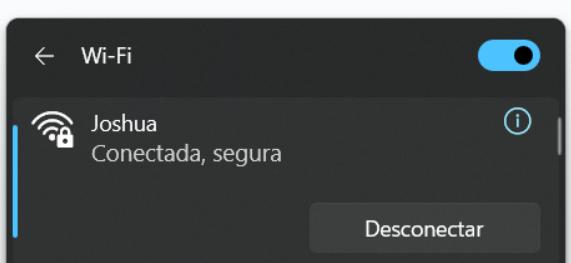
2. Conexión del nodo suscriptor al wifi:

En esta prueba es el mismo caso y la misma lógica.

```
13:45:12.929 -> ****
13:45:12.929 -> Conectado a la red WiFi: Joshua
13:45:12.929 -> IP: 192.168.41.13
13:45:12.929 -> macAddress: 98:F4:AB:F8:61:7A
13:45:12.929 -> ****
```

3. Conexión de los nodos al broker MQTT:

Para esto primero debemos asegurar que la máquina que aloja el protocolo MQTT de mosquitto esté conectada al wifi la cual los nodos también están conectados.



Aquí se comprueba que la máquina local la cual aloja el MQTT está conectada al wifi designado.

Adicionalmente comprobamos que la ip de la máquina que servirá como broker sea la misma que se le dijo a los nodos que se conectarán.

ip del broker:

```
Adaptador de LAN inalámbrica Wi-Fi:
Sufijo DNS específico para la conexión. . .
Vínculo: dirección IPv6 local. . . : fe80::87e8:93ce:292f:5eb4%8
Dirección IPv4. . . . . : 192.168.41.135
Máscara de subred . . . . . : 255.255.255.0
```

ip del broker dicho a los nodos en arduino:

```
const char* mqtt_server = "192.168.41.135";
```

Ya con esto solo se deben iniciar los nodos y se debe mostrar un mensaje que diga que se conectó al servidor MQTT.

```
13:51:57.862 -> **** Intentando conexión MQTT... Conectado
```

Aquí muestra el mensaje que evidentemente se conectó al servidor MQTT.

4. Prueba de detección de temperatura y publicación del topico.

Para esta prueba se trabaja solamente con el nodo 1 y aun no se evidencia la aplicación java, primero debemos definir el tópico que se publicará y mediante código adicionalmente programar la detección de temperatura.

```
float temp = dht.readTemperature();
String tempa= String(temp).c_str();
if (client.connected()) {
    client.publish(mqtt_topic_temp, string(temp).c_str());
}
ThingSpeak.setField1("temp");
const char* mqtt_topic_temp = "temperatura";
```

Adicionalmente se explicó que el tópico es temperatura.

De esta manera se evidencia 3 formas de que el tópico se publique de manera correcta.

```
13:55:23.537 -> Temperatura: 24.90 °C, Humedad: 56.30 %, Sensación térmica: 19.74 °C
13:55:29.027 -> Intentando conexión MQTT... Conectado
13:55:29.168 -> Temperatura: 24.80 °C, Humedad: 56.40 %, Sensación térmica: 19.63 °C
13:55:34.682 -> Intentando conexión MQTT... Conectado
13:55:34.715 -> Temperatura: 24.80 °C, Humedad: 56.40 %, Sensación térmica: 19.63 °C
13:55:40.486 -> Intentando conexión MQTT... Conectado
```

La primera es mediante la terminal serial , el cual nos muestra que evidentemente si está detectando los datos pero aun no sabemos si está publicando de manera correcta



La segunda es mediante la base de datos que está recopilando cada vez que se mandan los mensajes, acá también podemos evidenciar que efectivamente se detecta bien la temperatura sin embarazo aún debemos ver si un suscriptor está agarrando bien los datos y el tópico.

```
C:\Users\joshu\cd c:/program files/mosquitto
c:\Program Files\mosquitto>mosquitto_sub -d -t temperatura
Client null sending CONNACK (0)
Client null received SUBACK (0)
Client null received SUBSCRIBE (Mid: 1, Topic: temperatura, QoS: 0, Options: 0x00)
Client null received PUBACK (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
Subscribed (mid: 1): 0
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.30
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.39
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.30
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.30
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.30
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.30
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
24.30
Client null received PUBLISH (d0, q0, r0, m0, 'temperatura', ... (5 bytes))
```

Para esto es la misma máquina local nos suscribimos al tópico temperatura y evidenciamos que efectivamente se están recibiendo los datos que se están publicando.

De esta manera tenemos evidencia total de la publicación de mensajes correctamente.

5. Publicación y suscripción de la aplicación java, adicionalmente el envío de los datos de temperatura a telegram.

Para esto debemos en java programar la conexión al broker MQTT, suscribirse al tópico temperatura y luego publicar al tópico peligro que recibirá el segundo nodo sin embargo esa evidencia de publicación se hará en otra prueba, por último cada vez que se reciba un dato se enviará el mensaje al bot de telegram.

```
Cliente = new MqttAsyncClient("tcp://192.168.41.135:1883", UUID.randomUUID().toString());
```

Conexion al broker.

```
Cliente.subscribe("temperatura", 0);
```

Suscripción al tópico

```
byte[] payload = "false".getBytes();
MqttMessage mensajepel = new MqttMessage(payload);
App.Cliente.publish("peligro", mensajepel);
```

Publicación del topico

```
tele.sendMessageToTelegram("CUIDADO, HAY PELIGRO DE POSIBLE INCENDIO FORESTAL, LA TEMPERATURA ESTA EN ALTA");
```

Envío del mensaje al bot de telegram.

```
Bot de Telegram registrado
Se publico no hay peligro 24.8
Enviando mensaje de peligro a Telegram.
Mensaje de peligro en Telegram ACTUALMENTE,NO HAY PELIGRO DE INCENDIO, TU TRANQUILO YO NERVIOSO: 24.8
Se publico el mensaje
Se publico no hay peligro 24.8
Enviando mensaje de peligro a Telegram.
```

Acá se evidencia en la terminal que el bot se suscribe de manera correcta, agarra el dato que

recibe, crea el mensaje para el bot de telegram y publica el mensaje del tópico peligro que más adelante se evidenciará cómo lo recibe el segundo nodo.



Por último se adjunta la captura de pantalla del mensaje recibido a telegram.

6. Publicación del tópico peligro pero ahora en peligro.

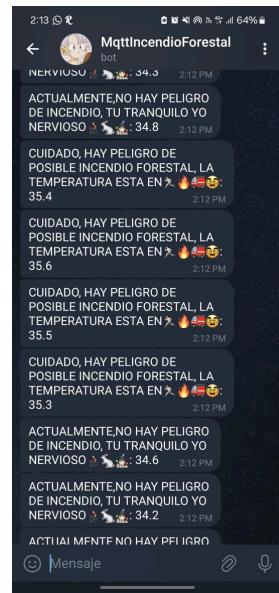
Para esta prueba se definen 2 if el cual diferenciarán si la temperatura es considerada un peligro de incendio o no, en la anterior prueba se evidenció que los mensajes no mostraban peligro de incendio, para esta prueba mediante un encendedor se subirá la temperatura con el fin de evidenciar cuales son los mensajes que muestran tanto en la serial del primer nodo, la aplicación de java y los mensajes de telegram.

Se publicó no hay peligro 34.8
Enviendo mensaje de peligro a Telegram.
Mensaje enviado a Telegram: ACTUALMENTE, NO HAY PELIGRO DE INCENDIO, TU TRANQUILO YO NERVIOSO. a: 34.8
Se publicó no hay peligro 35.4
Enviendo mensaje de peligro a Telegram.
Mensaje enviado a Telegram: CUIDADO, HAY PELIGRO DE POSIBLE INCENDIO FORESTAL, LA TEMPERATURA ESTA EN: 35.4
Se publicó el mapa de peligro
Se publicó hoy peligro 35.6
Enviendo mensaje de peligro a Telegram.

Acá se evidencia el cambio en el mensaje en la aplicación de java.

```
14:11:54.637 -> Intentando conexión MQTT...Conectado  
14:11:54.714 -> Temperatura: 34.80 °C, Humedad: 40.30 %, Sensación térmica: 30.25 °C  
14:12:00.387 -> Intentando conexión MQTT...Conectado  
14:12:00.526 -> Temperatura: 35.40 °C, Humedad: 49.30 %, Sensación térmica: 30.96 °C  
14:12:06.004 -> Intentando conexión MQTT...Conectado
```

Aquí se evidencia el cambio de temperatura en el nodo



Y por último acá se evidencia los mensajes recibidos en telegram y el cambio que existe cuando es menor y cuando es mayor a 35 grados.

7. Ahora se evidenciará la suscripción del nodo 2 al tópico peligro.

```

void callback(char* topic, byte* payload, unsigned int length) {
    char message[length + 1];
    memcpy(message, payload, length);
    message[length] = '\0';

    messageString = String(message);

    if(strcmp(topic, mqtt_topic_bool) == 0){
        if (messageString == "true") {
            peligro = true;
        } else if (messageString == "false") {
            peligro = false;
        }
        if(peligro==true){
            //suena musica de peligro
            // c_dball_();
            Serial.print("SE HA REPORTADO UN CAMBIO, HAY INCENDIO ");
        }
        else{
            //suena musica normal
            Serial.print("SE HA REPORTADO UN CAMBIO, TODO ESTA TRANQUILO ");
            // c_starWars_();
        }
    }
}

```

Para esto dentro del código del nodo se hace el recibimiento del tópico peligro y se hacen las diferentes sentencias para recibir este y como traducirlo (si es un peligro o no).

14:16:52.743 -> No hay de que preocuparse, no hay peligro de incendio
14:16:57.761 -> No hay de que preocuparse, no hay peligro de incendio
14:17:02.761 -> No hay de que preocuparse, no hay peligro de incendio

En este primer caso se evidencia que no hay peligro

Y de igual manera en java eso se muestra, ya que la aplicación java es la que está enviando los

mensajes.
Se publicó no hay peligro 34.8
Envío de Mensaje a Telégrafos,
Mensaje enviado a Telégrafos: ACTUALMENTE,NO HAY PELIGRO DE INCENDIO, TU TRANQUILO YO NERVICIOSO. C: 34.8
Se publicó el mensaje
Se publicó hay peligro 35.2
Envío de Mensaje a Telégrafos,
Mensaje enviado a Telégrafos.

Mensaje enviado a Telégrafos: CUIDADO, HAY PELIGRO DE POSIBLE INCENDIO FORESTAL, LA TEMPERATURA ESTA EN ALTA; 35.2
Se publicó el mensaje

14:28:37.950 - No hay de que preocuparse, no hay peligro de incendio
14:28:42.921 - No hay de que preocuparse, no hay peligro de incendio
14:28:47.926 - SE HA REPORTEADO UN CAMEJO, HAY INCENDIO la temperatura esta alta, hay peligro de incendio
14:28:52.924 - SE HA REPORTEADO UN CAMEJO, HAY INCENDIO la temperatura esta alta, hay peligro de incendio

Aca se evidencia el cambio de temperatura en el nodo 2 y la publicacion de java demostrando asi que hay conectividad total en el publicador, java y el suscriptor.

8. Prueba definitiva

A continuación se adjuntará un video mostrando evidencia de todo el proyecto y mostrando el cómo todo funciona a la par, la base de datos, la app java, los 2 nodos, la detección de temperatura a tiempo real y la acción del segundo nodo haciendo sonar el buzzer.

Link del video: [Video Demostrativo Del Proyecto](#)

Registro final del video (que no se alcanza a ver):



XVIII. REFERENCIAS

[1]«COMPROMISO DE CALIDAD Y SEGURIDAD DE BIC®», BIC. Accedido: 5 de mayo de 2024. [En línea]. Disponible en: https://eu.bic.com/es-es/lighters/bicr-lighters-com_mitments

[2]J. Soto, «Índice de Sensación Térmica “Heat Index”», clima.bio - alter technica ING. Accedido: 5 de mayo de 2024. [En línea]. Disponible en: <https://clima.bio/indice-de-sensacion-termica-heat-index/>

[3]“Introducción a MQTT”. Goto IoT. Accedido el 6 de mayo de 2024. [En línea]. Disponible: https://www.gotiot.com/pages/articles/mqtt_intro_index.html

[4] “MQTT”. paessler. Accedido el 6 de mayo de 2024. [En línea]. Disponible: <https://www.paessler.com/es/it-explained/mqtt>

[5]“Matternet Revolutionizes Drone Transportation with HiveMQ”. HiveMQ – The Most Trusted MQTT platform to Transform Your Business. Accedido el 9 de mayo de 2024. [En línea]. Disponible: <https://www.hivemq.com/case-studies/matternet/>

[6]M. C. Collado, “Monitorización de sensores con arduino utilizando el protocolo MQTT”, trabajo de grado, UPC Esc. Ing. Telecomunicacion Aeroesp. Castelldefels, 2019. Accedido el 7 de mayo de

2024. [En línea]. Disponible: <https://upcommons.upc.edu/bitstream/handle/2117/134193/memoria.pdf?sequence=1&isAllowed=y>

[7]“Fundamentos de seguridad del MQTT en la automatización industrial”. industrialshields. Accedido el 5 de mayo de 2024. [En línea]. Disponible: https://www.industrialshields.com/es_ES/blog/blog-industrial-open-source-1/fundamentos-de-seguridad-del-mqtt-en-la-automatizacion-industrial-235

[8]E. Buetas Sanjuan, “MQTT-SCACAUTH: Esquema de seguridad para el protocolo MQTT y su uso en el entorno del IIoT”, trabajo de grado, 2019. Accedido el 6 de mayo de 2024. [En línea]. Disponible:

http://e-spacio.uned.es/fez/eserv/tesisuned:ED-Pg-IngSisCon-Ebuetas/BUETAS_SANJUAN_Eduardo_Tesis.pdf

[9]R. X. AMAGUAYA RAMOS, “ANÁLISIS COMPARATIVO A NIVEL TRANSACCIONAL DE BROKERS MQTT (MOSQUITTO, MOSCA Y EMQ) CON RESPECTO A LA DISPONIBILIDAD EN INFRAESTRUCTURAS IoT ANTE ATAQUES DDoS.”, trabajo de grado, ESC. POLITEC. NAC., Quito, 2020. Accedido el 8 de mayo de 2024. [En línea]. Disponible: <https://bibdigital.epn.edu.ec/bitstream/15000/21374/1/CD%2010446.pdf>