Escuela Superior Politécnica Del Litoral

Facultad De Ingeniería En Electricidad Y Computación

Proyecto Parcial | Sistemas Operativos | PAO II 2025

Proyecto: Tiny_IoT

Presentado por:

Sebastian Holguin Vargas - sebhvarg@espol.edu.ec

Profesor:

Msc. Ronald Criollo Bonilla

Guayaquil – Ecuador

2025

Contenido

Introducción

Este proyecto implementa un sistema IoT distribuido que simula la recolección de métricas (temperatura y humedad) de múltiples sensores y su transmisión a través de una arquitectura de Gateway y Broker hasta los suscriptores. Utilizando un protocolo ligero de MQTT desarrollado en C. [1]

Descripción del proyecto

La solución de protocolo ligero de MQTT se divide en los siguientes componentes:

- **Publisher:** Los Publishers son nodos que obtienen datos del sensor de temperatura ESP32 u otros sensores de temperatura de húmedad. Estos datos se envían al Gateway al cual el Publisher se encuentra enviando información.
- **Gateway:** Recibe datos crudos, los procesa y los publica en el Broker. Es el intermediario entre los Publishers y el Broker. Agrega un identificador del Gateway al topic.
- **Broker:** Gestiona las suscripciones y distribuye los mensajes a los clientes interesados. Almacena los mensajes y entrega los mensajes al suscriptor de acuerdo al tópico suscrito.
- **Subscriber:** Se conecta al Broker y recibe actualizaciones de los temas suscritos.

Cuando se crean los publishers estos son asignados a un gateway cada Gateway soporta 3 publishers. Es decir, si se crean 5 publishers 3 son asignados al Gateway1 y los restantes al Gateway 2.

Procesos (Fork) y Hilos (Threads): Se emplean para lograr concurrencia y paralelismo. El uso de fork permite crear procesos independientes (como los nodos sensores) que poseen su propio espacio de memoria, garantizando aislamiento de fallos; si un nodo colapsa, no afecta a los demás. Los hilos (pthreads) se utilizan dentro del Broker y Gateways para manejar múltiples conexiones simultáneas de manera eficiente (bajo consumo de recursos) compartiendo el mismo espacio de memoria para el acceso rápido a estructuras comunes.

IPC (Inter-Process Communication): Dado que los componentes (Publishers, Broker, Subscribers) operan como entidades independientes, se requiere un mecanismo de comunicación. Se eligieron Sockets TCP/IP porque proporcionan un flujo de datos fiable y ordenado, esencial para la integridad de las métricas. Además, permiten la distribución real del sistema, posibilitando que los nodos se ejecuten en diferentes máquinas o microcontroladores (como el ESP32) a través de la red.

Mutex y Semáforos: En un entorno concurrente donde múltiples hilos intentan acceder a recursos compartidos (como la lista de suscriptores o el buffer de mensajes del Broker), es crítico evitar condiciones de carrera. Los Mutex (pthread_mutex) garantizan la exclusión mutua, asegurando que solo un hilo pueda modificar estas estructuras a la vez, manteniendo la consistencia de los datos y la estabilidad del servidor.

Diagramas de solución

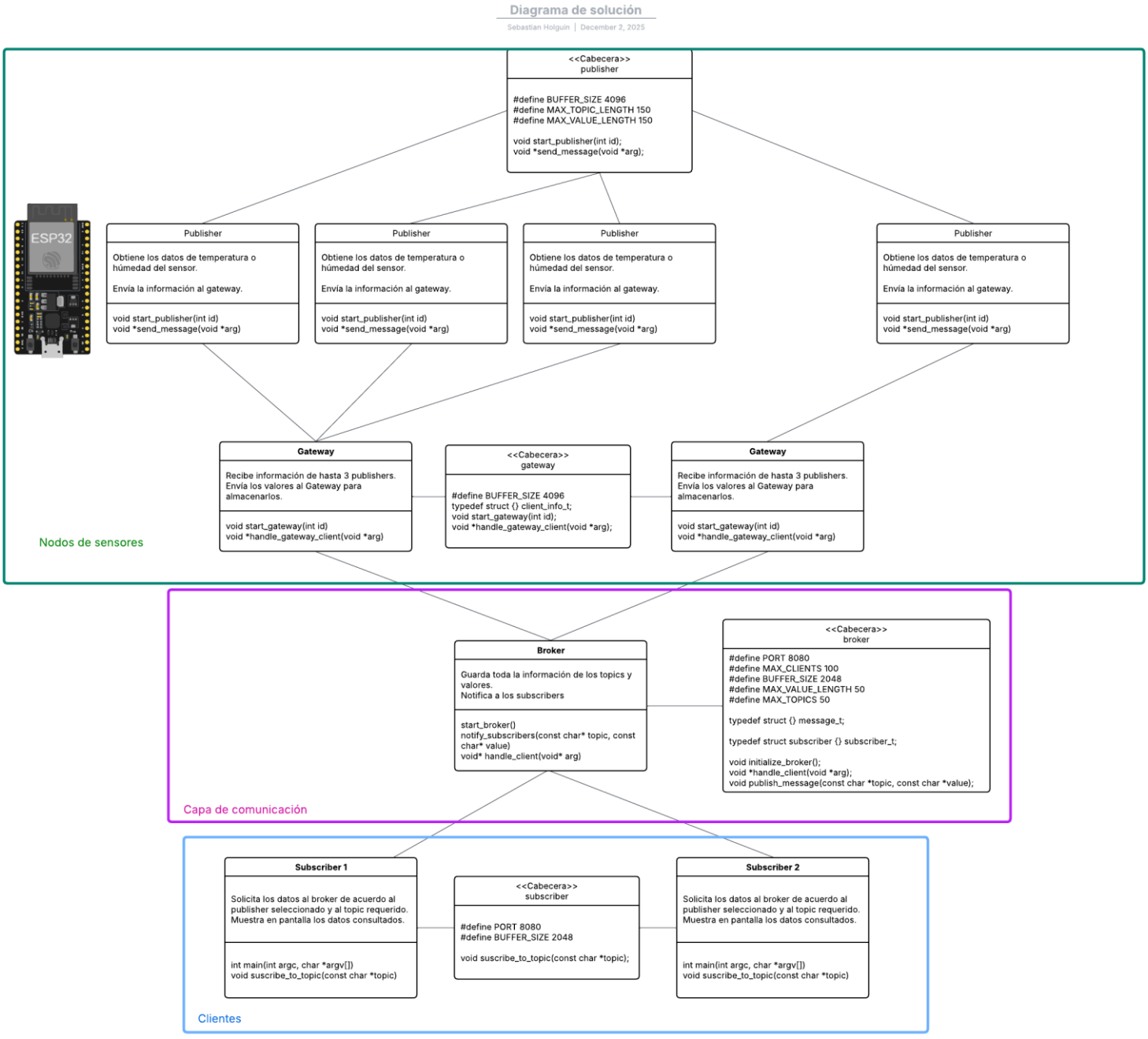


Diagrama del código de solución.

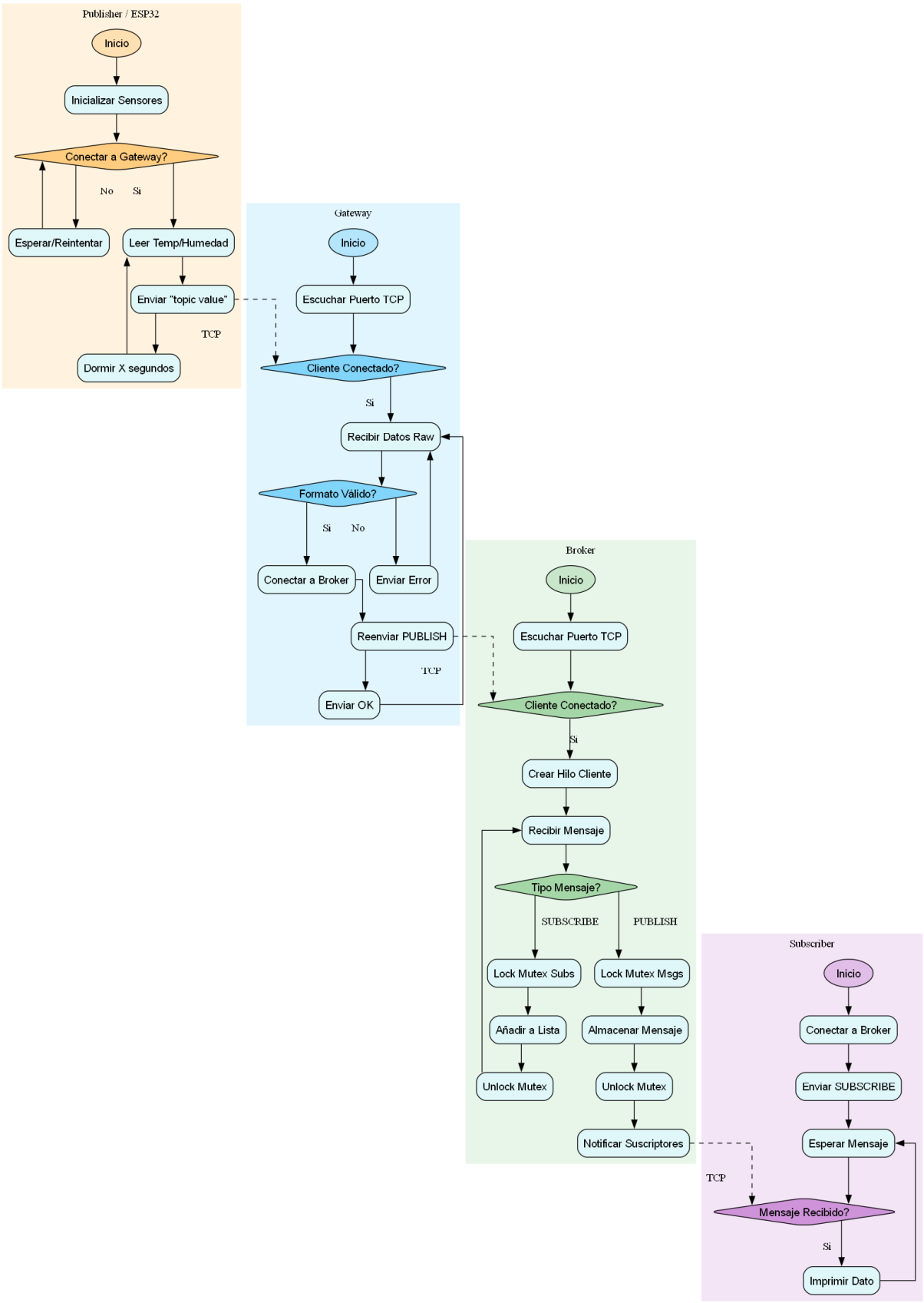


Diagrama de flujo de solución.

Código

Estructura del proyecto

```
|-- broker.c            # Implementación del Broker central
|-- broker.h            # Cabeceras del Broker
|-- channel.c           # Orquestador principal (crea procesos/hilos)
|-- esp32_main.ino      # Código para el nodo ESP32 (FreeRTOS)
|-- gateway.c           # Implementación del Gateway
|-- gateway.h           # Cabeceras del Gateway
|-- publisher.c         # Implementación de Publishers simulados
|-- publisher.h         # Cabeceras del Publisher
|-- subscriber.c        # Implementación de Subscriber
|-- subscriber.h        # Cabeceras del Subscriber
|-- Makefile            # Archivo de compilación
```

Explicación de código

- **Cabecera de Broker – broker.h**

```
You, yesterday | 1 author (You)
#ifndef BROKER_H
#define BROKER_H
#include <pthread.h>

#define PORT 8080
#define MAX_CLIENTS 100
#define BUFFER_SIZE 2048
#define MAX_VALUE_LENGTH 50
#define MAX_TOPICS 50

typedef struct {
    char topic[MAX_TOPICS];
    char value[MAX_VALUE_LENGTH];
} message_t;

typedef struct subscriber {
    int socket;
    char topics[MAX_TOPICS];

    struct subscriber *next;
} subscriber_t;

void initialize_broker();
void *handle_client(void *arg);
void publish_message(const char *topic, const char *value);

#endif // BROKER_H      You, yesterday • Headers …
```

En este archivo de cabecera que define las constantes, estructuras de datos (como subscriber_t y message_t) y prototipos de funciones utilizadas por el Broker.

- **broker.c**

```c
void initialize_broker() {
    int server_socket, new_socket;
    struct sockaddr_in server_addr;
    int addr_len = sizeof(server_addr);
    pthread_t thread_id;
    pthread_mutex_init(&msg_mutex, NULL);
    pthread_mutex_init(&subscribers_mutex, NULL);

    // Crear el socket del servidor
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Creacion del socket fallida");
        exit(EXIT_FAILURE);
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);
    // Enlazar el socket
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Fallo en el bind");
        close(server_socket);
        exit(EXIT_FAILURE);
    }
    // Escuchar conexiones entrantes
    if (listen(server_socket, MAX_CLIENTS) < 0) {
        perror("Fallo en el listen");
        close(server_socket);
        exit(EXIT_FAILURE);
    }
    printf("Broker escuchando en el puerto %d\n", PORT);
    while (1)
    {
        if ((new_socket = accept(server_socket, (struct sockaddr *)&server_addr, (socklen_t*)&addr_len))<0) {
            perror("Fallo en el accept");
            continue;
        }
        int *client_socket = malloc(sizeof(int));
        *client_socket = new_socket;

        if (pthread_create(&thread_id, NULL, handle_client, (void *)client_socket) != 0) {
            perror("Fallo en la creacion del hilo");
            free(client_socket);
        }
        pthread_detach(thread_id); // Desvincular el hilo para que libere recursos al terminar
    }

}
```

Implementa el servidor central (Broker) que gestiona las suscripciones de los clientes y distribuye los mensajes publicados. Se utilizaron hilos POSIX (pthread) para manejar la concurrencia.

```c
void* handle_client(void *arg) {
    int client_socket = *((int *)arg);
    free(arg);
    char buffer[BUFFER_SIZE];
    char topic[MAX_TOPICS];
    char value[MAX_VALUE_LENGTH];

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        int bytes_read = recv(client_socket, buffer, BUFFER_SIZE - 1, 0);
        if (bytes_read <= 0) {
            printf("Cliente desconectado\n");
            close(client_socket);
            return NULL;
        }
        buffer[bytes_read] = '\0';

        if (strncmp(buffer, "PUBLISH", 7) == 0) {
            sscanf(buffer + 8, "%s %s", topic, value);
            publish_message(topic, value);
        } else if (strncmp(buffer, "SUBSCRIBE", 9) == 0) {
            sscanf(buffer + 10, "%s", topic);
            subscriber_t *new_subscriber = malloc(sizeof(subscriber_t));
            new_subscriber->socket = client_socket;
            strncpy(new_subscriber->topics, topic, MAX_TOPICS);
            new_subscriber->next = NULL;

            pthread_mutex_lock(&subscribers_mutex);
            new_subscriber->next = subscribers_head;
            subscribers_head = new_subscriber;
            pthread_mutex_unlock(&subscribers_mutex);

            printf("Cliente suscrito al tema: %s\n", topic);
        }
    }
    close(client_socket);
    return NULL;
}
```

Para garantizar la integridad de los datos compartidos en el Broker (lista de suscriptores y buffer de mensajes), se utilizaron Mutex (pthread_mutex_t).

- **Cabecera Publisher -publisher.h**

```c
#ifndef PUBLISHER_H
#define PUBLISHER_H

#define PUBLISHER_BUFFER_SIZE 4096
#define MAX_TOPICS 50
#define MAX_VALUE_LENGTH 50
#define MAX_CLIENTS 100

typedef struct {
    int publisher_id;
    int gateway_port;
} publisher_config_t;

void *send_messages(void *arg);

#endif // PUBLISHER_H
```

Archivo de cabecera que define la configuración y las estructuras necesarias para los publicadores simulados.

- **publisher.c**

```c
void *send_messages(void *arg) {
    publisher_config_t *config = (publisher_config_t *)arg;
    int publisher_id = config->publisher_id;
    int gateway_port = config->gateway_port;
    free(config);

    int gateway_socket;
    struct sockaddr_in gateway_addr;
    char buffer[PUBLISHER_BUFFER_SIZE];
    srand(time(NULL) + publisher_id); // Semilla para valores aleatorios

    printf("Publisher %d iniciado, conectando al Gateway en puerto %d\n", publisher_id, gateway_port);

    // Conectar al gateway
    while (1) {
        gateway_socket = socket(AF_INET, SOCK_STREAM, 0);
        if (gateway_socket < 0) {
            perror("Error creando socket");
            sleep(2);
            continue;
        }

        gateway_addr.sin_family = AF_INET;
        gateway_addr.sin_port = htons(gateway_port);
        inet_pton(AF_INET, "127.0.0.1", &gateway_addr.sin_addr);

        if (connect(gateway_socket, (struct sockaddr *)&gateway_addr, sizeof(gateway_addr)) < 0) {
            printf("Publisher %d: esperando al Gateway en puerto %d...\n", publisher_id, gateway_port);
            close(gateway_socket);
            sleep(2);
            continue;
        }

        printf("Publisher %d: conectado al Gateway en puerto %d\n", publisher_id, gateway_port);
        break;
    }
```

Simula nodos publicadores que generan datos aleatorios de temperatura y humedad. Estos nodos se conectan a un Gateway asignado para transmitir su información. Los publishers generan datos aleatorios para simular sensores de temperatura y humedad. Estos datos se envían en formato de texto plano topic value.

```c
void *send_messages(void *arg) {
    while (1) {
        // Simular variación de temperatura
        temperature = 22 + (rand() % 10);
        snprintf(buffer, sizeof(buffer), "publisher%d/temperature %d°C\n", publisher_id, temperature);

        if (send(gateway_socket, buffer, strlen(buffer), 0) < 0) {
            printf("Publisher %d: Error enviando temperatura, reconectando...\n", publisher_id);
            close(gateway_socket);
            sleep(2);
            // Intentar reconectar
            gateway_socket = socket(AF_INET, SOCK_STREAM, 0);
            gateway_addr.sin_family = AF_INET;
            gateway_addr.sin_port = htons(gateway_port);
            inet_pton(AF_INET, "127.0.0.1", &gateway_addr.sin_addr);
            connect(gateway_socket, (struct sockaddr *)&gateway_addr, sizeof(gateway_addr));
            continue;
        }
        printf("Publisher %d TX: %s", publisher_id, buffer);
        sleep(1);

        // Simular variación de humedad
        humidity = 45 + (rand() % 20);
        snprintf(buffer, sizeof(buffer), "publisher%d/humidity %d%%\n", publisher_id, humidity);

        if (send(gateway_socket, buffer, strlen(buffer), 0) < 0) {
            printf("Publisher %d: Error enviando humedad, reconectando...\n", publisher_id);
            close(gateway_socket);
            sleep(2);
            continue;
        }
        printf("Publisher %d TX: %s", publisher_id, buffer);
        sleep(4); // Esperar antes del siguiente ciclo
    }

    close(gateway_socket);
    return NULL;
}
```

La comunicación entre procesos (o hilos en este caso, simulando nodos distribuidos) se realiza mediante Sockets TCP/IP (AF_INET, SOCK_STREAM).

- **Cabecera Gateway – Gateway.h**

```c
You, yesterday | 1 author (You)
#ifndef GATEWAY_H
#define GATEWAY_H

#define GATEWAY_BUFFER_SIZE 1024

typedef struct {
    int id;
    int client_socket;
} client_info_t;

void initialize_gateway(int port);
void *handle_client_connection(void *arg);

#endif // GATEWAY_H        You, yesterday • Headers …
```

Archivo de cabecera para el Gateway, donde se definen las configuraciones y declaraciones de funciones necesarias para su funcionamiento.

- **gateway.c**

```c
void initialize_gateway(int port) {
    int server_socket, new_socket;
    struct sockaddr_in server_addr;
    int addr_len = sizeof(server_addr);
    pthread_t thread_id;
    // Crear el socket del servidor
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Creacion del socket fallida");
        exit(EXIT_FAILURE);
    }
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(port);

    // Enlazar el socket
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Fallo en el bind");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    // Escuchar conexiones entrantes
    if (listen(server_socket, MAX_CLIENTS) < 0) {
        perror("Fallo en el listen");
        close(server_socket);
        exit(EXIT_FAILURE);
    }
    printf("Gateway TCP escuchando en el puerto %d\n", port);

    while (1)
    {
        if ((new_socket = accept(server_socket, (struct sockaddr *)&server_addr, (socklen_t*)&addr_len))<0) {
            perror("Fallo en el accept");
            continue;
        }
        client_info_t *client_info = malloc(sizeof(client_info_t));
        client_info->client_socket = new_socket;
        client_info->id = new_socket; // Usar el socket como ID temporal

        if (pthread_create(&thread_id, NULL, handle_client_connection, (void *)client_info) != 0) {
            perror("Fallo en la creacion del hilo");
            free(client_info);
        }
        pthread_detach(thread_id); // Desvincular el hilo para que libere recursos al terminar
    }
}
```

Implementa el Gateway intermedio que recibe los datos crudos de los sensores (Publishers/ESP32). Procesa estos mensajes y los reenvía al Broker utilizando el protocolo definido.

```c
void *handle_client_connection(void *arg) {
    client_info_t *client_info = (client_info_t *)arg;
    int client_socket = client_info->client_socket;
    char buffer[GATEWAY_BUFFER_SIZE];
    char topic[MAX_TOPICS];
    char value[MAX_VALUE_LENGTH];
    ssize_t read_size;

    printf("Cliente TCP conectado: ID %d\n", client_info->id);

    // Leer datos TCP raw (el ESP32 puede enviar múltiples líneas)
    while ((read_size = recv(client_socket, buffer, GATEWAY_BUFFER_SIZE - 1, 0)) > 0) {
        buffer[read_size] = '\0';

        // Procesar cada línea recibida
        char *line = strtok(buffer, "\n");
        while (line != NULL) {
            // Eliminar carriage return si existe
            char *cr = strchr(line, '\r');
            if (cr) *cr = '\0';

            // Parsear el formato: "topic value"
            if (sscanf(line, "%s %[^\n]", topic, value) == 2) {
                printf("Mensaje TCP recibido del cliente %d: Topic: %s, Value: %s\n",
                       client_info->id, topic, value);

                // Publicar el mensaje en el broker
                publish_message(topic, value);

                // Enviar respuesta TCP simple
                const char *response = "OK\n";
                send(client_socket, response, strlen(response), 0);

                // Notificar al broker
                int broker_socket = socket(AF_INET, SOCK_STREAM, 0);
                struct sockaddr_in broker_addr;
                broker_addr.sin_family = AF_INET;
                broker_addr.sin_port = htons(PORT);
                inet_pton(AF_INET, "127.0.0.1", &broker_addr.sin_addr);

                if (connect(broker_socket, (struct sockaddr *)&broker_addr, sizeof(broker_addr)) < 0) {
                    perror("Fallo en la conexion al broker");
                    close(broker_socket);
                } else {
                    char broker_msg[GATEWAY_BUFFER_SIZE];
                    snprintf(broker_msg, sizeof(broker_msg), "PUBLISH %s %s", topic, value);
                    send(broker_socket, broker_msg, strlen(broker_msg), 0);
                    close(broker_socket);
                }
            } else if (strlen(line) > 0) {
                printf("Formato TCP invalido del cliente %d: %s\n", client_info->id, line);
                const char *response = "ERROR\n";
                send(client_socket, response, strlen(response), 0);
            }

            line = strtok(NULL, "\n");
        }
    }
```

- **Cabecera Subscriber – subscriber.h**

```c
#ifndef SUBSCRIBER_H
#define SUBSCRIBER_H

#define PORT 8080
#define BUFFER_SIZE 2048

void suscribe_to_topic(const char *topic);

#endif // SUBSCRIBER_H
```

Archivo de cabecera para el suscriptor, definiendo constantes y prototipos de funciones para la conexión y recepción de mensajes.

- **subscriber.c**

```c
void suscribe_to_topic(const char *topic) {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[BUFFER_SIZE] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Error al crear socket \n");
        return;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convertir direcciones IPv4 e IPv6 de texto a binario
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\nDireccion invalida\n");
        return;
    }
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("\nConexion fallida \n");
        return;
    }

    char subscribe_msg[BUFFER_SIZE];
    snprintf(subscribe_msg, sizeof(subscribe_msg), "SUBSCRIBE %s", topic);
    send(sock , subscribe_msg , strlen(subscribe_msg) , 0 );
    printf("Suscrito al topic: %s\n", topic);

    while (1) {
        int valread = read( sock , buffer, BUFFER_SIZE);
        if (valread > 0) {
            buffer[valread] = '\0';
            printf("Mensaje recibido en el tema %s: %s\n", topic, buffer);
        } else if (valread == 0) {
            printf("Servidor cerro la conexion\n");
            break;
        } else {
            perror("Error de lectura");
            break;
        }
    }
    close(sock);
}
```

Implementa el cliente suscriptor que se conecta al Broker. Permite suscribirse a temas específicos y permanece a la espera de recibir notificaciones sobre dichos temas.

- **channel.c**

```c
void initialize_broker();
void initialize_gateway(int port);
void start_publisher(int port);

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Uso: %s <num_publishers>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    int num_p = atoi(argv[1]);

    pthread_t broker_thread;
    if (pthread_create(&broker_thread, NULL, (void *)initialize_broker, NULL) != 0) {
        perror("Fallo en la creacion del hilo del broker");
        exit(EXIT_FAILURE);
    }
    sleep(1); // Esperar a que el broker inicie
```

Funciona como el orquestador principal del sistema. Es responsable de inicializar y crear los hilos correspondientes para el Broker, los Gateways y los Publishers simulados.

```c
    // Crear 1 gateway por cada 3 publishers (incluyendo el reservado ID 1)
    int total_publishers = num_p + 1; // +1 por el ESP32 (ID 1)
    int num_gateways = (total_publishers + 2) / 3;  // Ceiling division
    for (int i = 1; i <= num_gateways; i++) {
        pthread_t gateway_thread;
        int gateway_port = 9000 + i; // Puertos 9001, 9002, ...
        if (pthread_create(&gateway_thread, NULL, (void *)initialize_gateway, (void *)(intptr_t)gateway_port) != 0) {
            perror("Fallo en la creacion del hilo del gateway");
            exit(EXIT_FAILURE);
        }
        printf("Gateway %d creado en puerto %d\n", i, gateway_port);
        sleep(1); // Esperar a que el gateway inicie
    }

    // Crear publishers simulados (IDs 2, 3, ...) y asignarlos a gateways
    for (int i = 1; i <= num_p; i++) {
        pthread_t publisher_thread;

        int publisher_id = i + 1; // IDs comienzan en 2 (1 reservado para ESP32)

        // Calcular a qué gateway debe conectarse este publisher
        int gateway_index = (publisher_id - 1) / 3 + 1;
        int gateway_port = 9000 + gateway_index;

        // Crear configuración para el publisher
        publisher_config_t *config = malloc(sizeof(publisher_config_t));
        config->publisher_id = publisher_id;
        config->gateway_port = gateway_port;

        if (pthread_create(&publisher_thread, NULL, send_messages, (void *)config) != 0) {
            perror("Fallo en la creacion del hilo del publisher");
            free(config);
            exit(EXIT_FAILURE);
        }
        printf("Publisher %d creado, asignado a Gateway en puerto %d\n", publisher_id, gateway_port);
        pthread_detach(publisher_thread);
        sleep(1); // Esperar a que el publisher inicie
    }
    pthread_join(broker_thread, NULL);
    return 0;
}
```

Se utilizaron hilos POSIX (pthread) para manejar la concurrencia. El archivo channel.c actúa como orquestador, creando hilos para el broker, múltiples gateways y múltiples publishers.

- **esp32_main.ino**

```
// FreeRTOS
TaskHandle_t tempTaskHandle = nullptr;
TaskHandle_t humidTaskHandle = nullptr;
TaskHandle_t sendTaskHandle = nullptr;
```

```
void setup() {
  Serial.begin(115200);
  delay(200);
  connectWifi();

  // Inicia las tareas
  xTaskCreatePinnedToCore(tempTask,   "tempTask",  2048, nullptr, 1, &tempTaskHandle, 1);
  xTaskCreatePinnedToCore(humidTask,  "humidTask", 2048, nullptr, 1, &humidTaskHandle, 1);
  xTaskCreatePinnedToCore(sendTask,   "sendTask",  4096, nullptr, 1, &sendTaskHandle, 0);
}
```

Este es el código fuente para el microcontrolador ESP32 utilizando FreeRTOS. Simula la lectura de sensores de temperatura y humedad mediante tareas independientes y envía los datos al Gateway vía TCP. Se implementó un nodo sensor utilizando FreeRTOS en un ESP32 (simulado). Se definieron tareas independientes para la lectura de sensores y el envío de datos.

Demo

Se compila con el uso de make en la consola

```
sebas@Seb:/mnt/c/Users/sebas/OneDrive/Documentos/Practicas_So/Proyecto_SO1P$ make
gcc -Wall -Wextra -pthread -c channel.c
gcc -Wall -Wextra -pthread -c broker.c
gcc -Wall -Wextra -pthread -c gateway.c
gcc -Wall -Wextra -pthread -c publisher.c
gcc -Wall -Wextra -pthread -o channel channel.o broker.o gateway.o publisher.o
gcc -Wall -Wextra -pthread -c subscriber.c
gcc -Wall -Wextra -pthread -o subscriber subscriber.o
gcc -Wall -Wextra -pthread -DSTANDALONE -o publisher publisher.c
```

Se ejecuta .\channel n donde n es número de pubishers a crear. Por cada 3 publishers se crea un Gateway. Hay que tener en cuenta que el Publisher 1 está reservado para el ESP-32

```
sebas@Seb:/mnt/c/Users/sebas/OneDrive/Documentos/Practicas_So/Proyecto_SO1P$ ./channel 2
```

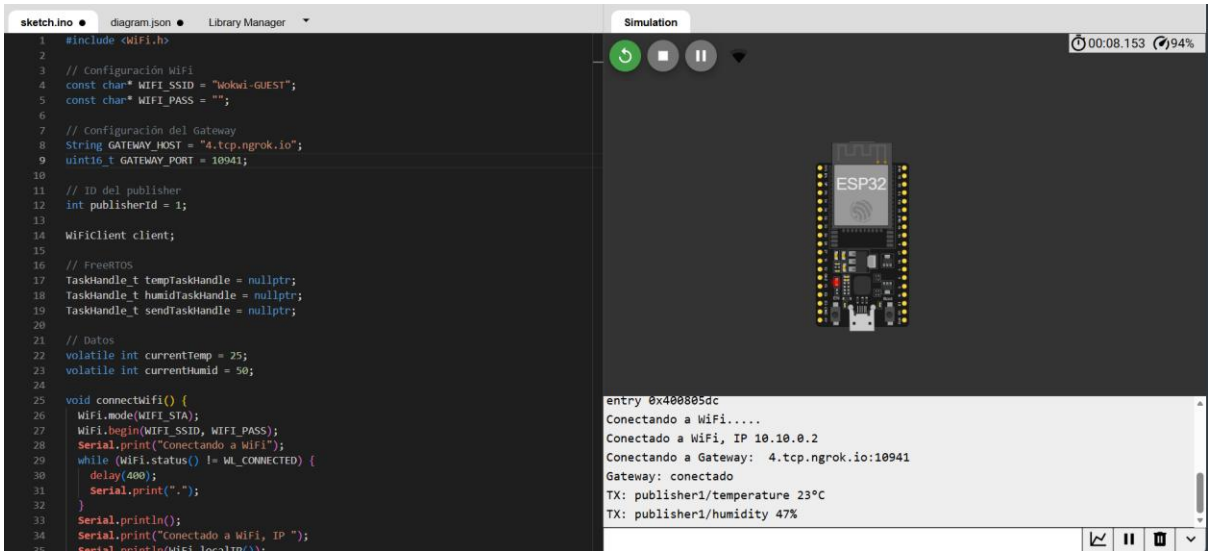
```
sebas@Seb:/mnt/c/Users/sebas/OneDrive/Documentos/Practicas_So/Proyecto_SO1P$ ./channel 2
Broker escuchando en el puerto 8080
Gateway 1 creado en puerto 9001
Gateway TCP escuchando en el puerto 9001
Publisher 2 creado, asignado a Gateway en puerto 9001
Publisher 2 iniciado, conectando al Gateway en puerto 9001
Publisher 2: conectado al Gateway en puerto 9001
Publisher 2 TX: publisher2/temperature 24°C
Cliente TCP conectado: ID 6
Mensaje TCP recibido del cliente 6: Topic: publisher2/temperature, Value: 24°C
Mensaje publicado en el broker: Topic: publisher2/temperature, Value: 24°C
Mensaje publicado en el broker: Topic: publisher2/temperature, Value: 24°C
Cliente desconectado
Publisher 3 creado, asignado a Gateway en puerto 9001
Publisher 3 iniciado, conectando al Gateway en puerto 9001
Publisher 3: conectado al Gateway en puerto 9001
Publisher 3 TX: publisher3/temperature 26°C
Cliente TCP conectado: ID 8
Mensaje TCP recibido del cliente 8: Topic: publisher3/temperature, Value: 26°C
Mensaje publicado en el broker: Topic: publisher3/temperature, Value: 26°C
Publisher 2 TX: publisher2/humidity 62%
Mensaje TCP recibido del cliente 6: Topic: publisher2/humidity, Value: 62%
Mensaje publicado en el broker: Topic: publisher2/humidity, Value: 62%
Mensaje publicado en el broker: Topic: publisher2/humidity, Value: 62%
Cliente desconectado
Mensaje publicado en el broker: Topic: publisher3/temperature, Value: 26°C
Cliente desconectado
```

Utilizar el comando ngrok tcp 9001 para habilitar el túnel y establecer la comunicación con el ESP-32

```
sebas@Seb:/mnt/c/Users/sebas/OneDrive/Documentos/Practicas_So/Proyecto_SO1P$ ngrok tcp 9001
```

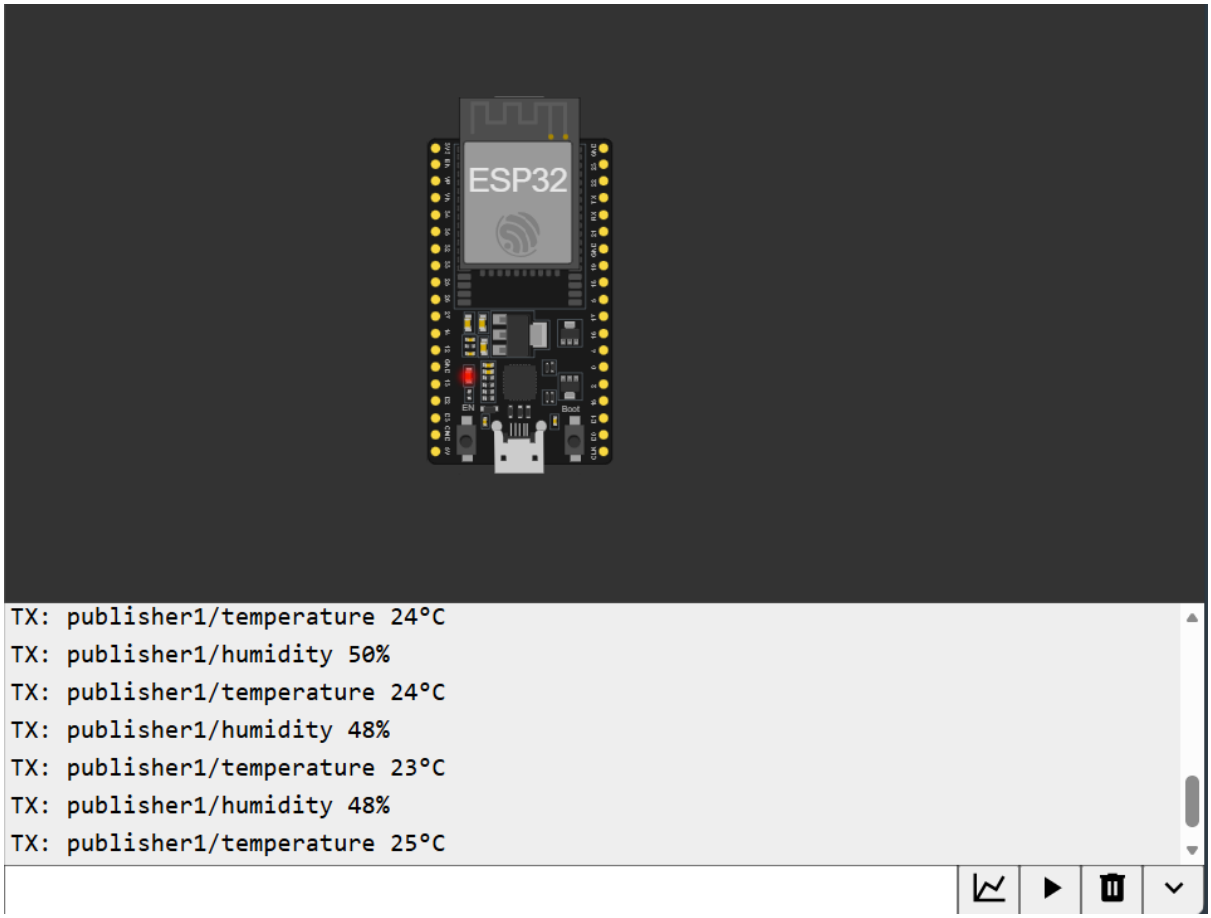
```
ngrok                                                                    (Ctrl+C to quit)

Call internal services from your gateway: https://ngrok.com/r/http-request

Session Status                online
Account                       Sebastian (Plan: Free)
Version                       3.33.1
Region                        United States (us)
Web Interface                 http://127.0.0.1:4041
Forwarding                    tcp://6.tcp.ngrok.io:15776 -> localhost:9001

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Se copia las credenciales en el código ESP-32 y luego se ejecuta:




```
#include <WiFi.h>

// Configuración WiFi
const char* WIFI_SSID = "Wokwi-GUEST";
const char* WIFI_PASS = "";

// Configuración del Gateway
String GATEWAY_HOST = "4.tcp.ngrok.io";
uint16_t GATEWAY_PORT = 10941;

// ID del publisher
int publisherId = 1;

WiFiClient client;

// FreeRTOS
TaskHandle_t tempTaskHandle = nullptr;
TaskHandle_t humidTaskHandle = nullptr;
TaskHandle_t sendTaskHandle = nullptr;

// Datos
volatile int currentTemp = 25;
volatile int currentHumid = 50;

void connectWifi() {
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Conectando a WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(400);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Conectado a WiFi, IP ");
```

Podemos visualizar en la consola del simulador los datos enviados:



Finalmente ejecutamos en otra consola .\suscriber [publisher n/topic]. En dónde n es el número de Publisher y el topic es el tema ya sea la temperatura o húmedad.

Aquí se realiza la consulta al Broker y este entrega la información:

```
PS C:\Users\sebas\OneDrive\Documentos\Practicas_So\Proyecto_SO1P> wsl
sebas@Seb:/mnt/c/Users/sebas/OneDrive/Documentos/Practicas_So/Proyecto_SO1P$ ./subscriber publisher1/temperature
Suscrito al topic: publisher1/temperature
Mensaje recibido en el tema publisher1/temperature: 24°C
Mensaje recibido en el tema publisher1/temperature: 24°C
Mensaje recibido en el tema publisher1/temperature: 23°C
Mensaje recibido en el tema publisher1/temperature: 23°C
Mensaje recibido en el tema publisher1/temperature: 25°C
Mensaje recibido en el tema publisher1/temperature: 25°C
Mensaje recibido en el tema publisher1/temperature: 22°C
Mensaje recibido en el tema publisher1/temperature: 22°C
```

Bibliografía

[1] OASIS, «MQTT version 3.1. 1,» *Standar OASIS,* vol. 1, p. 19, 2014.

[2] AWS Open Source, «FreeRTOS,» Noviembre 2025. [En línea]. Available: https://www.freertos.org/. [Último acceso: 01 12 2025].

[3] C. O. Biale y A. Lutenberg, «Control de cultivos hortícolas con sensores y actuadores mediante una plataforma en la nube basada en contenedores,» Buenos Aires, 2023.

[4] Espressif Systems (Shanghai) , «ESP-IDF Programming Guide,» 2025. [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html. [Último acceso: 2 Diciembre 2025].