Internet de las Cosas (GII-IC)

Sistema de Domótica con LoRa y MQTT

Autores
José Manuel Díaz Hernández
Nicolás Rey Alonso
Santiago Galindo Peralta
Alberto Martel Rodríguez

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este documento se describe el desarrollo de un sistema de domótica distribuido basado en tecnologías IoT. El objetivo del trabajo es diseñar y poner en funcionamiento una arquitectura completa que permita, a partir de sensores físicos de luz y proximidad, controlar automáticamente la iluminación y la apertura de una puerta utilizando comunicaciones LoRa y un broker MQTT sobre una Raspberry Pi.

El sistema se compone de varios nodos cooperando entre sí: un nodo de sensores dividido en maestro y esclavo, un gateway LoRa–MQTT, un nodo actuador y una capa de aplicaciones cliente. El enlace de largo alcance entre los nodos físicos se realiza mediante LoRa, sin embargo, entre el esclavo y el maestro la comunicación es serial. Por otro lado, la integración lógica y el acceso desde el exterior se articula a través de topics MQTT en un broker Mosquitto.

Desde el punto de vista docente, el proyecto permite aplicar de forma práctica conceptos de comunicaciones de bajo consumo, diseño de protocolos ligeros, integración de servicios mediante MQTT y desarrollo de dashboards web en tiempo real. Además, muestra cómo separar responsabilidades entre captura de datos, lógica de decisión, transporte y presentación.

El resto de la memoria se organiza de la siguiente forma. En el Capítulo 2 se presenta una descripción general del sistema y los objetivos perseguidos. El Capítulo 3 detalla el diseño hardware y software adoptado, incluyendo el protocolo maestro–esclavo y el formato de las tramas LoRa. En el Capítulo **??** se describe la arquitectura global y los flujos de comunicación entre nodos y el broker MQTT. El Capítulo **??** resume los datos intercambiados y la codificación utilizada en cada capa. El Capítulo **??** recoge los aspectos prácticos de implementación y pruebas, mientras que el Capítulo **??** muestra las interfaces más relevantes, como el dashboard web. Finalmente, en el Capítulo **??** se presentan las conclusiones y posibles líneas de mejora.

Capítulo 2

Descripción de la aplicación y objetivos

En este trabajo se ha desarrollado un sistema de domótica distribuido orientado a un entorno doméstico sencillo: una puerta de acceso y un punto de iluminación. El sistema toma decisiones automáticamente a partir de sensores de distancia (ultrasonidos) y un sensor de luz (LDR), y además permite forzar manualmente los estados desde aplicaciones externas vía MQTT (por ejemplo, un dashboard web o Node-RED).

El público objetivo es principalmente docente: estudiantes de la asignatura de Internet de las Cosas que necesiten un ejemplo completo de arquitectura IoT (sensores, pasarela, protocolo de campo, broker MQTT y aplicaciones cliente). No obstante, la solución es extensible a escenarios reales de monitorización y control de acceso en pequeña escala.

Contexto y Motivación

El Internet de las Cosas (IoT) ha revolucionado la forma en que interactuamos con nuestro entorno, permitiendo la automatización de tareas cotidianas y la monitorización remota de espacios. En el ámbito doméstico, los sistemas de domótica ofrecen beneficios tangibles:

- **Eficiencia energética**: encender luces solo cuando es necesario reduce el consumo eléctrico.
- **Comodidad**: puertas que se abren automáticamente al detectar presencia.
- **Seguridad**: monitorización remota del estado del hogar.
- **Accesibilidad**: control desde dispositivos móviles para personas con movilidad reducida.

La tecnología LoRa (Long Range) resulta especialmente adecuada para este tipo de aplicaciones por su largo alcance (hasta varios kilómetros en condiciones ideales), bajo consumo energético y capacidad de penetración en interiores. Combinada con MQTT, un protocolo de mensajería ligero diseñado para dispositivos con recursos limitados, permite construir sistemas escalables y flexibles.

Objetivos

Los objetivos principales del proyecto son:

- Diseñar y desplegar una arquitectura IoT completa que conecte sensores físicos, un nodo actuador y una pasarela basada en Raspberry Pi usando LoRa como red de campo y MQTT como capa de integración.
- Implementar un protocolo maestro–esclavo ligero para la lectura y configuración de sensores (ultrasonidos y LDR) y un formato de tramas LoRa con direcciones, identificadores de mensaje y ACK de aplicación.
- Integrar un broker MQTT (Mosquitto) y un puente serie–MQTT que permita exponer los datos de los sensores y controlar el actuador desde aplicaciones externas de forma desacoplada.
- Desarrollar una interfaz web sencilla que visualice en tiempo real el estado de luz y puerta, y que permita enviar comandos manuales reutilizando la infraestructura MQTT existente.

Funcionalidades del Sistema

Desde el punto de vista funcional, el sistema ofrece:

- **Control automático de iluminación**: encendido y apagado de la luz en función del nivel medido por la LDR. Cuando el sensor detecta oscuridad (valor inferior a 500), se enciende el LED; cuando hay luz suficiente, se apaga.
- **Control automático de puerta**: apertura y cierre de una puerta simulada mediante un servomotor SG90. Los sensores ultrasónicos SRF01 y SRF02 detectan la presencia de objetos a menos de 100 cm, lo que activa la apertura.
- **Publicación MQTT**: los estados lógicos (luz=0/1, puerta=0/1) se publican en topics MQTT para monitorización, registro histórico e integración con otras aplicaciones.
- **Control manual**: posibilidad de forzar los estados desde cualquier cliente MQTT (dashboard web, mosquitto_pub, Node-RED, Home Assistant) sin necesidad de acceder físicamente a los nodos.

Casos de Uso

A continuación se describen los principales casos de uso del sistema:

Tabla 2.1: Casos de uso del sistema de domótica.

Caso de Uso	**Descripción**
CU-01: Detectar oscuridad	El sensor LDR detecta un nivel de luz bajo y el sistema enciende automáticamente el LED.
CU-02: Detectar presencia	El sensor ultrasónico detecta un objeto cercano y el sistema abre la puerta.
CU-03: Monitorizar estado	El usuario accede al dashboard web y visualiza el estado actual de luz y puerta.
CU-04: Forzar luz	El usuario pulsa un botón en el dashboard para encender o apagar la luz manualmente.
CU-05: Forzar puerta	El usuario pulsa un botón en el dashboard para abrir o cerrar la puerta manualmente.

Alcance y Limitaciones

El sistema desarrollado tiene las siguientes características y limitaciones:

- **Alcance**: sistema funcional con dos sensores (luz y distancia) y dos actuadores (LED y servo), comunicación LoRa bidireccional con ACK, integración MQTT completa y dashboard web en tiempo real.
- **Limitaciones**: no implementa cifrado en LoRa, no tiene persistencia de datos históricos, y el dashboard no incluye autenticación de usuarios.

Capítulo 3

Diseño

En esta sección se describen las principales decisiones de diseño tanto a nivel lógico como de interacción con el sistema.

Patrones y decisiones

A nivel lógico se ha optado por una arquitectura por capas:

- **Capa de sensores**: formada por un esclavo y un maestro (MKR WAN 1310). El esclavo encapsula el acceso hardware a los sensores (I^2C para SRF01/SRF02 y analógico para la LDR) detrás de un protocolo serie simple. El maestro actúa como único punto que interpreta las medidas, aplica umbrales y decide estados lógicos.
- **Capa de transporte LoRa**: el maestro y el gateway intercambian tramas LoRa con cabeceras explícitas (direcciones, identificador de mensaje, longitud), lo que permite dirigir mensajes a distintos nodos (gateway, actuador) y asociar ACKs a comandos concretos.
- **Capa de integración MQTT**: en la Raspberry Pi, un proceso puente traduce entre el protocolo serie del gateway y mensajes MQTT, mapeando topics internos (`sensor/0`, `sensor/1`) a topics lógicos (`sensores/puerta`, `sensores/luz`). Esta capa aplica además políticas de *rate limiting* y eliminación de duplicados para no saturar LoRa.
- **Capa de presentación**: formada por el dashboard web y herramientas como `mosquitto_sub`/`mosquitto_pub`. Estas aplicaciones solo ven topics MQTT y no necesitan conocer detalles de LoRa ni del protocolo maestro–esclavo.

Esta separación responde a dos decisiones clave:

- Mantener los nodos embebidos (maestro, esclavo, actuador) lo más simples posible, delegando el tratamiento de mensajes, la conversión de formatos y la integración con otras aplicaciones en la Raspberry Pi.
- Utilizar formatos binarios compactos en los enlaces de menor ancho de banda (LoRa y UART) y formatos más ricos (cadenas y JSON) en MQTT, donde el coste en bytes es menos crítico.

En cuanto al control, el maestro solo envía nuevas órdenes cuando detecta un cambio de estado (por ejemplo, de luz=0 a luz=1), y el actuador siempre responde con un ACK con el mismo identificador de mensaje. El gateway reintenta automáticamente las transmisiones si no recibe ese ACK dentro de un tiempo máximo.

Prototipos/Mockups

Aunque el proyecto no es una aplicación móvil, se ha diseñado una interfaz web sencilla como dashboard de usuario final. La página muestra dos tarjetas principales:

- **Sensor de luz**: icono y estado textual (*oscuro* / *iluminado*), con dos botones para enviar manualmente valores 0 o 1 sobre el topic `sensores/luz`.
- **Sensor de puerta**: icono de puerta y estado (*cerrada/nadie* o *abierta/gente*), de nuevo con botones para forzar los estados 0 o 1 sobre `sensores/puerta`.

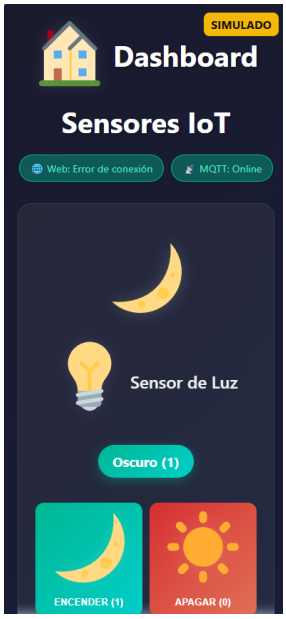
Además, el dashboard incluye indicadores de conexión (estado de la conexión web y del broker MQTT) y un panel de log donde se registran los eventos relevantes (nuevos valores recibidos, comandos enviados, pérdidas de conexión).

A nivel de diseño de interacción se ha priorizado:

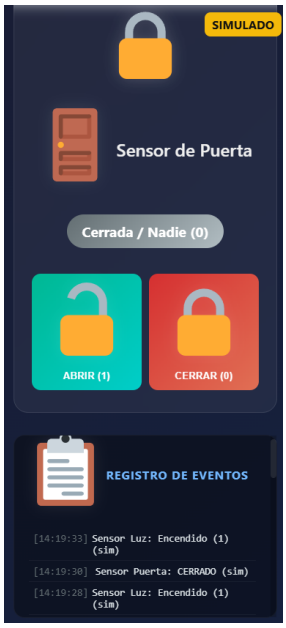
- Representar los estados con emojis e indicadores de color para facilitar una lectura rápida.
- Mostrar claramente si el sistema está conectado al broker MQTT y si la web está sincronizada con el estado actual.
- Registrar un histórico de eventos para ayudar en la depuración durante las pruebas.

(a) Dashboard Escritorio (1)

(b) Dashboard Escritorio (2)



(c) Dashboard Móvil (3)



(d) Dashboard Móvil (4)

Figura 3.1: Capturas del dashboard en escritorio (12) y móvil (34).

Diseño de la arquitectura del sistema

La arquitectura del sistema se basa en una comunicación jerárquica y modular entre los distintos componentes, cada uno con responsabilidades claras:

- **Esclavo de sensores**: encargado de la adquisición de datos desde los sensores físicos (LDR y SRF01/SRF02) y de enviarlos al maestro a través del canal serie.
- **Maestro de sensores**: recibe los datos del esclavo, procesa las mediciones aplicando umbrales para determinar estados lógicos, y gestiona la comunicación inalámbrica mediante LoRa con el gateway.

- **Gateway LoRa**: actúa como intermediario entre el maestro y la Raspberry Pi, recibiendo los mensajes LoRa y retransmitiéndolos a través del puerto serie y recibiendo los mensajes MQTT desde la raspberry y retransmitiéndolos por LoRa.
- **Raspberry Pi**: ejecuta un proceso puente que traduce los mensajes del gateway a formato MQTT, aplicando políticas de filtrado y mapeo de topics. Además, aloja el broker MQTT y el dashboard web.
- **Dashboard web**: proporciona una interfaz de usuario para visualizar el estado de los sensores y enviar comandos manuales a través de MQTT.

Diagrama de arquitectura del sistema

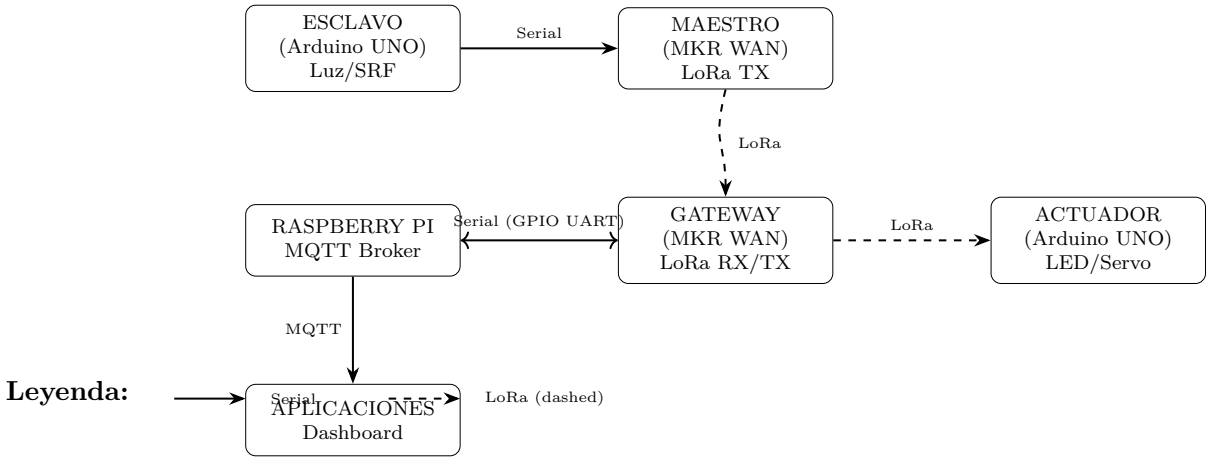


Figura 3.2: Diagrama de arquitectura del sistema. Líneas continuas: comunicación serial/MQTT. Líneas discontinuas: LoRa inalámbrico.

Capítulo 4

Arquitectura

En este capítulo se describe la arquitectura global del sistema, los flujos de comunicación entre los distintos nodos y los protocolos utilizados en cada capa.

Visión General

El sistema sigue una arquitectura distribuida de tipo *sensor-gateway-actuador* con integración en la nube a través de MQTT. Los componentes principales son:

1. **Nodo de sensores**: compuesto por un esclavo (Arduino UNO) y un maestro (MKR WAN 1310).
2. **Gateway LoRa**: MKR WAN 1310 conectado a la Raspberry Pi.
3. **Raspberry Pi**: ejecuta el broker MQTT y el puente serie-MQTT.
4. **Nodo actuador**: Arduino UNO con módulo LoRa externo.
5. **Aplicaciones cliente**: dashboard web, Node-RED, Home Assistant.

Diagrama de Flujo de Datos

El flujo de datos desde los sensores hasta las aplicaciones cliente sigue el siguiente recorrido:

Figura 4.1: Flujo de datos desde sensores hasta la Raspberry Pi.

Protocolo Maestro-Esclavo (Serial)

La comunicación entre el esclavo y el maestro utiliza un protocolo serie sencillo basado en tramas estructuradas. El esclavo envía periódicamente las lecturas de los sensores al maestro.

Tabla 4.1: Estructura de respuesta del esclavo al maestro.

Campo	Tamaño	Descripción
Código respuesta	1 byte	Tipo de dato (distancia, luz, error)
ID Sensor	1 byte	Identificador del sensor
Dato (MSB)	1 byte	Byte alto de la medida
Dato (LSB)	1 byte	Byte bajo de la medida

Protocolo LoRa

Las comunicaciones LoRa entre el maestro, el gateway y el actuador utilizan un formato de paquete común con cabecera explícita:

Tabla 4.2: Formato de paquete LoRa.

Campo	Tamaño	Descripción
Destino	1 byte	Dirección del nodo destino
Origen	1 byte	Dirección del nodo emisor
Msg ID (MSB)	1 byte	Identificador de mensaje (byte alto)
Msg ID (LSB)	1 byte	Identificador de mensaje (byte bajo)
Longitud	1 byte	Tamaño del payload en bytes
Payload	N bytes	Datos del mensaje

Las direcciones asignadas a cada nodo son:

Tabla 4.3: Direcciones LoRa de los nodos del sistema.

Nodo	Dirección
Maestro (sensores)	`0x04`
Gateway	`0x05`
Actuador	`0x06`
Broadcast	`0xFF`

Protocolo Serial Gateway-Raspberry

El gateway y la Raspberry Pi se comunican mediante un protocolo binario delimitado por caracteres STX/ETX:

Tabla 4.4: Formato de trama serial entre Gateway y Raspberry Pi.

Campo	Valor/Tamaño	Descripción
STX	`0x02`	Inicio de trama
Tipo	1 byte	R=RX, T=TX, A=ACK, N=NACK, S=Status
Topic Len	1 byte	Longitud del topic
Topic	N bytes	Nombre del topic
Payload Len	1 byte	Longitud del payload
Payload	N bytes	Datos
ETX	`0x03`	Fin de trama

Integración MQTT

El broker MQTT (Mosquitto) en la Raspberry Pi expone los siguientes topics:

Tabla 4.5: Topics MQTT del sistema.

Topic	Dirección	Descripción
`sensores/luz`	Publicación	Estado del sensor de luz (0/1)
`sensores/puerta`	Publicación	Estado del sensor de proximidad (0/1)
`lora/rx`	Publicación	Mensajes LoRa en bruto (JSON)
`lora/tx`	Suscripción	Enviar mensaje LoRa genérico
`actuador/comando`	Suscripción	Comandos para el actuador

El puente MQTT-LoRa (`mqtt_lora_bridge.py`) realiza el mapeo entre los topics internos de LoRa y los topics MQTT legibles:

- `sensor/0` → `sensores/puerta`
- `sensor/1` → `sensores/luz`

Flujo de Control (Actuador)

Cuando una aplicación cliente desea controlar el actuador, el flujo es el siguiente:

1. La aplicación publica en `sensores/luz` o `sensores/puerta`.
2. El bridge recibe el mensaje MQTT y lo convierte a trama serial.
3. El gateway recibe la trama y la transmite por LoRa al actuador.
4. El actuador ejecuta la acción y responde con ACK.

5. El gateway recibe el ACK y notifica al bridge.
6. Si no hay ACK en el tiempo límite, el gateway reintenta (hasta 3 veces).

Políticas de Calidad de Servicio

Para garantizar la fiabilidad del sistema se implementan las siguientes políticas:

- **Rate limiting**: el bridge limita los envíos a un mínimo de 150 ms entre mensajes para no saturar LoRa.
- **Eliminación de duplicados**: mensajes idénticos recibidos en una ventana de 2 segundos se ignoran.
- **Reintentos con ACK**: el gateway reintenta hasta 3 veces si no recibe confirmación del actuador.
- **Timeout configurable**: el tiempo máximo de espera de ACK es de 2 segundos, ajustado para SF10 y BW 62.5 kHz.

Diagrama de Secuencia

A continuación se muestra el diagrama de secuencia para un ciclo completo de detección y actuación:

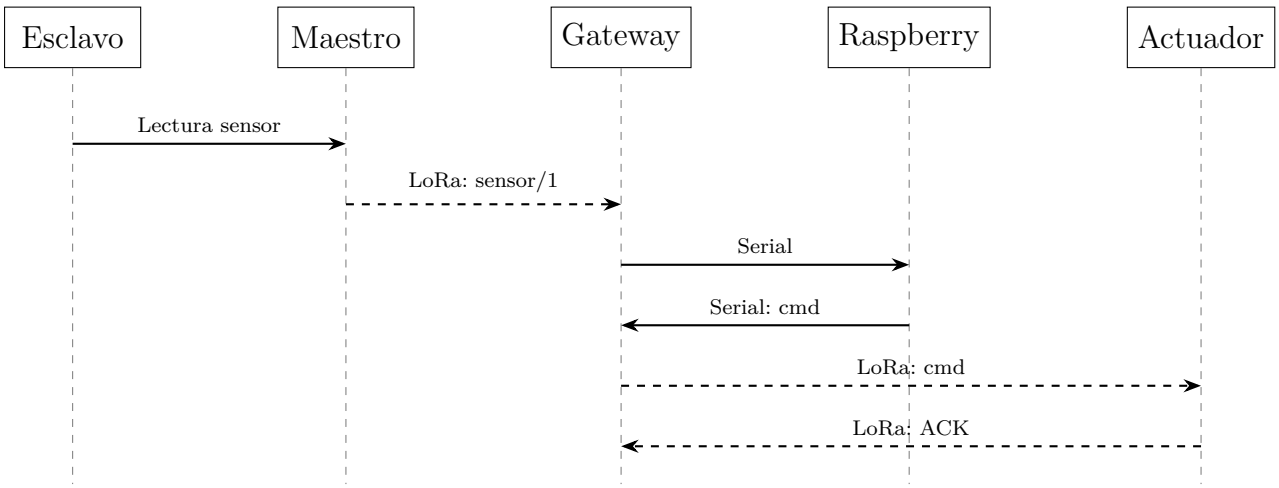


Figura 4.2: Diagrama de secuencia: detección de luz y actuación.

Consideraciones de Seguridad

Aunque el sistema no implementa cifrado en las comunicaciones LoRa (fuera del alcance de este proyecto académico), se aplican medidas básicas:

- Filtrado por dirección de origen en el actuador.
- Sync Word compartido (`0x12`) que actúa como identificador de red.

- Comunicación MQTT local (localhost) sin exposición a Internet.

En un despliegue real se recomendaría añadir cifrado AES en la capa de aplicación y autenticación MQTT con usuario/contraseña o certificados TLS.

Capítulo 5

Modelo de Datos

En este capítulo se describe el modelo de datos utilizado en el sistema, incluyendo la estructura de configuración de los sensores, los formatos de mensajes intercambiados y la codificación utilizada en cada capa.

Notas Técnicas

- Por incompatibilidades de hardware, se usó `SoftwareSerial.h` para reemplazar `Serial1` en el Arduino UNO.
- Todas las placas conectadas a la antena LoRa están conectadas a una batería para manejar la tensión y no quemar la placa.
- Se heredó código de las prácticas 2 y 3 de la asignatura.

Sensores Utilizados

El sistema utiliza tres sensores físicos para la adquisición de datos del entorno:

Tabla 5.1: Sensores del sistema.

Sensor	**Tipo**	**Interfaz**	**Función**
SRF01	Ultrasónico	I^2C	Detección de distancia
SRF02	Ultrasónico	I^2C	Detección de distancia
LDR	Fotorresistencia	Analógico (A1)	Nivel de luz ambiental

Esclavo (`Esclavo.ino`)

El esclavo se encarga de la gestión directa de sensores. Cada sensor dispone de una estructura de configuración propia que almacena su dirección, unidad de medida, retardo mínimo, modo de funcionamiento (periódico o puntual), periodo de muestreo y último valor leído.

Listing 5.1: Estructura de configuración de sensores.

```
1 struct SensorConfig {
```

```
  uint8_t address;       // Direccion I2C del sensor
  uint8_t unit;          // Unidad de medida (cm, inch, us)
  uint16_t delayMs;      // Retardo minimo entre lecturas
  bool periodic;         // Modo periodico activo
  uint16_t periodMs;     // Periodo de muestreo (ms)
  unsigned long lastShot; // Timestamp ultima lectura
  uint16_t lastMeasure; // Ultimo valor leido
  bool active;           // Sensor detectado/activo
  char name[8];          // Nombre del sensor
};
```

Durante la fase de inicialización, el dispositivo detecta automáticamente la presencia de los sensores I^2C y habilita por defecto las mediciones periódicas (1 segundo) para aquellos sensores activos.

Funciones de Lectura

Las lecturas de los sensores se realizan mediante las siguientes funciones:

Listing 5.2: Funciones de lectura de sensores.

```
// Lectura de sensores ultrasonicos SRF02
uint16_t readSRF02(uint8_t address, uint8_t unit) {
  Wire.beginTransmission(address);
  Wire.write(0x00);      // Registro de comando
  Wire.write(unit);      // 0x51=cm, 0x50=inch, 0x52=us
  Wire.endTransmission();
  delay(70);             // Tiempo de medicion
  Wire.requestFrom(address, 2);
  uint16_t distance = (Wire.read() << 8) | Wire.read();
  return distance;
}

// Lectura del fotorresistor
uint16_t readLDR() {
  return analogRead(A1);  // Valor 0-1023
}
```

Protocolo de Respuesta

El esclavo envía las lecturas al maestro mediante un protocolo estructurado:

Listing 5.3: Envío de respuesta al maestro.

```
void sendResponse(uint8_t code, uint8_t* data, uint8_t len) {
  Serial1.write(RESP_START);   // Marcador inicio
  Serial1.write(code);         // Codigo de respuesta
  Serial1.write(len);          // Longitud de datos
  if (len > 0 && data != nullptr) {
    Serial1.write(data, len);
  }
  Serial1.write(RESP_END);     // Marcador fin
}
```

Maestro (`Maestro.ino`)

El maestro supervisa el sistema de sensores, interpreta las mediciones recibidas y las publica como pares etiqueta-valor a través de la red LoRa.

Umbrales de Decisión

Las medidas crudas recibidas del esclavo son procesadas aplicando umbrales para convertirlas en estados binarios:

Listing 5.4: Umbrales de decisión.

```
#define LIGHT_THRESHOLD    500    // luz < 500 -> oscuro (1)
#define DISTANCE_THRESHOLD 100    // distancia < 100cm -> cerca (1)
```

Tabla 5.2: Interpretación de valores de sensores.

Sensor	Condición	Estado	Acción
LDR	valor <500	1 (oscuro)	Encender LED
LDR	valor $\geq$ 500	0 (iluminado)	Apagar LED
SRF	distancia <100 cm	1 (cerca)	Abrir puerta
SRF	distancia $\geq$ 100 cm	0 (lejos)	Cerrar puerta

Estructura de Mensajes

Para la comunicación LoRa, el maestro utiliza una estructura que mantiene el estado de cada sensor:

Listing 5.5: Estructura de mensaje de sensor.

```
struct SensorMessage {
  uint8_t payload;         // Valor a enviar (0 o 1)
  bool pending;            // Hay mensaje pendiente
  uint8_t lastSentValue;   // Ultimo valor enviado
};

// Mensajes pendientes para cada sensor
SensorMessage sensorMessages[2];  // [0]=puerta, [1]=luz
```

Configuración LoRa

El maestro configura el módulo LoRa integrado en el MKR WAN 1310 con los siguientes parámetros:

Listing 5.6: Configuración LoRa del maestro.

```
#define LORA_LOCAL_ADDRESS      0x04  // Direccion del maestro
#define LORA_GATEWAY_ADDRESS    0x05  // Direccion del gateway
#define LORA_FREQUENCY          868E6 // 868 MHz (Europa)
#define LORA_BW                 62.5E3 // Ancho de banda
#define LORA_SF                 10    // Spreading Factor
```

```
#define LORA_CR                 5      // Coding Rate 4/5
#define LORA_TP                 2      // Potencia TX (dBm)
#define LORA_SYNC_WORD          0x12   // Palabra de
    sincronizacion
#define LORA_PREAMBLE_LENGTH    8      // Longitud preambulo
```

Formato del Payload LoRa

El payload enviado por el maestro al gateway tiene el siguiente formato:

Tabla 5.3: Formato del payload de sensores.

Campo	Tamaño	Descripción
TopicLen	1 byte	Longitud del nombre del topic
Topic	N bytes	Nombre del topic (ej: "sensor/1")
Valor	1 byte	Estado binario del sensor (0 o 1)

Política de Envío

El sistema de envío está diseñado para ser reactivo y eficiente:

- Solo se envía un mensaje cuando hay un **cambio de estado** (de 0 a 1 o viceversa).
- Se mantiene un **intervalo mínimo** de 500 ms entre envíos para no saturar el canal.
- Los envíos se **alternan entre sensores** para distribuir la carga.

Listing 5.7: Control de envío LoRa.

```
const unsigned long LORA_SEND_INTERVAL = 500;  // 500ms minimo
volatile bool sendDistanceNext = false;        // Alternar
    sensores
```

Resumen del Modelo de Datos

Tabla 5.4: Resumen de topics y valores.

Topic Interno	Topic MQTT	Valores	Significado
sensor/0	sensores/puerta	0, 1	0=nadie, 1=presencia
sensor/1	sensores/luz	0, 1	0=iluminado, 1=oscuro

Capítulo 6

Desarrollo

En este capítulo se describe el desarrollo del nodo actuador, encargado de ejecutar las acciones físicas en respuesta a los estados detectados por los sensores. El actuador recibe comandos a través de LoRa desde el gateway y controla dos elementos: un LED para la iluminación y un servomotor para la apertura y cierre de una puerta.

Hardware del Actuador

El nodo actuador está compuesto por los siguientes elementos:

- **Arduino UNO**: microcontrolador principal que ejecuta la lógica de control.
- **Módulo LoRa SX1276**: receptor de comunicaciones inalámbricas a 868 MHz.
- **LED**: indicador luminoso conectado al pin 6, representa la iluminación del entorno.
- **Servo SG90**: servomotor conectado al pin 7, simula el mecanismo de apertura de una puerta.

Las conexiones del módulo LoRa al Arduino UNO siguen el estándar SPI:

Tabla 6.1: Conexiones del módulo LoRa SX1276 al Arduino UNO.

Pin LoRa	**Pin Arduino**
NSS	Pin 10
MOSI	Pin 11
MISO	Pin 12
SCK	Pin 13
RST	Pin 9
DIO0	Pin 2

Configuración LoRa

El actuador debe utilizar exactamente la misma configuración LoRa que el resto de nodos del sistema para garantizar la interoperabilidad:

Listing 6.1: Configuración LoRa del actuador.

```
LoRaConfig_t nodeConfig = {6, 10, 5, 2};
// BW = 62.5 kHz (indice 6)
// SF = 10 (Spreading Factor)
// CR = 4/5 (Coding Rate)
// TxPwr = 2 dBm
```

La dirección LoRa asignada al actuador es `0x06`, mientras que solo acepta mensajes provenientes del gateway (`0x05`). Esto proporciona una capa básica de seguridad al filtrar emisores no autorizados.

Protocolo de Comandos

Los comandos recibidos por el actuador siguen un formato binario compacto de dos bytes:

Tabla 6.2: Formato del payload de comandos para el actuador.

Byte	Campo	Descripción
0	Tipo	0 = Luz, 1 = Puerta
1	Valor	Estado a aplicar

Para el control de luz:

- Valor 0: Apagar LED.
- Valor 1: Encender LED.

Para el control de puerta:

- Valor 0: Cerrar puerta (servo a 10ř).
- Valor 1, 2 o 3: Abrir puerta (servo a 160ř).

Sistema de ACK

El actuador implementa un sistema de confirmación (ACK) para garantizar la entrega fiable de comandos. Cada vez que recibe un paquete válido, responde inmediatamente con un ACK que incluye:

- Dirección de destino (el gateway).
- Dirección de origen (el actuador).
- Identificador de mensaje (el mismo que el comando recibido).
- Marcador de ACK (`0xAC`).
- Estado (0 = éxito, 1 = error).

Listing 6.2: Función de envío de ACK.

```
void sendAck(uint8_t dest, uint16_t msgId, uint8_t status) {
  LoRa.beginPacket();
  LoRa.write(dest);
  LoRa.write(localAddress);
  LoRa.write((uint8_t)(msgId >> 8));
  LoRa.write((uint8_t)(msgId & 0xFF));
  LoRa.write((uint8_t)2);
  LoRa.write(ACK_MARKER);
  LoRa.write(status);
  LoRa.endPacket();
}
```

Control de Duplicados

Para evitar que un mismo comando se ejecute múltiples veces (por ejemplo, debido a retransmisiones del gateway), el actuador mantiene el identificador del último mensaje procesado. Si recibe un paquete con el mismo `msgId`, responde con ACK pero no vuelve a ejecutar la acción:

Listing 6.3: Detección de mensajes duplicados.

```
uint16_t lastProcessedMsgId = 0xFFFF;

// En el bucle de recepcion:
bool isDuplicate = (msgId == lastProcessedMsgId);
if (!isDuplicate) {
  // Procesar comando
  lastProcessedMsgId = msgId;
}
// SIEMPRE enviar ACK
sendAck(sender, msgId, ackStatus);
```

Funciones de Control

Las funciones que aplican los estados físicos son directas y robustas:

Listing 6.4: Control del LED y servo.

```
void aplicarLuz(uint8_t v) {
  if (v == 1) {
    digitalWrite(pinLed, HIGH);
  } else if (v == 0) {
    digitalWrite(pinLed, LOW);
  }
}

void aplicarPuerta(uint8_t v) {
  if (v == 0) {
    posicion = 10;
```

```
12      miServo.write(posicion);
13    } else if (v == 1 || v == 2 || v == 3) {
14      posicion = 160;
15      miServo.write(posicion);
16    }
17  }
```

Flujo de Operación

El actuador opera en modo *polling*, verificando continuamente si hay paquetes LoRa disponibles:

1. Inicialización del hardware (servo, LED, LoRa).
2. Bucle principal: verificar si hay paquete LoRa disponible.
3. Si hay paquete: leer cabecera y verificar destinatario.
4. Verificar que el emisor sea el gateway autorizado.
5. Extraer tipo y valor del payload.
6. Si no es duplicado: ejecutar acción correspondiente.
7. Enviar ACK al gateway.
8. Volver al paso 2.

Este diseño garantiza que el actuador responda rápidamente a los comandos mientras mantiene un consumo de recursos bajo, adecuado para un microcontrolador con recursos limitados como el Arduino UNO.

Capítulo 7

Interfaces

En este capítulo se describen las interfaces de software del sistema: el gateway LoRa, el puente MQTT-LoRa y el dashboard web de monitorización.

Gateway LoRa (MKR WAN 1310)

El gateway actúa como intermediario bidireccional entre la red LoRa y la Raspberry Pi. Su función principal es traducir entre el protocolo binario de LoRa y el protocolo serial estructurado que entiende el puente MQTT.

Funcionalidades principales

- **Recepción LoRa**: escucha continuamente mensajes de los nodos sensores y los reenvía por serial.
- **Transmisión LoRa**: recibe comandos por serial y los transmite a los actuadores.
- **Gestión de ACK**: espera confirmación del actuador y reintenta si es necesario.
- **Mapeo de topics**: traduce los topics internos (`sensor/0`) a topics MQTT (`sensores/puerta`).

Protocolo serial

El gateway implementa un protocolo binario delimitado:

Listing 7.1: Constantes del protocolo serial.

```
#define STX 0x02  // Start of Text
#define ETX 0x03  // End of Text

#define MSG_TYPE_LORA_RX  'R'  // Mensaje recibido de LoRa
#define MSG_TYPE_LORA_TX  'T'  // Mensaje a transmitir
#define MSG_TYPE_ACK      'A'  // Acknowledgment
#define MSG_TYPE_NACK     'N'  // Negative acknowledgment
#define MSG_TYPE_STATUS   'S'  // Estado del sistema
```

Sistema de reintentos

Para garantizar la entrega fiable de comandos, el gateway implementa un sistema de reintentos con timeout:

Listing 7.2: Configuración de reintentos.

```
const unsigned long ACK_TIMEOUT_MS = 2000;  // 2 segundos
const uint8_t MAX_ACK_RETRIES = 3;          // 3 intentos
const unsigned long TX_COOLDOWN_MS = 50;    // Cooldown entre TX
```

Puente MQTT-LoRa (Raspberry Pi)

El script `mqtt_lora_bridge.py` ejecutándose en la Raspberry Pi conecta el mundo LoRa con el ecosistema MQTT.

Arquitectura del puente

El puente utiliza dos hilos principales:

- **Hilo serial**: lee continuamente del puerto GPIO UART y parsea las tramas del gateway.
- **Hilo MQTT**: gestiona la conexión al broker y procesa los mensajes suscritos.

Listing 7.3: Configuración del puente.

```
SERIAL_PORT = "/dev/serial0"  # GPIO UART
SERIAL_BAUD = 115200

MQTT_BROKER = "localhost"
MQTT_PORT = 1883

# Topics de suscripcion para reenviar por LoRa
TOPICS_TO_LORA = [
    "lora/tx",
    "actuador/comando",
    "sensores/luz",
    "sensores/puerta"
]
```

Control de rate limiting

Para evitar saturar el canal LoRa, el puente implementa políticas de limitación:

Listing 7.4: Políticas de rate limiting.

```
MIN_TX_INTERVAL = 0.15   # Minimo 150ms entre envios
DUPLICATE_WINDOW = 2.0   # Ignorar duplicados en 2 segundos
```

Clases principales

- `SerialProtocol`: gestiona la comunicación serial con el gateway, incluyendo conexión, desconexión y parseo de tramas.
- `LoRaMessage`: estructura de datos para mensajes LoRa con campos para emisor, RSSI, SNR, topic y payload.
- `MQTTLoRaBridge`: clase principal que coordina serial y MQTT, implementa el mapeo de topics y las políticas de QoS.

Dashboard Web

El dashboard proporciona una interfaz visual para monitorizar el estado de los sensores y enviar comandos manuales.

Tecnologías utilizadas

- **Flask**: framework web ligero para Python.
- **Flask-SocketIO**: extensión para comunicación en tiempo real mediante WebSockets.
- **Paho MQTT**: cliente MQTT para Python.
- **HTML/CSS/JavaScript**: interfaz de usuario responsive.

Arquitectura del dashboard

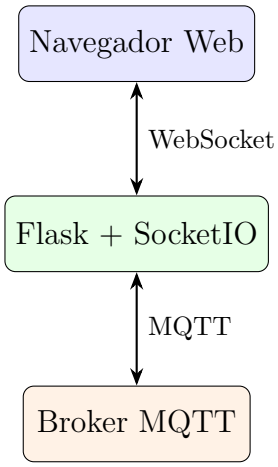


Figura 7.1: Arquitectura del dashboard web.

Funcionalidades

El dashboard ofrece las siguientes características:

- **Visualización en tiempo real**: muestra el estado actual de luz y puerta con emojis e indicadores de color.

- **Control manual**: botones para forzar estados 0 o 1 en cada sensor/actuador.
- **Indicadores de conexión**: muestra el estado de la conexión WebSocket y MQTT.
- **Log de eventos**: registro histórico de mensajes recibidos y enviados.

Interfaz de usuario

La interfaz está diseñada para ser intuitiva y accesible desde cualquier dispositivo:

- Diseño responsive con CSS Grid.
- Tema oscuro con gradientes para reducir fatiga visual.
- Tarjetas con bordes redondeados y efecto glassmorphism.
- Emojis grandes (100px) para identificación rápida del estado.
- Transiciones suaves para cambios de estado.

Listing 7.5: Estructura de una tarjeta de sensor.

```
<div class="card">
  <div class="emoji" id="emoji-luz"></div>
  <div class="titulo-sensor">Sensor de Luz</div>
  <div class="estado" id="estado-luz">Desconocido</div>
  <div class="botones">
    <button onclick="publicarLuz(0)">Iluminado (0)</button>
    <button onclick="publicarLuz(1)">Oscuro (1)</button>
  </div>
</div>
```

Comunicación en tiempo real

La actualización del dashboard se realiza mediante WebSockets con SocketIO:

Listing 7.6: Manejo de eventos SocketIO en el cliente.

```
socket.on('update_luz', function(data) {
  const valor = data.valor;
  document.getElementById('emoji-luz').textContent =
    valor === 1 ? '' : '';
  document.getElementById('estado-luz').textContent =
    valor === 1 ? 'Oscuro' : 'Iluminado';
});

socket.on('update_puerta', function(data) {
  const valor = data.valor;
  document.getElementById('emoji-puerta').textContent =
    valor === 1 ? '' : '';
  document.getElementById('estado-puerta').textContent =
    valor === 1 ? 'Abierta/Gente' : 'Cerrada/Nadie';
});
```

Eventos del servidor

El servidor Flask gestiona los siguientes eventos:

Tabla 7.1: Eventos SocketIO del dashboard.

Evento	Dirección	Descripción
`connect`	Cliente → Servidor	Cliente web conectado
`estado_inicial`	Servidor → Cliente	Envía estado actual
`update_luz`	Servidor → Cliente	Actualiza estado de luz
`update_puerta`	Servidor → Cliente	Actualiza estado de puerta
`publicar_luz`	Cliente → Servidor	Envía comando de luz
`publicar_puerta`	Cliente → Servidor	Envía comando de puerta
`mqtt_status`	Servidor → Cliente	Estado conexión MQTT

Instalación y Ejecución

Requisitos

Los requisitos de software para ejecutar el sistema completo son:

Listing 7.7: Contenido de requirements.txt

```
paho-mqtt>=1.6.0
pyserial>=3.5
flask>=2.0.0
flask-socketio>=5.0.0
```

Configuración de la Raspberry Pi

1. Deshabilitar la consola serial en `raspi-config` y habilitar el puerto “serial0“ para uso general.

2. Instalar el broker Mosquitto y Python 3:

```
sudo apt install mosquitto mosquitto-clients python3-pip
```

3. Habilitar Mosquitto para que arranque al inicio:

```
sudo systemctl enable mosquitto
```

4. Instalar dependencias Python:

```
pip3 install -r requirements.txt
```

5. Ejecutar el puente y el dashboard:

```
python3 mqtt_lora_bridge.py &
python3 web_dashboard.py
```

Acceso al dashboard

El dashboard es accesible desde cualquier dispositivo en la misma red local mediante:

`http://<IP_RASPBERRY>:5000`

La aplicación está configurada con `cors_allowed_origins="*"` para permitir conexiones desde cualquier origen durante el desarrollo.

Capítulo 8

Conclusiones

En este capítulo se presentan las conclusiones del proyecto, los objetivos alcanzados, las dificultades encontradas y las posibles líneas de mejora.

Objetivos Alcanzados

El proyecto ha cumplido satisfactoriamente los objetivos planteados al inicio:

- Se ha diseñado e implementado una arquitectura IoT completa que integra sensores físicos, comunicación LoRa, un broker MQTT y aplicaciones cliente.
- Se ha desarrollado un protocolo maestro-esclavo eficiente para la lectura de sensores ultrasónicos (SRF01/SRF02) y analógicos (LDR), con soporte para lecturas periódicas y configuración remota.
- Se ha implementado un formato de tramas LoRa con direccionamiento, identificadores de mensaje y sistema de ACK que garantiza la entrega fiable de comandos.
- Se ha desplegado un broker MQTT (Mosquitto) y un puente serie-MQTT que expone los datos de los sensores y permite el control del actuador desde aplicaciones externas.
- Se ha desarrollado un dashboard web interactivo que visualiza en tiempo real el estado de luz y puerta, y permite enviar comandos manuales.
- El sistema responde automáticamente a los cambios detectados por los sensores, encendiendo o apagando la luz y abriendo o cerrando la puerta según los umbrales configurados.

Dificultades Encontradas

Durante el desarrollo del proyecto se encontraron las siguientes dificultades:

- **Incompatibilidad de SoftwareSerial**: el Arduino UNO no dispone de un segundo puerto serie hardware, lo que obligó a utilizar la librería `SoftwareSerial` con sus limitaciones de velocidad y fiabilidad.

- **Gestión de energía del MKR WAN 1310**: las placas MKR WAN conectadas al módulo LoRa requieren una fuente de alimentación externa (batería) para manejar correctamente los picos de consumo durante la transmisión y evitar daños.
- **Sincronización de parámetros LoRa**: todos los nodos deben utilizar exactamente la misma configuración de frecuencia, ancho de banda, spreading factor y sync word. Pequeñas discrepancias impiden la comunicación.
- **Tiempos de propagación LoRa**: con SF10 y BW 62.5 kHz, el tiempo de transmisión de un paquete puede superar varios cientos de milisegundos, lo que requiere ajustar los timeouts de ACK y las políticas de rate limiting.
- **Compatibilidad de versiones de Paho MQTT**: la librería Paho MQTT para Python cambió la firma de los callbacks entre las versiones 1.x y 2.x, lo que requirió adaptar el código para soportar ambas.

Conocimientos Aplicados

El proyecto ha permitido aplicar de forma práctica conceptos de diversas áreas:

- **Comunicaciones inalámbricas**: configuración de módulos LoRa, comprensión de parámetros como spreading factor, ancho de banda y coding rate.
- **Protocolos de comunicación**: diseño de protocolos binarios compactos, sistemas de ACK/NACK, y manejo de duplicados.
- **Arquitectura IoT**: separación de responsabilidades entre capas (sensores, transporte, integración, presentación).
- **Sistemas embebidos**: programación de Arduino con restricciones de memoria y procesamiento, uso de interrupciones y temporizadores.
- **Desarrollo web**: creación de aplicaciones en tiempo real con Flask, SocketIO y WebSockets.
- **Integración de sistemas**: conexión de componentes heterogéneos mediante protocolos estándar (MQTT, Serial).

Líneas de Mejora

El sistema actual es funcional y cumple los requisitos académicos, pero existen varias áreas de mejora para un despliegue en producción:

- **Seguridad**: implementar cifrado AES-128 en la capa de aplicación LoRa y autenticación MQTT con TLS.
- **Persistencia**: almacenar el histórico de lecturas en una base de datos (InfluxDB, SQLite) para análisis posterior.
- **Escalabilidad**: soportar múltiples actuadores y sensores con descubrimiento automático de nodos.

- **Interfaz móvil**: desarrollar una aplicación nativa o PWA para control desde dispositivos móviles.
- **Integración con asistentes**: conectar con Home Assistant, Google Home o Alexa para control por voz.
- **Bajo consumo**: implementar modos de sleep en los nodos LoRa para maximizar la duración de baterías.
- **Redundancia**: añadir un segundo gateway para tolerancia a fallos.
- **Monitorización**: integrar métricas de sistema (RSSI, SNR, latencia) en un dashboard de operaciones.

Valoración Personal

El desarrollo de este proyecto ha sido una experiencia enriquecedora que ha permitido integrar conocimientos de múltiples asignaturas del grado. La combinación de hardware embebido, comunicaciones inalámbricas y desarrollo web ofrece una visión completa del ecosistema IoT.

El trabajo en equipo ha sido fundamental para abordar las distintas áreas del proyecto, distribuyendo las tareas según las fortalezas de cada miembro y manteniendo una comunicación fluida para integrar los componentes.

El sistema resultante, aunque sencillo en su funcionalidad (control de luz y puerta), demuestra que es posible construir soluciones domóticas completas con hardware de bajo coste y software libre, sentando las bases para proyectos más ambiciosos en el futuro.