Capítulo 17 · MQTT: jerarquía de temas, QoS, LWT y Mosquitto

Conceptos clave

Define temas por activo y variable. Usa LWT para anunciar desconexiones y adapta el QoS al efecto que tendría recibir duplicados.



GUÍA PASO A PASO DE LABORATORIO — MÓDULO 05

> **Despliegue de Broker Eclipse Mosquitto en Raspberry Pi, Endurecimiento de Seguridad y Puente Bidireccional Modbus RTU ↔ MQTT**

1. Objetivo General del Laboratorio

Implementar una arquitectura completa de mensajería industrial **Publicador/Suscriptor (MQTT v3.1.1 / v5.0)** sobre una **Raspberry Pi 4 Model B / 5**, integrando el nodo de campo **Arduino UNO R3 (Esclavo Modbus RTU ID=1)** construido en el Módulo 04 hacia un **Broker Eclipse Mosquitto 2.x** asegurado con autenticación por contraseña cifrada (`SHA-512`), listas de control de acceso por tópico (**ACL**) y un servicio demonio en Python 3 (`puente_modbus_a_mqtt_edge.py`) con exclusión mutua (`threading.Lock`), mensajes retenidos (**Retained**) y testamento de última voluntad (**LWT**).

2. Arquitectura de Red y Mapa de Tópicos del Laboratorio

Parámetro de Red / Serial	Valor de Producción en Laboratorio
Interfaz Serial RS-485 en Raspberry Pi	`/dev/ttyUSB0` (9600 bps, formato `8N1` , Half-Duplex)
Dirección Esclavo Modbus RTU (Arduino UNO)	`ID = 1` (Registros Holding `40001` a `40005`)
Dirección IP del Broker Mosquitto (RPi)	`127.0.0.1` (local) / `192.168.1.100` (LAN Planta)
Puertos TCP del Broker	`1883` (LAN autenticada) / `8883` (TLS 1.3 con X.509)
Raíz Jerárquica de Tópicos (ISA-95)	`invernadero/zona_norte/nodo_01/`
Tópico de Estado y Testamento (LWT)	`invernadero/zona_norte/nodo_01/estado/conexion` (`QoS 1, Retain=True`)
Tópico de Telemetría Agregada JSON	`invernadero/zona_norte/nodo_01/estado/resumen` (`QoS 1, Retain=False`)
Tópico de Mando Descendente (Actuadores)	`invernadero/zona_norte/nodo_01/actuadores/comando` (`QoS 1, Retain=False`)
Tópico de Confirmación de Escritura `0x06`	`invernadero/zona_norte/nodo_01/actuadores/confirmacion` (`QoS 1, Retain=False`)

3. Fase 1: Instalación del Broker Eclipse Mosquitto 2.x en Raspberry Pi OS

Conéctese por SSH a la Raspberry Pi y ejecute la actualización de repositorios e instalación tanto del servidor demonizado (`mosquitto`) como de las herramientas de diagnóstico por línea de comandos (`mosquitto-clients`):

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y mosquitto mosquitto-clients python3-pip python3-venv
sudo systemctl enable mosquitto
```

Verifique que el binario del servidor corresponde a la serie **Eclipse Mosquitto 2.x**, la cual por diseño de ciberseguridad **restringe el tráfico no autenticado a la interfaz `loopback` (`127.0.0.1`)** hasta que se configure explícitamente el archivo de seguridad:

```
mosquitto -h | head -n 3
sudo systemctl status mosquitto
```

---

4. Fase 2: Creación de Cuentas Cifradas (`mosquitto_passwd`) y Listas ACL

Nunca opere un broker industrial con `allow_anonymous true`. Crearemos tres identidades con privilegios segregados según el principio de mínimo privilegio:

- `puente_edge`: Cuenta exclusiva del demonio Python en la Raspberry Pi.
- `scada_central`: Cuenta del sistema supervisor / Node-RED con permiso de mando.
- `monitor_operador`: Cuenta de solo lectura para paneles de visualización.

```
# 1. Crear archivo de contraseñas cifradas con hash SHA-512 + Salt (-c crea el archivo)
sudo mosquitto_passwd -c /etc/mosquitto/passwd puente_edge
sudo mosquitto_passwd /etc/mosquitto/passwd scada_central
sudo mosquitto_passwd /etc/mosquitto/passwd monitor_operador

# 2. Copiar los archivos de configuración industrial y ACL del material complementario
sudo cp mosquitto_industrial.conf /etc/mosquitto/conf.d/mosquitto_industrial.conf
sudo cp acl_invernadero.conf /etc/mosquitto/acl_invernadero.conf

# 3. Asegurar permisos estrictos de propietario para que Mosquitto pueda leerlos
sudo chown mosquitto:mosquitto /etc/mosquitto/passwd /etc/mosquitto/acl_invernadero.conf
sudo chmod 0600 /etc/mosquitto/passwd /etc/mosquitto/acl_invernadero.conf

# 4. Reiniciar el broker y verificar que escucha en 0.0.0.0:1883
sudo systemctl restart mosquitto
sudo ss -tulpn | grep 1883
```

---

5. Fase 3: Verificación del Árbol de Tópicos, QoS y Retención por CLI

Abra **dos terminales SSH** en la Raspberry Pi para comprobar el enrutamiento y los permisos ACL antes de conectar el bus físico:

Terminal 1 (Suscriptor Supervisor con comodín multinivel `#`):

```
mosquitto_sub -h localhost -p 1883 \
  -u "scada_central" -P "TuClaveSeguraSCADA" \
  -t "invernadero/zona_norte/nodo_01/#" -q 1 -v
```

Terminal 2 (Prueba de publicación retenida y prueba de bloqueo ACL):

```
# Publicar configuración retenida (-r) con QoS 1 (-q 1)
mosquitto_pub -h localhost -p 1883 \
  -u "puente_edge" -P "ClaveSeguraIndustrial2026" \
  -t "invernadero/zona_norte/nodo_01/configuracion/limites" \
  -m '{"consigna_temp_c": 27.5, "histeresis_c": 1.0}' -q 1 -r

# Verificar que monitor_operador NO puede publicar comandos (ACL deniega escritura)
mosquitto_pub -h localhost -p 1883 \
  -u "monitor_operador" -P "ClaveOperador" \
  -t "invernadero/zona_norte/nodo_01/actuadores/comando" \
  -m '{"ventilador": 1}' -q 1
```

---

6. Fase 4: Despliegue del Puente Bidireccional Python (`Modbus RTU ⇔ MQTT`)

- Agregue el usuario `pi` al grupo `dialout` para acceso directo al conversor USB-RS485 (`/dev/ttyUSB0`):

  ```bash
  sudo usermod -a -G dialout pi
  ```

- Instale las bibliotecas de comunicación industrial en un entorno virtual o del sistema:

  ```bash
  pip3 install minimalmodbus pyserial paho-mqtt
  ```

- Copie `puente_modbus_a_mqtt_edge.py` a `/opt/iot_invernadero/` y ejecútelo inicialmente en primer plano para observar el ciclo de sondeo cada $2.0\ s$ (`Función 0x03`) y el mensaje de nacimiento (**Birth Message**):

  ```bash
  sudo mkdir -p /opt/iot_invernadero
  sudo cp puente_modbus_a_mqtt_edge.py /opt/iot_invernadero/
  sudo chown -R pi:pi /opt/iot_invernadero
  python3 /opt/iot_invernadero/puente_modbus_a_mqtt_edge.py
  ```

---

7. Fase 5: Prueba de Mando Bidireccional (`MQTT → Mutex → Modbus 0x06 → Relé`)

Con el puente en ejecución y el suscriptor abierto en la Terminal 1, envíe desde otra consola una orden JSON para encender el extractor forzado y actualizar la consigna térmica a $26.0\ ^\circ C$:

```
mosquitto_pub -h localhost -p 1883 \
  -u "scada_central" -P "TuClaveSeguraSCADA" \
  -t "invernadero/zona_norte/nodo_01/actuadores/comando" \
  -m '{"ventilador": 1, "bomba_riego": 0, "consigna_c": 26.0}' -q 1
```

Comportamiento esperado:

- El callback asíncrono `al_recibir_mensaje` adquiere `candado_serial` (`threading.Lock()`), esperando a que finalice cualquier lectura `0x03` en curso (~$32\ ms$).
- Envía la trama **Modbus** `0x06` **(** `Write Single Register` **)** al registro `40003` (valor `1`) y al registro `40005` (valor `260` , coma fija $\times\ 10$).
- El Arduino UNO R3 conmuta físicamente el **Relé K1** en la protoboard, encendiendo el **motor DC del ventilador extractor**, y devuelve el eco Modbus `0x06` .
- El puente libera `candado_serial` y publica en `.../actuadores/confirmacion` el objeto JSON con `"resultado": "OK_MODBUS_0x06"` .

---

8. Fase 6: Demonización 24/7 con `systemd` y Prueba de Testamento (LWT)

Instale la unidad `modbus-mqtt-bridge.service` para arranque automático y recuperación ante fallos:

```
sudo cp modbus-mqtt-bridge.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable modbus-mqtt-bridge.service
sudo systemctl start modbus-mqtt-bridge.service
sudo journalctl -u modbus-mqtt-bridge.service -f
```

Para validar el **Testamento de Última Voluntad (LWT)**, simule una caída abrupta del proceso enviando la señal `SIGKILL` (`kill -9`), la cual impide enviar el paquete `DISCONNECT` :

```
sudo kill -9 $(pgrep -f puente_modbus_a_mqtt_edge.py)
```

Tras $T_{LWT} = 1.5 \times T_{keepalive}$ (o inmediatamente al cerrarse el descriptor TCP del kernel), el Broker Mosquitto publicará automáticamente `"DESCONECTADO_INESPERADO"` con bandera retenida en `invernadero/zona_norte/nodo_01/estado/conexion`, y `systemd` relanzará el puente a los $5\ s$.

LISTADO DE MATERIALES (BOM) — MÓDULO 05

Hardware de Campo, Pasarela Edge Raspberry Pi, Red RS-485 y Software Broker MQTT

1. Unidad Central de Borde (Gateway IIoT & Broker MQTT)

Ítem	Cant.	Componente / Especificación Técnica	Función en la Arquitectura del Módulo 05
01	1	**Raspberry Pi 4 Model B (4 GB / 8 GB RAM)** o **Raspberry Pi 5**	Ejecuta en simultáneo el Broker **Eclipse Mosquitto 2.x** y el demonio **Puente Python Modbus ↔ MQTT**.
02	1	**Fuente Oficial USB-C 5.1 V / 3.0 A (15.3 W)** o Riel DIN 5 V / 4 A	Alimentación regulada libre de caídas de tensión durante ráfagas de red y escritura en disco.
03	1	**Tarjeta microSD Industrial A2 / U3 (32 GB)** o SSD NVMe por USB 3.0	Almacenamiento de Raspberry Pi OS (64-bit), base de persistencia `mosquitto.db` y registros `systemd`.
04	1	**Carcasa de Aluminio Pasiva o Riel DIN** + Cable Ethernet Cat 6 RJ45	Disipación térmica continua 24/7 y enlace cableado determinista hacia el switch de planta.

2. Interfaz de Acoplamiento Físico RS-485 (Borde ↔ Campo)

Ítem	Cant.	Componente / Especificación Técnica	Función en la Arquitectura del Módulo 05
05	1	**Conversor Industrial USB a RS-485** (Chipset FT232RL o CH340G con TVS)	Expone el puerto `/dev/ttyUSB0` en Linux con bornera de tornillo (`A+` , `B-` , `GND`).
06	1	**Módulo Transceptor MAX485CPA** (DIP-8 en protoboard o PCB con bornera)	Convierte niveles UART TTL ($0–5\ V$) del Arduino UNO a señal diferencial RS-485 ($\pm 1.5\ V$ a $\pm 5\ V$).
07	2	**Resistencias de Terminación** $R_T = 120\ \Omega$ **(**$1/4\ W$**, 1%)**	Adaptación de impedancia característica ($Z_0 = 120\ \Omega$) en ambos extremos físicos del bus.
08	2	**Resistencias de Polarización Fail-Safe (**$680\ \Omega$**,** $1/4\ W$**)**	Mantienen $V_{AB} > +200\ mV$ cuando todos los nodos están en alta impedancia (`DE=0`).
09	5 m	**Cable de Par Trenzado Apantallado (STP Belden 9841 / Cat 5e)**	Transporte diferencial de las líneas `A+` , `B-` y drenaje de referencia `GND` .

3. Nodo Esclavo de Campo en Protoboard (Arduino UNO ID=1 + Sensores y Actuadores)

Ítem	Cant.	Componente / Especificación Técnica	Pin Arduino / Registro Modbus / Tópico MQTT
10	1	**Arduino UNO R3 (ATmega328P, 16 MHz)** con cable USB-B	Esclavo Modbus RTU (`ID=1`): pines `D0 (RX)` , `D1 (TX)` y `D2 (DE/RE)` hacia el MAX485.
11	2	**Protoboards (Solderless Breadboards)** de 830 y 400 puntos	Montaje físico de relés, transceptor, divisores resistivos, pulsadores y diodos LED testigo.
12	2	**Relés de 5 V DC (`K52B-M-DC5` / Módulo 2 Canales Optoacoplado)** + `2N2222` + `1N4007`	Conmutación de potencia controlada por `D8` (**Reg 40003**, Extractor) y `D4` (**Reg 40004**, Bomba).
13	2	**Motores DC de 12 V con Engranaje Reductor / Ventilador Brushless**	Actuadores electromecánicos de extracción forzada y bombeo gobernados desde MQTT (`0x06`).
14	1	**Sensor de Temperatura NTC** $10\ k\Omega$ **/ Sonda Impermeable DS18B20**	Entrada `A0` : temperatura en décimas de grado (**Reg 40001** → `.../sensores/temperatura`).
15	1	**Fotorresistor LDR GL5528** + Resistencia $10\ k\Omega$ (Divisor)	Entrada `A1` : porcentaje de radiación lumínica (**Reg 40002** → `.../sensores/iluminacion`).
16	1	**Sensor Ultrasónico HC-SR04** (Nivel de estanque de riego)	Pines `D3 (TRIG)` y `A2 (ECHO)` : nivel de agua en cm (**Reg 40005** → `.../sensores/nivel_tanque`).
17	1	**Micro Servomotor SG90 (PWM** $50\ Hz$**)**	Pin `D5` : apertura proporcional de escotilla cenital (**Reg 40006**).
18	4	**Diodos LED 5 mm** (Azul, Verde, Ámbar, Rojo) + 4 Resistencias $220\ \Omega$	Pines `D9..D12` : indicadores visuales de tráfico MQTT, estado Online, trama Modbus y Alarma.
19	2	**Pulsadores Táctiles** $6\times 6\ mm$ + 2 Resistencias Pull-Down $10\ k\Omega$	Pines `D6` y `D7` : parada local de emergencia y rearme manual en protoboard.
20	1	**Fuente de Banco / Auxiliar DC (**$12\ V$ **/** $5\ V$**)** con masa común (`GND`)	Alimentación dedicada para bobinas de relés, servomotor y motores DC sin inducir caída en el MCU.

4. Pila de Software e Instrumentación Lógica

Paquete / Herramienta	Versión Recomendada	Propósito en el Laboratorio
Eclipse Mosquitto	`>= 2.0.18`	Broker MQTT ligero en C con soporte TLS 1.3, autenticación `SHA-512` y listas ACL.
mosquitto-clients	`>= 2.0.18`	Herramientas CLI `mosquitto_pub` , `mosquitto_sub` y `mosquitto_passwd` .
Python 3 + MinimalModbus	`Python 3.11+` / `2.1.1`	Maestro Modbus RTU sobre `/dev/ttyUSB0` con cálculo y verificación automática de CRC-16.
Eclipse Paho-MQTT Python	`>= 1.6.1 / 2.0`	Cliente asíncrono MQTT con soporte nativo de `will_set` (LWT), reconexión y niveles QoS 0/1/2.

MANUAL DE ARQUITECTURA DE TÓPICOS, CALIDAD DE SERVICIO (QoS) Y SEGURIDAD MQTT

Diseño de Espacios de Nombres ISA-95, Dimensionamiento de Tráfico, Retención, LWT y Endurecimiento de Mosquitto

---

1. Paradigma Publicador/Suscriptor frente a Sondeo Maestro-Esclavo

En el nivel de campo (**Tecnología de Operación — OT**), protocolos como **Modbus RTU sobre RS-485** utilizan un esquema *Maestro-Esclavo por sondeo secuencial (*polling*) **altamente determinista, pero con acoplamiento directo y una capacidad limitada a un único maestro físico en el bus. Cuando múltiples sistemas de nivel superior (**SCADA, Historiador de Base de Datos, Analítica en la Nube y Paneles Móviles**) necesitan consumir las variables del proceso simultáneamente, interrogar de forma directa el bus serial resulta inviable.

El protocolo *MQTT (*Message Queuing Telemetry Transport*, estándar OASIS / ISO/IEC 20922) **resuelve esta frontera mediante un** Broker centralizado **que desacopla a los productores de datos (**Publicadores**) de los consumidores (**Suscriptores**) en tres dimensiones fundamentales:

- **Desacoplamiento de Espacio:** El nodo de borde (Raspberry Pi) no necesita conocer las direcciones IP ni la cantidad de clientes SCADA o bases de datos existentes; únicamente publica en la IP del Broker.
- **Desacoplamiento de Tiempo:** Gracias a las sesiones persistentes (`clean_session=False` / `Session Expiry Interval`) y a los **mensajes retenidos (`retain=True`)**, un suscriptor puede conectarse horas después de la emisión y recibir instantáneamente el último estado conocido.
- **Desacoplamiento de Sincronización:** Las publicaciones y recepciones operan de manera asíncrona mediante eventos sin bloquear el bucle de adquisición de control.

---

2. Modelado Matemático de Sobrecarga y Caudal de Red en MQTT

Una trama MQTT posee una cabecera fija ultracompacta de tan solo $2\ \text{bytes}$ (1 byte de tipo de control y banderas + 1 a 4 bytes de longitud restante codificada en base 128). Para un mensaje de publicación (`PUBLISH`), el tamaño total en bytes de aplicación L_{MQTT} viene dado por:

$$L_{MQTT} = L_{fija} + \leq ft(2 + L_{tópico} \right) + L_{PacketID} + L_{payload}$$

Donde:

- $L_{fija} \in [2,\ 5]\ \text{bytes}$ es la cabecera fija.
- $2 + L_{tópico}$ corresponde a los 2 bytes de longitud UTF-8 más la cadena del tópico (por ejemplo, `invernadero/zona_norte/nodo_01/estado/resumen` tiene $L_{tópico} = 45\ \text{bytes}$).
- $L_{PacketID} = 0\ \text{bytes}$ para **QoS 0**, y $L_{PacketID} = 2\ \text{bytes}$ para **QoS 1** y **QoS 2**.
- $L_{payload}$ es la longitud útil del cuerpo JSON (por ejemplo, $160\ \text{bytes}$).

Si el sistema cuenta con N nodos publicando cada T_{pub} segundos hacia M suscriptores con un nivel de calidad de servicio $q \in \{0,\ 1,\ 2\}$, el número total de paquetes MQTT intercambiados por ciclo de publicación es:

$$N_{paquetes}(q,\ M) = 2^q \cdot (1 + M)$$

- **En QoS 0 ($q=0$):** $1\ \text{paquete}$ del emisor al broker (`PUBLISH`) + $M\ \text{paquetes}$ del broker a los suscriptores $= 1 + M$.

- **En QoS 1 ($q=1$):** 2 paquetes por salto (`PUBLISH` + `PUBACK`) $= 2 \cdot (1 + M)$.
- **En QoS 2 ($q=2$):** 4 paquetes por salto (`PUBLISH` → `PUBREC` → `PUBREL` → `PUBCOMP`) $= 4 \cdot (1 + M)$.

---

3. Reglas de Ingeniería para Jerarquía de Tópicos (Alineación ISA-95)

- **Estructura de General a Específico:**
  `sitio/area/unidad_borde/categoria/variable`
  Ejemplo: `invernadero/zona_norte/nodo_01/sensores/temperatura`
- **Prohibición de Barra Inicial (`/`):** Un tópico que inicia con `/invernadero/...` introduce un nivel vacío `""` innecesario en el árbol de enrutamiento del broker.
- **Caracteres ASCII Minúsculos y Guion Bajo (`snake_case`):** Evite espacios en blanco, tildes o caracteres especiales dentro del nombre del tópico para garantizar compatibilidad total con puentes SQL, InfluxDB y Kafka.
- **Uso Correcto de Comodines (*Wildcards — *Exclusivos para `SUBSCRIBE`):**
- **Comodín de Un Nivel (`+`):** Sustituye exactamente un nivel entre barras. Ejemplo: `invernadero/+/+/sensores/temperatura` suscribe la temperatura de todos los nodos en todas las zonas.
- **Comodín Multinivel (`#`):** Sustituye cero o más niveles y **debe ubicarse obligatoriamente al final del tópico**. Ejemplo: `invernadero/zona_norte/nodo_01/#` .

---

4. Semántica de Mensajes Retenidos (`Retain`) y Testamento (`LWT`)

Mecanismo	Configuración Recomendada	Propósito Operativo Industrial
Telemetría Periódica (`.../estado/resumen`)	`QoS = 1` , `Retain = False`	Evita que un cliente reciba una lectura histórica como si acabara de ocurrir en el milisegundo actual.
Parámetros y Límites (`.../configuracion/limites`)	`QoS = 1` , `Retain = True`	Permite que cualquier panel HMI recién abierto conozca de inmediato la consigna vigente sin esperar cambios.
Estado de Salud + LWT (`.../estado/conexion`)	`QoS = 1` , `Retain = True`	Combina el **Birth Message** (`"CONECTADO"`) con el **LWT** (`"DESCONECTADO_INESPERADO"`) almacenado en el Broker.
Comandos de Actuación (`.../actuadores/comando`)	`QoS = 1` , `Retain = False`	**Crítico:** Jamás usar `Retain = True` en tópicos de comando directo para evitar re-ejecuciones accidentales al reconectar el puente.

El tiempo máximo de detección de caída abrupta por parte del Broker Mosquitto obedece la especificación OASIS MQTT:

> $$T_{\text{detección LWT}} = 1.5 \times T_{\text{keepalive}}$$

Para $T_{\text{keepalive}} = 60\ \text{s}$, si el puente sufre un corte eléctrico o pérdida de enlace físico, el Broker espera exactamente $90\ \text{s}$ sin recibir paquetes de control ni `PINGREQ` antes de despachar el mensaje LWT a todos los clientes supervisores.

PLANO Y ESQUEMÁTICO DEL CIRCUITO — MÓDULO 05

> ## Montaje en Protoboard (Estilo Tinkercad / Fritzing), Placa de Circuito Impreso Soldada (PCB FR-4), Interconexión RS-485 y Concurrencia del Puente

1. Descripción General del Sistema Físico

El sistema físico del Módulo 05 integra dos dominios de hardware interconectados mediante un bus diferencial **RS-485 Half-Duplex**:

- **Dominio de Campo OT (Arduino UNO R3 en Protoboard o Placa PCB FR-4 Soldada):** Ejecuta el firmware esclavo Modbus RTU (`ID = 1`), adquiriendo en tiempo real la temperatura (`A0`), iluminación (`A1`) y nivel de estanque ultrasónico (`D3/A2`), y gobernando las etapas de potencia mediante relés (`K52B-M-DC5` / `SRD-05VDC-SL-C`) para el motor extractor (`D8`), la bomba de riego (`D4`), el servomotor de escotilla (`D5`) y la botonera/indicadores LED locales (`D6..D12`).
- **Dominio de Borde IT/OT (Conversor USB-RS485 + Raspberry Pi 4B):** Acopla el par trenzado diferencial (`A+` , `B-` , `GND`) hacia el puerto `/dev/ttyUSB0` , donde el servicio `puente_modbus_a_mqtt_edge.py` arbitra el puerto serie con un cerrojo de exclusión mutua (`threading.Lock`) y publica/suscribe hacia el **Broker Eclipse Mosquitto 2.x**.

2. Plano 1: Montaje Completo en Protoboard del Nodo Esclavo (Estilo Tinkercad / Fritzing)

📐 Plano 1: Montaje en Protoboard Estilo Tinkercad / Fritzing

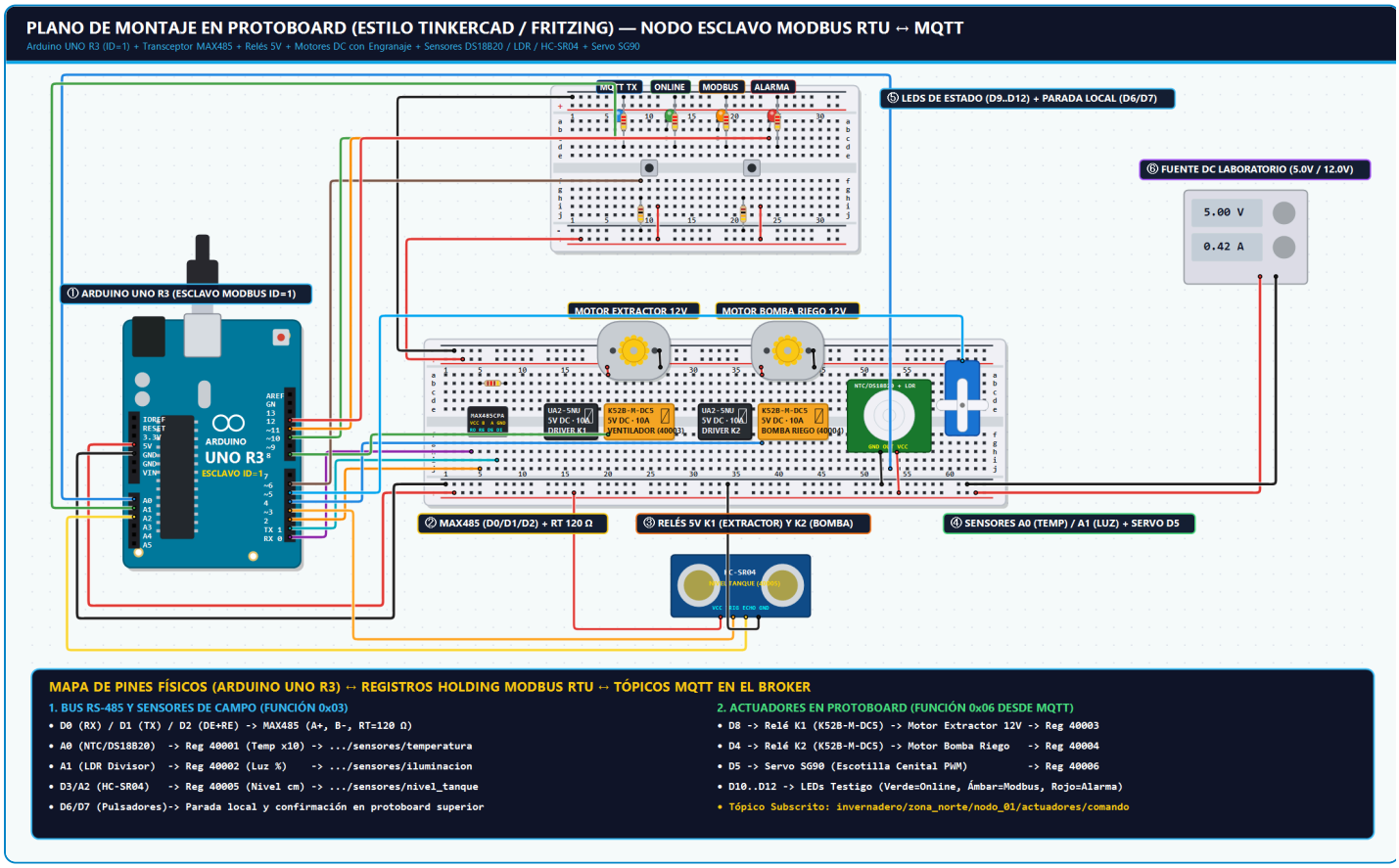


Tabla de Asignación de Pines del Arduino UNO R3 y Registros Modbus

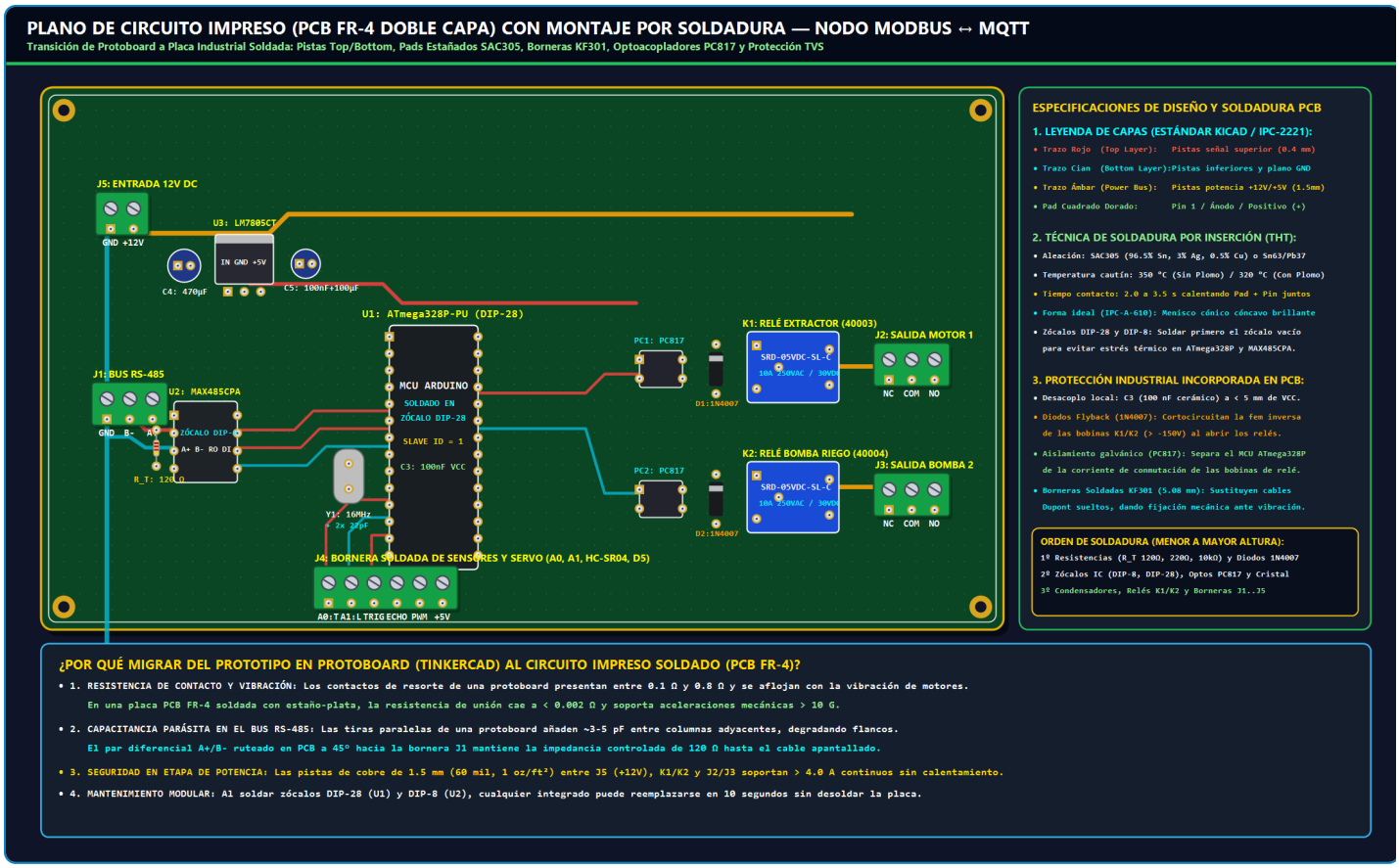
Pin Arduino UNO	Dispositivo en Protoboard / PCB	Registro Holding Modbus	Dirección de Datos y Tópico MQTT Asociado
`D0 (RX)` / `D1 (TX)`	Transceptor **MAX485** (`RO` / `DI`)	—	Tramas serie Modbus RTU a $9600\ bps$ (`8N1`).
`D2`	Transceptor **MAX485** (`DE + /RE` puenteados)	—	Control de dirección Half-Duplex (`HIGH` =TX, `LOW` =RX).
`A0`	Sensor Térmico **NTC / DS18B20**	`40001` (Índice 0)	Lectura `0x03` ($\times\ 10$) → `.../sensores/temperatura`
`A1`	Divisor Fotorresistivo **LDR +** $10\ k\Omega$	`40002` (Índice 1)	Lectura `0x03` ($0-100\ \%$) → `.../sensores/iluminacion`
`D8`	**Relé K1 (`K52B-M-DC5`)** → Motor Extractor 12V	`40003` (Índice 2)	Lectura `0x03` / Escritura `0x06` $\leq ftarrow$ `.../actuadores/comando`
`D4`	**Relé K2 (`K52B-M-DC5`)** → Motor Bomba Riego	`40004` (Índice 3)	Lectura `0x03` / Escritura `0x06` $\leq ftarrow$ `.../actuadores/comando`
`D3 (TRIG)` / `A2 (ECHO)`	Sensor Ultrasónico **HC-SR04** / Consigna	`40005` (Índice 4)	Nivel de estanque / Consigna térmica ($\times\ 10$) vía `0x06` .
`D5 (PWM)`	Micro Servomotor **SG90** (Escotilla)	`40006` (Índice 5)	Apertura proporcional de ventilación cenital ($0^\circ-90^\circ$).
`D6` , `D7`	Pulsadores Táctiles con Pull-Down $10\ k\Omega$	—	Parada local de emergencia y confirmación manual en planta.
`D9..D12`	Diodos LED 5 mm con resistencias de $220\ \Omega$	—	Indicadores luminosos: `MQTT TX` , `ONLINE` , `MODBUS` , `ALARMA` .

---

3. Plano 2: Circuito Impreso Industrial (PCB FR-4 Doble Capa) con Montaje por Soldadura

Para despliegues definitivos en invernaderos o naves industriales sometidas a vibraciones mecánicas, humedad relativa elevada y ciclos térmicos, el prototipo en protoboard se transfiere a una **placa de circuito impreso (PCB FR-4 de doble capa) con componentes soldados por inserción (`Through-Hole Technology - THT`) y bornes de tornillo (`KF301` de $5.08\ mm$)**:

📐 Plano 2: Circuito Impreso PCB FR-4 con Montaje por Soldadura



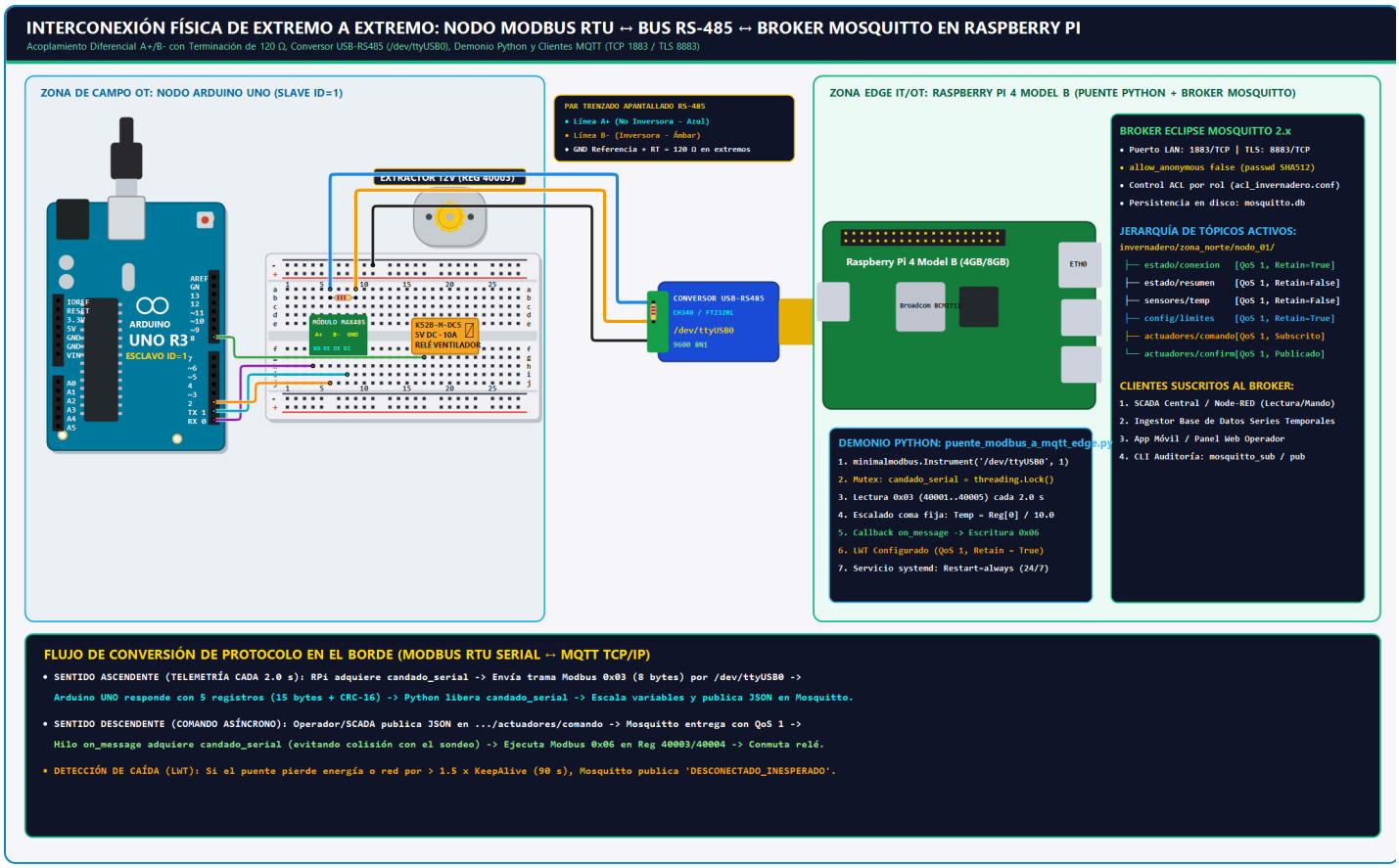
Especificaciones de Fabricación PCB y Proceso de Soldadura (`IPC-2221` / `IPC-A-610`)

Subsistema en PCB	Referencia Serigrafía	Especificación de Ruteo y Soldadura	Función de Ingeniería
Microcontrolador	`U1` (`ATmega328P-PU`) + `Y1` (16 MHz)	Soldar primero zócalo `DIP-28` sin el chip; insertar `U1` en frío. Condensador `C3` (100 nF) a < 5 mm de `VCC`.	Evita daño térmico al silicio durante la soldadura a $350\ ^\circ C$ y permite reemplazo rápido en campo.
Transceptor RS-485	`U2` (`MAX485CPA`) + `R_T` ($120\ \Omega$) + `J1`	Pistas diferenciales `A+` / `B-` de 0.6 mm ruteadas con codos a 45° hacia bornera soldada `J1`.	Elimina reflexiones de impedancia y capacitancias parásitas propias de las protoboards (3–5 pF).
Etapa de Potencia Aislada	`PC1/PC2` (`PC817`) + `D1/D2` (`1N4007`) + `K1/K2`	Pistas de potencia en cobre de 1.5 mm (60 mil, $1\ oz/ft^2$) entre relés `SRD-05VDC-SL-C` y borneras `J2` / `J3`.	Aísla galvánicamente el microcontrolador y suprime picos inductivos inversos (> -150 V) al desconectar motores.
Regulación de Energía	`U3` (`LM7805CT`) + `C4` ($470\ \mu F$) + `J5`	Pad cuadrado dorado identifica el terminal positivo (`+12V` / Ánodo). Aleación recomendada: `SAC305` o `Sn63/Pb37`.	Convierte la línea industrial de 12 V DC en 5.0 V estabilizados con resistencia de contacto $< 0.002\ \Omega$.

---

4. Plano 3: Acoplamiento Físico RS-485 hacia Raspberry Pi y Broker Mosquitto

📐 Plano 3: Acoplamiento RS-485 a Broker Mosquitto en Raspberry Pi



Reglas de Cableado e Integridad de Señal en el Bus RS-485

- **Polaridad Homóloga Directa (Sin Cruce):** A diferencia de UART punto a punto (donde `TX` se cruza con `RX`), en el bus diferencial RS-485 el borne `A+` **del módulo MAX485 se conecta directamente al borne** `A+` **del conversor USB-RS485**, y el borne `B-` **se conecta directamente al borne** `B-`.
- **Doble Terminación Resistiva ($R_T = 120\ \Omega$):** Instale una resistencia de $120\ \Omega$ entre `A+` y `B-` en el nodo Arduino UNO (final de línea) y otra de $120\ \Omega$ en la bornera del conversor USB-RS485 (cabecera de línea en la Raspberry Pi).
- **Referencia Común de Masa (`GND`):** Conecte siempre un tercer conductor de referencia `GND` entre la masa lógica del nodo de campo y el terminal `GND` del conversor USB-RS485 para mantener el voltaje de modo común dentro del rango operativo del transceptor ($-7\ V$ a $+12\ V$).

---

5. Plano 4: Jerarquía de Tópicos ISA-95, Niveles QoS y Máquina de Estados LWT

📐 Plano 4: Mapa de Jerarquía de Tópicos y Flujo QoS MQTT

ARQUITECTURA DE TÓPICOS JERÁRQUICOS ISA-95, COMODINES (+ / #), NIVELES QoS Y CICLO LWT

Diseño de Espacio de Nombres MQTT, Handshakes de Entrega (QoS 0, QoS 1, QoS 2), Mensajes Retenidos y Testamento de Última Voluntad

1. ÁRBOL JERÁRQUICO DE TÓPICOS (PLANTA → ZONA → NODO → RECURSO)

```
invernadero/                          [Nivel 1: Planta / Sitio]
└── zona_norte/                       [Nivel 2: Área / Célula]
    └── nodo_01/                      [Nivel 3: Unidad Edge / ID=1]
        ├── estado/
        │   ├── conexion              -> QoS 1 | Retain=True  (Online / LWT)
        │   └── resumen               -> QoS 1 | Retain=False (JSON Completo)
        ├── sensores/
        │   ├── temperatura           -> QoS 1 | Retain=False (24.5 °C)
        │   ├── iluminacion           -> QoS 1 | Retain=False (68 %)
        │   └── nivel_tanque          -> QoS 0 | Retain=False (142 cm)
        ├── configuracion/
        │   └── limites               -> QoS 1 | Retain=True  (Consigna 28.0 °C)
        └── actuadores/
            ├── comando               -> QoS 1 | Retain=False (Hacia Puente)
            └── confirmacion          -> QoS 1 | Retain=False (Eco Modbus 0x06)
```

REGLAS DE SUSCRIPCIÓN CON COMODINES (SOLO EN SUBSCRIBE):

- **Comodín Un Nivel (+):** `invernadero/+/+/sensores/temperatura`
- **Comodín Multinivel (#):** `invernadero/zona_norte/nodo_01/#`
- Antipatrones Prohibidos: Nunca iniciar con '/' ni usar espacios o '#'

2. NIVELES DE CALIDAD DE SERVICIO (QoS 0, QoS 1 Y QoS 2)

QoS 0 — A LO SUMO UNA VEZ (AT MOST ONCE / FIRE AND FORGET)

```
Emisor  ──────[ PUBLISH (QoS=0) ]──────►  Broker Mosquitto   (1 paquete MQTT, sin acuse)
```

Uso recomendado: Variables de alta frecuencia no críticas donde el siguiente muestreo reemplaza al anterior.

QoS 1 — AL MENOS UNA VEZ (AT LEAST ONCE — ESTÁNDAR INDUSTRIAL IIoT)

```
Emisor  ──────[ 1. PUBLISH (PacketId=K, DUP=0) ]──────►  Broker Mosquitto
Emisor  ◄──────[ 2. PUBACK  (PacketId=K)        ]──────  Broker Mosquitto   (2 paquetes MQTT)
```

Uso recomendado: Telemetría agregada, alarmas, LWT y comandos de actuación con idempotencia.

QoS 2 — EXACTAMENTE UNA VEZ (EXACTLY ONCE — HANDSHAKE DE 4 FASES)

```
Emisor ──[1. PUBLISH]──► Broker ──[2. PUBREC]──► Emisor ──[3. PUBREL]──► Broker ──[4. PUBCOMP]
```

Sobrecarga: 4 paquetes por mensaje. Reservado para facturación fiscal o dosificación no repetible.

Fórmula Paquetes Totales (1 Emisor + M Suscriptores en QoS q): $N_{paq} = (2^q) * (1 + M)$

3. MENSAJES RETENIDOS (RETAIN=TRUE) Y MÁQUINA DE ESTADOS DEL TESTAMENTO (LWT)

- FASE 1 (CONEXIÓN INICIAL): Puente envía paquete CONNECT registrando el LWT en RAM del Broker:
  Tópico LWT: .../estado/conexion | Payload: 'DESCONECTADO_INESPERADO' | QoS=1 | Retain=True
- FASE 2 (MENSAJE DE NACIMIENTO / BIRTH): Inmediatamente tras recibir CONNACK, el puente publica:
  Tópico: .../estado/conexion | Payload: 'CONECTADO' | QoS=1 | Retain=True
- FASE 3 (SUPERVISIÓN KEEP-ALIVE): El cliente envía PINGREQ si no hay tráfico en T_keepalive = 60 s.
- FASE 4 (DISPARO AUTOMÁTICO LWT): Si transcurre T_LWT = 1.5 * T_keepalive = 90 s sin PINGREQ ni DISCONNECT,
  el Broker Mosquitto cierra el socket y publica automáticamente el LWT retenido a todos los SCADA suscritos.

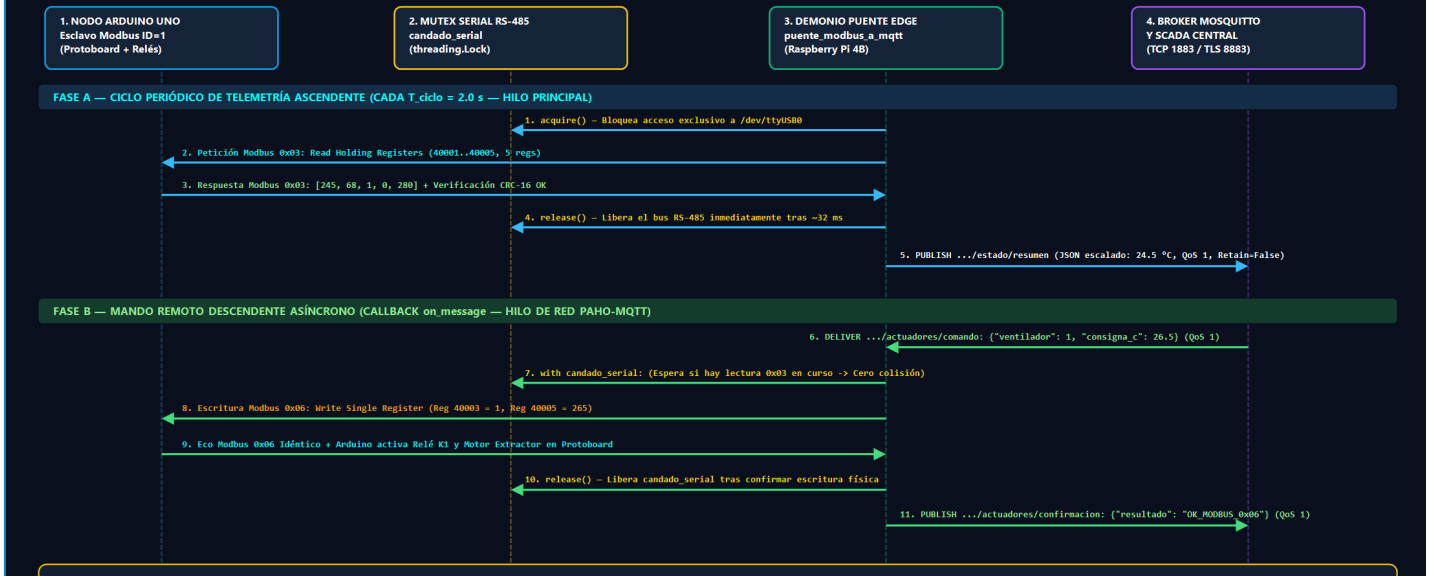
---

6. Plano 5: Secuencia Bidireccional y Exclusión Mutua (`threading.Lock`)

📐 Plano 5: Diagrama de Secuencia del Puente Bidireccional Modbus a MQTT

DIAGRAMA DE SECUENCIA BIDIRECCIONAL Y EXCLUSIÓN MUTUA (MODBUS RTU ↔ PUENTE PYTHON ↔ MOSQUITTO)

Arbitraje del Bus RS-485 con threading.Lock() entre el Bucle de Sondeo (0x03) y el Hilo Asíncrono de Comandos MQTT (0x06)



¿POR QUÉ ES OBLIGATORIO EL MUTEX (threading.Lock) EN EL PUENTE MODBUS ↔ MQTT?

- En paho-mqtt, client.loop_start() ejecuta las recepciones de red (on_message) en un hilo secundario asíncrono, mientras que el bucle principal while True ejecuta el sondeo periódico de los sensores cada 2.0 segundos.
- El bus RS-485 es estrictamente Half-Duplex: si el hilo on_message intenta enviar una trama 0x06 por /dev/ttyUSB0 en el mismo milisegundo en que el bucle principal espera los 15 bytes de respuesta de una lectura 0x03, los bytes de ambas tramas se entrelazan en el UART, corrompiendo el CRC-16 y provocando una excepción de Timeout.
- Con 'with candado_serial:', toda transacción Petición-Respuesta + Silencio de 3.5 caracteres es 100% ATÓMICA.
- Latencia máxima de encolamiento del comando: T_espera_max <= T_transaccion_0x03 (~32 ms a 9600 bps).