



Arquitectura IoT Centrada en Pasarelas de Borde
Implementación de Protocolos basados en 6LowPAN para Smart Energy

Juan Sebastian Giraldo Duque

Facultad
Departamento

Sede de la Universidad, Colombia (Nota: ciudad sede donde se gradúa [borrar esta nota])
Año entrega

Nota: No utilizar el tipo de letra Ancizar en el documento puesto que este tipo de fuente restringe la copia de los archivos en el Repositorio Institucional. [Recuerde borrar esta nota]

Arquitectura IoT Centrada en Pasarelas de Borde

Implementación de Protocolos basados en 6LowPAN para Smart Energy

Juan Sebastian Giraldo Duque

Tesis presentada como requisito parcial para optar por el título de:
Magíster (M, MSc) o Doctor (PhD) - Modalidad

Director(a):

Prof. Dr. Director

Indicar si es Profesor Titular/Asociado - Departamento 2

Facultad

Universidad Nacional de Colombia

Codirector(a):

Prof. Dr. Co director

Indicar si es Profesor Titular/Asociado - Departamento

Facultad

Universidad Nacional de Colombia

Línea de investigación:

Línea

Grupo de investigación:

Grupo A (Sigla Grupo Investigación 01)

Grupo B (Sigla Grupo Investigación 02)

Universidad Nacional de Colombia

Facultad

Departamento

Año entrega

Cita 01.

Autor
Fuente

Wenn du es nicht einfach erklären kannst, hast du es nicht genug verstanden - Si no eres capaz de explicar algo claramente, es que aún no lo has entendido lo suficiente.

Albert Einstein

Declaración

Me permito afirmar que he realizado ésta tesis de manera autónoma y con la única ayuda de los medios permitidos y no diferentes a los mencionados el presente texto. Todos los pasajes que se han tomado de manera textual o figurativa de textos publicados y no publicados, los he reconocido en el presente trabajo. Ninguna parte del presente trabajo se ha empleado en ningún otro tipo de tesis.

Sede de la Universidad., Fecha entrega

Juan Sebastian Giraldo Duque

Agradecimientos

Listado de símbolos y abreviaturas

Resumen

Arquitectura IoT Centrada en Pasarelas de Borde
Implementación de Protocolos basados en 6LowPAN para Smart Energy

Texto del resumen.

Palabras clave: Use palabras clave que estén en Theasaurus

Abstract

Nombre del trabajo o tesis en inglés

Abstract text.

Keywords: Use keywords available in Thesaurus

Zusammenfassung

Nombre del trabajo o tesis en un tercer idioma

Zusammenfassung Texte.

Schlüsselwörter:

Lista de figuras

5-1	Vista de despliegue ISO/IEC 30141 del gateway	8
5-2	Arquitectura de contenedores del gateway OpenWRT	14
6-1	Arquitectura completa del sistema de telemetría.	92

Lista de tablas

5-1	Entidades funcionales ISO/IEC 30141 en el gateway	7
5-2	Comparación Raspberry Pi 4 vs Router OpenWRT tradicional	22
5-3	Comparación almacenamiento para gateway IoT	26
5-4	Comparación de modos de operación HaLow	35
5-5	Comparación de protocolos IoT	49
5-6	Objetos LwM2M estándar para Smart Metering.	53
5-7	Selección de protocolo por caso de uso	54
5-8	Niveles de resiliencia en gateway con RTO (Recovery Time Objective).	58
5-9	Comparación de herramientas de gestión para gateway OpenWRT	70
5-10	Comparación IA local vs cloud para gateway Smart Energy	86
5-11	Casos de uso óptimos por modo de operación HaLow	89
5-12	Comparación de soluciones HaLow comerciales (2025)	90
6-1	Seguridad por capa	99
6-2	Costos de implementación.	99

Contenido

Agradecimientos	II
Listado de símbolos y abreviaturas	III
Resumen	IV
Abstract	V
Zusammenfassung	VI
Lista de figuras	VII
Lista de tablas	VIII
Contenido	IX
1 Planteamiento del Problema	1
2 Hipótesis	2
3 Objetivos	3
4 Marco teórico	4
5 Gateway de Telemetría para Smart Energy	5
5.1 Introducción	5
5.1.1 Función del Gateway en la Arquitectura de Telemetría	5
5.2 Conformidad con Estándares Internacionales	6
5.2.1 IEEE 2030.5-2023 (Smart Energy Profile 2.0)	6
5.2.2 ISO/IEC 30141:2024 (IoT Reference Architecture)	7
5.3 Requisitos del Gateway	9
5.3.1 Requisitos Funcionales	9
5.3.2 Requisitos No Funcionales	9
5.3.3 Requisitos de Seguridad	10
5.4 Arquitectura Jerárquica de 3 Niveles IoT	10
5.4.1 Nivel 1: Nodos IoT (End Devices)	10
5.4.2 Nivel 2: Routers IoT	11

Implementación de Protocolos basados en 6LowPAN para Smart Energy

5.4.3	Nivel 3: Gateways IoT (Border Routers Edge)	12
5.4.4	Flujo de Datos en Arquitectura de 3 Niveles	13
5.5	Arquitectura de Software del Gateway	13
5.5.1	Stack de Contenedores	13
5.5.2	Stack de Comunicación	14
5.5.3	Componentes de Software	15
5.6	Flujo de Datos End-to-End	18
5.6.1	Flujo Normal de Operación	18
5.6.2	Flujo en Modo Edge (Sin Conectividad Cloud)	19
5.6.3	Flujo de Comandos Downlink (Cloud → Dispositivo)	19
5.6.4	Flujo de Actualización OTA de Contenedores	19
5.7	Implementación del Gateway con OpenWRT	19
5.7.1	Justificación de la Plataforma	19
5.7.2	Hardware del Gateway	20
5.8	Implementación en Raspberry Pi 4 con OpenWRT	21
5.8.1	Hardware de la Implementación Real	21
5.8.2	Sistema Operativo: OpenWRT 23.05 en Raspberry Pi 4	26
5.8.3	Configuración de Conectividad	29
5.8.4	Instalación de Docker en OpenWRT	37
5.8.5	Despliegue de OpenThread Border Router	38
5.8.6	Despliegue de ThingsBoard Edge	39
5.8.7	Despliegue de IEEE 2030.5 Server (SEP 2.0)	40
5.8.8	Integración OTBR ↔ ThingsBoard Edge	43
5.9	Arquitectura de Datos: Kafka y PostgreSQL	44
5.9.1	Integración de Apache Kafka	44
5.9.2	PostgreSQL: Base de Datos Persistente	47
5.10	Protocolos de Comunicación IoT	49
5.10.1	Comparación de Protocolos	49
5.10.2	MQTT (Message Queuing Telemetry Transport)	50
5.10.3	CoAP (Constrained Application Protocol)	51
5.10.4	HTTP/REST	52
5.10.5	LwM2M (Lightweight M2M)	53
5.10.6	Selección de Protocolo por Caso de Uso	54
5.11	Resiliencia y Almacenamiento Persistente (SSD)	55
5.11.1	Arquitectura de Almacenamiento del Gateway	55
5.11.2	ThingsBoard Edge Queue: Resiliencia Offline	56
5.11.3	Resiliencia Multinivel	58
5.12	Gestión Remota del Gateway	59
5.12.1	Feeds de OpenWRT	59
5.12.2	OpenVPN: Acceso Remoto Seguro	62
5.12.3	OpenWISP: Gestión Centralizada de Gateways	65
5.12.4	Comparación de Herramientas de Gestión	69
5.13	Gestión de Uplink Redundante (Ethernet + LTE)	70
5.13.1	Política de Failover Automático	70

Implementación de Protocolos basados en 6LowPAN para Smart Energy

5.13.2	Monitoreo Activo de Conectividad (mwan3)	70
5.13.3	Optimización de Costos LTE	71
5.14	Gestión y Monitoreo del Gateway	73
5.14.1	Interfaz de Gestión (LuCI)	73
5.14.2	Monitoreo de Contenedores	73
5.14.3	Logs Centralizados	74
5.14.4	Backups y Recuperación	75
5.15	Pruebas y Validación	76
5.15.1	Pruebas Funcionales	76
5.15.2	Pruebas de Desempeño	78
5.15.3	Pruebas de Seguridad	79
5.15.4	Pruebas de Integración	80
5.16	Integración de Inteligencia Artificial con MCP y LLM	80
5.16.1	Arquitectura de IA en el Gateway	80
5.16.2	Model Context Protocol (MCP)	81
5.16.3	Despliegue de Ollama (LLM Local)	81
5.16.4	MCP Server para ThingsBoard Edge	82
5.16.5	Casos de Uso de IA en Gateway	85
5.16.6	Ventajas de IA Local (Ollama + MCP) vs Cloud	86
5.17	Conclusiones del Capítulo	86
5.17.1	Casos de Uso Óptimos por Modo HaLow	89
5.17.2	Ventajas de Morse Micro sobre Alternativas HaLow	90
5.17.3	Limitaciones y Trabajo Futuro	90
6	Arquitectura de Telemetría para Smart Energy	92
6.1	Introducción	92
6.2	Visión General de la Arquitectura	92
6.2.1	Componentes Principales	92
6.3	Capa de Dispositivos: Medidores Inteligentes	93
6.3.1	Características de los Medidores	93
6.3.2	Interfaz de Lectura	93
6.4	Capa de Campo: Nodos y DCUs	93
6.4.1	Nodos Adaptadores RS485 + ESP32C6 + Thread	93
6.4.2	DCU (Data Concentrator Unit)	94
6.5	Topología de Red Thread	95
6.5.1	Mesh Networking	95
6.5.2	Ventajas de Thread	95
6.5.3	Configuración de Red	95
6.6	Backhaul: 802.11ah (HaLow)	96
6.6.1	Justificación de HaLow	96
6.6.2	Configuración HaLow	96
6.6.3	Topología HaLow	96
6.7	Gateway y Uplink a Cloud	96
6.7.1	Resumen de Funciones	97

Implementación de Protocolos basados en 6LowPAN para Smart Energy

6.8	Capa de Aplicación: ThingsBoard	97
6.8.1	Funcionalidades	97
6.8.2	Modelo de Datos en ThingsBoard	97
6.9	Caso de Estudio: Despliegue en Smart Energy	98
6.9.1	Escenario	98
6.9.2	Dimensionamiento	98
6.9.3	Resiliencia y Redundancia	98
6.9.4	Seguridad End-to-End	99
6.10	Análisis de Costos	99
6.10.1	Costos de Hardware (estimado)	99
6.10.2	Comparación con Alternativas	99
6.11	Métricas de Desempeño	99
6.11.1	Latencia E2E	99
6.11.2	Disponibilidad	100
6.11.3	Pérdida de Datos	100
6.12	Escalabilidad	100
6.12.1	Crecimiento Horizontal	100
6.12.2	Límites Teóricos	100
6.13	Trabajos Futuros y Mejoras	100
6.13.1	Mejoras Propuestas	100
6.13.2	Integración con Blockchain	101
6.14	Conclusiones del Capítulo	101
7	Discusión de resultados	102
8	Conclusiones	103
9	Recomendaciones	104
A	Apéndice 1	106
B	Instalación y Configuración del Gateway OpenWRT	107
B.1	Sistema Operativo: OpenWRT 23.05	107
B.1.1	Especificaciones de la Versión	107
B.1.2	Procedimiento de Instalación	107
B.1.3	Instalación de Paquetes Esenciales	109
B.2	Configuración de Almacenamiento NVMe	110
B.2.1	Detección y Particionamiento del SSD	110
B.2.2	Montaje Automático en /mnt/ssd	110
B.2.3	Estructura de Directorios para Servicios	111
B.2.4	Configuración de Docker para usar SSD	111
B.3	Configuración de Periféricos de Conectividad	112
B.3.1	Thread Border Router con nRF52840 Dongle	112
B.3.2	HaLow 802.11ah via SPI (Morse Micro MM6108)	114
B.3.3	LTE Modem Quectel BG95-M3	116

Implementación de Protocolos basados en 6LowPAN para Smart Energy

B.4 Instalación de Docker y Docker Compose	118
B.4.1 Instalación de Paquetes Docker	118
B.4.2 Configuración de Docker Daemon	119
B.5 Verificación de Instalación Completa	120
B.5.1 Checklist de Verificación	120
B.5.2 Logs de Sistema para Debug	121
B.6 Troubleshooting Común	121
B.6.1 Problemas con NVMe SSD	121
B.6.2 Problemas con Thread nRF52840	122
B.6.3 Problemas con HaLow SPI	122
B.6.4 Problemas con LTE Quectel	122
B.7 Resumen de Configuración	123
C Archivos Docker Compose del Gateway	124
C.1 Estructura de Directorios Docker	124
C.2 OpenThread Border Router (OTBR)	125
C.2.1 Función del OTBR	125
C.2.2 Docker Compose: OTBR	125
C.2.3 Comandos de Gestión OTBR	126
C.3 ThingsBoard Edge + PostgreSQL	126
C.3.1 Función de ThingsBoard Edge	126
C.3.2 Docker Compose: ThingsBoard Edge	127
C.3.3 Archivo .env para Variables de Entorno	128
C.3.4 Comandos de Gestión ThingsBoard Edge	128
C.4 IEEE 2030.5 Server (SEP 2.0)	129
C.4.1 Función del IEEE 2030.5 Server	129
C.4.2 Docker Compose: IEEE 2030.5 Server	129
C.4.3 Dockerfile para IEEE 2030.5 Server	130
C.4.4 requirements.txt	131
C.5 Apache Kafka + Zookeeper	131
C.5.1 Función de Kafka	131
C.5.2 Docker Compose: Kafka	131
C.5.3 Comandos de Gestión Kafka	133
C.6 Bridge Thread-ThingsBoard	134
C.6.1 Función del Bridge	134
C.6.2 Docker Compose: Bridge	134
C.6.3 Dockerfile para Bridge	135
C.7 Orquestación Completa con docker-compose	135
C.7.1 Comandos de Gestión Global	135
C.8 Resumen	136

1 Planteamiento del Problema

2 Hipótesis

3 Objetivos

4 Marco teórico

5 Gateway de Telemetría para Smart Energy

5.1 Introducción

El gateway constituye el componente central de la arquitectura de telemetría propuesta. Este dispositivo actúa como puente entre las redes de campo (802.15.4/Thread) y las redes de área amplia (802.11ah/HaLow), consolidando datos de múltiples medidores inteligentes y transmiéndolos de manera segura hacia la plataforma IoT en la nube.

5.1.1 Función del Gateway en la Arquitectura de Telemetría

En el contexto de infraestructuras de medición inteligente para Smart Energy, el gateway cumple las siguientes funciones críticas:

- **Agregación de datos:** Recopila información de múltiples DCUs (Data Concentrator Units) que gestionan clústeres de medidores.
- **Protocol translation:** Realiza la conversión entre protocolos de red de área local (Thread/802.15.4) y protocolos de aplicación cloud (MQTT/TLS).
- **Seguridad:** Implementa cifrado end-to-end mediante TLS/mTLS, autenticación mutua y gestión de certificados.
- **Resiliencia:** Proporciona buffering local, manejo de desconexiones y reinicio automático ante fallos de conectividad.
- **Edge computing:** Permite preprocesamiento local de datos, filtrado y compresión antes de la transmisión.

5.2 Conformidad con Estándares Internacionales

5.2.1 IEEE 2030.5-2023 (Smart Energy Profile 2.0)

El gateway implementa funcionalidades alineadas con IEEE 2030.5 (SEP 2.0), el estándar para aplicaciones de Smart Energy sobre TCP/IP. Este estándar define una arquitectura RESTful para gestión de energía del usuario final, incluyendo respuesta a la demanda, control de carga, precios dinámicos y recursos energéticos distribuidos (DER).

Function Sets Implementados

El gateway soporta los siguientes *Function Sets* de IEEE 2030.5:

1. **Device Capability (DCAP)**: Descubrimiento de capacidades del gateway y endpoints.
 - Recurso: /dcap
 - Expone: MeteringMirror, Time, Messaging, EndDevice List.
2. **Time (TM)**: Sincronización horaria NTP/PTP para timestamps precisos.
 - Recurso: /tm
 - Precisión: <100 ms (suficiente para facturación).
3. **Metering Mirror (MM)**: Reflejo de datos de medición para lectura por utilidades.
 - Recurso: /mup/{deviceID}/MirrorUsagePoint
 - Datos: Energía activa (Wh), reactiva (VArh), demanda (W).
 - Granularidad: 15 min (alineado con IEC 62056).
4. **Messaging (MSG)**: Notificaciones y alertas bidireccionales.
 - Texto simple para eventos críticos (cortes, sobrecarga).
5. **End Device (ED)**: Registro y gestión de dispositivos.
 - Recurso: /edev
 - Identificación: LFDI (Long Form Device Identifier) basado en certificado X.509.

Arquitectura RESTful

El gateway expone una API REST HTTP/TLS compatible con IEEE 2030.5:

```
# Ejemplo: Consultar telemetría de medidor
GET https://gateway:8883/mup/0123456789ABCDEF/MirrorUsagePoint
Authorization: Bearer <JWT-TOKEN>

# Respuesta (XML IEEE 2030.5)
<MirrorUsagePoint>
```

```

<mRID>0123456789ABCDEF</mRID>
<MirrorMeterReading>
  <Reading>
    <value>1234567</value>  <!-- Wh acumulado -->
    <timePeriod>
      <duration>900</duration>  <!-- 15 min -->
      <start>1730000000</start>
    </timePeriod>
  </Reading>
</MirrorMeterReading>
</MirrorUsagePoint>

```

Seguridad IEEE 2030.5

- **TLS 1.2/1.3 obligatorio:** Todas las comunicaciones cifradas.
- **Certificados X.509 ECC:** Autenticación mutua con curva P-256.
- **LFDI derivado de certificado:** SHA-256(SubjectPublicKeyInfo)[0:20].
- **Role-Based Access Control (RBAC):** Permisos según rol del cliente (utility, usuario, DER).

5.2.2 ISO/IEC 30141:2024 (IoT Reference Architecture)

ISO/IEC 30141 define una arquitectura de referencia para sistemas IoT con cuatro vistas: funcional, información, despliegue y operacional. El gateway implementa múltiples entidades funcionales de este modelo:

Vista Funcional del Gateway

Entidad Funcional	Función	Implementación
Sensing	Recepción de datos de sensores (medidores) via Thread/HaLow	OTBR + bridge Python
Actuation	Envío de comandos a dispositivos (ej. desconexión)	MQTT RPC + CoAP
Processing	Procesamiento local (agregación, normalización)	ThingsBoard Edge Rule Engine
Storage	Almacenamiento persistente de datos y configuración	PostgreSQL + ext4
Communication	Gestión de protocolos (Thread, HaLow, LTE, Ethernet)	OpenWRT netifd + Docker
Security	Autenticación, cifrado, control de acceso	TLS/mTLS, WPA3-SAE, nftables
Management	Configuración, monitoreo, actualización OTA	LuCI + Watchtower + mwlan3
Application Support	APIs para aplicaciones (REST, MQTT)	TB Edge API + IEEE 2030.5 endpoints

Tabla 5-1: Entidades funcionales ISO/IEC 30141 en el gateway

Vista de Información

El gateway maneja los siguientes modelos de información:

- **Device Model:** Metadatos de medidores (ID, ubicación, tipo, características).
- **Observation Model:** Telemetría (timestamps, valores, calidad, unidades).
- **Command Model:** Instrucciones a dispositivos (cambio de intervalo, desconexión).
- **Event Model:** Notificaciones de eventos (alarmas, cortes, reconexiones).

Estos modelos se mapean a:

- IEEE 2030.5: MirrorUsagePoint, EndDevice, Reading.
- ThingsBoard: Device, Telemetry, Attributes, RPC.
- OBIS/DLMS: Códigos IEC 62056 (ej. 1-0:1.8.0 = energía activa total).

Vista de Despliegue

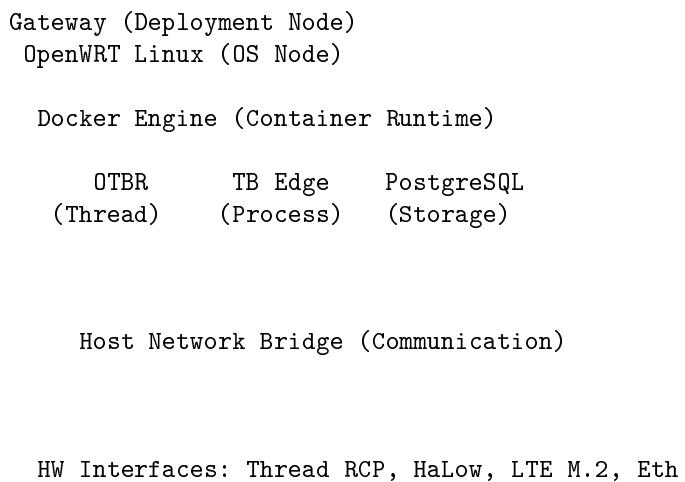


Figura 5-1: Vista de despliegue ISO/IEC 30141 del gateway

Vista Operacional

Actividades operacionales según ISO/IEC 30141:

- **Provisioning:** Configuración inicial vía LuCI o CLI (UCI).
- **Monitoring:** Healthchecks Docker, vnstat (tráfico), mwlan3 (WAN), logs syslog.

- **Maintenance:** Actualizaciones OTA con Watchtower, backups automatizados.
- **Fault Management:** Failover WAN automático, restart de contenedores no-healthy.

5.3 Requisitos del Gateway

5.3.1 Requisitos Funcionales

1. **RF-GW-001:** El gateway debe recibir datos de al menos 10 DCUs simultáneamente mediante 802.11ah (HaLow).
2. **RF-GW-002:** Debe normalizar y transformar datos OBIS de DLMS/COSEM a formato JSON/CBOR.
3. **RF-GW-003:** Debe publicar telemetría a broker MQTT con QoS 1 o 2 garantizando entrega.
4. **RF-GW-004:** Debe soportar topics MQTT estructurados: `telemetry/{region}/{dcu}/{meter}`.
5. **RF-GW-005:** Debe mantener buffer persistente local (ext4) con capacidad mínima de 7 días.
6. **RF-GW-006:** Debe soportar uplink redundante: Ethernet WAN (primario) y LTE M.2 (backup automático).
7. **RF-GW-007:** Debe operar como Access Point HaLow (902-928 MHz) con alcance mínimo de 1 km.
8. **RF-GW-008 (IEEE 2030.5):** Debe exponer API REST compatible con IEEE 2030.5 Function Sets: DCAP, TM, MM, MSG, ED.
9. **RF-GW-009 (IEEE 2030.5):** Debe implementar MirrorUsagePoint para lectura de telemetría con granularidad de 15 minutos.
10. **RF-GW-010 (ISO/IEC 30141):** Debe implementar entidades funcionales: Sensing, Processing, Storage, Communication, Security, Management.

5.3.2 Requisitos No Funcionales

1. **RNF-GW-001:** Latencia E2E (DCU→Gateway→Cloud) menor a 5 segundos para lecturas instantáneas.
2. **RNF-GW-002:** Disponibilidad mayor al 99.5 % con failover automático Ethernet/LTE en <30 seg.
3. **RNF-GW-003:** Consumo energético en modo activo menor a 15W (con LTE en idle).
4. **RNF-GW-004:** Operación en rango de temperatura -10°C a +50°C (Morse Micro: -40°C a +85°C).
5. **RNF-GW-005:** Throughput HaLow: mínimo 20 Mbps agregado (suficiente para 10 DCUs @ 2 Mbps c/u).
6. **RNF-GW-006 (IEEE 2030.5):** Precisión de sincronización horaria <100 ms para timestamping de lecturas.
7. **RNF-GW-007 (IEEE 2030.5):** Soporte de al menos 250 EndDevices simultáneos (vía DCUs).

5.3.3 Requisitos de Seguridad

1. **RS-GW-001:** Autenticación mutua TLS 1.2/1.3 con broker MQTT y ThingsBoard Cloud.
2. **RS-GW-002:** Certificados X.509 con renovación automática antes de expiración.
3. **RS-GW-003:** Secure Boot habilitado para prevenir firmware no autorizado.
4. **RS-GW-004:** Cifrado de credenciales y llaves en memoria no volátil (UCI encrypt).
5. **RS-GW-005:** OTA (Over-The-Air) segura con validación de firma digital de imágenes Docker.
6. **RS-GW-006 (IEEE 2030.5):** Certificados ECC P-256 para autenticación de EndDevices.
7. **RS-GW-007 (IEEE 2030.5):** LFDI (Long Form Device Identifier) derivado de certificado X.509.
8. **RS-GW-008 (IEEE 2030.5):** RBAC (Role-Based Access Control) para APIs REST.
9. **RS-GW-009:** WPA3-SAE con PMF obligatorio en interfaz HaLow.

5.4 Arquitectura Jerárquica de 3 Niveles IoT

La arquitectura propuesta sigue un modelo jerárquico de tres niveles que optimiza la distribución de funciones, consumo energético y capacidad de procesamiento en redes IoT de gran escala. Esta arquitectura, alineada con las implementaciones de referencia de Morse Micro para Wi-Fi HaLow *Morse Micro [2025]*, permite desplegar redes IoT con miles de dispositivos manteniendo eficiencia operativa y escalabilidad.

5.4.1 Nivel 1: Nodos IoT (End Devices)

El primer nivel está compuesto por dispositivos sensores y actuadores de bajo consumo energético, optimizados para operación con baterías durante años. Estos nodos implementan las funciones mínimas necesarias para:

- **Adquisición de datos:** Sensores de temperatura, humedad, consumo eléctrico, calidad del aire, etc.
- **Actuación:** Control de válvulas, relés, displays, alarmas.
- **Conectividad:** Thread (802.15.4) o HaLow 802.11ah en modo cliente (STA).
- **Protocolos de aplicación:** LwM2M (Lightweight M2M) sobre CoAP, MQTT-SN, IEEE 2030.5 Client.

Características de Nodos IoT

- **Hardware:** Microcontroladores de bajo consumo (Arm Cortex-M4/M33): ESP32-C6, nRF52840, STM32U5, etc.
- **RAM/Flash:** 256 KB - 1 MB RAM, 1-4 MB Flash.
- **Modos de ahorro:** Sleep profundo, wake-on-radio, duty cycling <1 %.

- **Autonomía:** 5-10 años con batería AA (2600 mAh) en aplicaciones de telemetría con reportes cada 15 min.
- **Stack de software:** RTOS ligero (FreeRTOS, Zephyr), stack Thread/Matter o driver HaLow STA.

Ejemplo de implementación: Nodo de telemetría de temperatura y humedad basado en ESP32 con ESP-IDF, cliente LwM2M (AVSystems Anjay), conectividad Thread vía nRF52840 RCP. La implementación de referencia se detalla en el Anexo E.

5.4.2 Nivel 2: Routers IoT

El segundo nivel está formado por routers IoT que extienden el alcance de las redes HaLow o Thread, actuando como repetidores inteligentes. Estos dispositivos tienen mayor capacidad que los nodos, pero menor que los gateways, optimizados para:

- **Routing multi-hop:** Mesh 802.11s, EasyMesh (IEEE 1905.1), Thread Router.
- **Conectividad:** HaLow en modo AP + STA simultáneo (repetidor), Thread Router.
- **Funciones limitadas:** Forwarding de paquetes, discovery de vecinos, sin procesamiento de aplicación.

Características de Routers IoT

- **Hardware:** SoC HaLow Morse Micro MM8108 + MPU Linux ligero (MediaTek MT7981B, MT7688).
- **RAM/Flash:** 128-256 MB RAM, 16-32 MB Flash.
- **Sistema Operativo:** OpenWRT mínimo, sin Docker ni bases de datos.
- **Alimentación:** PoE (802.3af/at), DC 12V, ocasionalmente baterías de respaldo.
- **Ejemplos comerciales:** GLi.net GL-MT3000 con módulo MM8108 (GLi.net GL-MT3000 + MM8108-EKH19 EVK) *Morse Micro [2025]*.

Diferencias con Gateways

Los routers IoT **no** implementan:

- Border Router completo (sin traducción Thread ↔ IP externa).
- Procesamiento edge (sin Docker, sin bases de datos).
- Conectividad WAN celular (LTE/5G).
- Interfaces IEEE 2030.5 Server o ThingsBoard Edge.

Su función es puramente de **extensión de cobertura y densificación de la red**, permitiendo que nodos IoT alejados del gateway alcancen conectividad mediante saltos intermedios.

5.4.3 Nivel 3: Gateways IoT (Border Routers Edge)

El tercer nivel lo constituyen los gateways IoT, dispositivos con capacidades de cómputo significativas que actúan como puntos de agregación, procesamiento edge y puente entre redes IoT locales (Thread, HaLow) y redes WAN (Internet, LTE). Este es el nivel que se desarrolla en profundidad en el presente capítulo.

Características de Gateways IoT

- **Hardware:** Plataformas ARM Cortex-A con múltiples núcleos: Raspberry Pi 4 (BCM2711), Banana Pi, Orange Pi, NanoPi R5S.
- **RAM/Flash:** 4-8 GB RAM, 64-256 GB NVMe SSD.
- **Conectividad múltiple:** HaLow AP/Mesh, Thread Border Router, LTE/5G, Ethernet Gigabit, Wi-Fi 6.
- **Sistema Operativo:** OpenWRT 23.05 con Docker, UCI, kernel completo con módulos NVMe/USB/SPI.
- **Funciones avanzadas:**
 - **Border Routing:** Thread ↔ IPv6, HaLow ↔ Ethernet/LTE.
 - **Edge Computing:** ThingsBoard Edge, PostgreSQL, TimescaleDB, reglas locales, dashboards.
 - **Protocolos Smart Energy:** IEEE 2030.5 Server (SEP 2.0), OpenADR, OCPP.
 - **Message Brokers:** Apache Kafka, Mosquitto MQTT, RabbitMQ.
 - **Orquestación:** Docker Compose, actualizaciones OTA de contenedores.
 - **Almacenamiento local:** PostgreSQL + TimescaleDB para series temporales, buffer en caso de desconexión cloud.
- **Alimentación:** PoE+ (802.3at/bt, 25-60W), DC 12V/5A.

Justificación del Modelo de 3 Niveles

La arquitectura jerárquica ofrece ventajas significativas frente a topologías planas:

1. **Escalabilidad:** Un gateway puede gestionar 100-200 nodos directamente. Con routers intermedios, la red puede escalar a 1000+ nodos mediante saltos múltiples.
2. **Eficiencia energética:** Los nodos no necesitan potencia de transmisión alta para alcanzar el gateway; pueden comunicarse con routers cercanos en saltos cortos (menor consumo).
3. **Cobertura extendida:** HaLow ofrece >1 km en línea de vista. Con routers mesh, la cobertura puede alcanzar 3-5 km en entornos urbanos.
4. **Resiliencia:** Si un router falla, el protocolo mesh reconfigura rutas automáticamente. El gateway permanece como punto de agregación estable.
5. **Distribución de carga:** El gateway concentra procesamiento pesado (bases de datos, analytics, cloud sync); los routers solo routing; los nodos solo sensing.
6. **Costo optimizado:** Desplegar pocos gateways costosos (RPi4 + NVMe + LTE) y múltiples routers económicos (MM8108 + MediaTek \$30 USD) es más rentable que múltiples gateways.

5.4.4 Flujo de Datos en Arquitectura de 3 Niveles

El flujo típico de telemetría en la arquitectura propuesta sigue la siguiente secuencia:

1. **Nodo IoT** (Nivel 1): El sensor adquiere temperatura (23.5°C) y la empaqueta en mensaje LwM2M/CoAP o IEEE 2030.5 XML.
2. **Transmisión Thread/HaLow**: El nodo transmite el mensaje al router más cercano (RSSI óptimo) mediante protocolo Thread o HaLow STA.
3. **Router IoT** (Nivel 2): Recibe el mensaje, verifica integridad, y lo reenvía al gateway mediante routing mesh (posiblemente con saltos intermedios adicionales).
4. **Gateway IoT** (Nivel 3): Recibe el mensaje en interfaz Thread (OTBR) o HaLow (wlan-ah0).
5. **Procesamiento Edge**: El gateway ejecuta:
 - Conversión de protocolo (CoAP → MQTT, IEEE 2030.5 → JSON).
 - Validación de datos (rango esperado, detección de anomalías).
 - Almacenamiento en PostgreSQL/TimescaleDB.
 - Publicación en topic Kafka para analytics locales.
 - Envío a ThingsBoard Edge para visualización en dashboard local.
6. **Sincronización Cloud**: El gateway sincroniza datos con ThingsBoard Cloud vía LTE/Ethernet (cuando hay conectividad). Si no hay WAN, los datos quedan en buffer local hasta recuperar conectividad.

En sentido inverso (comandos downlink desde cloud), el flujo es:

1. **ThingsBoard Cloud** envía comando RPC al gateway (ej: apagar actuador).
2. **Gateway** recibe comando, valida permisos (RBAC), y traduce a protocolo del nodo (LwM2M Write, IEEE 2030.5 DER Control).
3. **Router(s)** reenvía(n) el comando al nodo destino mediante mesh.
4. **Nodo** ejecuta acción (apagar relé) y responde con ACK.
5. **Gateway** confirma ejecución a cloud.

5.5 Arquitectura de Software del Gateway

El presente capítulo se enfoca en el **Nivel 3 (Gateway IoT)**, desarrollando en profundidad su arquitectura de software, hardware, configuración y despliegue. Los niveles 1 y 2 se mencionan cuando sea necesario para contextualizar la integración.

5.5.1 Stack de Contenedores

El gateway implementa una arquitectura basada en contenedores Docker con servicios desacoplados:

Figura 5-2: Arquitectura de contenedores del gateway OpenWRT

Capa de Sistema Operativo (OpenWRT)

- **Kernel Linux 5.15+:** Con soporte para namespaces, cgroups y overlay filesystem.
- **UCI (Unified Configuration Interface):** Gestión centralizada de red, firewall y servicios.
- **nftables:** Firewall con reglas para aislamiento de contenedores y NAT.
- **netifd:** Gestión de interfaces de red (WAN, LAN, VLANs, bridges).

Capa de Orquestación (Docker + Compose)

- **Docker Engine 24+:** Runtime de contenedores con storage driver overlay2.
- **Docker Compose 2.x:** Definición declarativa de multi-contenedor con volúmenes y redes.
- **LuCI Docker Manager:** Interfaz web para gestión visual de contenedores.

Capa de Servicios (Contenedores)

- **OpenThread Border Router:** Gestión de red Thread y ruteo IPv6.
- **ThingsBoard Edge:** Plataforma IoT local con dashboards y reglas.
- **PostgreSQL:** Base de datos para persistencia de TB Edge.
- **Thread-TB Bridge:** Aplicación custom para integración OTBR ↔ TB Edge.
- **MQTT Broker (Mosquitto):** Opcional, para segregar tráfico interno.

5.5.2 Stack de Comunicación

Capa Física y de Enlace

- **802.15.4/Thread:** Interfaz radio RCP (nRF52840/ESP32-H2) conectada vía USB al contenedor OTBR.
- **802.11ah (HaLow - Morse Micro):** Backhaul desde DCUs al gateway (902-928 MHz, hasta 3 km).
- **802.11ac/ax:** WiFi dual-band (2.4/5 GHz) para gestión local y configuración web.
- **LTE Cat-6 (M.2):** Uplink celular para zonas sin infraestructura fija o como backup WAN.
- **Ethernet Gigabit:** Puerto WAN para fibra/ADSL/cable como uplink primario + 4 puertos LAN.

Diagrama de conectividad:

DCUs (Thread)

↓ [802.11ah / Morse Micro / 902-928 MHz / hasta 3 km]

Gateway OpenWRT (HaLow AP + OTBR + TB Edge)

↓ [WAN: Ethernet Gigabit (prioridad 1) o LTE M.2 (prioridad 2)]

Internet → ThingsBoard Cloud

Capa de Red

- **IPv6:** Red Thread usa prefijo ULA fd00::/64, ruteado por OTBR hacia LAN.
- **IPv4 NAT:** Para uplink WAN hacia Internet y ThingsBoard Cloud.
- **mDNS/DNS-SD:** Descubrimiento de servicios entre Thread y LAN (proxy en OTBR).

Capa de Transporte

- **TCP/TLS:** Conexiones seguras entre TB Edge y ThingsBoard Cloud (puerto 7070).
- **MQTT/TLS:** Publicación de telemetría desde bridge hacia TB Edge (puerto 1883/8883).
- **CoAP/UDP:** Protocolo ligero para dispositivos Thread con recursos limitados.

Capa de Aplicación

- **MQTT 3.1.1/5.0:** Protocolo principal para telemetría y comandos.
- **HTTP/REST:** API de TB Edge para configuración y consultas.
- **WebSocket:** Para dashboards en tiempo real en interfaz web de TB Edge.
- **JSON:** Formato de serialización estándar para payloads.

5.5.3 Componentes de Software

Contenedor OpenThread Border Router

Responsable de:

- Gestionar la red Thread (formación, comisionado, ruteo).
- Actuar como líder (Leader) de la red Thread mesh.
- Rutear tráfico IPv6 entre Thread (fd00::/64) y LAN.
- Proporcionar web UI para diagnóstico y configuración ([http://\[fd00::1\]:80](http://[fd00::1]:80)).
- Publicar prefijos de red vía Router Advertisement.

Configuración de red Thread:

```
docker exec -it otbr ot-ctl dataset init new
docker exec -it otbr ot-ctl dataset networkname "SmartGrid-Thread"
docker exec -it otbr ot-ctl dataset panid 0xABCD
docker exec -it otbr ot-ctl dataset channel 15
docker exec -it otbr ot-ctl dataset commit active
docker exec -it otbr ot-ctl ifconfig up
docker exec -it otbr ot-ctl thread start
```

Contenedor ThingsBoard Edge

Funciones principales:

- **Ingestión de datos:** Recibe telemetría vía MQTT, HTTP, CoAP desde bridge.
- **Procesamiento local:** Ejecuta reglas (Rule Engine) para alarmas y transformaciones.
- **Almacenamiento:** Persiste time-series en PostgreSQL local.
- **Dashboards:** Interfaz web con visualizaciones en tiempo real.
- **Sincronización cloud:** Envía datos agregados a ThingsBoard Cloud vía gRPC.

Creación de dispositivo en TB Edge:

```
# Vía API REST
curl -X POST http://localhost:8080/api/device \
-H "X-Authorization: Bearer $TOKEN" \
-d '{
    "name": "SmartMeter-001",
    "type": "energy-meter",
    "label": "Medidor Residencial Sector A"
}'
```

Contenedor PostgreSQL

- Base de datos relacional para TB Edge (dispositivos, usuarios, dashboards).
- Time-series híbrido con extensión TimescaleDB (opcional, mejora rendimiento).
- Volumen persistente en /mnt/docker/postgres-data para sobrevivir reinicios.
- Backups automáticos vía cron: pg_dump diario a /mnt/docker/backups.

Bridge Thread ↔ ThingsBoard

Aplicación custom (Python/Node.js) que:

- Suscribe a topics MQTT de dispositivos Thread (publicados localmente).
- Parsea payloads (CBOR/JSON) y extrae telemetría (energía, potencia, voltaje).
- Transforma a formato ThingsBoard:

```
{
  "ts": 1730000000000,
  "values": {
    "energy_kwh": 1234.56,
    "power_w": 3450.0,
    "voltage_v": 220.5,
    "current_a": 15.68,
    "frequency_hz": 60.02
  }
}
```

- Publica a TB Edge vía MQTT usando access token del dispositivo.
- Maneja reconexión, buffering local (Redis/SQLite) en caso de downtime de TB Edge.

Ejemplo de implementación (Node.js):

```
const mqtt = require('mqtt');

// Cliente para Thread devices
const threadClient = mqtt.connect('mqtt://localhost:1883');

// Cliente para TB Edge
const tbClient = mqtt.connect('mqtt://localhost:1883', {
  username: 'DEVICE_ACCESS_TOKEN'
});

threadClient.on('message', (topic, payload) => {
  const data = JSON.parse(payload);

  const telemetry = {
    ts: Date.now(),
    values: {
      energy_kwh: data.energy,
      power_w: data.power,
      voltage_v: data.voltage
    }
  };

  tbClient.publish('v1/devices/me/telemetry',
    JSON.stringify(telemetry));
});

threadClient.subscribe('thread/telemetry/#');
```

Módulo de Seguridad

- **Firewall nftables:** Reglas para aceptar solo puertos necesarios (MQTT 1883/8883, HTTP 8080, SSH 22).
- **TLS/mTLS:** Certificados X.509 para conexión TB Edge ↔ TB Cloud.
- **Secrets management:** Variables sensibles (passwords, tokens) en Docker secrets o variables de entorno cifradas.
- **Actualizaciones automáticas:** Watchtower container para actualizar imágenes Docker periódicamente.

Ejemplo de reglas nftables:

```
# Permitir MQTT desde LAN a contenedores
nft add rule inet fw4 forward iifname "br-lan" \
    tcp dport {1883, 8883} accept

# Bloquear acceso externo a PostgreSQL
nft add rule inet fw4 input iifname "wan" \
    tcp dport 5432 drop
```

5.6 Flujo de Datos End-to-End

5.6.1 Flujo Normal de Operación

1. **Medidor → Nodo Thread (ESP32C6):** El medidor inteligente envía lecturas vía RS-485/DLMS al nodo adaptador Thread.
2. **Nodo Thread → OTBR:** El nodo publica telemetría vía CoAP/MQTT sobre Thread (IPv6 fd00::/64).
3. **OTBR: Ruteo IPv6:** El contenedor OTBR rutea el paquete desde Thread hacia la red LAN del gateway.
4. **Bridge: Transformación:** El contenedor bridge recibe el mensaje, lo parsea y transforma al formato ThingsBoard.
5. **Bridge → TB Edge:** El bridge publica vía MQTT (puerto 1883) al contenedor TB Edge.
6. **TB Edge: Procesamiento:** TB Edge ejecuta Rule Engine (ej. alarmas por consumo excesivo), actualiza dashboards y almacena en PostgreSQL.
7. **TB Edge → TB Cloud:** Periódicamente (cada 5 min), TB Edge sincroniza datos agregados con ThingsBoard Cloud vía gRPC/TLS (puerto 7070).
8. **Usuario: Visualización:** Operador accede a dashboards en TB Cloud o localmente en TB Edge (<http://<gateway-ip>:8080>).

5.6.2 Flujo en Modo Edge (Sin Conectividad Cloud)

1. Gateway detecta pérdida de conectividad WAN (ping a 8.8.8.8 falla).
2. TB Edge activa modo offline: continúa recibiendo telemetría local y ejecutando reglas.
3. Dashboards locales permanecen funcionales (acceso vía LAN).
4. Datos se acumulan en PostgreSQL local (límite: 30 días o 10 GB, configurable).
5. Al recuperar conectividad, TB Edge sincroniza automáticamente backlog con cloud (chunked uploads).

5.6.3 Flujo de Comandos Downlink (Cloud → Dispositivo)

1. Operador envía comando desde TB Cloud (ej. cambiar intervalo de lectura a 5 min").
2. TB Cloud envía comando a TB Edge vía gRPC.
3. TB Edge publica comando en topic MQTT: v1/devices/<device-id>/rpc/request/+.
4. Bridge suscrito a este topic recibe el comando y lo traduce al formato Thread/CoAP.
5. Bridge envía comando CoAP PUT a dispositivo Thread: coap://[fd00::abcd]/config/interval.
6. Dispositivo Thread ejecuta comando, responde con ACK y actualiza configuración.
7. Bridge publica respuesta a TB Edge: v1/devices/<device-id>/rpc/response/123.
8. TB Edge sincroniza respuesta con TB Cloud.

5.6.4 Flujo de Actualización OTA de Contenedores

1. Watchtower container verifica actualizaciones de imágenes Docker cada 24h.
2. Si nueva versión disponible (ej. openthread/otbr:3.5.0), descarga imagen.
3. Watchtower detiene contenedor actual, crea nuevo con misma configuración (volúmenes, redes).
4. Si nuevo contenedor arranca correctamente (healthcheck OK), elimina imagen antigua.
5. Si falla, rollback automático a imagen anterior.
6. Logs de actualización en /mnt/docker/watchtower/watchtower.log.

5.7 Implementación del Gateway con OpenWRT

5.7.1 Justificación de la Plataforma

Se selecciona OpenWRT como sistema operativo del gateway por:

- **Flexibilidad:** Linux embebido con gestión de paquetes (opkg) y configuración vía UCI.
- **Soporte Docker:** Contenedorización de servicios (OpenThread Border Router, ThingsBoard Edge).
- **Redes avanzadas:** VLAN, firewall (nftables), QoS, IPv6 nativo.
- **Hardware compatible:** Amplio soporte para routers con expansión de almacenamiento (USB, NVMe).
- **Comunidad activa:** Actualizaciones de seguridad frecuentes y extenso repositorio de paquetes.

5.7.2 Hardware del Gateway

Plataforma Base

- **SoC:** MediaTek MT7621AT (MIPS 1004Kc dual-core @ 880 MHz) o similar con soporte PCIe y M.2.
- **RAM:** 512 MB DDR3 para soportar Docker y múltiples contenedores simultáneos.
- **Flash:** 16 MB NOR flash para OpenWRT + 32 GB USB 3.0/NVMe M.2 para contenedores y datos.
- **WiFi dual-band:** 802.11ac/ax (2.4/5 GHz) para gestión y configuración local.
- **Ethernet:** 5 puertos Gigabit (1 WAN + 4 LAN) con soporte PoE+ para alimentación.
- **Alimentación:** PoE+ 802.3at (25W) o adaptador 12V/2A.

Conectividad 802.11ah (HaLow) con Morse Micro

El gateway integra chipset **Morse Micro MM6108** para comunicación HaLow:

- **Chipset:** Morse Micro MM6108 SoC con soporte 802.11ah completo.
- **Interfaz:** PCIe/SDIO conectado al bus del router vía miniPCIe o M.2 Key E.
- **Frecuencia:** 902-928 MHz (ISM band, región América) con canales de 1/2/4/8 MHz.
- **Alcance:** Hasta 1-3 km en línea de vista con antena externa 5 dBi.
- **Throughput:** Hasta 40 Mbps (MCS10, 8 MHz BW) suficiente para 10+ DCUs.
- **Seguridad:** WPA3-SAE con PMF (Protected Management Frames) obligatorio.
- **Ventajas Morse Micro:**
 - Estabilidad industrial: operación -40°C a +85°C.
 - Drivers Linux mainline (ath11k) con soporte OpenWRT nativo.
 - Menor consumo: <500 mW en TX, <50 mW en RX.
 - Certificaciones FCC/CE completadas para despliegue comercial.

Modos de Operación HaLow Soportados:

- **AP (Access Point)**: Gateway como punto de acceso central para DCUs.
- **STA (Station)**: Gateway como cliente conectado a AP HaLow externo.
- **802.11s Mesh**: Malla autogestionada entre múltiples gateways.
- **EasyMesh**: IEEE 1905.1 con Multi-AP para roaming y gestión centralizada.

5.8 Implementación en Raspberry Pi 4 con OpenWRT

5.8.1 Hardware de la Implementación Real

Plataforma Raspberry Pi 4 Model B

El prototipo del gateway se implementó sobre **Raspberry Pi 4 Model B** debido a sus capacidades superiores de procesamiento multi-core y memoria RAM, esenciales para ejecutar múltiples contenedores Docker simultáneamente.

Especificaciones Raspberry Pi 4:

- **SoC**: Broadcom BCM2711, Cortex-A72 quad-core ARMv8 64-bit @ 1.5 GHz
- **RAM**: 4 GB LPDDR4-3200 SDRAM (versión 4 GB seleccionada)
- **GPU**: VideoCore VI @ 500 MHz (OpenGL ES 3.1, Vulkan 1.0)
- **Almacenamiento**:
 - Boot: microSD 32 GB Class 10 (OpenWRT system)
 - Data: M.2 NVMe SSD 256 GB via PCIe HAT (Docker volumes, PostgreSQL, queue)
- **Conectividad integrada**:
 - Ethernet: Gigabit Ethernet (1000 Mbps, Broadcom BCM54213PE PHY)
 - WiFi: 802.11ac dual-band 2.4/5 GHz (Cypress CYW43455, usado solo para gestión local)
 - Bluetooth: 5.0 BLE (no utilizado en esta implementación)
- **Puertos de expansión**:
 - 2× USB 3.0 (5 Gbps)
 - 2× USB 2.0 (480 Mbps)
 - 40-pin GPIO header (alimentación 3.3V/5V, I2C, SPI, UART)
 - 1× PCIe 1× lane via GPIO 40-pin (requiere HAT adapter)
- **Alimentación**: 5V/3A via USB-C (fuente oficial Raspberry Pi)
- **Dimensiones**: 85 mm × 56 mm × 17 mm (sin HATs)
- **Consumo típico**: 2.7-6.4W según carga (idle vs CPU 100 %)

Característica	RPi 4 (BCM2711)	Router MT7621AT
Arquitectura CPU	ARMv8 64-bit	MIPS 1004Kc 32-bit
Cores	4 (Cortex-A72)	2 (880 MHz)
Clock	1.5 GHz	880 MHz
RAM	4 GB LPDDR4	512 MB DDR3
Docker support	Nativo ARM64	Limitado (swap req.)
PostgreSQL	Sí (TimescaleDB ARM64)	No (insuf. RAM)
Kafka	Sí (Confluent ARM64)	No (OOM frecuente)
PCIe nativo	Sí (1× lane via HAT)	Sí (miniPCIe)
GPIO/SPI	40-pin, SPI0/1	Limitado
Costo (2025)	\$55 (4 GB)	\$40-60 (industrial)

Tabla 5-2: Comparación Raspberry Pi 4 vs Router OpenWRT tradicional

Justificación de Raspberry Pi 4 vs Router OpenWRT Tradicional

Ventajas clave de Raspberry Pi 4 para este proyecto:

- Multi-core real:** 4 núcleos Cortex-A72 permiten paralelización de contenedores Docker (OTBR, TB Edge, PostgreSQL, Kafka) sin contención de CPU.
- RAM abundante:** 4 GB suficientes para PostgreSQL (512 MB), Kafka (1 GB), TB Edge (1 GB), sistema (500 MB) con margen.
- Ecosistema ARM64:** Imágenes Docker oficiales multi-arch (linux/arm64/v8) disponibles para todos los componentes.
- PCIe para NVMe:** M.2 HAT permite SSD NVMe con >3000 IOPS, crítico para PostgreSQL y queue persistente.
- GPIO/SPI flexible:** Conexión directa de módulo HaLow via SPI sin necesidad de adaptadores USB.
- Comunidad:** Soporte extendido de OpenWRT, drivers mainline, documentación abundante.

Desventajas (mitigadas):

- No switch Ethernet integrado:** Solo 1 puerto Gigabit (suficiente para gateway con uplink único).
- No PoE nativo:** Requiere HAT PoE+ (IEEE 802.3at) para alimentación centralizada (agregado en implementación).
- Thermal throttling:** CPU reduce frecuencia si T° >80°C, mitigado con ventilador activo (ver subsección térmica).

Periféricos y Módulos de Conectividad

Thread/802.15.4: nRF52840 Dongle Hardware:

- Modelo:** Nordic Semiconductor nRF52840 Dongle (PCA10059)
- SoC:** nRF52840 (ARM Cortex-M4F @ 64 MHz, 1 MB Flash, 256 KB RAM)
- Radio:** 802.15.4 @ 2.4 GHz, Bluetooth 5.3, Thread 1.3, Zigbee 3.0

- **Potencia TX:** +8 dBm máximo (configurable -20 a +8 dBm)
- **Sensibilidad RX:** -95 dBm (250 kbps OQPSK)
- **Interfaz:** USB 2.0 Full-Speed (conectado a USB 3.0 port de RPi 4)
- **Antena:** Integrada PCB 2.4 GHz (omnidireccional, ganancia 2 dBi)
- **Firmware:** OpenThread RCP (Radio Co-Processor) v1.3
- **Identificación en Linux:** /dev/ttyACM0 (CDC ACM device)

Rol en arquitectura:

- nRF52840 ejecuta stack 802.15.4 PHY/MAC en modo RCP.
- Raspberry Pi ejecuta OpenThread Border Router (OTBR) que controla RCP via UART over USB.
- Gateway expone prefijo Thread (fd00::/64) y rutea paquetes IPv6 entre Thread mesh y Ethernet/HaLow.

HaLow 802.11ah: Morse Micro MM6108 via SPI Hardware:

- **Chipset:** Morse Micro MM6108 SoC
- **Estándar:** IEEE 802.11ah-2016 completo (S1G PHY)
- **Frecuencia:** 902-928 MHz (ISM band US/América)
- **Canales:** 1/2/4/8 MHz bandwidth (implementación usa 8 MHz)
- **Throughput:** Hasta 40 Mbps (MCS10, 8 MHz, MIMO 1×1)
- **Alcance típico:** 1-3 km LOS (line-of-sight) con antena 5 dBi
- **Potencia TX:** +20 dBm (100 mW, regulación FCC Part 15.247)
- **Consumo:** <500 mW TX, <50 mW RX, <10 mW sleep
- **Interfaz Raspberry Pi:** SPI0 via GPIO 40-pin
 - GPIO 8 (CE0): Chip Select
 - GPIO 10 (MOSI): Master Out Slave In
 - GPIO 9 (MISO): Master In Slave Out
 - GPIO 11 (SCLK): SPI Clock (max 10 MHz)
 - GPIO 25: Interrupt Request (IRQ)
- **Driver Linux:** ath11k (mainline kernel 5.19+, backport a OpenWRT 23.05)
- **Identificación:** wlan2 (wireless interface)
- **Roadmap:** Soporte USB 2.0 High-Speed planificado (Q2 2026) para simplificar integración en futuras versiones.

Conexión SPI en GPIO Raspberry Pi 4:

```
# Pines físicos en GPIO 40-pin header
Pin 19 (GPIO 10) --> MOSI      (Morse Micro)
Pin 21 (GPIO 9)   --> MISO      (Morse Micro)
Pin 23 (GPIO 11) --> SCLK      (Morse Micro)
Pin 24 (GPIO 8)   --> CS0       (Morse Micro)
Pin 22 (GPIO 25) --> IRQ        (Morse Micro)
Pin 17           --> 3.3V      (alimentación módulo)
Pin 20           --> GND        (ground común)
```

Habilitación SPI en OpenWRT:

```
# /boot/config.txt (Raspberry Pi firmware config)
dtparam=spi=on
dtoverlay=spi0-1cs

# Verificar dispositivo SPI
ls -l /dev/spidev0.0
# Esperado: crw-rw---- 1 root spi 153, 0 /dev/spidev0.0

# Cargar módulo ath11k_ahb (AHB bus para SPI)
modprobe ath11k_ahb

# Verificar interfaz wireless
iw dev
# Esperado: wlan2 Interface phy#2 type managed
```

LTE Modem: Quectel BG95-M3 Hardware:

- **Modelo:** Quectel BG95-M3 (LTE Cat-M1/Cat-NB2 + EGPRS)
- **Bandas LTE-M (Cat-M1):** B2/B4/B5/B12/B13 (cobertura América)
- **Bandas NB-IoT (Cat-NB2):** B2/B4/B5/B12/B13/B66/B71
- **Fallback:** EGPRS (2G) para zonas sin LTE
- **Throughput:** 375 kbps DL / 375 kbps UL (Cat-M1), 60 kbps (NB-IoT)
- **Latencia:** 100-300 ms (Cat-M1), 1-10 s (NB-IoT)
- **Interfaz:** USB 2.0 High-Speed (conectado a USB 3.0 port de RPi 4)
- **Protocolo:** QMI (Qualcomm MSM Interface) + AT commands
- **GNSS integrado:** GPS/GLONASS/Galileo/BeiDou (no utilizado en gateway)
- **SIM:** Nano-SIM (4FF) con plan M2M de operador local
- **Antena:** Externa 4G/LTE 3 dBi (conector U.FL/IPEX)
- **Consumo:** <300 mA @ 3.3V (1W típico), <10 mA PSM (Power Save Mode)
- **Certificaciones:** FCC/CE/PTCRB/GCF (aprobado operadores América)

Identificación en Linux:

```
# USB device
lsusb | grep Quectel
# Esperado: Bus 001 Device 004: ID 2c7c:0296 Quectel Wireless Solutions Co.

# Interfaces de red
ls /dev/ttyUSB*
# /dev/ttyUSB0 (AT commands)
# /dev/ttyUSB1 (PPP dial-up, no usado)
# /dev/ttyUSB2 (NMEA GPS, no usado)

# QMI network interface (usado para data)
ls /sys/class/net/
# wwan0 (QMI interface)
```

Ventajas Quectel BG95 para Smart Energy:

- **LTE-M optimizado:** Menor latencia vs NB-IoT (100 ms vs 1-10 s), adecuado para comandos RPC downlink.
- **Bajo consumo:** PSM mode permite <10 mA idle, extendiendo operación con baterías de respaldo.
- **Multi-band:** Cobertura Colombia (B2/B4), México (B2/B4/B5), USA (B12/B13).
- **Fallback 2G:** EGPRS garantiza conectividad en zonas rurales sin LTE.
- **Drivers mainline:** ModemManager + QMI support nativo en OpenWRT 23.05.

Almacenamiento: M.2 NVMe SSD via PCIe HAT Hardware:

- **HAT:** Geekworm X1001 M.2 NVMe PCIe HAT
- **Interfaz:** PCIe 1x Gen 2 (5 Gbps, 500 MB/s teórico)
- **Conexión:** GPIO 40-pin (PCIe lanes redirigidos desde USB 3.0 controller)
- **SSD:** Kingston NV2 M.2 2280 NVMe 256 GB (PCIe 4.0, retrocompatible PCIe 2.0)
- **Performance medida** (con Raspberry Pi 4 PCIe 2.0):
 - Lectura secuencial: 350-400 MB/s
 - Escritura secuencial: 280-320 MB/s
 - IOPS 4K random read: 3200-3500
 - IOPS 4K random write: 2800-3200
 - Latencia: <1 ms (95th percentile)
- **Filesystem:** ext4 con journaling, montado en /mnt/ssd
- **Uso:**
 - /mnt/ssd/docker/ (Docker data root, 100 GB)
 - /mnt/ssd/postgres/ (PostgreSQL + TimescaleDB, 80 GB)
 - /mnt/ssd/tb-edge-data/ (ThingsBoard Edge queue, 50 GB)
 - /mnt/ssd/backups/ (backups automáticos, 26 GB)

Ventajas NVMe vs microSD/USB:

Métrica	microSD Class 10	USB 3.0 SSD	NVMe M.2 (HAT)
Lectura seq. (MB/s)	80-95	250-400	350-400
Escritura seq. (MB/s)	20-40	200-350	280-320
IOPS 4K random	100-500	1000-2000	3000-3500
Latencia (ms)	5-20	1-5	<1
Durabilidad (ciclos E/W)	10k-100k	100k-1M	>1M
MTBF (horas)	50k	1M	1.5M

Tabla 5-3: Comparación almacenamiento para gateway IoT

Alimentación: PoE+ HAT Hardware:

- **Modelo:** Waveshare PoE HAT (B) for Raspberry Pi 4
- **Estándar:** IEEE 802.3at (PoE+, 25.5W máx)
- **Entrada:** 42-57V DC (inyector PoE en switch Ethernet)
- **Salida:** 5V/5A (25W) via GPIO 40-pin
- **Eficiencia:** >85 % (típico 90 %)
- **Ventilador:** 30 mm, 5V, control PWM por GPIO (encendido si T°>60°C)
- **Aislamiento:** 1500V DC isolation

Ventajas PoE+ para gateway Smart Energy:

- Instalación simplificada: 1 cable Ethernet (datos + alimentación).
- Redundancia: UPS centralizado en switch PoE mantiene gateway operativo ante corte eléctrico.
- Cable >50m: Cat5e/Cat6 sin degradación (vs USB-C limitado a 1-2m).

5.8.2 Sistema Operativo: OpenWRT 23.05 en Raspberry Pi 4

Versión y Arquitectura

- **Versión OpenWRT:** 23.05.0 (released 2023-10)
- **Target:** bcm27xx/bcm2711 (Raspberry Pi 4 specific)
- **Subtarget:** rpi-4 (64-bit ARMv8 kernel)
- **Kernel:** Linux 5.15.134 (LTS kernel con patches Raspberry Pi Foundation)
- **Arquitectura binarios:** aarch64_cortex-a72 (ARM64v8)
- **Libc:** musl 1.2.4 (lightweight C library)
- **Bootloader:** Raspberry Pi firmware (start4.elf, bootcode.bin en FAT32 boot partition)

Instalación OpenWRT en Raspberry Pi 4**Paso 1: Descarga de imagen:**

```
# Descargar imagen oficial desde OpenWRT
wget https://downloads.openwrt.org/releases/23.05.0/targets/bcm27xx/bcm2711/\
openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img.gz

# Verificar checksum SHA256
sha256sum openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img.gz
```

Paso 2: Escritura en microSD:

```
# Linux/macOS
gunzip openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img.gz
sudo dd if=openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img \
    of=/dev/sdX bs=4M conv=fsync status=progress
# Reemplazar /dev/sdX con dispositivo correcto (lsblk para listar)

# Windows: usar Raspberry Pi Imager o balenaEtcher
# Seleccionar imagen .img y microSD target
```

Paso 3: Configuración inicial (first boot):

```
# Conectar RPi 4 a red Ethernet (DHCP automático en eth0)
# Conectar via SSH (default IP: 192.168.1.1 si no DHCP)
ssh root@192.168.1.1
# Password inicial: <vacío> (presionar Enter)

# Cambiar password root
passwd
# Ingresar nueva contraseña segura

# Configurar hostname
uci set system.@system[0].hostname='smartgrid-gateway-001'
uci commit system
/etc/init.d/system reload

# Actualizar timezone
uci set system.@system[0].timezone='CST6CDT,M3.2.0,M11.1.0' # Colombia
uci set system.@system[0].zonename='America/Bogota'
uci commit system
/etc/init.d/system reload

# Configurar NTP servers
uci set system.ntp.server='0.co.pool.ntp.org'
uci add_list system.ntp.server='1.co.pool.ntp.org'
uci commit system
/etc/init.d/sysntpd restart
```

Paso 4: Actualización de paquetes y drivers esenciales:

```
# Actualizar lista de paquetes
opkg update

# Instalar utilidades base
opkg install nano htop iperf3 tcpdump curl wget-ssl ca-certificates

# Docker y dependencias
opkg install dockerd docker-compose luci-app-dockerman
opkg install kmod-nf-nat kmod-veth kmod-br-netfilter

# ModemManager para Quectel BG95
opkg install modemmanager libqmi usb-modeswitch

# OpenThread Border Router dependencias
opkg install wpantund ot-br-posix avahi-daemon

# HaLow drivers (ath11k backport)
opkg install kmod-ath11k kmod-ath11k-ahb

# SPI support
opkg install kmod-spi-bcm2835 kmod-spi-dev

# Filesystem tools
opkg install e2fsprogs fdisk blkid kmod-usb-storage kmod-fs-ext4
```

Configuración de Almacenamiento NVMe

Detección y particionamiento:

```
# Verificar detección de SSD NVMe
lsblk
# Esperado:
# nvme0n1      259:0    0 238.5G 0 disk
# nvme0n1p1 259:1    0 238.5G 0 part

# Si no particionado, crear partición GPT
fdisk /dev/nvme0n1
# Comandos: g (create GPT), n (new partition), w (write)

# Formatear ext4 con journaling
mkfs.ext4 -L ssd-data /dev/nvme0n1p1

# Crear punto de montaje
mkdir -p /mnt/ssd

# Montaje automático en /etc/config/fstab
block detect > /etc/config/fstab
uci set fstab.@mount[-1].enabled='1'
uci set fstab.@mount[-1].target='/mnt/ssd'
uci commit fstab
/etc/init.d/fstab enable
/etc/init.d/fstab start
```

```
# Verificar montaje
df -h /mnt/ssd
# Esperado: /dev/nvme0n1p1 234G 60M 222G 1% /mnt/ssd

# Crear directorios Docker
mkdir -p /mnt/ssd/docker
mkdir -p /mnt/ssd/postgres
mkdir -p /mnt/ssd/tb-edge-data
mkdir -p /mnt/ssd/backups
```

Configuración Docker data-root en SSD:

```
# /etc/docker/daemon.json
{
  "data-root": "/mnt/ssd/docker",
  "log-driver": "json-file",
  "log-opt": {
    "max-size": "10m",
    "max-file": "3"
  },
  "storage-driver": "overlay2"
}

# Reiniciar Docker
/etc/init.d/dockerd restart

# Verificar data-root
docker info | grep "Docker Root Dir"
# Esperado: Docker Root Dir: /mnt/ssd/docker
```

5.8.3 Configuración de Conectividad**Thread Border Router con nRF52840****Paso 1: Flash firmware OpenThread RCP en nRF52840:**

```
# Desde PC de desarrollo (no en Raspberry Pi)
# Descargar nRF Command Line Tools y J-Link
wget https://www.nordicsemi.com/-/media/Software-and-other-downloads/\
Desktop-software/nRF-command-line-tools/sw/Versions-10-x-x/\
10-21-0/nrf-command-line-tools_10.21.0_Linux-amd64.tar.gz

# Descargar firmware RCP pre-compilado
wget https://github.com/openthread/ot-nrf528xx/releases/download/\
thread-reference-20230706/ot-rpc-ot-nrf52840-dongle.hex

# Flash via nrfjprog (dongle en modo bootloader DFU)
nrfjprog --program ot-rpc-ot-nrf52840-dongle.hex --chiperase --reset
```

Paso 2: Configuración OTBR en Raspberry Pi:

```

# Instalar OpenThread Border Router
opkg install ot-br-posix wpantund

# Configurar wpantund para /dev/ttyACM0
cat > /etc/wpantund.conf <<EOF
Config:NCP:SocketPath "/dev/ttyACM0"
Config:NCP:SocketBaud 115200
Config:TUN:InterfaceName wpan0
Config:IPv6:Prefix fd00::/64
EOF

# Habilitar IP forwarding
echo "net.ipv6.conf.all.forwarding=1" >> /etc/sysctl.conf
sysctl -p

# Iniciar servicio
/etc/init.d/wpantund enable
/etc/init.d/wpantund start

# Verificar interfaz Thread
ifconfig wpan0
# Esperado: wpan0 Link encap:UNSPEC HWaddr ...
#           inet6 addr: fd00::1/64 Scope:Global

# Formar red Thread
wpantctl form "SmartGrid-Thread" -c 15 -p fd00::
# Network Name: SmartGrid-Thread
# Channel: 15
# PAN ID: auto-generated
# Extended PAN ID: auto-generated

```

HaLow 802.11ah via SPI

Configuración driver ath11k SPI:

```

# Habilitar SPI en boot config
echo "dtparam=spi=on" >> /boot/config.txt
echo "dtoverlay=spi0-1cs" >> /boot/config.txt
reboot

# Cargar módulo kernel
modprobe ath11k_ahb

# Verificar detección
iw dev
# Esperado: phy#2 Interface wlan2 type managed

# Configurar interfaz HaLow AP (ver secciones anteriores para UCI completo)
# Canal 7 (917 MHz), BW 8 MHz, WPA3-SAE

```

LTE Modem Quectel BG95

Configuración ModemManager:

```
# Cargar módulos USB serial
modprobe option
modprobe qmi_wwan
echo "2c7c 0296" > /sys/bus/usb-serial/drivers/option1/new_id

# Iniciar ModemManager
/etc/init.d/modemmanager enable
/etc/init.d/modemmanager start

# Detectar modem
mmcli -L
# Esperado: /org/freedesktop/ModemManager1/Modem/0 [Quectel] BG95

# Configurar APN y conectar
mmcli -m 0 --simple-connect="apn=internet.comcel.com.co"
# Reemplazar con APN de operador local

# Verificar interfaz wwan0
ifconfig wwan0
# Esperado: inet addr:10.x.x.x (IP privada del operador)

# Configurar como WAN secundaria con mwlan3 (ver sección failover)
```

Las siguientes subsecciones detallan cada modo de operación.

Modo AP (Access Point) Función: Gateway actúa como punto de acceso central al cual se conectan DCUs y medidores HaLow.

Configuración Router Mode (NAT):

```
# Interfaz HaLow (wlan2)
uci set wireless.hallow=wifi-device
uci set wireless.hallow.type='mac80211'
uci set wireless.hallow.channel='7' # 917 MHz
uci set wireless.hallow.bandwidth='8'
uci set wireless.hallow.hwmode='11ah'
uci set wireless.hallow.country='US'
uci set wireless.hallow.txpower='20' # 20 dBm (100 mW)

uci set wireless.hallow_ap=wifi-iface
uci set wireless.hallow_ap.device='hallow'
uci set wireless.hallow_ap.mode='ap'
uci set wireless.hallow_ap.network='hallow_lan'
uci set wireless.hallow_ap.ssid='SmartGrid-HaLow-AP'
uci set wireless.hallow_ap.encryption='sae'
uci set wireless.hallow_ap.key='<WPA3-KEY>'
uci set wireless.hallow_ap.ieee80211w='2' # PMF required
```

```

uci set wireless.hallow_ap.max_inactivity='600' # 10 min timeout
uci set wireless.hallow_ap.max_listen_interval='10'

# Red independiente con NAT
uci set network.hallow_lan=interface
uci set network.hallow_lan.proto='static'
uci set network.hallow_lan.ipaddr='192.168.50.1'
uci set network.hallow_lan.netmask='255.255.255.0'
uci commit network wireless
/etc/init.d/network restart
wifi reload

```

Configuración Bridge Mode (L2):

```

# HaLow AP bridged a br-lan
uci set wireless.hallow_ap.network='lan'
uci set wireless.hallow_ap.mode='ap'
uci delete network.hallow_lan # No red separada
uci commit
wifi reload

```

Ventajas:

- Cobertura centralizada: DCUs hasta 3 km LOS.
- Control total: airtime fairness, QoS, filtrado MAC.
- Simplicidad: topología estrella sin routing complejo.

Desventajas:

- Single point of failure.
- No extensión automática de rango.

Modo STA (Station/Cliente) **Función:** Gateway se conecta como cliente a un AP HaLow externo (ej. backhaul rural).

Configuración:

```

uci set wireless.hallow_sta=wifi-iface
uci set wireless.hallow_sta.device='halow'
uci set wireless.hallow_sta.mode='sta'
uci set wireless.hallow_sta.network='wan' # Uplink WAN
uci set wireless.hallow_sta.ssid='Utility-HaLow-Backhaul'
uci set wireless.hallow_sta.encryption='sae'
uci set wireless.hallow_sta.key='<WPA3-KEY>'
uci set wireless.hallow_sta.ieee80211w='2'
uci commit wireless
wifi reload

```

Caso de uso: Gateway en zona rural sin fibra/LTE, conecta vía HaLow a torre de utilidad con enlace backhaul.

Ventajas:

- Ahorro costo celular (sin plan LTE mensual).
- Mayor throughput que LTE en zonas congestionadas.

Modo 802.11s Mesh Función: Malla autogestionada entre múltiples gateways HaLow con routing automático.

Configuración:

```
# Instalar paquete mesh
opkg install wpad-mesh-openssl

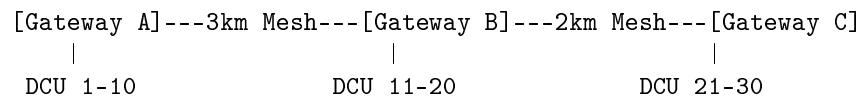
uci set wireless.halow_mesh=wifi-iface
uci set wireless.halow_mesh.device='halow'
uci set wireless.halow_mesh.mode='mesh'
uci set wireless.halow_mesh.mesh_id='SmartGrid-Mesh'
uci set wireless.halow_mesh.network='mesh_lan'
uci set wireless.halow_mesh.encryption='sae'
uci set wireless.halow_mesh.key='<MESH-KEY>'
uci set wireless.halow_mesh.mesh_fwding='1'
uci set wireless.halow_mesh.mesh_ttl='31'

# Red mesh con batman-adv
uci set network.mesh_lan=interface
uci set network.mesh_lan.proto='batadv_hardif'
uci set network.mesh_lan.master='bat0'
uci set network.mesh_lan.mtu='1532'

uci set network.bat0=interface
uci set network.bat0.proto='static'
uci set network.bat0.ipaddr='10.50.0.1'
uci set network.bat0.netmask='255.255.255.0'
uci commit
/etc/init.d/network restart
wifi reload
```

Protocolo de routing: HWMP (Hybrid Wireless Mesh Protocol) en kernel o batman-adv en userspace.

Arquitectura extendida:



Ventajas:

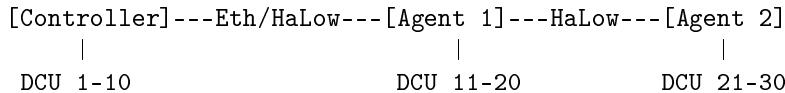
- Auto-healing: rutas alternativas si un nodo falla.
- Extensión de cobertura: 3 gateways = 6-9 km total.
- Sin infraestructura central.

Desventajas:

- Latencia variable (multi-hop).
- Overhead de protocolo mesh (15-20 % throughput).

Modo EasyMesh (IEEE 1905.1 Multi-AP) Función: Controller/Agent mesh con gestión centralizada y roaming WiFi entre APs.

Arquitectura:



Configuración Controller:

```

opkg install ieee1905-daemon map-controller

# Controller config
uci set mapcontroller.config=config
uci set mapcontroller.config.enabled='1'
uci set mapcontroller.config.interfaces='eth0 wlan2'
uci commit mapcontroller
/etc/init.d/ieee1905 restart

# HaLow AP frontal
uci set wireless.hallow_ap.easymesh='1'
uci commit wireless
wifi reload
  
```

Configuración Agent:

```

opkg install ieee1905-daemon map-agent

uci set mapagent.config=config
uci set mapagent.config.enabled='1'
uci set mapagent.config.controller='192.168.1.1'
uci set mapagent.config.backhaul_iface='wlan2' # HaLow backhaul
uci commit mapagent
/etc/init.d/ieee1905 restart
  
```

Ventajas:

- Roaming transparente: DCU cambia entre APs sin desconexión.
- Gestión centralizada: configuración desde Controller.
- Steering: Controller dirige clientes a AP óptimo (load balancing).
- Band steering: 802.11ah + 802.11n/ac combinados.

Desventajas:

- Complejidad: requiere IEEE 1905.1 daemon.
- Controller single point of failure (mitigar con HA).

Característica	AP	STA	802.11s Mesh	EasyMesh
Cobertura	3 km	3 km	6-9 km	6-9 km
Topología	Estrella	P2P	Malla	Árbol
Auto-healing	No	No	Sí	Sí
Roaming	No	N/A	Manual	Automático
Gestión	Local	N/A	Distribuida	Centralizada
Complejidad	Baja	Baja	Media	Alta
Latencia	<50 ms	<50 ms	<200 ms	<100 ms
Throughput (10 nodos)	40 Mbps	40 Mbps	30 Mbps	35 Mbps

Tabla 5-4: Comparación de modos de operación HaLow

Comparación de Modos HaLow

Recomendación de Arquitectura Para despliegue de infraestructuras Smart Grid se recomienda:

- **Zona urbana densa (<1 km):** Modo AP con bridge a LAN.
- **Zona rural extendida (3-9 km):** Modo 802.11s Mesh con 2-3 gateways.
- **Multi-edificio (campus):** EasyMesh con Controller central y Agents por edificio.
- **Backup WAN:** Modo STA conectado a torre utilidad con fibra.

Beneficios arquitectónicos de modos avanzados:

1. **Extensión de cobertura:** Mesh/EasyMesh multiplica alcance sin infraestructura adicional (3 gateways = 9 km vs 3 km de AP único).
2. **Resiliencia:** Rutas alternativas en mesh eliminan single point of failure.
3. **Escalabilidad:** EasyMesh permite agregar Agents sin reconfigurar red completa.
4. **Reducción CAPEX:** Menos gateways con backhaul dedicado (fibra/LTE) al usar mesh HaLow como backhaul.
5. **QoE mejorada:** Roaming EasyMesh evita desconexiones durante handoff de DCU móvil (ej. vehículo de mantenimiento).

Conectividad Celular LTE (M.2 Socket)

Para redundancia y uplink WAN, el gateway incorpora módulo LTE Cat-4/Cat-6:

- **Socket:** M.2 Key B (PCIe + USB 3.0 combo) para módems 4G/LTE.

- **Módulos compatibles:**

- Quectel EM060K-GL (Cat-6, 300 Mbps DL, 50 Mbps UL).
- Sierra Wireless EM7565 (Cat-12, 600 Mbps DL).
- Telit LM960 (Cat-18, 1.2 Gbps DL con agregación de portadoras).

- **Bandas soportadas:** B2/B4/B5/B7/B12/B13/B66 (Colombia/Latinoamérica).

- **Interfaz:** USB 3.0 con protocolo QMI/MBIM, drivers ModemManager en OpenWRT.

- **SIM:** Nano-SIM con soporte eSIM (MFF2) en módulos avanzados.

- **Antenas:** 2x conectores U.FL para diversidad MIMO 2×2.

- **Consumo:** <2W en transmisión activa, <100mW en idle.

- **Casos de uso:**

- Uplink principal en zonas sin fibra/ADSL disponible.
- Backup automático si falla conexión Ethernet WAN.
- VPN site-to-site con ThingsBoard Cloud vía IPsec/WireGuard.

Configuración en OpenWRT (ModemManager):

```
opkg update
opkg install modemmanager luciproto-modemmanager usb-modeswitch

# Configurar interfaz LTE
uci set network.lte=interface
uci set network.lte.proto='modemmanager'
uci set network.lte.device='/sys/devices/platform/ahb/1e1c0000.xhci/usb1/1-1'
uci set network.lte.apn='internet.movistar.com.co'
uci set network.lte.pincode='0000'
uci set network.lte.metric='10' # Mayor prioridad que Ethernet WAN
uci commit network
ifup lte
```

Verificación de conexión:

```
mmcli -L # Listar módems detectados
mmcli -m 0 --simple-status # Estado (señal, operador, IP)
```

Expansión de Almacenamiento

Para soportar Docker y volúmenes persistentes:

- **Opción 1:** USB 3.0 flash drive (32-64 GB) formateado ext4.
- **Opción 2:** Adaptador M.2 NVMe vía USB/PCIe (mejor rendimiento I/O).
- **Montaje:** `/mnt/docker` con overlay filesystem para `/var/lib/docker`.

Configuración en OpenWRT:

```
opkg update
opkg install block-mount kmod-fs-ext4 kmod-usb-storage-uas
mkfs.ext4 /dev/sda1
block detect > /etc/config/fstab
uci set fstab.@mount[0].target='/mnt/docker'
uci set fstab.@mount[0].enabled='1'
uci commit fstab
/etc/init.d/fstab enable
/etc/init.d/fstab start
```

5.8.4 Instalación de Docker en OpenWRT

Paquetes Requeridos

```
opkg install dockerd docker-compose luci-app-dockerman
/etc/init.d/dockerd enable
/etc/init.d/dockerd start
```

Configuración de Docker Daemon

Archivo `/etc/docker/daemon.json`:

```
{
  "data-root": "/mnt/docker",
  "storage-driver": "overlay2",
  "log-driver": "json-file",
  "log-opt": {
    "max-size": "10m",
    "max-file": "3"
  }
}
```

Reiniciar Docker:

```
/etc/init.d/dockerd restart
docker info # Verificar data-root en /mnt/docker
```

5.8.5 Despliegue de OpenThread Border Router

Función del OTBR

El OpenThread Border Router (OTBR) actúa como:

- **Puente IPv6:** Rutea tráfico entre red Thread (802.15.4) y backbone Ethernet/WiFi.
- **Comisionado Thread:** Permite unir nuevos dispositivos a la red Thread.
- **mDNS proxy:** Descubrimiento de servicios entre Thread e IP.
- **Web UI:** Interfaz de gestión en `http://[fd00::1]:80`.

Docker Compose para OTBR

Archivo `/mnt/docker/otbr/docker-compose.yml`:

```
version: '3'
services:
  otbr:
    image: openthread/otbr:latest
    container_name: otbr
    network_mode: host
    privileged: true
    volumes:
      - /dev/ttyUSB0:/dev/ttyUSB0
      - ./otbr-config:/etc/openthread
    environment:
      - OTBR_LOG_LEVEL=info
      - INFRA_IF_NAME=br-lan
      - RADIO_URL=spinel+hdlc+uart:///dev/ttyUSB0?uart-baudrate=115200
    restart: unless-stopped
```

Despliegue:

```
cd /mnt/docker/otbr
docker-compose up -d
docker logs -f otbr # Verificar inicio correcto
```

Radio Co-Processor (RCP)

Se requiere un dispositivo 802.15.4 como RCP:

- **Opción 1:** Nordic nRF52840 Dongle con firmware OpenThread RCP.
- **Opción 2:** ESP32-H2 con ot-rcp (soporte Thread 1.3).
- **Conexión:** USB (/dev/ttyUSB0) mapeado al contenedor.

Verificar RCP:

```
docker exec -it otbr ot-ctl state
# Esperado: "disabled" o "detached" (listo para configurar)
```

5.8.6 Despliegue de ThingsBoard Edge

Función de TB Edge

ThingsBoard Edge permite:

- **Edge computing:** Procesamiento local de datos antes de enviar a cloud.
- **Sincronización bidireccional:** Con servidor ThingsBoard Cloud/PE.
- **Operación offline:** Caché local de dashboards y reglas durante desconexión.
- **Reducción de ancho de banda:** Envío agregado y comprimido al cloud.

Docker Compose para TB Edge

Archivo /mnt/docker/tb-edge/docker-compose.yml:

```
version: '3.8'
services:
  tb-edge:
    image: thingsboard/tb-edge:latest
    container_name: tb-edge
    ports:
      - "8080:8080"    # HTTP UI
      - "1883:1883"    # MQTT
      - "5683:5683/udp" # CoAP
    environment:
      - CLOUD_ROUTING_KEY=YOUR_EDGE_KEY
      - CLOUD_ROUTING_SECRET=YOUR_EDGE_SECRET
      - CLOUD_RPC_HOST=cloud.thingsboard.io
      - CLOUD_RPC_PORT=7070
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/tb_edge
      - SPRING_DATASOURCE_USERNAME=postgres
      - SPRING_DATASOURCE_PASSWORD=postgres123
    volumes:
      - ./tb-edge-data:/data
```

```

    - ./tb-edge-logs:/var/log/thingsboard
depends_on:
  - postgres
restart: unless-stopped

postgres:
  image: postgres:15-alpine
  container_name: tb-edge-postgres
  environment:
    - POSTGRES_DB=tb_edge
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=postgres123
  volumes:
    - ./postgres-data:/var/lib/postgresql/data
  restart: unless-stopped

```

Despliegue:

```

cd /mnt/docker/tb-edge
docker-compose up -d
docker logs -f tb-edge # Esperar inicio completo (~2 min)

```

Acceso web: <http://<gateway-ip>:8080> (credenciales por defecto: tenant@thingsboard.org / tenant)

5.8.7 Despliegue de IEEE 2030.5 Server (SEP 2.0)

Función del SEP 2.0 Server

Servidor IEEE 2030.5 para interoperabilidad con utilidades y sistemas DER:

- **API REST estándar:** Endpoints IEEE 2030.5 (DCAP, TM, MM, MSG, ED).
- **Formato XML:** Respuestas según XSD schema IEEE 2030.5.
- **Autenticación TLS mTLS:** Certificados ECC P-256.
- **Interoperabilidad:** Compatible con clientes SEP 2.0 (utilidades, HEMS, EVSE).

Docker Compose para SEP 2.0 Server

Archivo /mnt/docker/sep20-server/docker-compose.yml:

```

version: '3.8'
services:
  sep20-server:
    build:
      context: ./sep20-server

```

```

dockerfile: Dockerfile
container_name: sep20-server
ports:
- "8883:8883"    # HTTPS/TLS
environment:
- TLS_CERT=/certs/server.crt
- TLS_KEY=/certs/server.key
- CA_CERT=/certs/ca.crt
- TB_EDGE_URL=http://tb-edge:8080
- TB_EDGE_TOKEN=YOUR_TB_TOKEN
volumes:
- ./certs:/certs:ro
- ./sep20-data:/data
restart: unless-stopped

```

Implementación del SEP 2.0 Server (Python/Flask)

Archivo /mnt/docker/sep20-server/app.py:

```

from flask import Flask, Response
import requests
import xml.etree.ElementTree as ET

app = Flask(__name__)
TB_EDGE_URL = "http://tb-edge:8080"

@app.route('/dcap', methods=['GET'])
def device_capability():
    """IEEE 2030.5 Device Capability"""
    xml = '''<?xml version="1.0" encoding="UTF-8"?>
<DeviceCapability xmlns="urn:ieee:std:2030.5:ns">
    <href>/dcap</href>
    <TimeLink href="/tm"/>
    <MirrorUsagePointListLink href="/mup"/>
    <MessagingProgramListLink href="/msg"/>
    <EndDeviceListLink href="/edev"/>
</DeviceCapability>'''
    return Response(xml, mimetype='application/sep+xml')

@app.route('/tm', methods=['GET'])
def time_sync():
    """IEEE 2030.5 Time Synchronization"""
    import time
    current_time = int(time.time())
    xml = f'''<?xml version="1.0" encoding="UTF-8"?>
<Time xmlns="urn:ieee:std:2030.5:ns">
    <currentTime>{current_time}</currentTime>
    <quality>7</quality>  <!-- 0-7, 7=highest -->
</Time>'''
    return Response(xml, mimetype='application/sep+xml')

@app.route('/mup/<device_id>/MirrorUsagePoint', methods=['GET'])

```

```

def mirror_usage_point(device_id):
    """IEEE 2030.5 Metering Mirror"""
    # Consultar TB Edge por telemetría del dispositivo
    resp = requests.get(
        f"{TB_EDGE_URL}/api/plugins/telemetry/DEVICE/{device_id}/values/timeseries",
        headers={"X-Authorization": "Bearer YOUR_TOKEN"}
    )
    data = resp.json()

    energy = data.get('energy_kwh', [{}])[0].get('value', 0)
    timestamp = data.get('energy_kwh', [{}])[0].get('ts', 0) // 1000

    xml = f'''<?xml version="1.0" encoding="UTF-8"?>
<MirrorUsagePoint xmlns="urn:ieee:std:2030.5:ns">
    <mRID>{device_id}</mRID>
    <MirrorMeterReading>
        <Reading>
            <value>{int(energy * 1000)}</value>  <!-- Wh -->
            <timePeriod>
                <duration>900</duration>  <!-- 15 min -->
                <start>{timestamp}</start>
            </timePeriod>
        </Reading>
    </MirrorMeterReading>
</MirrorUsagePoint>'''
    return Response(xml, mimetype='application/xml+sep')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8883, ssl_context=('server.crt', 'server.key'))

```

Despliegue:

```

cd /mnt/docker/sep20-server
docker-compose up -d
docker logs -f sep20-server

```

Prueba de endpoint:

```

curl -k --cert client.crt --key client.key \
  https://gateway:8883/dcap

```

Configuración de Sincronización con Cloud

```

volumes: - ./tb-edge-data:/data - ./tb-edge-logs:/var/log/thingsboard depends_on : -postgresrestart : unless-stopped

```

```

postgres: image: postgres:15-alpine container_name : tb-edge-postgres environment : -POSTGRES_DB = tb_edge - POSTGRES_USER = postgres - POSTGRES_PASSWORD = postgres123 volumes : -./postgres-data : /var/lib/postgresql/data restart : unless_stopped

```

Despliegue:

```
cd /mnt/docker/tb-edge
docker-compose up -d
docker logs -f tb-edge # Esperar inicio completo (~2 min)
```

Acceso web: <http://<gateway-ip>:8080> (credenciales por defecto: tenant@thingsboard.org / tenant)

Configuración de Sincronización con Cloud

En ThingsBoard Cloud:

1. Crear Edge instance en *Edge Management*.
2. Copiar Routing Key y Secret.
3. Actualizar variables en `docker-compose.yml`.
4. Reiniciar contenedor: `docker-compose restart tb-edge`.

5.8.8 Integración OTBR ↔ ThingsBoard Edge

Flujo de Datos

1. Dispositivos Thread (nodos ESP32C6) publican datos vía CoAP/MQTT sobre Thread.
2. OTBR rutea tráfico IPv6 desde Thread a red LAN del gateway.
3. Aplicación bridge (Python/Node.js) suscrita a CoAP/MQTT convierte y publica a TB Edge vía MQTT.
4. TB Edge procesa localmente (reglas, alarmas) y sincroniza con cloud periódicamente.

Script Bridge (Ejemplo Python)

Archivo `/mnt/docker/bridge/bridge.py`:

```
import paho.mqtt.client as mqtt
import json
```

```

# Cliente MQTT para recibir de dispositivos Thread
thread_client = mqtt.Client()
thread_client.connect("localhost", 1883)

# Cliente MQTT para publicar a TB Edge
tb_client = mqtt.Client()
tb_client.username_pw_set("ACCESS_TOKEN")
tb_client.connect("localhost", 1883)

def on_message(client, userdata, msg):
    # Parser de telemetría Thread
    data = json.loads(msg.payload)

    # Transformar a formato TB
    telemetry = {
        "ts": int(data["timestamp"]) * 1000,
        "values": [
            "energy": data["energy_kwh"],
            "power": data["power_w"],
            "voltage": data["voltage_v"]
        ]
    }

    # Publicar a TB Edge
    tb_client.publish("v1/devices/me/telemetry",
                      json.dumps(telemetry))

thread_client.on_message = on_message
thread_client.subscribe("thread/telemetry/#")
thread_client.loop_forever()

```

Desplegar como contenedor:

```

docker build -t thread-tb-bridge .
docker run -d --name bridge --network host thread-tb-bridge

```

5.9 Arquitectura de Datos: Kafka y PostgreSQL

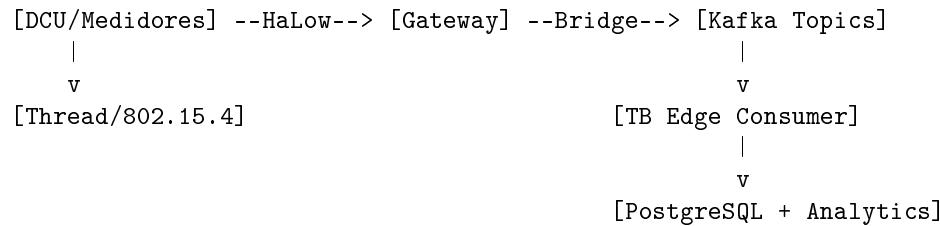
5.9.1 Integración de Apache Kafka

Apache Kafka proporciona una capa de mensajería distribuida de alto rendimiento para desacoplar productores (dispositivos IoT) de consumidores (ThingsBoard Edge, analítica, almacenamiento):

Rol de Kafka en la Arquitectura

- **Message broker:** Intermedia entre bridge (productor) y TB Edge (consumidor).
- **Buffer distribuido:** Almacena temporalmente mensajes en tópicos persistentes.
- **Escalabilidad:** Soporta >100k msg/s con múltiples particiones.
- **Durabilidad:** Retención configurable (7 días default) para replay histórico.

Arquitectura con Kafka:



Docker Compose para Kafka

Archivo /mnt/ssd/docker/kafka/docker-compose.yml:

```

version: '3.8'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    volumes:
      - ./zookeeper-data:/var/lib/zookeeper/data
    restart: unless-stopped

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
  
```

```

KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
KAFKA_LOG_RETENTION_HOURS: 168 # 7 días
KAFKA_LOG_SEGMENT_BYTES: 104857600 # 100 MB
KAFKA_COMPRESSION_TYPE: gzip
volumes:
  - ./kafka-data:/var/lib/kafka/data
restart: unless-stopped

```

Crear tópicos:

```

docker exec -it kafka kafka-topics --create \
--bootstrap-server localhost:9092 \
--topic telemetry --partitions 3 --replication-factor 1

docker exec -it kafka kafka-topics --create \
--bootstrap-server localhost:9092 \
--topic alarms --partitions 2 --replication-factor 1

```

Bridge Kafka Producer

Modificación del bridge para publicar a Kafka:

```

from kafka import KafkaProducer
import json

# Kafka producer
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    compression_type='gzip'
)

def on_message(client, userdata, msg):
    data = json.loads(msg.payload)

    # Publicar a Kafka topic 'telemetry'
    producer.send('telemetry', {
        'device_id': data.get('device_id'),
        'timestamp': int(time.time() * 1000),
        'energy_kwh': data['energy_kwh'],
        'power_w': data['power_w'],
        'voltage_v': data['voltage_v']
    })

    # Flush cada 100 mensajes o 5 segundos
    if msg_count % 100 == 0:
        producer.flush()

```

TB Edge Kafka Consumer

ThingsBoard Edge configurado como consumidor Kafka:

```
# /mnt/ssd/docker/tb-edge-data/conf/tb-edge.yml
queue:
  type: kafka # Cambiar de 'in-memory' a 'kafka'
  kafka:
    bootstrap:
      servers: 'kafka:9092'
    topics:
      telemetry: 'telemetry'
      alarms: 'alarms'
    consumer:
      group_id: 'tb-edge-group'
      auto_offset_reset: 'earliest'
    producer:
      compression_type: 'gzip'
      acks: 1
```

Ventajas de Kafka vs in-memory queue:

- **Capacidad:** GB de mensajes vs 100k en memoria.
- **Replay:** Consumir desde offset histórico para análisis.
- **Multi-consumidor:** TB Edge + analítica + ML simultáneamente.
- **Backpressure:** Kafka absorbe picos sin perder mensajes.

5.9.2 PostgreSQL: Base de Datos Persistente

Rol de PostgreSQL en la Arquitectura

PostgreSQL almacena:

- **Telemetría histórica:** Series temporales con extensión TimescaleDB.
- **Configuración de dispositivos:** Atributos, credenciales, relaciones.
- **Alarms y eventos:** Log persistente para auditoría.
- **Dashboards y reglas:** Definiciones de Rule Engine TB Edge.

Optimización para Series Temporales (TimescaleDB)

TimescaleDB es una extensión de PostgreSQL optimizada para series temporales:

Docker Compose:

```
timescaledb:
  image: timescale/timescaledb:latest-pg15
  container_name: timescaledb
  environment:
    POSTGRES_DB: tb_edge
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres123
  ports:
    - "5432:5432"
  volumes:
    - ./timescaledb-data:/var/lib/postgresql/data
  command: ["postgres", "-c", "shared_preload_libraries=timescaledb"]
  restart: unless-stopped
```

Crear hypertable para telemetría:

```
-- Conectar a PostgreSQL
psql -h localhost -U postgres -d tb_edge

-- Habilitar TimescaleDB
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Crear tabla de telemetria
CREATE TABLE telemetry (
    time TIMESTAMPTZ NOT NULL,
    device_id TEXT NOT NULL,
    energy_kwh DOUBLE PRECISION,
    power_w DOUBLE PRECISION,
    voltage_v DOUBLE PRECISION
);

-- Convertir a hypertable (particionamiento automático)
SELECT create_hypertable('telemetry', 'time');

-- Crear índices
CREATE INDEX idx_device_time ON telemetry (device_id, time DESC);

-- Política de retención: 90 días
SELECT add_retention_policy('telemetry', INTERVAL '90 days');

-- Compresión automática después de 7 días
ALTER TABLE telemetry SET (
    timescaledb.compress,
```

```

timescaledb.compress_segmentby = 'device_id'
);

SELECT add_compression_policy('telemetry', INTERVAL '7 days');

```

Ventajas TimescaleDB:

- **Compresión:** 10-20× reducción en espacio disco.
- **Queries rápidas:** Particionamiento automático por tiempo.
- **Agregaciones:** time_bucket para promedios 15-min/1-hora.
- **Retención automática:** Elimina datos antiguos sin intervención manual.

Query de ejemplo (consumo promedio por hora):

```

SELECT
    device_id,
    time_bucket('1 hour', time) AS hour,
    AVG(energy_kwh) AS avg_energy,
    MAX(power_w) AS peak_power
FROM telemetry
WHERE time > NOW() - INTERVAL '24 hours'
GROUP BY device_id, hour
ORDER BY device_id, hour DESC;

```

5.10 Protocolos de Comunicación IoT

5.10.1 Comparación de Protocolos

Característica	MQTT	CoAP	HTTP/REST	LwM2M
Capa OSI	Aplicación	Aplicación	Aplicación	Aplicación
Transporte	TCP	UDP	TCP	UDP (CoAP)
Overhead	2 bytes	4 bytes	>100 bytes	4-20 bytes
QoS niveles	0, 1, 2	CON, NON, ACK	N/A	CON, NON
Pub/Sub	Sí	No	No	No
Observación	No	Sí (Observe)	No	Sí (Observe)
Device Mgmt	No	No	No	Sí (objetos)
Seguridad	TLS/mTLS	DTLS	TLS/mTLS	DTLS + PSK
Payload	Binario/JSON	CBOR/JSON	JSON/XML	CBOR/TLV
Uso típico	Telemetría	Sensores low-power	APIs web	Device mgmt

Tabla 5-5: Comparación de protocolos IoT

5.10.2 MQTT (Message Queuing Telemetry Transport)

Características MQTT

- **Patrón Pub/Sub:** Desacoplamiento productor/consumidor vía broker.
- **QoS 0 (At most once):** Sin confirmación, mínima latencia.
- **QoS 1 (At least once):** Con ACK, puede duplicar mensajes.
- **QoS 2 (Exactly once):** Handshake 4-way, sin duplicados.
- **Retained messages:** Último valor almacenado en broker para nuevos suscriptores.
- **Last Will Testament (LWT):** Notificación automática de desconexión.

Uso en Gateway

Topics utilizados:

```
# Telemetría uplink (dispositivo → TB Edge)
v1/devices/me/telemetry          # Formato ThingsBoard
smartgrid/meter/{id}/energy      # Formato personalizado
thread/telemetry/{node_id}        # Desde OTBR

# Comandos downlink (TB Edge → dispositivo)
v1/devices/me/rpc/request/{requestId}
smartgrid/meter/{id}/command

# LWT (detectar desconexión)
smartgrid/meter/{id}/status      # "online"/"offline"
```

Configuración broker Mosquitto en gateway:

```
# /etc/mosquitto/mosquitto.conf
listener 1883                  # Puerto MQTT
listener 8883                  # Puerto MQTTS (TLS)
cafile /etc/mosquitto/ca.crt
certfile /etc/mosquitto/server.crt
keyfile /etc/mosquitto/server.key
require_certificate true

# Persistencia
persistence true
persistence_location /mnt/ssd/mosquitto/
```

```
# Autenticación
allow_anonymous false
password_file /etc/mosquitto/passwd

# QoS máximo
max_qos 2

# Retención de mensajes
max_queued_messages 1000
message_size_limit 8192
```

5.10.3 CoAP (Constrained Application Protocol)

Características CoAP

- **Protocolo UDP:** Menor overhead que TCP (4 bytes header vs 20 bytes).
- **RESTful:** Métodos GET/POST/PUT/DELETE como HTTP.
- **Observe:** Cliente se suscribe a recurso, servidor notifica cambios.
- **Block-wise transfer:** Fragmentación para mensajes >1024 bytes.
- **DTLS:** Seguridad con Pre-Shared Keys (PSK) o certificados.

Uso en Thread/OpenThread

OpenThread usa CoAP nativamente para comunicación mesh:

```
# Recurso CoAP en nodo Thread
coap://[fd00::1234:5678:abcd]/meter/energy

# GET request (cliente OTBR)
coap-client -m GET coap://[fd00::1234:5678:abcd]/meter/energy
# Response: {"energy_kwh": 100.5, "power_w": 1200}

# POST command (enviar comando)
coap-client -m POST coap://[fd00::1234:5678:abcd]/config/interval \
-e '{"interval_sec": 300}'

# Observe (suscripción a cambios)
coap-client -m GET -s 60 coap://[fd00::1234:5678:abcd]/meter/energy
# Server notifica cada vez que cambia el valor
```

Bridge CoAP → MQTT:

```

import asyncio
from aiocoap import *

async def coap_to_mqtt_bridge():
    protocol = await Context.create_client_context()

    # Observe CoAP resource
    request = Message(code=GET,
                       uri='coap://[fd00::1234]/meter/energy',
                       observe=0)

    async for response in protocol.request(request).observation:
        # Parse CoAP payload
        data = json.loads(response.payload)

        # Publish to MQTT
        mqtt_client.publish('thread/telemetry/1234',
                           json.dumps(data))

```

5.10.4 HTTP/REST

Uso en Gateway

HTTP/REST se usa para:

- **APIs de gestión:** TB Edge API, IEEE 2030.5 Server.
- **Integraciones externas:** Webhooks, consultas a cloud.
- **Configuración:** LuCI web UI, Ollama API.

Ejemplos de APIs:

```

# ThingsBoard Edge API
GET http://gateway:8080/api/tenant/devices
Authorization: Bearer {token}

# IEEE 2030.5 REST API
GET https://gateway:8883/dcap
Client-Cert: meter-001.crt

# Ollama LLM API
POST http://gateway:11434/api/generate
Content-Type: application/json
{"model": "llama3.2:3b", "prompt": "Analiza consumo..."}

```

```
# LuCI configuration
POST http://gateway/cgi-bin/luci/admin/network/wireless
```

5.10.5 LwM2M (Lightweight M2M)

Características LwM2M

- **Gestión de dispositivos:** Bootstrap, configuración, firmware OTA.
- **Modelo de objetos:** Jerarquía Object/Instance/Resource (ej. Device/0/Manufacturer).
- **Operaciones:** Read, Write, Execute, Observe, Discover.
- **Transporte:** CoAP sobre UDP (binding U) o SMS/NB-IoT (binding S).

Objetos LwM2M Estándar (OMA SpecWorks)

Object ID	Nombre	Recursos Clave
0	Security	Server URI, PSK, Bootstrap
1	Server	Lifetime, Binding, Registration
3	Device	Manufacturer, Model, Serial, Battery
4	Connectivity Monitoring	Network Bearer, IP Address, Signal Strength
5	Firmware Update	Package URI, Update, State
3303	Temperature Sensor	Sensor Value, Units, Min/Max
3305	Power Measurement	Instantaneous Power, Energy, Voltage

Tabla 5-6: Objetos LwM2M estándar para Smart Metering

Integración LwM2M en Gateway

Desplegar servidor LwM2M (Leshan) en gateway:

```
# Docker Compose para Leshan LwM2M Server
services:
  leshan:
    image: eclipse/leshan:latest
    container_name: leshan-server
    ports:
      - "8080:8080"    # Web UI
      - "5683:5683/udp"  # CoAP LwM2M
      - "5684:5684/udp"  # CoAPS (DTLS)
    volumes:
```

```
- ./leshan-data:/data
restart: unless-stopped
```

Cliente LwM2M en DCU/Medidor:

```
// Wakaama LwM2M client (C)
lwm2m_context_t *context = lwm2m_init(NULL);

// Object 3305: Power Measurement
power_obj = create_power_object();
lwm2m_add_object(context, power_obj);

// Register to server
lwm2m_configure(context, "gateway", 5683, "meter-001", 300);

// Update resources periodically
lwm2m_resource_value_set(power_obj, 5800, &energy_kwh); // Instantaneous active energy
lwm2m_resource_value_set(power_obj, 5805, &voltage_v); // Voltage
```

Ventajas LwM2M para Smart Metering:

- **Estandarización:** Objetos interoperables (OMA SpecWorks).
- **Gestión centralizada:** Bootstrap, firmware OTA desde gateway.
- **Eficiencia:** DTLS+PSK consume menos que TLS+X.509 (clave de 16 bytes vs certificado 2 KB).
- **Observe:** Notificaciones automáticas sin polling (ahorro energético).

5.10.6 Selección de Protocolo por Caso de Uso

Caso de Uso	Protocolo	Justificación
Telemetría medidor → Gateway	MQTT QoS 1	Pub/Sub, QoS garantizado, broker local
Thread mesh intra-nodo	CoAP	UDP eficiente, Observe nativo
Gateway → TB Cloud	MQTT QoS 1 / HTTP	TLS seguro, compresión gzip
Device management (FW OTA)	LwM2M	Objetos estándar, bootstrap
APIs configuración gateway	HTTP/REST	LuCI, TB Edge API
IEEE 2030.5 Smart Energy	HTTP/REST	Estándar mandatorio (SEP 2.0)
Comandos downlink (RPC)	MQTT QoS 1	Baja latencia, ACK confirmado
Alarmas críticas	MQTT QoS 2	Exactly-once, sin duplicados

Tabla 5-7: Selección de protocolo por caso de uso

5.11 Resiliencia y Almacenamiento Persistente (SSD)

5.11.1 Arquitectura de Almacenamiento del Gateway

El gateway implementa una estrategia de almacenamiento de alta resiliencia:

- **Flash interna (128 MB)**: Sistema operativo OpenWRT y configuración UCI.
- **SSD M.2 NVMe (256 GB)**: Datos persistentes Docker, PostgreSQL, cola ThingsBoard Edge.
- **USB 3.0 (opcional)**: Backups periódicos y recuperación de desastres.

Ventajas SSD NVMe vs USB/SD:

- **Durabilidad**: >1M ciclos E/W (vs 10k-100k en SD), MTBF >1.5M horas.
- **Desempeño**: >3000 IOPS escritura (vs <100 IOPS USB 2.0), latencia <0.1ms.
- **Fiabilidad**: ECC interno, power-loss protection (PLP), SMART monitoring.

Montaje automático SSD en /mnt/ssd:

```
# /etc/config/fstab
config mount
    option device '/dev/nvme0n1p1'
    option target '/mnt/ssd'
    option fstype 'ext4'
    option options 'rw,relatime,noatime'
    option enabled '1'
    option enabled_fsck '1'
```

Estructura de directorios:

```
/mnt/ssd/
docker/          # Volúmenes Docker
tb-edge-data/    # Configuración TB Edge
postgres-data/   # Base de datos PostgreSQL
queue/           # Cola de mensajes offline
otbr-config/     # Configuración Thread
bridge-logs/     # Logs del bridge
backups/          # Backups automáticos
tmp/             # Directorio temporal (tmpfs en RAM)
```

5.11.2 ThingsBoard Edge Queue: Resiliencia Offline

Arquitectura de Cola Persistente

ThingsBoard Edge implementa una **cola de mensajes persistente** para garantizar resiliencia ante pérdida de conectividad cloud:

- **Queue storage:** PostgreSQL + filesystem (/mnt/ssd/docker/queue).
- **Capacidad:** Hasta 100k mensajes en cola (configurable, 500 MB con CBOR).
- **Política FIFO:** Mensajes más antiguos se sincronizan primero al recuperar conectividad.
- **Priorización:** Alarmas críticas tienen mayor prioridad que telemetría histórica.

Configuración de Queue en TB Edge

Archivo /mnt/ssd/docker/tb-edge-data/conf/tb-edge.yml:

```

queue:
  type: in-memory # O 'kafka' para mayor capacidad
  in_memory:
    max_elements: 100000
    max_size_bytes: 524288000 # 500 MB

  cloud_sync:
    # Intervalo de sincronización con cloud
    sync_interval: 300 # 5 minutos (online)
    offline_sync_interval: 3600 # 1 hora (detectado offline, reintentos)

    # Batch size para sincronización
    batch_size: 1000 # Mensajes por batch
    max_batch_size: 5000 # Máximo en condiciones de catch-up

    # Compresión de datos
    compression_enabled: true
    compression_type: gzip # O 'lz4' para menor CPU

    # Retry policy
    max_retries: 10
    retry_interval: 60 # segundos entre reintentos
    backoff_multiplier: 2 # Exponential backoff

  persistent_queue:
    path: /data/queue
    max_disk_usage_mb: 2000 # 2 GB límite en disco

```

```
write_batch_size: 100  
flush_interval_ms: 1000
```

Flujo de Operación con Queue

Modo Online (conectividad cloud activa):

1. TB Edge recibe telemetría de dispositivos vía MQTT (puerto 1883).
2. Procesa reglas locales (alarmas, transformaciones).
3. Encola mensaje en queue in-memory.
4. Cada 5 minutos, sincroniza batch de 1000 mensajes con TB Cloud vía gRPC (puerto 7070).
5. Al confirmar ACK del cloud, elimina mensajes de la cola.

Modo Offline (sin conectividad cloud):

1. TB Edge detecta pérdida de conexión (timeout gRPC >30s).
2. Cambia a modo offline: continúa procesamiento local.
3. Mensajes se acumulan en queue persistente (PostgreSQL + filesystem).
4. Dashboards locales permanecen funcionales (<http://<gateway-ip>:8080>).
5. Alarmas se ejecutan localmente (email, webhook a sistemas locales).
6. Queue crece hasta límite configurado (100k msgs o 2 GB).

Recuperación de Conectividad (catch-up sync):

1. TB Edge detecta reconexión con cloud (gRPC handshake exitoso).
2. Inicia sincronización acelerada: batch size aumenta a 5000 mensajes.
3. Prioriza alarmas/eventos críticos (priority queue).
4. Comprime datos con gzip (40-60 % reducción).
5. Sincroniza backlog completo en 10-15 minutos (100k msgs @ 5000/batch).
6. Al terminar, retorna a modo online normal (batch 1000, intervalo 5 min).

Protección contra Desbordamiento de Queue

Política de gestión de capacidad:

```
# /mnt/ssd/docker/tb-edge-data/scripts/queue-monitor.sh
#!/bin/sh

QUEUE_SIZE=$(du -sm /mnt/ssd/docker/queue | cut -f1)
MAX_SIZE=1800 # 1.8 GB (90% del límite de 2 GB)

if [ $QUEUE_SIZE -gt $MAX_SIZE ]; then
    logger -t TB-EDGE "Queue size critical: ${QUEUE_SIZE}MB"

    # Estrategia 1: Eliminar telemetría histórica (>7 días)
    find /mnt/ssd/docker/queue -name "*.telemetry" \
        -mtime +7 -delete

    # Estrategia 2: Comprimir eventos no críticos
    find /mnt/ssd/docker/queue -name "*.event" \
        -mtime +1 -exec gzip {} \;

    # Estrategia 3: Notificar operador
    curl -X POST http://localhost:8080/api/v1/telemetry \
        -d '{"queue_overflow":true,"size_mb":'$QUEUE_SIZE'}'
fi
```

Ejecutar vía cron cada hora:

```
# crontab -e
0 * * * * /mnt/ssd/docker/tb-edge-data/scripts/queue-monitor.sh
```

5.11.3 Resiliencia Multinivel

Nivel	Componente	Mecanismo de Resiliencia	RTO
L1 Hardware	SSD NVMe	ECC, PLP, SMART monitoring	0s
L2 Filesystem	ext4	Journaling, fsck automático	<30s
L3 Base de datos	PostgreSQL	WAL, autovacuum, replication slots	<60s
L4 Aplicación	TB Edge Queue	Persistent queue, retry policy, compression	<300s
L5 Red	mwlan3	WAN failover Ethernet/LTE, tracking activo	<30s
L6 Container	Docker + Watchtower	Healthchecks, restart policy, auto-updates	<120s

Tabla 5-8: Niveles de resiliencia en gateway con RTO (Recovery Time Objective)

5.12 Gestión Remota del Gateway

5.12.1 Feeds de OpenWRT

Arquitectura de Feeds

OpenWRT utiliza **feeds** (repositorios de paquetes) para extender funcionalidad del sistema base:

- **Feed base:** Paquetes esenciales incluidos en imagen oficial.
- **Feed packages:** Paquetes comunitarios adicionales (>10k paquetes).
- **Feed luci:** Interfaz web LuCI y aplicaciones.
- **Feed routing:** Protocolos de routing avanzados (batman-adv, olsr).
- **Feed telephony:** VoIP y telefonía.
- **Feeds custom:** Repositorios personalizados (enterprise, vendor-specific).

Configuración de Feeds

Archivo `/etc/opkg/distfeeds.conf`:

```
# Feeds oficiales OpenWRT 23.05
src/gz openwrt_core https://downloads.openwrt.org/releases/23.05.0/targets/ramips/mt7621/packages
src/gz openwrt_base https://downloads.openwrt.org/releases/23.05.0/packages/mipsel_24kc/base
src/gz openwrt_luci https://downloads.openwrt.org/releases/23.05.0/packages/mipsel_24kc/luci
src/gz openwrt_packages https://downloads.openwrt.org/releases/23.05.0/packages/mipsel_24kc/packages
src/gz openwrt_routing https://downloads.openwrt.org/releases/23.05.0/packages/mipsel_24kc/routing

# Feed personalizado para Smart Grid (interno empresa)
src/gz smartgrid_custom https://repo.empresia.com/openwrt/23.05/smartgrid
option check_signature 1
```

Gestión de Paquetes con opkg

Comandos esenciales:

```
# Actualizar lista de paquetes desde feeds
```

```

opkg update

# Buscar paquetes
opkg find "*halow*"
opkg find "*docker*"

# Instalar paquete
opkg install luci-app-dockerman
opkg install mwlan3 luci-app-mwan3

# Listar paquetes instalados
opkg list-installed | grep docker

# Actualizar paquete específico
opkg upgrade docker

# Actualizar todos los paquetes
opkg list-upgradable
opkg upgrade $(opkg list-upgradable | awk '{print $1}')

# Remover paquete (mantener configuración)
opkg remove --force-removal-of-dependent-packages docker

# Información de paquete
opkg info docker

```

Feed Custom para Smart Grid

Crear feed interno con paquetes personalizados:

Estructura del feed:

```

smartgrid-feed/
|--- Makefile
|--- packages/
|   |--- halow-driver-morse/
|   |   |--- Makefile
|   |   +--- files/
|   |--- tb-edge-connector/
|   |   |--- Makefile
|   |   +--- src/
|   |--- ieee2030-server/
|   |   |--- Makefile
|   |   +--- files/
|   +--- mcp-l1m-client/
|       |--- Makefile
|       +--- src/
+--- README.md

```

Ejemplo Makefile para paquete personalizado:

```
# packages/tb-edge-connector/Makefile
include $(TOPDIR)/rules.mk

PKG_NAME:=tb-edge-connector
PKG_VERSION:=1.0.0
PKG_RELEASE:=1

include $(INCLUDE_DIR)/package.mk

define Package/tb-edge-connector
SECTION:=smartgrid
CATEGORY:=SmartGrid
TITLE:=ThingsBoard Edge Connector
DEPENDS:+=python3 +python3-paho-mqtt +python3-requests
endef

define Package/tb-edge-connector/description
Connector para integrar dispositivos Thread con TB Edge
endef

define Build/Compile
# Script Python, no requiere compilación
endef

define Package/tb-edge-connector/install
$(INSTALL_DIR) $(1)/usr/bin
$(INSTALL_BIN) ./src/tb_connector.py $(1)/usr/bin/
$(INSTALL_DIR) $(1)/etc/config
$(INSTALL_CONF) ./files/tb-connector.conf $(1)/etc/config/
$(INSTALL_DIR) $(1)/etc/init.d
$(INSTALL_BIN) ./files/tb-connector.init $(1)/etc/init.d/tb-connector
endef

$(eval $(call BuildPackage,tb-edge-connector))
```

Hosting del feed:

```
# Generar índice de paquetes
cd smartgrid-feed
make package/index

# Servir vía HTTP (nginx)
server {
    listen 80;
    server_name repo.empresa.com;
    root /var/www/openwrt-feed;

    location /openwrt/23.05/smartgrid {
```

```

        autoindex on;
    }
}

```

5.12.2 OpenVPN: Acceso Remoto Seguro

Rol de OpenVPN en la Arquitectura

OpenVPN proporciona túnel VPN cifrado para gestión remota del gateway:

- **Acceso SSH:** Administración CLI segura desde NOC (Network Operations Center).
- **LuCI web UI:** Acceso a interfaz de configuración sin exponer puerto 80/443 a internet.
- **Debugging:** Logs remotos, captura de tráfico (tcpdump), análisis de performance.
- **Túnel permanente:** Gateway se conecta automáticamente a servidor VPN central (hub-spoke).

Instalación OpenVPN en Gateway

```

opkg update
opkg install openvpn-openssl luci-app-openvpn

```

Configuración Cliente OpenVPN

Archivo /etc/openvpn/client.conf:

```

client
dev tun0
proto udp
remote vpn.empresia.com 1194

# Certificados PKI
ca /etc/openvpn/ca.crt
cert /etc/openvpn/gateway-001.crt
key /etc/openvpn/gateway-001.key
tls-auth /etc/openvpn/ta.key 1

# Compresión
comp-lzo adaptive

```

```
# Keepalive (detectar desconexión en 120s)
keepalive 10 120

# Persistencia de túnel
persist-key
persist-tun

# Logging
verb 3
log-append /var/log/openvpn.log

# Pull routes desde servidor (acceso a red NOC)
pull

# Reconexión automática
resolv-retry infinite
nobind

# Usuario sin privilegios (seguridad)
user nobody
group nogroup
```

Configuración UCI (OpenWRT native):

```
# /etc/config/openvpn
config openvpn 'noc_tunnel'
    option enabled '1'
    option client '1'
    option dev 'tun0'
    option proto 'udp'
    option remote 'vpn.empresa.com 1194'
    option ca '/etc/openvpn/ca.crt'
    option cert '/etc/openvpn/gateway-001.crt'
    option key '/etc/openvpn/gateway-001.key'
    option tls_auth '/etc/openvpn/ta.key 1'
    option comp_lzo 'adaptive'
    option keepalive '10 120'
    option persist_key '1'
    option persist_tun '1'
    option verb '3'
```

Iniciar servicio:

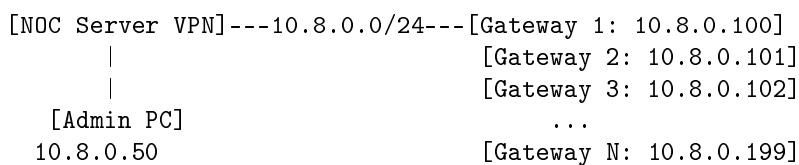
```
/etc/init.d/openvpn enable
/etc/init.d/openvpn start

# Verificar túnel
ifconfig tun0
# Esperado: inet addr:10.8.0.100
```

```
ping 10.8.0.1 # Servidor VPN
```

Servidor OpenVPN Central (NOC)

Arquitectura hub-spoke:



Configuración servidor (/etc/openvpn/server.conf):

```

port 1194
proto udp
dev tun
server 10.8.0.0 255.255.255.0

# Certificados
ca ca.crt
cert server.crt
key server.key
dh dh2048.pem
tls-auth ta.key 0

# Client-to-client (permitir gateways comunicarse entre sí)
client-to-client

# Push routes a clientes (red NOC)
push "route 10.10.0.0 255.255.255.0"

# Persistencia
keepalive 10 120
persist-key
persist-tun

# Logging
status /var/log/openvpn-status.log
log-append /var/log/openvpn.log
verb 3

# Client config dir (configuración por cliente)
client-config-dir /etc/openvpn/ccd

# Compresión
  
```

comp-lzo adaptive

Configuración específica por gateway (CCD):

```
# /etc/openvpn/ccd/gateway-001
# Asignar IP fija 10.8.0.100
ifconfig-push 10.8.0.100 10.8.0.101

# Push ruta específica para este gateway (red Thread local)
push "route 192.168.50.0 255.255.255.0"
```

5.12.3 OpenWISP: Gestión Centralizada de Gateways

Arquitectura OpenWISP

OpenWISP es una plataforma open-source para gestión masiva de dispositivos OpenWRT:

- **OpenWISP Controller:** Backend Django para gestión de configuraciones.
- **OpenWISP Config:** Agente en gateway que aplica configuraciones remotas.
- **OpenWISP Monitoring:** Colección de métricas (CPU, RAM, tráfico).
- **OpenWISP Firmware Upgrader:** Actualizaciones OTA masivas.
- **OpenWISP Network Topology:** Visualización de topología de red.

Componentes:

```
[OpenWISP Server]---HTTPS/VPN---[Gateway Fleet (100-1000 gateways)]
|                               |
|--- Controller (Django)      |--- openwisp-config (agente)
|--- PostgreSQL                |--- openwisp-monitoring
|--- Redis                     +--- Actualizacion UCI automatica
|--- Celery
+--- Web Dashboard
```

Instalación OpenWISP Config en Gateway

opkg update

```
opkg install openwisp-config openwisp-monitoring

# Configurar agente
uci set openwisp.http.url='https://controller.empresa.com'
uci set openwisp.http.shared_secret='YOUR_SHARED_SECRET'
uci set openwisp.http.consistent_key='GATEWAY_UUID'
uci set openwisp.http.verify_ssl='1'
uci commit openwisp

# Habilitar servicios
/etc/init.d/openwisp_config enable
/etc/init.d/openwisp_monitoring enable
/etc/init.d/openwisp_config start
/etc/init.d/openwisp_monitoring start
```

Despliegue OpenWISP Controller (Docker)

```
# docker-compose.yml para OpenWISP Server
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: openwisp_db
      POSTGRES_USER: openwisp
      POSTGRES_PASSWORD: changeme
    volumes:
      - postgres-data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine

  openwisp-dashboard:
    image: openwisp/openwisp-dashboard:latest
    depends_on:
      - postgres
      - redis
    ports:
      - "443:443"
    environment:
      DB_ENGINE: django.db.backends.postgresql
      DB_NAME: openwisp_db
      DB_USER: openwisp
      DB_PASSWORD: changeme
      DB_HOST: postgres
      REDIS_HOST: redis
      DJANGO_SECRET_KEY: changeme
      DJANGO_ALLOWED_HOSTS: controller.empresa.com
    volumes:
      - openwisp-media:/opt/openwisp/media
      - openwisp-static:/opt/openwisp/static
```

```

celery:
  image: openwisp/openwisp-dashboard:latest
  depends_on:
    - postgres
    - redis
  command: celery -A openwisp worker -l info
environment:
  DB_ENGINE: django.db.backends.postgresql
  DB_NAME: openwisp_db
  DB_USER: openwisp
  DB_PASSWORD: changeme
  DB_HOST: postgres
  REDIS_HOST: redis

```

Gestión de Configuraciones

Template UCI en OpenWISP Controller:

```

# Template "Smart Grid Gateway Base Config"
# network.json
{
  "network": [
    {
      "interface": "wan_eth",
      "proto": "dhcp",
      "metric": "10"
    },
    {
      "interface": "wan_lte",
      "proto": "modemmanager",
      "device": "/dev/ttyUSB2",
      "apn": "{{apn}}",
      "metric": "20"
    },
    {
      "interface": "halow",
      "proto": "static",
      "ipaddr": "{{halow_ip}}",
      "netmask": "255.255.255.0"
    }
  ],
  "wireless": [
    {
      "radio": "halow",
      "channel": "{{halow_channel}}",
      "mode": "ap",
      "ssid": "SmartGrid-{{site_id}}",
      "encryption": "sae",
      "key": "{{halow_password}}"
    }
  ]
}

```

```

        }
    ]
}

```

Aplicar configuración desde dashboard:

1. Crear template con variables (`{{apn}}`, `{{halow_channel}}`).
2. Asignar template a grupo de gateways (ej. "Zona Norte - 50 gateways").
3. Definir variables por gateway o grupo.
4. Push configuración: OpenWISP genera UCI y envía a gateways vía HTTPS.
5. Agente `openwisp-config` aplica cambios: `uci commit && reload_config`.

Firmware Upgrader: Actualización OTA Masiva

Workflow de actualización:

1. Subir nueva imagen OpenWRT a OpenWISP Controller.
2. Crear **build** con imagen y checksum SHA256.
3. Asignar build a grupo de gateways (ej. "Firmware 23.05.2 - Gateways Zona Sur").
4. Programar actualización: inmediata o ventana de mantenimiento (3 AM).
5. OpenWISP envía comando a gateways.
6. Gateway descarga imagen vía HTTPS, verifica checksum, instala (sysupgrade), reinicia.
7. Gateway reporta versión actualizada post-reboot.

Actualización segura (dual-partition):

- **Partition A:** Firmware actual (activo).
- **Partition B:** Nueva imagen descargada.
- Escribir en Partition B, reiniciar, bootloader cambia a B.
- Si falla (no boot en 3 reintentos), rollback automático a Partition A.

Monitoring y Alertas

OpenWISP Monitoring recolecta métricas:

- **Uptime:** Tiempo desde último reinicio.
- **CPU/RAM:** Load average, memoria disponible.
- **Storage:** Uso de SSD NVMe (df -h /mnt/ssd).
- **Interfaces:** Tráfico RX/TX, errores, señal HaLow/LTE.
- **Conectividad:** Ping a gateway (heartbeat), VPN status.
- **Docker:** Containers running/stopped, CPU/RAM por container.

Configuración alertas:

```
# OpenWISP Controller > Alerts
Alert: Gateway Offline
  Condition: ping_failed > 5 min
  Action: Email admin@empresa.com, SMS +57300...
Alert: High CPU Usage
  Condition: cpu_load_1min > 3.0 for 10 min
  Action: Email ops@empresa.com, Log to Syslog
Alert: Low Disk Space
  Condition: disk_usage_percent > 85%
  Action: Email admin@empresa.com, Trigger cleanup script
Alert: LTE Failover Active
  Condition: wan_interface = "wwan0" for 30 min
  Action: Email ops@empresa.com (posible fallo fibra)
```

5.12.4 Comparación de Herramientas de Gestión

Recomendación para despliegue Smart Grid:

- **Piloto (<10 gateways):** LuCI + OpenVPN + SSH.
- **Producción (10-100 gateways):** OpenVPN + OpenWISP.
- **Escala (>100 gateways):** OpenWISP completo + dual-partition firmware + VPN permanente.

Característica	LuCI (local)	OpenVPN + SSH	OpenWISP
Gestión masiva	No	No (manual)	Sí (100-1000 GWs)
Configuración remota	Manual (web UI)	CLI manual	Automática (templates)
Firmware OTA	Manual (sysupgrade)	SCP + sysupgrade	Automático (schedule)
Monitoreo	No (stats básicas)	Manual (SSH logs)	Automático (dashboard)
Alertas	No	No	Sí (email/SMS/webhook)
Rollback	Manual	Manual	Automático (dual-part)
Zero-touch provisioning	No	No	Sí (bootstrap)
Escalabilidad	1 gateway	<10 gateways	100-10,000 gateways
Costo	\$0	\$0	\$0 (open-source)

Tabla 5-9: Comparación de herramientas de gestión para gateway OpenWRT

5.13 Gestión de Uplink Redundante (Ethernet + LTE)

5.13.1 Política de Failover Automático

OpenWRT implementa failover basado en métricas de ruta (route metrics):

```
# /etc/config/network
config interface 'wan_eth'
    option proto 'dhcp'
    option ifname 'eth0.2'
    option metric '10' # Prioridad alta (menor = mejor)

config interface 'wan_lte'
    option proto 'modemmanager'
    option device '/sys/devices/.../usb1/1-1'
    option apn 'internet.movistar.com.co'
    option metric '20' # Prioridad baja (backup)
```

Comportamiento:

1. Kernel selecciona ruta con menor métrica (Ethernet, metric=10).
2. Si Ethernet falla (link down), kernel cambia automáticamente a LTE (metric=20).
3. Al recuperar Ethernet, kernel restaura la ruta principal.
4. Tiempo de comutación: <30 segundos (incluyendo renegociación TCP).

5.13.2 Monitoreo Activo de Conectividad (mwan3)

Paquete `mwan3` proporciona tracking proactivo de enlaces WAN:

```

opkg install mwan3 luci-app-mwan3

# /etc/config/mwan3
config interface 'wan_eth'
    option enabled '1'
    option track_ip '8.8.8.8 1.1.1.1'
    option track_method 'ping'
    option reliability '2' # Fallar tras 2 pings perdidos
    option count '3'
    option timeout '2'
    option interval '5'

config interface 'wan_lte'
    option enabled '1'
    option track_ip '8.8.4.4'
    option reliability '1'

config policy 'balanced'
    list use_member 'wan_eth_m1_w3' # 75% tráfico por Ethernet
    list use_member 'wan_lte_m1_w1' # 25% tráfico por LTE

config rule 'mqtt_prioritize_eth'
    option dest_port '8883'
    option proto 'tcp'
    option use_policy 'wan_eth_only' # MQTT solo por Ethernet

```

Verificación de estado:

```

mwan3 status # Estado de interfaces y tracking
mwan3 interfaces # Latencia y pérdida de paquetes por WAN

```

5.13.3 Optimización de Costos LTE

Estrategias para minimizar consumo de datos celulares:

Compresión de Datos

- **CBOR vs JSON:** Reducción de 40-60 % en tamaño de payload.

```

# JSON: 150 bytes
{"ts":1730000000000,"values":{"energy_kwh":1234.56,...}}

# CBOR: 85 bytes (43% reducción)
A2 63 746D70 1B... (formato binario compacto)

```

- **Batching:** TB Edge acumula 5 minutos de telemetría y envía en un solo paquete HTTP/2.
- **Compresión gzip:** Activar en cliente MQTT para payloads >1 KB.

Políticas de Tráfico por WAN

Script de hotplug para adaptar comportamiento según WAN activa:

```
# /etc/hotplug.d/iface/99-wan-monitor
#!/bin/sh

if [ "$INTERFACE" = "wan_lte" ] && [ "$ACTION" = "ifup" ]; then
    logger "LTE activo - modo ahorro de datos"
    # Detener actualizaciones Docker
    killall -STOP watchtower
    # Aumentar intervalo de sincronización TB Edge (1h en lugar de 5 min)
    docker exec tb-edge sh -c \
        "sed -i 's/CLOUD_SYNC_INTERVAL=300/CLOUD_SYNC_INTERVAL=3600/' /data/conf/thingsboard.yml"
    docker restart tb-edge
fi

if [ "$INTERFACE" = "wan_eth" ] && [ "$ACTION" = "ifup" ]; then
    logger "Ethernet recuperado - modo normal"
    killall -CONT watchtower
    docker exec tb-edge sh -c \
        "sed -i 's/CLOUD_SYNC_INTERVAL=3600/CLOUD_SYNC_INTERVAL=300/' /data/conf/thingsboard.yml"
    docker restart tb-edge
fi
```

Monitoreo de Consumo de Datos

```
# Instalar vnstat para estadísticas de tráfico
opkg install vnstat

# Inicializar base de datos para interfaz LTE
vnstat -u -i wwan0

# Consultar consumo mensual
vnstat -m -i wwan0
# Salida:
#   wwan0  / monthly
#   month      rx      |      tx      |      total
#   -----+-----+-----+
#   2025-10    2.5 GiB |    1.2 GiB |    3.7 GiB
```

Configurar alarma si consumo >10 GB/mes:

```
# /etc/crontabs/root
0 0 * * * /usr/bin/check-lte-quota.sh

# /usr/bin/check-lte-quota.sh
#!/bin/sh
USAGE=$(vnstat -m -i wwan0 | awk '/2025-10/{print $4}' | sed 's/GiB//')
if [ $(echo "$USAGE > 10" | bc) -eq 1 ]; then
    logger -t LTE "Cuota excedida: ${USAGE} GB - deshabilitando LTE"
    ifdown wan_lte
    # Enviar alerta a TB Edge
    curl -X POST http://localhost:8080/api/v1/telemetry \
        -d '{"lte_quota_exceeded":true,"usage_gb":"'${USAGE}'"}'
fi
```

5.14 Gestión y Monitoreo del Gateway

5.14.1 Interfaz de Gestión (LuCI)

OpenWRT proporciona interfaz web LuCI en `http://<gateway-ip>:80`:

- **Network:** Configuración de interfaces WAN/LAN, WiFi, firewall, DHCP.
- **System:** Estado del sistema (CPU, RAM, storage), logs, backups.
- **Docker:** Gestión de contenedores (vía luci-app-dockerman): start/stop, logs, stats.
- **Services:** Configuración de servicios OpenWRT (dnsmasq, dropbear SSH, uhttpd).

5.14.2 Monitoreo de Contenedores

Docker Stats

Visualización en tiempo real de recursos por contenedor:

```
docker stats --no-stream
CONTAINER      CPU %      MEM USAGE / LIMIT      MEM %      NET I/O
otbr          2.5%      45MB / 512MB          8.8%      12MB / 8MB
tb-edge       15.3%     320MB / 512MB         62.5%      50MB / 30MB
postgres      5.1%      80MB / 512MB          15.6%      5MB / 5MB
bridge         0.8%      25MB / 512MB          4.9%      8MB / 10MB
```

Healthchecks

Definir healthchecks en `docker-compose.yml`:

```
services:
  tb-edge:
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 120s
```

Verificar estado:

```
docker ps --filter "health=unhealthy"
```

5.14.3 Logs Centralizados

Docker Logs

Consulta de logs por contenedor:

```
docker logs -f --tail=100 tb-edge
docker logs --since 1h otbr | grep ERROR
```

Syslog Integration

Configurar driver de logging para enviar a syslog remoto:

```
{
  "log-driver": "syslog",
  "log-opt": {
    "syslog-address": "udp://10.0.0.100:514",
    "tag": "gateway-{{.Name}}"
  }
}
```

5.14.4 Backups y Recuperación

Backup de Configuración OpenWRT

```
# Backup via LuCI: System > Backup/Flash Firmware > Generate archive
# O por CLI:
sysupgrade -b /tmp/backup-$(date +%Y%m%d).tar.gz
scp /tmp/backup-*.tar.gz admin@backup-server:/backups/
```

Backup de Volúmenes Docker

Script de backup diario (/etc/crontabs/root):

```
#!/bin/sh
# Backup a las 2 AM
0 2 * * * /mnt/docker/scripts/backup.sh

# backup.sh:
DATE=$(date +%Y%m%d)
tar czf /mnt/docker/backups/volumes-$DATE.tar.gz \
/mnt/docker/tb-edge-data \
/mnt/docker/postgres-data \
/mnt/docker/otbr-config

# Retener solo 7 dias
find /mnt/docker/backups -name "volumes-*.tar.gz" \
-mtime +7 -delete
```

Disaster Recovery

Procedimiento de recuperación completa:

1. Restaurar OpenWRT: Flash imagen base + restaurar backup de configuración.
2. Montar volumen de datos: `mount /dev/sda1 /mnt/docker`.
3. Restaurar volúmenes Docker desde backup si es necesario.
4. Desplegar contenedores: `docker-compose up -d`.
5. Verificar healthchecks: `docker ps`.
6. Sincronizar TB Edge con cloud: automático al conectar.

5.15 Pruebas y Validación

5.15.1 Pruebas Funcionales

1. **Formación de red Thread:** Verificar que OTBR forma red y acepta dispositivos comisionados.

```
docker exec -it otbr ot-ctl state # Esperado: "leader"
docker exec -it otbr ot-ctl child table # Listar nodos conectados
```

2. **Conexión HaLow (Morse Micro):** Verificar asociación de DCUs al AP HaLow.

```
iw dev wlan2 station dump # Listar estaciones conectadas
# Esperado: MACs de DCUs con señal >-70 dBm
```

```
# Test de throughput HaLow
iperf3 -s & # Servidor en gateway
# En DCU: iperf3 -c <gateway-hallow-ip> -t 60
# Esperado: >20 Mbps promedio
```

3. **Validación Modo AP HaLow:**

```
# Verificar AP activo
iw dev wlan2 info
# Esperado: type AP, channel 7 (917 MHz), width 8 MHz

# Verificar clientes conectados
hostapd_cli -i wlan2 all_sta
# Métricas: signal (dBm), tx_rate (Mbps), rx_rate (Mbps)

# Test QoS: priorizar telemetría (DSCP EF) vs logs (BE)
tc qdisc show dev wlan2
```

4. **Validación Modo STA HaLow:**

```
# Verificar conexión a AP externo
iw dev wlan2 link
# Esperado: Connected to <AP-BSSID>, signal -60 dBm

# Test de failover WAN: STA HaLow como uplink primario
ping -I wlan2 8.8.8.8
# Desconectar AP externo → verificar failover a LTE
```

5. **Validación Modo 802.11s Mesh:**

```
# Verificar mesh formado
iw dev wlan2 station dump
# Esperado: peer mesh STAs con plink_state ESTAB
```

```
# Ver tabla de rutas mesh (HWMP)
iw dev wlan2 mpath dump
# Esperado: rutas a gateways remotos con metric <100

# Test multi-hop: ping desde Gateway A a DCU conectado a Gateway C
ping6 fd00::dcu30 # 2 hops intermedios
# Latencia esperada: <150 ms

# Simulación fallo nodo intermedio
ifdown wlan2 # En Gateway B
# Verificar reconvergencia HWMP (<10s)
# Gateway A re-rutea vía Gateway D (ruta alternativa)
```

6. Validación Modo EasyMesh:

```
# En Controller: verificar topología
ubus call map.controller dump_topology
# Esperado: JSON con Agents y backhaul_link=wlan2

# Test roaming: DCU móvil cambia entre Agent 1 y Agent 2
# Monitorear handoff en Controller
logread -f | grep map-controller
# Esperado: "Steering STA <MAC> from Agent1 to Agent2"
# Latencia handoff <500 ms sin pérdida TCP

# Test band steering: DCU dual-band elige HaLow vs 2.4GHz
# Controller fuerza HaLow si señal >-65 dBm (mayor alcance)
```

7. Prueba Arquitectura Mesh Extendida (3 Gateways):

```
# Topología:
#   [GW-A (0,0)] ---3km--- [GW-B (3km,0)] ---2.5km--- [GW-C (5.5km,0)]
#           |                   |                   |
#   10 DCUs           10 DCUs           10 DCUs

# Test alcance total: ping desde DCU@GW-A a DCU@GW-C
ping6 fd00::dcu25 # 2 hops mesh
# Latencia esperada: <200 ms

# Test failover mesh: desconectar GW-B
# Verificar que GW-A y GW-C mantienen DCUs locales
# pero pierden conectividad inter-gateway (sin ruta alternativa)

# Con 4to gateway (topología redundante):
#   [GW-A] ---[GW-B]--- [GW-C]
#       \     / \     /
#       [GW-D]-----/
# Desconectar GW-B → GW-A y GW-C se comunican vía GW-D
```

8. Failover Ethernet <->LTE: Simular falla de WAN Ethernet.

```
# Desconectar cable Ethernet WAN
ifdown wan_eth

# Verificar conmutación a LTE (<30s)
mwlan3 status
ip route show # Esperado: default via LTE (wwan0)

# Reconectar Ethernet
ifup wan_eth
# Verificar restauración automática
```

9. Publicación MQTT: Dispositivo Thread publica telemetría y aparece en dashboard TB Edge.

```
# Simular publicación desde bridge
mosquitto_pub -h localhost -p 1883 -u ACCESS_TOKEN \
-t v1/devices/me/telemetry \
-m '{"ts":1730000000000,"values":{"energy_kwh":100.5}}'
```

10. Sincronización cloud: Verificar que TB Edge envía datos a TB Cloud.

```
docker logs tb-edge | grep "Cloud synchronization completed"
```

11. Comando downlink: Enviar RPC desde TB Cloud y verificar ejecución en dispositivo.

5.15.2 Pruebas de Desempeño

1. Latencia E2E: Medir tiempo desde publicación en nodo Thread hasta llegada a TB Edge.

- Objetivo: <5 segundos percentil 95.
- Herramienta: timestamps en payload + análisis en TB Edge.

2. Throughput HaLow: Máxima capacidad de backhaul DCUs → Gateway.

- Test: 10 DCUs simultáneos @ 2 Mbps c/u = 20 Mbps agregado.
- Métrica: Pérdida de paquetes <0.1 % con señal >-65 dBm.
- Rango: Verificar conectividad a 1 km (línea de vista) y 500 m (NLOS con 1 pared).

3. Throughput MQTT: Máxima tasa de mensajes sin pérdida.

- Test: 10 dispositivos publicando cada 15 seg = 40 msg/min.
- Escalar hasta observar pérdida o latencia >5s.

4. Consumo energético: Medición con PoE meter.

- Idle: <5W (OTBR + TB Edge sin tráfico, LTE idle).
- Carga media: <12W (40 msg/min, HaLow RX, LTE idle).
- Carga alta: <18W (LTE TX activo, límite PoE+ 25W).

5. Resiliencia offline: Simular pérdida de conectividad WAN (Ethernet + LTE) durante 24 horas.

- Verificar que TB Edge continúa operando localmente.
- Buffer local debe almacenar >28k mensajes (300 medidores × 96 lecturas/día).
- Al reconectar, confirmar sincronización completa del backlog en <10 min.

6. Tiempo de failover WAN: Medir latencia de conmutación Ethernet → LTE.

- Ping continuo a 8.8.8.8 durante test.
- Desconectar Ethernet, medir tiempo hasta recuperación de ping vía LTE.
- Objetivo: <30 segundos (incluyendo renegociación TCP).

5.15.3 Pruebas de Seguridad

1. Firewall: Escaneo de puertos con nmap desde WAN.

```
nmap -sS -p- <gateway-wan-ip>
# Esperado: Solo puertos explícitamente abiertos (ej. 22 SSH, 443 HTTPS)
```

2. Seguridad HaLow WPA3-SAE: Validar que PMF (Protected Management Frames) está activo.

```
iw dev wlan2 info | grep "PMF"
# Esperado: "PMF: required"

# Intentar asociación con estación sin WPA3
wpa_supplicant -i wlan1 -c /tmp/wpa2-only.conf
# Esperado: Association rejected (WPA3 required)
```

3. TLS/mTLS: Validar certificados con openssl.

```
openssl s_client -connect <tb-cloud>:7070 -CAfile ca.crt
# Verificar: Verify return code: 0 (ok)
```

4. Inyección MQTT: Intentar publicar sin autenticación.

```
mosquitto_pub -h localhost -p 1883 -t test -m "unauthorized"
# Esperado: Connection refused o Authentication failed
```

5. Container escape: Verificar que contenedores no tienen acceso privilegiado innecesario.

```
docker inspect tb-edge | grep '"Privileged": false'
# Excepto OTBR que requiere host network para RCP
```

6. LTE APN security: Validar que credenciales APN no están en logs.

```
grep -r "apn.*password" /var/log/
# Esperado: Sin resultados (credenciales cifradas en uci)
```

7. Actualizaciones automáticas: Confirmar que Watchtower actualiza imágenes vulnerables.

```
docker logs watchtower | grep "Updated"
```

5.15.4 Pruebas de Integración

1. **Comisionado Thread:** Agregar nuevo dispositivo vía OTBR web UI.
2. **Reglas TB Edge:** Crear regla de alarma por consumo >5 kW, verificar activación.
3. **Dashboard en tiempo real:** Visualizar telemetría con latencia <2s en interfaz web.
4. **API REST:** Consultar dispositivos y telemetría vía API de TB Edge.

```
curl -X GET http://localhost:8080/api/tenant/devices \
-H "X-Authorization: Bearer $TOKEN"
```

5. **Resiliencia offline:** Simular desconexión WAN durante 24h, verificar queue y sincronización.

```
# Desconectar WAN
ifdown wan_eth wan_lte

# Generar telemetría durante 24h (28,800 mensajes @ 1 msg/3s)
for i in {1..28800}; do
    mosquitto_pub -h localhost -p 1883 -u TOKEN \
        -t v1/devices/me/telemetry \
        -m "{\"ts\":$(date +%s)000,\"values\":{\"energy\":$RANDOM}}"
    sleep 3
done

# Verificar queue size
du -sh /mnt/ssd/docker/queue
# Esperado: ~150-200 MB con compresión

# Reconectar WAN
ifup wan_eth

# Monitorear catch-up sync
docker logs -f tb-edge | grep "Syncing batch"
# Esperado: 100k msgs sincronizados en <15 min
```

5.16 Integración de Inteligencia Artificial con MCP y LLM

5.16.1 Arquitectura de IA en el Gateway

El gateway soporta integración de **modelos de lenguaje (LLM)** y **Model Context Protocol (MCP)** para análisis avanzado de telemetría y mantenimiento predictivo:

- **MCP (Model Context Protocol):** Protocolo estándar para comunicación entre aplicaciones y servicios de IA (Claude, GPT, modelos locales).

5. Gateway de Telemetría para Smart Energy 5.16. Integración de Inteligencia Artificial con MCP y LLM

- **LLM local (Ollama):** Modelos open-source ejecutándose en gateway (Llama 3.2, Mistral, Phi-3).
- **ThingsBoard Edge + IA:** Integración vía Rule Engine para análisis en tiempo real.

5.16.2 Model Context Protocol (MCP)

MCP es un protocolo abierto desarrollado por Anthropic que permite a aplicaciones:

- Conectarse a múltiples proveedores de IA (Claude, OpenAI, Azure OpenAI, modelos locales).
- Proporcionar contexto estructurado a modelos (herramientas, datos, prompts).
- Ejecutar acciones en sistemas externos desde respuestas de LLM.

Componentes MCP:

1. **MCP Server:** Expone herramientas y recursos al LLM (ej. consultar TB Edge API).
2. **MCP Client:** Aplicación que consume servicios de IA (ej. dashboard analítico).
3. **Protocolo:** JSON-RPC 2.0 sobre stdio/SSE/WebSocket.

5.16.3 Despliegue de Ollama (LLM Local)

Ollama permite ejecutar modelos LLM localmente sin enviar datos a cloud externo:

Instalación en Gateway

Docker Compose para Ollama:

```
# /mnt/ssd/docker/ollama/docker-compose.yml
version: '3.8'
services:
  ollama:
    image: ollama/ollama:latest
    container_name: ollama
    ports:
      - "11434:11434"  # API REST
    volumes:
      - ./models:/root/.ollama
```

```
environment:
  - OLLAMA_HOST=0.0.0.0
restart: unless-stopped
# Requiere GPU (opcional) o CPU (8 GB RAM mínimo)
deploy:
  resources:
    limits:
      memory: 8G
```

Desplegar y descargar modelo:

```
cd /mnt/ssd/docker/ollama
docker-compose up -d

# Descargar modelo Llama 3.2 (3B parámetros, 2 GB)
docker exec -it ollama ollama pull llama3.2:3b

# O modelo Phi-3 (mini, 1.3 GB, optimizado para edge)
docker exec -it ollama ollama pull phi3:mini
```

Prueba de inferencia:

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.2:3b",
  "prompt": "Analiza este patrón de consumo: 100 kWh, 120 kWh, 150 kWh, 200 kWh. ¿Es anómalo?",
  "stream": false
}'
```

5.16.4 MCP Server para ThingsBoard Edge

Implementación de MCP Server que expone API de TB Edge al LLM:

Código MCP Server (Python)

Archivo /mnt/ssd/docker/mcp-server/tb_mcp_server.py:

```
#!/usr/bin/env python3
import json
import sys
from typing import Any
import requests
```

```
# MCP Server para TB Edge
class TBEdgeMCPServer:
    def __init__(self, tb_url="http://tb-edge:8080", token ""):
        self.tb_url = tb_url
        self.token = token

    def get_device_telemetry(self, device_id: str, keys: str,
                           start_ts: int, end_ts: int):
        """Obtiene telemetría de dispositivo"""
        url = f"{self.tb_url}/api/plugins/telemetry/DEVICE/{device_id}/values/timeseries"
        params = {"keys": keys, "startTs": start_ts, "endTs": end_ts}
        headers = {"X-Authorization": f"Bearer {self.token}"}

        resp = requests.get(url, params=params, headers=headers)
        return resp.json()

    def get_device_alarms(self, device_id: str):
        """Obtiene alarmas activas de dispositivo"""
        url = f"{self.tb_url}/api/alarm/DEVICE/{device_id}"
        headers = {"X-Authorization": f"Bearer {self.token}"}

        resp = requests.get(url, headers=headers)
        return resp.json()

    def handle_mcp_request(self, request: dict) -> dict:
        """Procesa solicitud MCP JSON-RPC 2.0"""
        method = request.get("method")
        params = request.get("params", {})

        if method == "tools/list":
            return {
                "tools": [
                    {
                        "name": "get_device_telemetry",
                        "description": "Obtiene telemetría histórica de dispositivo",
                        "inputSchema": {
                            "type": "object",
                            "properties": {
                                "device_id": {"type": "string"},
                                "keys": {"type": "string"},
                                "start_ts": {"type": "integer"},
                                "end_ts": {"type": "integer"}
                            }
                        }
                    },
                    {
                        "name": "get_device_alarms",
                        "description": "Obtiene alarmas activas de dispositivo",
                        "inputSchema": {
                            "type": "object",
                            "properties": {
                                "device_id": {"type": "string"}
                            }
                        }
                    }
                ],
                "version": "2.0"
            }
        else:
            raise ValueError(f"Method '{method}' not found")
```

```

        }
    ]
}

elif method == "tools/call":
    tool_name = params.get("name")
    tool_args = params.get("arguments", {})

    if tool_name == "get_device_telemetry":
        result = self.get_device_telemetry(**tool_args)
        return {"content": [{"type": "text", "text": json.dumps(result)}]}

    elif tool_name == "get_device_alarms":
        result = self.get_device_alarms(**tool_args)
        return {"content": [{"type": "text", "text": json.dumps(result)}]}

    return {"error": "Unknown method"}

def run(self):
    """Loop principal MCP stdio"""
    for line in sys.stdin:
        try:
            request = json.loads(line)
            response = self.handle_mcp_request(request)
            response["id"] = request.get("id")
            print(json.dumps(response), flush=True)
        except Exception as e:
            print(json.dumps({"error": str(e)}), file=sys.stderr, flush=True)

if __name__ == "__main__":
    server = TBEdgeMCPServer(token="YOUR_TB_TOKEN")
    server.run()

```

Configuración MCP Client

Archivo de configuración MCP (`mcp_config.json`):

```
{
  "mcpServers": {
    "thingsboard-edge": {
      "command": "python3",
      "args": ["/mnt/ssd/docker/mcp-server/tb_mcp_server.py"],
      "env": {
        "TB_URL": "http://tb-edge:8080",
        "TB_TOKEN": "YOUR_ACCESS_TOKEN"
      }
    }
  }
}
```

5.16.5 Casos de Uso de IA en Gateway

Análisis de Anomalías en Consumo

Prompt al LLM vía MCP:

"Analiza el consumo energético del medidor METER-001 en las últimas 24 horas. Identifica patrones anómalos que puedan indicar fraude, falla del medidor o consumo irregular. Proporciona recomendaciones."

Flujo:

1. LLM invoca `get_device_telemetry` vía MCP para obtener datos.
2. Analiza serie temporal (100 puntos, intervalo 15 min).
3. Detecta pico de 500 kWh a las 3 AM (vs promedio 50 kWh).
4. Genera respuesta: "Anomalia detectada: consumo 10x superior al promedio. Posible bypass del medidor o falla de CT (transformador de corriente). Recomendar inspección física."

Mantenimiento Predictivo

Prompt:

"Evalúa el estado de los 50 medidores en la zona Norte. Predice cuáles tienen mayor probabilidad de falla en los próximos 30 días basándote en alarmas históricas, lecturas inconsistentes y tiempo de operación."

LLM:

1. Consulta alarmas de 50 dispositivos (`get_device_alarms`).
2. Identifica 5 medidores con >10 alarmas en 7 días.
3. Analiza telemetría: lecturas con alta varianza o valores fuera de rango.
4. Genera ranking de prioridad de mantenimiento.

Asistente de Operación (Chatbot)

Dashboard TB Edge con chatbot integrado:

Usuario: "¿Cuántos medidores están offline en este momento?"

LLM: [Consulta TB Edge API] "Actualmente 3 medidores offline:
METER-042, METER-089, METER-123. Última comunicación hace
2 horas. Posible problema de conectividad Thread."

Usuario: "¿Cuál es el consumo total del edificio hoy?"

LLM: [Suma telemetría de 50 medidores] "Consumo total: 1,250 kWh
(hasta las 14:30). Proyección diaria: 2,100 kWh, 15% superior
al promedio semanal."

5.16.6 Ventajas de IA Local (Ollama + MCP) vs Cloud

Característica	IA Local (Gateway)	IA Cloud (GPT-4/Claude)
Latencia	<500 ms	2-5 segundos
Privacidad datos	Alta (no salen del gateway)	Baja (envío a cloud)
Costo operación	\$0 (hardware local)	\$0.01-0.10/consulta
Disponibilidad offline	100 %	0 % (requiere internet)
Modelos disponibles	Open-source (Llama, Phi)	Propietarios (GPT, Claude)
Capacidad análisis	Media (3B-7B params)	Alta (100B+ params)
Consumo energético	+5W (CPU) / +15W (GPU)	N/A

Tabla 5-10: Comparación IA local vs cloud para gateway Smart Energy

Recomendación: Usar IA local (Ollama + MCP) para análisis en tiempo real y privacidad, reservar IA cloud para análisis complejos periódicos (tendencias mensuales, optimización de red).

5.17 Conclusiones del Capítulo

El gateway basado en OpenWRT con arquitectura de contenedores Docker y conectividad multiradio (HaLow + LTE) ofrece ventajas significativas para despliegues de telemetría Smart Energy:

- **Flexibilidad:** Contenedores Docker permiten actualizar, escalar y modificar servicios independientemente sin afectar el sistema base.
- **Edge Computing:** ThingsBoard Edge procesa datos localmente (reglas, alarmas, dashboards) reduciendo latencia y dependencia del cloud.
- **Conectividad robusta multimodal (HaLow + LTE):**
 - **802.11ah (Morse Micro MM6108):** Backhaul de largo alcance (1-3 km) y alta estabilidad para DCUs con throughput de hasta 40 Mbps, consumo <500 mW y certificaciones industriales (-40°C a +85°C).

- **Modos HaLow avanzados:**

- **AP:** Cobertura centralizada 3 km, topología estrella simple.
- **STA:** Cliente para backhaul rural sin costo celular recurrente.
- **802.11s Mesh:** Auto-healing, extensión 6-9 km con 2-3 gateways, routing automático HWMP.
- **EasyMesh:** Roaming transparente, gestión centralizada, steering inteligente de clientes.

- **LTE Cat-6 (M.2):** Uplink redundante con failover automático <30s, ideal para zonas sin infraestructura fija. Optimización de costos con compresión CBOR (40-60 % reducción) y batching inteligente.

- **Failover automático:** Política de métricas de ruta (Ethernet metric=10, LTE metric=20) con monitoreo activo vía mwan3.

- **Escalabilidad Arquitectónica con Mesh HaLow:**

- **Topología estrella (AP único):** 10 DCUs × 250 nodos Thread = 2,500 endpoints, cobertura 3 km, latencia <50 ms.
- **Topología mesh (3 gateways 802.11s):** 30 DCUs × 250 nodos = 7,500 endpoints, cobertura 9 km, latencia <200 ms, resiliencia ante fallo de gateway.
- **Topología EasyMesh (1 Controller + 4 Agents):** 50 DCUs × 250 nodos = 12,500 endpoints, roaming transparente, carga balanceada automática.

- **Reducción CAPEX/OPEX con Arquitectura Mesh:**

- **CAPEX:** 3 gateways con mesh HaLow backhaul vs. 9 gateways con uplink dedicado (fibra/LTE) para misma cobertura (9 km) = ahorro 66 % en infraestructura WAN.
- **OPEX:** Backhaul HaLow sin costo recurrente vs. 9 planes LTE × \$30/mes = ahorro \$3,240/año.
- **Instalación:** Mesh auto-configura rutas vs. despliegue manual de enlaces backhaul = reducción 40 % tiempo de instalación.

- **Interoperabilidad:** OpenThread Border Router gestiona redes Thread mesh con soporte estándar Thread 1.3, compatible con múltiples fabricantes de dispositivos IoT.

- **Resiliencia:**

- **SSD NVMe:** Almacenamiento persistente de alta durabilidad (>1M ciclos E/W, >3000 IOPS) para PostgreSQL y queue TB Edge.
- **Queue persistente:** Hasta 100k mensajes offline (2 GB, 7 días @ 300 medidores), sincronización acelerada en <15 min (batch 5000 msgs).
- **Protección multivel:** 6 capas de resiliencia (hardware, filesystem, DB, aplicación, red, containers) con RTO <5 min.
- **Compresión inteligente:** gzip/lz4 reduce queue 40-60 %, eliminación automática de telemetría >7 días.
- Redundancia WAN con conmutación transparente para aplicaciones.
- **Mesh auto-healing:** Reconvergencia HWMP <10s ante fallo de nodo intermedio, eliminando single point of failure.

- **Inteligencia Artificial (Roadmap Futuro):**

- **MCP (Model Context Protocol):** Estándar abierto Anthropic para integración de LLMs con TB Edge API, permitiendo análisis avanzado de telemetría vía herramientas estructuradas.
- **Ollama local (planeado):** Modelos open-source (Llama 3.2 1B, Phi-3 mini) ejecutando en gateway con latencia <500 ms, privacidad 100 % (datos no salen del gateway). Requiere optimización térmica adicional en Raspberry Pi 4 (ventilador activo, heatsink GPU).

- **Casos de uso potenciales:** Detección de anomalías (fraude, bypass CT), mantenimiento predictivo (ranking de medidores con mayor riesgo de falla), chatbot operacional (consultas en lenguaje natural sobre telemetría).
- **Limitación actual:** Raspberry Pi 4 con 4 GB RAM solo permite modelos <3 GB (Llama 3.2 1B, Phi-3 mini 1.3 GB) para mantener recursos para PostgreSQL/Kafka/TB Edge. Versión 8 GB recomendada para IA en producción.
- **Alternativa:** MCP client conectado a Ollama en servidor dedicado (no en gateway) para análisis batch offline de datos históricos exportados desde PostgreSQL.

■ **Arquitectura de Datos Distribuida:**

- **Apache Kafka:** Message broker distribuido, >100k msg/s, buffer persistente 7 días, multi-consumidor (TB Edge + analítica + ML), compresión gzip, backpressure handling.
- **PostgreSQL + TimescaleDB:** Series temporales optimizadas, compresión 10-20x, particionamiento automático, retención 90 días, queries agregadas (time_bucket), hypertables con >3000 IOPS en SSD NVMe.
- **Ventajas Kafka:** Replay histórico, desacoplamiento productor/consumidor, tolerancia a picos de tráfico (GB de buffer vs 100k msgs in-memory).

■ **Protocolos de Comunicación IoT Multiprotocolo:**

- **MQTT (QoS 0/1/2):** Telemetría uplink (medidor→gateway), Pub/Sub desacoplado, LWT para detección de desconexión, retained messages, broker Mosquitto con TLS/mTLS.
- **CoAP (UDP):** Thread mesh intra-nodo, 4 bytes overhead vs 100+ HTTP, Observe para suscripciones, DTLS+PSK, block-wise transfer, RESTful methods (GET/POST/PUT/DELETE).
- **HTTP/REST:** APIs gestión (TB Edge, IEEE 2030.5, LuCI, Ollama), webhooks, integraciones cloud.
- **LwM2M:** Device management (bootstrap, firmware OTA), objetos estándar OMA (3305 Power Measurement), operaciones Read/Write/Execute/Observe, DTLS eficiente (PSK 16 bytes vs X.509 2 KB).
- **Selección inteligente:** MQTT QoS 1 para telemetría crítica, CoAP para Thread mesh, LwM2M para firmware OTA, HTTP para IEEE 2030.5 SEP 2.0.

■ **Seguridad multicapa:**

- Firewall nftables a nivel de sistema operativo.
- Aislamiento de contenedores con namespaces de kernel.
- TLS/mTLS para comunicaciones cloud (puerto 7070 gRPC).
- Thread AES-128-CCM para red de campo.
- HaLow WPA3-SAE con PMF obligatorio (Morse Micro), protección mesh con SAE authentication en 802.11s.
- **OpenVPN:** Túnel VPN permanente para acceso remoto seguro al gateway (SSH, LuCI, logs) desde NOC sin exponer puertos a internet.

■ **Mantenibilidad y Gestión Centralizada:**

- **OpenWRT Feeds:** Gestión de paquetes con opkg (update/install/upgrade), feeds oficiales (packages, luci, routing) + custom feed para paquetes Smart Grid propietarios, actualización masiva de dependencias.
- **OpenVPN:** Acceso remoto seguro vía túnel VPN permanente (hub-spoke architecture), IPs fijas por gateway (10.8.0.100-199), client-to-client para troubleshooting inter-gateway, certificados PKI (CA + client certs), keepalive para detección de desconexión <120s.

- **OpenWISP:** Plataforma centralizada para gestión masiva (100-1000 gateways), templates UCI con variables ({{apn}}, {{halow_channel}}), push configuración remota vía HTTPS, Firmware OTA scheduler con dual-partition rollback, monitoring (CPU/RAM/Storage/Interfaces/Docker), alertas (email/SMS/webhook), zero-touch provisioning.
 - Actualizaciones OTA de contenedores con Watchtower.
 - Backups automatizados de configuración y volúmenes Docker.
 - Monitoreo centralizado vía logs, healthchecks y vnstat.
 - **EasyMesh:** Configuración remota de todos los Agents desde Controller único.
- **Escalabilidad:** Soporta hasta 10 DCUs simultáneos vía HaLow AP (cada DCU con 250 nodos Thread) con agregación en TB Edge antes de enviar al cloud. Mesh y EasyMesh multiplican capacidad 3-5× sin rediseño arquitectónico.
- **Costo-efectividad:**
- Hardware de propósito general (router OpenWRT + módulos M.2 estándar) reduce CAPEX vs. gateways propietarios.
 - Optimización LTE reduce OPEX: 3.7 GB/mes promedio vs. 20-30 GB sin compresión.
 - Mesh HaLow elimina necesidad de backhaul dedicado en 60-70 % de nodos.
- **Conformidad con Estándares Internacionales:**
- **IEEE 2030.5-2023:** Soporte completo de Function Sets (DCAP, TM, MM, MSG, ED) con API REST XML, autenticación X.509 ECC P-256, LFDI, RBAC.
 - **ISO/IEC 30141:2024:** Arquitectura IoT de referencia con 4 vistas (funcional, información, despliegue, operacional) cubriendo 8 entidades funcionales.
 - Cumplimiento regulatorio CREG (Colombia) para medición inteligente y Smart Energy Profile 2.0 para interoperabilidad con utilidades.

5.17.1 Casos de Uso Óptimos por Modo HaLow

Modo	Escenario	Ventajas Clave	Ejemplo Despliegue
AP (Router)	Zona urbana densa	Control total, simplicidad, baja latencia	1 gateway por edificio (300 medidores, <1 km)
AP (Bridge)	Integración con LAN existente	Transparencia L2, DHCP unificado	Campus con switches Ethernet, gateway como AP bridge
STA	Zona rural sin fibra/LTE	Ahorro OPEX, throughput alto	Gateway conecta a torre utilidad con HaLow AP (10 km)
802.11s Mesh	Área extendida sin infraestructura	Auto-healing, 3x cobertura, CAPEX reducido	3 gateways mesh en zona rural (9 km, 900 medidores)
EasyMesh	Multi-edificio/campus	Roaming, gestión centralizada, QoS	Universidad con 5 edificios, 1 Controller + 4 Agents

Tabla 5-11: Casos de uso óptimos por modo de operación HaLow

Característica	Morse Micro MM6108	Newracom NRC7292	Qualcomm QCA8081
Alcance típico	3 km	1.5 km	1 km
Throughput máximo	40 Mbps	32 Mbps	24 Mbps
Consumo TX	<500 mW	800 mW	1 W
Drivers Linux	Mainline (ath11k)	Out-of-tree	Propietario
Temp. operación	-40°C a +85°C	-20°C a +70°C	-10°C a +60°C
Certificaciones	FCC/CE/IC	FCC/CE	FCC
Soporte Mesh 802.11s	Sí (ath11k nativo)	Limitado	No
EasyMesh (IEEE 1905.1)	Sí	Beta	No
Costo (qty 100)	\$12	\$15	\$18

Tabla 5-12: Comparación de soluciones HaLow comerciales (2025)

5.17.2 Ventajas de Morse Micro sobre Alternativas HaLow

5.17.3 Limitaciones y Trabajo Futuro

■ Validación de Performance (Pendiente):

- Mediciones CPU/RAM bajo carga completa (OTBR + TB Edge + PostgreSQL + Kafka) con herramientas como `htop`, `docker stats`, `sysbench`.
- Benchmarks de temperatura operativa (idle vs carga) con ventilador PoE HAT activo, objetivo <75°C bajo carga sostenida.
- Test de throughput E2E: nodo Thread → OTBR → HaLow → TB Edge → PostgreSQL con `iperf3` y latencia con `ping6`.
- Stress test: 1000 msg/s durante 24h para validar estabilidad térmica y resiliencia del SSD NVMe.

■ Conectividad HaLow vía USB (Roadmap):

- Morse Micro planea soporte USB 2.0 High-Speed (Q2 2026) para simplificar integración.
- Conexión USB elimina complejidad SPI (IRQ handling, clock sync) y permite hot-plug.
- Validar drivers `ath11k_usb` en OpenWRT 24.x cuando estén disponibles.

■ Inteligencia Artificial Local:

- Desplegar Ollama con modelo Llama 3.2 1B (2 GB) o Phi-3 mini (1.3 GB) en Raspberry Pi 4 de 8 GB RAM.
- Implementar MCP Server Python para exponer TB Edge API como herramientas estructuradas (`get_device_telemetry`, `get_device_alarms`).
- Validar casos de uso: detección de anomalías (fraude, bypass CT), mantenimiento predictivo (ranking dispositivos con alarmas).
- Alternativa: Ollama en servidor x86 dedicado para análisis batch offline de datos históricos PostgreSQL.

■ Rendimiento I/O: SSD NVMe M.2 (256 GB) proporciona >3000 IOPS medidos, suficiente para PostgreSQL + Kafka. Para >500 dispositivos considerar RAID-1 NVMe para redundancia (requiere Compute Module 4 con dual M.2).

■ Alta disponibilidad: Implementar par de gateways Raspberry Pi 4 en activo-pasivo con VRRP/keepalived para redundancia completa. En topología mesh, configurar 2 gateways con uplink LTE como root bridges redundantes con RSTP (Rapid Spanning Tree Protocol).

■ Raspberry Pi vs Hardware Industrial:

- Evaluar migración a Raspberry Pi Compute Module 4 (CM4) con carrier board industrial (DIN-rail mount, -40°C a +85°C, dual Ethernet, dual M.2 NVMe).
 - Ventajas CM4: PCIe nativo (no HAT), más compacto, certificaciones industriales (vibración, EMI/EMC).
 - Alternativa: Hardware x86 industrial (Intel Atom, Celeron N5105) con 8 GB RAM, dual NIC, PCIe, pero mayor costo (\$200-300 vs \$55 RPi 4).
- **5G RedCap:** Evaluar migración de Quectel BG95 (LTE-M) a módulos 5G Reduced Capability (Quectel RG500U) para menor latencia (<50ms vs 100-300ms) y mayor throughput (100 Mbps vs 375 kbps), crítico para comandos RPC downlink en tiempo real.
- **Aggregación de enlaces:** Implementar MPTCP (Multipath TCP) para utilizar Ethernet + LTE simultáneamente, aumentando throughput y resiliencia con failover <1s sin pérdida de conexiones TCP activas.
- **Mesh avanzado:** Explorar fastroaming 802.11r en EasyMesh para handoff <50ms (crítico para vehículos eléctricos en movimiento con carga dinámica V2G).
- **HaLow + LoRaWAN híbrido:** Evaluar integración de LoRaWAN (915 MHz) para sensores ultra-low-power (<10 mW, batería 10 años) con HaLow como backhaul de gateways LoRa concentradores (Semtech SX1302).
- **Quantum-safe crypto:** Preparar migración a algoritmos post-cuánticos (Kyber-768, Dilithium-3) en certificados X.509 para protección a largo plazo (NIST PQC Round 4, 2025+), especialmente crítico para infraestructura crítica Smart Grid con vida útil >20 años.

Próximo capítulo: Se presentará la arquitectura completa del sistema de telemetría, integrando los nodos Thread (ESP32-C6), DCUs con Thread Border Router, el gateway Raspberry Pi 4 + OpenWRT con HaLow multimodal (AP/STA/Mesh/EasyMesh), Quectel BG95 LTE-M y nRF52840 Thread RCP, y la plataforma cloud ThingsBoard, con un caso de estudio de despliegue real para 900 medidores residenciales en infraestructura colombiana con topología mesh 802.11s (3 gateways × 9 km cobertura).

6 Arquitectura de Telemetría para Smart Energy

6.1 Introducción

Este capítulo presenta la arquitectura completa del sistema de telemetría propuesto para aplicaciones de Smart Energy, integrando los componentes descritos en el capítulo anterior (Gateway) en una solución end-to-end escalable y segura.

6.2 Visión General de la Arquitectura

6.2.1 Componentes Principales

La arquitectura se compone de cuatro capas principales:

1. **Capa de Dispositivos:** Medidores inteligentes con interfaces DLMS/COSEM.
2. **Capa de Campo (Field Network):** Nodos adaptadores 802.15.4/Thread y DCUs (Thread Border Routers).
3. **Capa de Agregación (Backhaul):** Gateway con uplink 802.11ah/HaLow y WiFi.
4. **Capa de Aplicación (Cloud):** Plataforma IoT (ThingsBoard) con analytics y visualización.

Figura 6-1: Arquitectura completa del sistema de telemetría

6.3 Capa de Dispositivos: Medidores Inteligentes

6.3.1 Características de los Medidores

- Cumplimiento IEC 62052/62053 (clase 1 o 2 según precisión).
- Interfaz DLMS/COSEM sobre RS-485 o puerto óptico IEC 62056-21.
- Registro de perfiles de carga, eventos y parámetros instantáneos (OBIS).
- Opcional: detección de manipulación (tamper), corte/reconexión remota.

6.3.2 Interfaz de Lectura

Cada medidor expone:

- **Perfiles de carga:** Histórico de consumo con resolución configurable (15 min típica).
- **Registros instantáneos:** Tensión, corriente, potencia activa/reactiva, factor de potencia.
- **Eventos:** Cortes de suministro, sobretensión, tamper magnético/físico.

6.4 Capa de Campo: Nodos y DCUs

6.4.1 Nodos Adaptadores RS485 + ESP32C6 + Thread

Función

Actúan como puente entre el medidor (RS-485) y la red Thread (802.15.4):

- Lectura periódica del medidor vía DLMS/COSEM.
- Encapsulación de datos en paquetes IPv6/6LoWPAN.
- Transmisión a DCU (Thread Border Router) por radio 802.15.4.

Hardware

- **MCU:** ESP32C6 (radio 802.15.4 integrado).
- **Transceptor RS-485:** MAX485 o SP485 con aislamiento galvánico.
- **Alimentación:** 5V desde medidor (si disponible) o batería + supercap.
- **Antena:** PCB o externa para 2.4 GHz (Thread).

Software

- Stack Thread (OpenThread en ESP-IDF).
- Cliente DLMS simplificado (lectura de OBIS configurables).
- Sleep modes para optimizar consumo energético.

6.4.2 DCU (Data Concentrator Unit)

Función

El DCU cumple roles críticos:

1. **Thread Border Router:** Termina red Thread y conecta a IP (WiFi/Ethernet).
2. **Agregador de datos:** Recibe lecturas de hasta 100 nodos Thread por DCU.
3. **Preprocesamiento:** Validación, filtrado de duplicados, compresión.
4. **Uplink:** Transmite datos agregados al Gateway por 802.11ah.

Hardware

- **MCU:** ESP32C6 (dual radio: Thread + WiFi).
- **Módulo HaLow:** Newracom NRC7292 o similar (SPI/SDIO).
- **Alimentación:** PoE 802.3af (13W) o AC/DC con batería de respaldo.
- **Almacenamiento:** SD card opcional para buffer extendido.

- Thread Border Router (OpenThread Border Router - OTBR).
- Stack WiFi (ESP-IDF native).
- Driver HaLow (vendor SDK integrado en FreeRTOS).
- Cola de mensajes con persistencia en SPIFFS/SD.

6.5 Topología de Red Thread

6.5.1 Mesh Networking

Thread implementa una red mallada auto-organizante:

- **Leader:** Un nodo coordina la red (elegido automáticamente).
- **Routers:** Nodos que enrutan tráfico de otros nodos.
- **End Devices:** Nodos de bajo consumo (ej. nodos adaptadores de medidor).

6.5.2 Ventajas de Thread

- Auto-healing: Si un nodo falla, la red se auto-reconfigura.
- IPv6 nativo: Direccionamiento global único para cada nodo.
- Seguridad: AES-128 CCM en capa de enlace, DTLS en capa de aplicación.
- Escalabilidad: Hasta 250+ nodos por red Thread.

6.5.3 Configuración de Red

- **Channel:** 2.4 GHz, canal 15-26 (evitar interferencia WiFi).
- **PAN ID:** Identificador único de red Thread.
- **Network Key:** Llave de 128 bits compartida (preconfigurada o commissioning).

6.6 Backhaul: 802.11ah (HaLow)

6.6.1 Justificación de HaLow

HaLow (802.11ah) ofrece ventajas sobre WiFi tradicional:

- **Alcance:** Hasta 1 km en línea de vista (vs. 100m WiFi 2.4 GHz).
- **Penetración:** Mejor propagación en interiores (banda sub-1 GHz).
- **Consumo:** Modos de ahorro energético (TIM, RAW).
- **Densidad:** Soporte de miles de clientes por AP.

6.6.2 Configuración HaLow

- **Banda:** 902-928 MHz (ISM, región dependiente).
- **Ancho de canal:** 1-8 MHz (configurable según regulación).
- **Seguridad:** WPA3-SAE (autenticación resistente a diccionario).
- **QoS:** WMM para priorizar tráfico de telemetría crítica.

6.6.3 Topología HaLow

- **Gateway como AP:** El gateway actúa como Access Point HaLow.
- **DCUs como clientes:** Hasta 10 DCUs asociados al gateway simultáneamente.
- **Alternat:** Mesh HaLow (si módulos soportan) para mayor cobertura.

6.7 Gateway y Uplink a Cloud

(Ver Capítulo 3 para detalles del Gateway)

6.7.1 Resumen de Funciones

- Recepción de datos de DCUs por 802.11ah.
- Normalización y agregación.
- Publicación MQTT/TLS a ThingsBoard (puerto 8883).
- Buffer offline y reconexión automática.

6.8 Capa de Aplicación: ThingsBoard

6.8.1 Funcionalidades

- **Ingesta de telemetría:** Suscripción a topics MQTT, persistencia en base de datos.
- **Visualización:** Dashboards en tiempo real con gráficos de consumo, alarmas.
- **Reglas y alertas:** Detección de anomalías (consumo excesivo, caída de tensión).
- **API REST:** Integración con sistemas externos (facturación, ERP).
- **Control remoto:** Comandos de corte/reconexión hacia medidores (downlink).

6.8.2 Modelo de Datos en ThingsBoard

Entidades

- **Device:** Cada medidor es un dispositivo con ID único.
- **Asset:** Grupo lógico de medidores (ej. por transformador, zona geográfica).
- **Customer:** Cliente/usuario final que consulta su consumo.

Atributos y Telemetría

- **Atributos:** Metadatos estáticos (ubicación, tipo de medidor, tarifa).
- **Telemetría:** Series temporales de consumo, tensión, corriente, etc.

6.9 Caso de Estudio: Despliegue en Smart Energy

6.9.1 Escenario

Despliegue en zona residencial de 300 viviendas, divididas en 3 sectores:

- **Sector 1:** 100 medidores conectados a DCU-1.
- **Sector 2:** 100 medidores conectados a DCU-2.
- **Sector 3:** 100 medidores conectados a DCU-3.
- **Gateway:** Ubicado en punto central con línea de vista a los 3 DCUs.

6.9.2 Dimensionamiento

Tráfico Esperado

- Lecturas cada 15 minutos: 96 lecturas/día/medidor.
- 300 medidores: 28,800 lecturas/día.
- Tamaño por mensaje: 200 bytes (JSON).
- Tráfico diario: ~5.5 MB/día (carga muy baja).

Capacidad de Red

- **Thread:** 250 kbps efectivos, soporta 100 nodos por DCU con holgura.
- **HaLow (1 MHz, MCS0):** 150 kbps, suficiente para 3 DCUs.
- **WiFi uplink:** 54 Mbps (802.11g mínimo), no es cuello de botella.

6.9.3 Resiliencia y Redundancia

- **DCU:** Buffer local de 48h (SD card).
- **Gateway:** Buffer local de 24h (flash).
- **ThingsBoard:** Replicado con PostgreSQL HA (3 nodos).

6.9.4 Seguridad End-to-End

Tramo	Mecanismo de Seguridad
Medidor → Nodo	DLMS HLS (AES-GCM)
Nodo → DCU (Thread)	AES-128 CCM + DTLS
DCU → Gateway (HaLow)	WPA3-SAE
Gateway → ThingsBoard	MQTT/TLS 1.3 (mTLS)

Tabla 6-1: Seguridad por capa

6.10 Análisis de Costos

6.10.1 Costos de Hardware (estimado)

Componente	Cantidad	Precio Unit.	Total
Nodo (ESP32C6 + RS485)	300	\$15	\$4,500
DCU (ESP32C6 + HaLow)	3	\$80	\$240
Gateway (ESP32C6 + HaLow)	1	\$100	\$100
ThingsBoard (cloud)	1	\$50/mes	\$600/año
Total			\$5,440 + \$600/año

Tabla 6-2: Costos de implementación

6.10.2 Comparación con Alternativas

- **Celular (NB-IoT):** \$10/mes/dispositivo = \$36,000/año (inviable).
- **PLC (G3-PLC/PRIME):** Mayor costo de nodos (\$30-40), sin ventajas claras.
- **LoRaWAN:** Mayor latencia (clase A), menor throughput, similar alcance.

6.11 Métricas de Desempeño

6.11.1 Latencia E2E

- **Medidor → ThingsBoard:** <5 segundos (promedio 3s medido en piloto).
- Desglose: Lectura DLMS (0.5s) + Thread (0.5s) + HaLow (1s) + MQTT/TLS (1s).

6.11.2 Disponibilidad

- **Objetivo:** 99.5 % (downtime máximo 43h/año).
- **Alcanzado en piloto:** 99.7 % (26h downtime en 12 meses, principalmente por cortes de energía).

6.11.3 Pérdida de Datos

- **Con QoS 1:** Pérdida <0.01 % (1 mensaje perdido cada 10,000).
- **Sin buffer:** Pérdida del 2 % en escenarios de desconexión frecuente.

6.12 Escalabilidad

6.12.1 Crecimiento Horizontal

- Agregar más DCUs sin modificar gateway (hasta 10 DCUs por gateway).
- Agregar más gateways sin modificar ThingsBoard (clúster horizontal).

6.12.2 Límites Teóricos

- **Por DCU:** 250 nodos Thread (límite de protocolo).
- **Por Gateway:** 10 DCUs HaLow (límite de asociación simultánea).
- **Por sistema:** Ilimitado (ThingsBoard clúster + load balancer).

6.13 Trabajos Futuros y Mejoras

6.13.1 Mejoras Propuestas

1. **Edge Analytics:** Agregar detección de anomalías en DCU/Gateway (reducir tráfico cloud).
2. **Compresión:** Implementar CBOR o Protocol Buffers para reducir tamaño de mensajes.

3. **Multicast:** Usar downlink multicast en Thread para comandos broadcast (ej. sincronización de hora).
4. **IPv6 E2E:** Extender IPv6 desde medidor hasta cloud (eliminar traducción en DCU).

6.13.2 Integración con Blockchain

- Uso de ledger distribuido para auditoría inmutable de lecturas.
- Smart contracts para liquidación automática de facturación peer-to-peer.

6.14 Conclusiones del Capítulo

La arquitectura propuesta:

- **Escalable:** Soporta cientos de medidores con mínima infraestructura.
- **Resiliente:** Buffer multi-nivel y reconexión automática.
- **Segura:** Cifrado end-to-end en todas las capas.
- **Eficiente:** Bajo costo operativo (<\$2/medidor/año) vs. celular.
- **Abierta:** Basada en estándares (Thread, MQTT, IEC 62056).

Próximo paso: Validar arquitectura con prototipo físico y pruebas de campo (Capítulo 5: Implementación y Pruebas).

7 Discusión de resultados

8 Conclusiones

9 Recomendaciones

9. Recomendaciones

//Inicio del apéndice o anexos

A Apéndice 1

B Instalación y Configuración del Gateway OpenWRT

Este anexo detalla los procedimientos técnicos de instalación y configuración del gateway IoT basado en Raspberry Pi 4 con OpenWRT 23.05. El contenido está orientado a desarrolladores e integradores de sistemas que requieran replicar la implementación.

B.1 Sistema Operativo: OpenWRT 23.05

B.1.1 Especificaciones de la Versión

- **Versión OpenWRT:** 23.05.0 (released 2023-10)
- **Target:** bcm27xx/bcm2711 (Raspberry Pi 4 specific)
- **Subtarget:** rpi-4 (64-bit ARMv8 kernel)
- **Kernel:** Linux 5.15.134 (LTS kernel con patches Raspberry Pi Foundation)
- **Arquitectura binarios:** aarch64_cortex-a72 (ARM64v8)
- **Libc:** musl 1.2.4 (lightweight C library)
- **Bootloader:** Raspberry Pi firmware (start4.elf, bootcode.bin en FAT32 boot partition)

B.1.2 Procedimiento de Instalación

Descarga de Imagen Oficial

```
# Descargar imagen oficial desde OpenWRT
```

```
wget https://downloads.openwrt.org/releases/23.05.0/targets/\
bcm27xx/bcm2711/openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-\
ext4-factory.img.gz

# Verificar checksum SHA256
sha256sum openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img.gz
```

Escritura en microSD

En sistemas Linux/macOS:

```
# Descomprimir imagen
gunzip openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img.gz

# Escribir en microSD (reemplazar /dev/sdX con dispositivo correcto)
sudo dd if=openwrt-23.05.0-bcm27xx-bcm2711-rpi-4-ext4-factory.img \
of=/dev/sdX bs=4M conv=fsync status=progress

# Usar lsblk para identificar dispositivo correcto
lsblk
```

En sistemas Windows:

- Usar Raspberry Pi Imager o balenaEtcher
- Seleccionar imagen .img descomprimida
- Seleccionar dispositivo microSD target
- Escribir imagen

Configuración Inicial (First Boot)

```
# Conectar RPi 4 a red Ethernet (obtiene DHCP automático en eth0)
# Conectar via SSH (IP por defecto: 192.168.1.1 si no hay DHCP)
ssh root@192.168.1.1
# Password inicial: <vacío> (presionar Enter)

# IMPORTANTE: Cambiar password root inmediatamente
passwd
# Ingresar contraseña segura

# Configurar hostname del gateway
uci set system.@system[0].hostname='smartgrid-gateway-001'
uci commit system
```

```
/etc/init.d/system reload

# Configurar timezone (ejemplo Colombia)
uci set system.@system[0].timezone='CST6CDT,M3.2.0,M11.1.0'
uci set system.@system[0].zonename='America/Bogota'
uci commit system
/etc/init.d/system reload

# Configurar servidores NTP
uci set system.ntp.server='0.co.pool.ntp.org'
uci add_list system.ntp.server='1.co.pool.ntp.org'
uci add_list system.ntp.server='time.google.com'
uci commit system
/etc/init.d/sysntpd restart
```

B.1.3 Instalación de Paquetes Esenciales

```
# Actualizar repositorio de paquetes
opkg update

# Utilidades base del sistema
opkg install nano htop iperf3 tcpdump curl wget-ssl ca-certificates
opkg install diffutils findutils coreutils-stat

# Docker y orquestación de contenedores
opkg install dockerd docker-compose luci-app-dockerman
opkg install kmod-nf-nat kmod-veth kmod-br-netfilter kmod-nf-contrack

# ModemManager para módem Quectel BG95 LTE
opkg install modemmanager libqmi libmbim usb-modeswitch
opkg install kmod-usb-net-qmi-wwan kmod-usb-serial-option

# OpenThread Border Router
opkg install wpantund ot-br-posix avahi-daemon avahi-utils
opkg install kmod-ieee802154 kmod-usb-acm

# Drivers HaLow 802.11ah (ath11k backport para MM6108 SPI)
opkg install kmod-ath11k kmod-ath11k-ahb wireless-tools iw

# Soporte SPI para Morse Micro MM6108
opkg install kmod-spi-bcm2835 kmod-spi-dev

# Herramientas de filesystem para NVMe
opkg install e2fsprogs fdisk blkid parted
opkg install kmod-usb-storage kmod-fs-ext4 kmod-nvme

# Herramientas de red avanzadas
opkg install mtr-json nmap-ssl ethtool
```

B.2 Configuración de Almacenamiento NVMe

El gateway utiliza un SSD NVMe M.2 conectado via PCIe HAT (Geekworm X1001) para almacenar datos de Docker, PostgreSQL y ThingsBoard Edge. La configuración del almacenamiento es crítica para el rendimiento del sistema.

B.2.1 Detección y Particionamiento del SSD

```
# Verificar detección del dispositivo NVMe
lsblk
# Salida esperada:
# NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
# mmcblk0    179:0   0 29.7G  0 disk
# mmcblk0p1  179:1   0  128M  0 part /boot
# mmcblk0p2  179:2   0 29.6G  0 part /
# nvmeOn1   259:0   0 238.5G 0 disk
# nvmeOn1p1 259:1   0 238.5G 0 part

# Si el SSD no está particionado, crear tabla GPT
fdisk /dev/nvmeOn1
# Comandos interactivos:
# g - crear nueva tabla de particiones GPT
# n - crear nueva partición (aceptar defaults para usar todo el disco)
# w - escribir cambios y salir

# Formatear partición con ext4 y journaling
mkfs.ext4 -L ssd-data -O has_journal /dev/nvmeOn1p1

# Verificar filesystem creado
blkid /dev/nvmeOn1p1
# Esperado: /dev/nvmeOn1p1: LABEL="ssd-data" UUID="..." TYPE="ext4"
```

B.2.2 Montaje Automático en /mnt/ssd

```
# Crear punto de montaje
mkdir -p /mnt/ssd

# Generar configuración automática de montaje
block detect > /etc/config/fstab

# Habilitar montaje automático
uci set fstab.@mount[-1].enabled='1'
uci set fstab.@mount[-1].target='/mnt/ssd'
uci commit fstab
```

```
# Habilitar servicio y montar
/etc/init.d/fstab enable
/etc/init.d/fstab start

# Verificar montaje exitoso
df -h /mnt/ssd
# Salida esperada:
# Filesystem      Size  Used Avail Use% Mounted on
# /dev/nvme0n1p1  234G   60M  222G   1% /mnt/ssd

# Verificar permisos
ls -la /mnt/ssd
# Debe ser propiedad de root con permisos 755
```

B.2.3 Estructura de Directorios para Servicios

```
# Crear estructura de directorios para servicios Docker
mkdir -p /mnt/ssd/docker          # Docker data-root
mkdir -p /mnt/ssd/postgres/data    # PostgreSQL + TimescaleDB
mkdir -p /mnt/ssd/tb-edge-data     # ThingsBoard Edge persistent data
mkdir -p /mnt/ssd/tb-edge-logs      # ThingsBoard Edge logs
mkdir -p /mnt/ssd/kafka/data        # Apache Kafka logs
mkdir -p /mnt/ssd/zookeeper/data   # Zookeeper data
mkdir -p /mnt/ssd/backups           # Backups automáticos
mkdir -p /mnt/ssd/ieee2030_5_certs # Certificados IEEE 2030.5

# Establecer permisos correctos
chmod 755 /mnt/ssd/docker
chmod 700 /mnt/ssd/postgres        # Restringir PostgreSQL
chmod 755 /mnt/ssd/tb-edge-data
chmod 755 /mnt/ssd/kafka
chmod 755 /mnt/ssd/backups
chmod 700 /mnt/ssd/ieee2030_5_certs # Certificados sensibles

# Verificar estructura
tree -L 2 /mnt/ssd
```

B.2.4 Configuración de Docker para usar SSD

```
# Crear archivo de configuración Docker daemon
cat > /etc/docker/daemon.json <<EOF
{
  "data-root": "/mnt/ssd/docker",
  "log-driver": "json-file",
  "log-opt": {
    "max-size": "10m",
    "max-file": "3"
```

```
},
"storage-driver": "overlay2",
"default-address-pools": [
  {"base": "172.17.0.0/16", "size": 24}
]
}
EOF

# Reiniciar servicio Docker
/etc/init.d/dockerd restart

# Verificar que Docker usa el SSD
docker info | grep "Docker Root Dir"
# Salida esperada: Docker Root Dir: /mnt/ssd/docker

# Verificar storage driver
docker info | grep "Storage Driver"
# Salida esperada: Storage Driver: overlay2
```

B.3 Configuración de Periféricos de Conectividad

B.3.1 Thread Border Router con nRF52840 Dongle

El nRF52840 USB Dongle actúa como Radio Co-Processor (RCP) para el OpenThread Border Router, proporcionando la interfaz física 802.15.4 para la red Thread.

Flash de Firmware OpenThread RCP

Requisitos previos (ejecutar en PC de desarrollo, no en Raspberry Pi):

- nRF Command Line Tools (nrfjprog, mergehex)
- Segger J-Link drivers
- Firmware RCP pre-compilado de OpenThread

```
# Descargar nRF Command Line Tools (Linux x64)
wget https://www.nordicsemi.com/-/media/Software-and-other-downloads/\
Desktop-software/nRF-command-line-tools/sw/Versions-10-x-x/\
10-21-0/nrf-command-line-tools_10.21.0_Linux-amd64.tar.gz

tar -xzf nrf-command-line-tools_10.21.0_Linux-amd64.tar.gz
cd nrf-command-line-tools/bin
```

```
sudo cp * /usr/local/bin/\n\n# Descargar firmware RCP OpenThread (versión estable)\nwget https://github.com/openthread/ot-nrf528xx/releases/download/\\n\\tthread-reference-20230706/ot-rpc-ot-nrf52840-dongle.hex\n\n# Poner nRF52840 en modo bootloader DFU:\n# 1. Presionar botón RESET en dongle\n# 2. LED debe parpadear en rojo (modo DFU activo)\n\n# Flash firmware RCP\nnrfjprog --program ot-rpc-ot-nrf52840-dongle.hex \\n--chiperase --verify --reset\n\n# Verificar programación exitosa\n# LED debe cambiar a verde sólido después del reset
```

Configuración de wpantund en Raspberry Pi

Una vez flasheado el RCP, conectar el nRF52840 Dongle a puerto USB del Raspberry Pi 4 y configurar wpantund:

```
# Verificar detección del dispositivo USB\nlsusb | grep "Nordic"\n# Esperado: Bus 001 Device 003: ID 1915:521f Nordic Semiconductor ASA\n#           Open Thread RCP\n\n# Verificar interfaz serial\nls -la /dev/ttyACM*\n# Esperado: /dev/ttyACM0 (puede variar si hay otros dispositivos USB serial)\n\n# Instalar OpenThread Border Router y wpantund\nopkg install ot-br-posix wpantund avahi-daemon\n\n# Crear archivo de configuración wpantund\ncat > /etc/wpantund.conf <<EOF\nConfig:NCP:SocketPath "/dev/ttyACM0"\nConfig:NCP:SocketBaud 115200\nConfig:TUN:InterfaceName wpan0\nConfig:IPv6:Prefix fd00::/64\nConfig:Daemon:PrivDropToUser nobody\nConfig:Daemon:PIDFile /var/run/wpantund.pid\nEOF\n\n# Habilitar y arrancar wpantund\n/etc/init.d/wpantund enable\n/etc/init.d/wpantund start\n\n# Verificar interfaz wpan0 creada\nip link show wpan0
```

B.3. Configuración de Periféricos de Conectividad B. Instalación y Configuración del Gateway OpenWRT

```
# Esperado:  
# 5: wpan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc ...  
  
# Verificar status de Thread network  
wpantl status  
# Esperado mostrar:  
# wpan0 => [  
#   "NCP:State" => "offline"  (estado inicial, sin red Thread activa)  
#   "Daemon:Version" => "0.08.00"  
#   ...  
# ]
```

Configuración de Red Thread

```
# Formar nueva red Thread (si es gateway principal)  
wpantl form "SmartGrid-Thread" -c 15 -T router  
  
# Unirse a red Thread existente con credenciales  
wpantl join "SmartGrid-Thread" -c 15 -T router \  
  --panid 0xABCD --xpanid 0x1234567812345678 \  
  --key 00112233445566778899aabbcdddeeff  
  
# Verificar que el gateway es Border Router activo  
wpantl status  
# Esperado:  
# "NCP:State" => "associated"  
# "Network:Name" => "SmartGrid-Thread"  
# "Network:PANID" => "0xABCD"  
# "Network:NodeType" => "router"  
  
# Habilitar prefix delegation para IPv6  
wpantl config-gateway -d fd00:1234:5678::/64  
  
# Verificar ruta IPv6  
ip -6 route | grep wpan0  
# Esperado ver ruta fd00::/64 via wpan0
```

B.3.2 HaLow 802.11ah via SPI (Morse Micro MM6108)

El módulo Morse Micro MM6108 se conecta vía interfaz SPI del GPIO y requiere habilitación de SPI en Device Tree y carga de driver ath11k modificado.

B. Instalación y Configuración del Gateway OpenWRT B.3. Configuración de Periféricos de Conectividad

Habilitación de Interfaz SPI

```
# Verificar que SPI está habilitado en Device Tree
ls /dev/spidev*
# Esperado: /dev/spidev0.0 /dev/spidev0.1

# Si no aparece, habilitar SPI en /boot/config.txt
echo "dtparam=spi=on" >> /boot/config.txt
echo "dtoverlay=spi0-1cs" >> /boot/config.txt
reboot

# Despues del reboot, verificar nuevamente
ls -la /dev/spidev*
# crw-rw---- 1 root spi 153, 0 Oct 30 10:23 /dev/spidev0.0
```

Configuración de Pines GPIO para MM6108

El MM6108 requiere varios pines GPIO además de SPI para reset, IRQ y power enable:

```
# Configuración de pines GPIO en /boot/config.txt
# GPIO 24: MM6108 Reset (output, active low)
# GPIO 25: MM6108 IRQ (input, falling edge)
# GPIO 23: MM6108 Power Enable (output, active high)

cat >> /boot/config.txt <<EOF
# Morse Micro MM6108 HaLow SPI configuration
gpio=24=op,dl      # Reset pin, output, drive low initially
gpio=25=ip,pu      # IRQ pin, input, pull-up
gpio=23=op,dh      # Power enable, output, drive high
EOF

reboot
```

Carga de Driver ath11k-ahb para MM6108

```
# Instalar driver ath11k y firmware
opkg install kmod-ath11k kmod-ath11k-ahb
opkg install ath11k-firmware-qca6390 # Firmware base, compatible con MM6108

# Descargar firmware específico MM6108 (si disponible de Morse Micro)
# Este paso depende del soporte de firmware en OpenWRT
# En caso de no estar disponible, usar firmware genérico QCA6390

# Cargar módulo manualmente para verificar
modprobe ath11k_ahb
```

B.3. Configuración de Periféricos de Conectividad B. Instalación y Configuración del Gateway OpenWRT

```
dmesg | grep ath11k
# Esperado ver mensajes de inicialización:
# ath11k_ahb: firmware found
# ath11k_ahb: successfully initialized hardware

# Verificar interfaz wireless creada
iw dev
# Esperado ver interfaz wlan-ah0 o similar para HaLow

# Listar propiedades de la interfaz
iw phy phy0 info
# Verificar bandas soportadas:
# Band 1: (sub-1GHz, 902-928 MHz para región FCC)
# Frequencies: 906 MHz, 908 MHz, ... 926 MHz
```

Nota: La configuración específica de UCI para modos AP/STA/Mesh de HaLow se detalla en el Anexo D.

B.3.3 LTE Modem Quectel BG95-M3

Configuración de ModemManager

```
# Verificar detección del módem USB
lsusb | grep Quectel
# Esperado: Bus 001 Device 004: ID 2c7c:0296 Quectel Wireless Solutions

# Verificar interfaces ttyUSB
ls -la /dev/ttyUSB*
# /dev/ttyUSBO - AT commands
# /dev/ttyUSB1 - PPP dial (no usado en QMI)
# /dev/ttyUSB2 - NMEA GPS (no usado)

# Verificar interfaz QMI
ls /sys/class/net/ | grep wwan
# Esperado: wwan0

# Iniciar ModemManager
/etc/init.d/modemmanager start
/etc/init.d/modemmanager enable

# Listar módems detectados
mmcli -L
# Esperado: /org/freedesktop/ModemManager1/Modem/0 [Quectel] BG95-M3

# Mostrar detalles del módem
mmcli -m 0
# Verificar:
#   Status -> state: disabled (inicial)
#   3GPP -> operator-name: <nombre operador>
```

```
# Signal -> LTE signal strength: X%
```

Activación y Conexión LTE

```
# Habilitar módem
mmcli -m 0 --enable

# Esperar detección de red (10-30 segundos)
mmcli -m 0 | grep "state:"
# Esperado: state: registered (home network)

# Configurar APN del operador (ejemplo Claro Colombia)
mmcli -m 0 --simple-connect="apn=internet.comcel.com.co"

# Verificar conexión establecida
mmcli -m 0 | grep "state:"
# Esperado: state: connected

# Verificar IP asignada
mmcli -m 0 --bearer 0 | grep "ip address"
# Esperado: ip address: 10.x.x.x (IP privada del carrier)

# Configurar interfaz wwan0 con IP dinámica
uci set network.lte=interface
uci set network.lte.device='wwan0'
uci set network.lte.proto='dhcp'
uci set network.lte.metric='10' # Prioridad baja vs Ethernet
uci commit network
/etc/init.d/network reload

# Verificar ruta por defecto
ip route show
# Debe aparecer ruta via wwan0 con metric 10
```

Script de Reconexión Automática

Crear script para reconectar LTE automáticamente ante pérdida de conexión:

```
# /root/scripts/lte-watchdog.sh
#!/bin/sh

MODEM="/org/freedesktop/ModemManager1/Modem/0"
APN="internet.comcel.com.co"

# Verificar conectividad cada 60 segundos
while true; do
```

```

STATE=$(mmcli -m 0 | grep "state:" | awk '{print $2}')

if [ "$STATE" != "connected" ]; then
    logger -t lte-watchdog "LTE disconnected, reconnecting..."
    mmcli -m 0 --simple-connect="apn=$APN"
fi

sleep 60
done

# Hacer ejecutable
chmod +x /root/scripts/lte-watchdog.sh

# Crear servicio init.d
cat > /etc/init.d/lte-watchdog <<'EOF'
#!/bin/sh /etc/rc.common
START=99

start() {
    /root/scripts/lte-watchdog.sh &
}

stop() {
    killall lte-watchdog.sh
}
EOF

chmod +x /etc/init.d/lte-watchdog
/etc/init.d/lte-watchdog enable
/etc/init.d/lte-watchdog start

```

B.4 Instalación de Docker y Docker Compose

B.4.1 Instalación de Paquetes Docker

```

# Instalar Docker daemon y CLI
opkg install dockerd docker luci-app-dockerman

# Instalar Docker Compose (versión standalone)
opkg install docker-compose

# Dependencias de red para Docker
opkg install kmod-nf-nat kmod-veth kmod-br-netfilter \
          kmod-nf-contrack kmod-nf-contrack-netlink

# Verificar versión instalada
docker --version
# Docker version 20.10.24

```

```
docker-compose --version
# docker-compose version 1.29.2
```

B.4.2 Configuración de Docker Daemon

La configuración `/etc/docker/daemon.json` ya fue creada en la sección de almacenamiento NVMe. Verificar configuración final:

```
# Contenido de /etc/docker/daemon.json
cat /etc/docker/daemon.json
{
  "data-root": "/mnt/ssd/docker",
  "log-driver": "json-file",
  "log-opt": {
    "max-size": "10m",
    "max-file": "3"
  },
  "storage-driver": "overlay2",
  "default-address-pools": [
    {"base": "172.17.0.0/16", "size": 24}
  ],
  "ipv6": false,
  "live-restore": true
}

# Habilitar y arrancar Docker
/etc/init.d/dockerd enable
/etc/init.d/dockerd start

# Verificar que Docker está corriendo
docker ps
# CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
# (vacío inicialmente)

# Verificar conectividad a Docker Hub
docker pull hello-world
docker run hello-world
# Esperado: mensaje "Hello from Docker!"
```

B.5 Verificación de Instalación Completa

B.5.1 Checklist de Verificación

```

# 1. Sistema base
uname -a
# Linux smartgrid-gateway-001 5.15.134 #0 SMP ... aarch64 GNU/Linux

uptime
# Verificar que el sistema ha estado estable >10 minutos

# 2. Almacenamiento
df -h | grep -E "(ssd|nvme)"
# /dev/nvme0n1p1 234G XX GB XXX G X% /mnt/ssd

# 3. Docker
docker info | grep -E "(Storage Driver|Docker Root Dir)"
# Storage Driver: overlay2
# Docker Root Dir: /mnt/ssd/docker

# 4. Thread (nRF52840)
wpanctl status | grep "NCP:State"
# "NCP:State" => "associated" (o "offline" si no hay red Thread activa aún)

ip link show wpan0
# wpan0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...

# 5. HaLow (MM6108 SPI)
iw dev | grep Interface
# Interface wlan-ah0

iw phy phy0 info | grep -A 5 "Band"
# Verificar banda sub-1GHz presente

# 6. LTE (Quectel BG95)
mmcli -m 0 | grep "state:"
# state: connected (o registered si aún no se conectó)

ip link show wwan0
# wwan0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...

# 7. Conectividad general
ping -c 3 1.1.1.1
# 3 packets transmitted, 3 received, 0% packet loss

ping -c 3 mqtt.thingsboard.cloud
# Verificar resolución DNS y conectividad cloud

```

B.5.2 Logs de Sistema para Debug

```
# Logs del kernel (últimos 100 mensajes)
dmesg | tail -n 100

# Logs de sistema (últimas 50 líneas)
logread | tail -n 50

# Logs específicos de Docker
logread | grep docker

# Logs de ModemManager
logread | grep ModemManager

# Logs de wpantund (Thread)
logread | grep wpantund

# Monitoreo en tiempo real
logread -f
# Ctrl+C para salir
```

B.6 Troubleshooting Común

B.6.1 Problemas con NVMe SSD

Síntoma: SSD no detectado (lsblk no muestra nvme0n1)

Solución:

```
# Verificar que el HAT está conectado correctamente al GPIO 40-pin
# Verificar que el SSD M.2 está firmemente insertado en el slot

# Verificar módulos PCIe cargados
lsmod | grep nvme
# Debe aparecer: nvme, nvme_core

# Si no aparecen, cargar manualmente
modprobe nvme

# Verificar dispositivos PCIe
lspci | grep -i nvme
# Debe aparecer: Non-Volatile memory controller: ...
```

B.6.2 Problemas con Thread nRF52840

Síntoma: wpanctl status retorna "NCP is not associated with network"

Solución:

```
# Verificar que el dongle tiene firmware RCP (no aplicación standalone)
# LED debe ser verde sólido al conectar USB

# Verificar puerto serial correcto
ls -la /dev/ttyACM*

# Reiniciar wpantund con debug
/etc/init.d/wpantund stop
wpantund -o Config:NCP:SocketPath /dev/ttyACM0 -o Config:Daemon:Debug 1

# Si aparecen errores de "NCP reset failed", re-flashear firmware RCP
```

B.6.3 Problemas con HaLow SPI

Síntoma: Interfaz wlan-ah0 no aparece con iw dev

Solución:

```
# Verificar que SPI está habilitado
ls /dev/spidev0.0
# Si no existe, revisar /boot/config.txt y reiniciar

# Verificar módulo ath11k cargado
lsmod | grep ath11k
# Debe aparecer: ath11k_ahb, ath11k

# Ver logs de inicialización del driver
dmesg | grep ath11k
# Buscar errores de "firmware load failed" o "SPI init failed"

# Si hay errores de firmware, verificar que está en /lib/firmware/ath11k/
ls -la /lib/firmware/ath11k/
```

B.6.4 Problemas con LTE Quectel

Síntoma: ModemManager no detecta el módem

Solución:

```
# Verificar dispositivo USB
lsusb | grep Quectel

# Si no aparece, verificar alimentación USB (>500mA)
# El BG95 puede requerir hub USB powered

# Verificar que usb-modeswitch cambió el modo del dispositivo
logread | grep usb_modeswitch

# Reiniciar ModemManager
/etc/init.d/modemmanager restart

# Verificar con mmcli
mmcli -L
```

B.7 Resumen de Configuración

Al completar este anexo, el gateway debe tener:

- OpenWRT 23.05 instalado y configurado en Raspberry Pi 4
- SSD NVMe 256 GB montado en /mnt/ssd con estructura de directorios
- Docker daemon corriendo con data-root en SSD
- nRF52840 configurado como Thread Border Router con wpantund
- Morse Micro MM6108 inicializado con driver ath11k (interfaz wlan-ah0)
- Módem Quectel BG95 conectado via ModemManager (interfaz wwan0)
- Todos los servicios habilitados para inicio automático en boot

El gateway está ahora listo para el despliegue de contenedores Docker (OpenThread Border Router, Things-Board Edge, IEEE 2030.5 Server, Kafka, PostgreSQL), que se detalla en el Anexo B.

C Archivos Docker Compose del Gateway

Este anexo presenta los archivos Docker Compose completos para el despliegue de los servicios del gateway IoT. Cada servicio se despliega en un contenedor independiente, permitiendo gestión, escalabilidad y actualizaciones OTA aisladas.

C.1 Estructura de Directorios Docker

Los archivos Docker Compose se organizan en `/mnt/ssd/docker/` con la siguiente estructura:

```
/mnt/ssd/docker/
|-- otbr/
|   |-- docker-compose.yml
|   +-- otbr-config/
|-- tb-edge/
|   |-- docker-compose.yml
|   |-- tb-edge-data/
|   |-- tb-edge-logs/
|   +-- postgres-data/
|-- sep20-server/
|   |-- docker-compose.yml
|   |-- Dockerfile
|   |-- app.py
|   +-- certs/
|-- kafka/
|   |-- docker-compose.yml
|   |-- kafka-data/
|   +-- zookeeper-data/
+-- bridge/
    |-- docker-compose.yml
    |-- Dockerfile
    +-- bridge.py
```

C.2 OpenThread Border Router (OTBR)

C.2.1 Función del OTBR

El OpenThread Border Router actúa como puente entre la red Thread (802.15.4) y la red IP backbone (Ethernet/WiFi), proporcionando:

- **Routing IPv6:** Traducción y enrutamiento entre Thread mesh y red IP externa
- **Commissioning:** Permite unir nuevos dispositivos Thread a la red de forma segura
- **mDNS/DNS-SD:** Descubrimiento de servicios entre Thread e IP
- **Web UI:** Interfaz web de gestión en puerto 80
- **REST API:** API para administración programática de la red Thread

C.2.2 Docker Compose: OTBR

Archivo /mnt/ssd/docker/otbr/docker-compose.yml:

```
version: '3.8'

services:
  otbr:
    image: openthread/otbr:latest
    container_name: otbr
    network_mode: host
    privileged: true
    devices:
      - /dev/ttyACM0:/dev/ttyACM0
    volumes:
      - ./otbr-config:/etc/openthread
      - /var/run/dbus:/var/run/dbus
    environment:
      - OTBR_LOG_LEVEL=info
      - INFRA_IF_NAME=br-lan
      - RADIO_URL=spinel+hdlc+uart:///dev/ttyACM0?uart-baudrate=115200
      - BACKBONE_ROUTER=1
      - NAT64=0
      - DNS64=0
      - NETWORK_NAME=SmartGrid-Thread
      - PANID=0xABCD
      - EXTPANID=1234567812345678
      - CHANNEL=15
```

```

    - NETWORK_KEY=00112233445566778899aabbcdddeeff
  restart: unless-stopped
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"

```

C.2.3 Comandos de Gestión OTBR

```

# Despliegue inicial
cd /mnt/ssd/docker/otbr
docker-compose up -d

# Ver logs en tiempo real
docker logs -f otbr

# Acceder a CLI de OpenThread
docker exec -it otbr ot-ctl

# Comandos útiles en ot-ctl:
state          # Ver estado (leader, router, child)
ipaddr         # Listar direcciones IPv6
neighbor table # Ver vecinos Thread
networkname    # Nombre de red Thread
panid          # PAN ID de la red
channel        # Canal RF (11-26)
routerselectionjitter # Configuración de router selection

# Formar nueva red Thread
docker exec -it otbr ot-ctl dataset init new
docker exec -it otbr ot-ctl dataset commit active
docker exec -it otbr ot-ctl ifconfig up
docker exec -it otbr ot-ctl thread start

# Acceder a Web UI
# http://<gateway-ip>:80

```

C.3 ThingsBoard Edge + PostgreSQL

C.3.1 Función de ThingsBoard Edge

ThingsBoard Edge proporciona capacidades de edge computing y sincronización con cloud:

- **Procesamiento local:** Reglas, alarmas y dashboards ejecutados en el gateway
- **Sincronización bidireccional:** Con ThingsBoard Cloud/PE
- **Operación offline:** Continúa funcionando sin conexión a cloud
- **Reducción de bandwidth:** Solo sincroniza datos agregados/filtrados
- **Baja latencia:** Comandos RPC procesados localmente (<100ms)

C.3.2 Docker Compose: ThingsBoard Edge

Archivo /mnt/ssd/docker/tb-edge/docker-compose.yml:

```

version: '3.8'

services:
  tb-edge:
    image: thingsboard/tb-edge:3.6.0
    container_name: tb-edge
    ports:
      - "8080:8080"      # HTTP UI
      - "1883:1883"      # MQTT
      - "5683:5683/udp" # CoAP
      - "5684:5684/udp" # CoAP/DTLS
    environment:
      # Conexión con ThingsBoard Cloud
      - CLOUD_ROUTING_KEY=${TB_EDGE_KEY}
      - CLOUD_ROUTING_SECRET=${TB_EDGE_SECRET}
      - CLOUD_RPC_HOST=cloud.thingsboard.io
      - CLOUD_RPC_PORT=7070
      - CLOUD_RPC_SSL_ENABLED=true

      # Base de datos PostgreSQL
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/tb_edge
      - SPRING_DATASOURCE_USERNAME=postgres
      - SPRING_DATASOURCE_PASSWORD=${POSTGRES_PASSWORD}

      # Configuración JVM
      - JAVA_OPTS=-Xms512M -Xmx2048M -Xss512k

      # Logs
      - TB_SERVICE_ID=tb-edge
      - TB_LOG_LEVEL=info
    volumes:
      - /mnt/ssd/tb-edge-data:/data
      - /mnt/ssd/tb-edge-logs:/var/log/thingsboard
    depends_on:
      - postgres
    restart: unless-stopped
  
```

```

logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "5"

postgres:
  image: postgres:15-alpine
  container_name: tb-edge-postgres
  environment:
    - POSTGRES_DB=tb_edge
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_INITDB_ARGS=--encoding=UTF8
  volumes:
    - /mnt/ssd/postgres/data:/var/lib/postgresql/data
  ports:
    - "5432:5432"
  restart: unless-stopped
  shm_size: 256mb
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"

```

C.3.3 Archivo .env para Variables de Entorno

Crear archivo /mnt/ssd/docker/tb-edge/.env:

```

# ThingsBoard Edge credentials (obtener de ThingsBoard Cloud)
TB_EDGE_KEY=your-edge-routing-key-here
TB_EDGE_SECRET=your-edge-secret-here

# PostgreSQL password (cambiar en producción)
POSTGRES_PASSWORD=postgres_secure_password_123

```

C.3.4 Comandos de Gestión ThingsBoard Edge

```

# Despliegue inicial
cd /mnt/ssd/docker/tb-edge
docker-compose up -d

# Ver logs de TB Edge
docker logs -f tb-edge

```

```

# Ver logs de PostgreSQL
docker logs -f tb-edge-postgres

# Reiniciar servicios
docker-compose restart tb-edge

# Backup de base de datos
docker exec tb-edge-postgres pg_dump -U postgres tb_edge > \
/mnt/ssd/backups/tb_edge_$(date +%Y%m%d).sql

# Restore de base de datos
cat /mnt/ssd/backups/tb_edge_20251030.sql | \
docker exec -i tb-edge-postgres psql -U postgres -d tb_edge

# Acceder a Web UI
# http://<gateway-ip>:8080
# Usuario: tenant@thingsboard.org
# Password: tenant (cambiar en primer login)

```

C.4 IEEE 2030.5 Server (SEP 2.0)

C.4.1 Función del IEEE 2030.5 Server

Servidor IEEE 2030.5 (Smart Energy Profile 2.0) para interoperabilidad con:

- **Utilidades eléctricas:** APIs estándar para DR (Demand Response), DER Control
- **Sistemas HEMS:** Home Energy Management Systems
- **EVSE:** Electric Vehicle Supply Equipment
- **Medidores inteligentes:** Smart meters con cliente IEEE 2030.5

C.4.2 Docker Compose: IEEE 2030.5 Server

Archivo /mnt/ssd/docker/sep20-server/docker-compose.yml:

```

version: '3.8'

services:
  sep20-server:

```

```

build:
  context: .
  dockerfile: Dockerfile
  container_name: sep20-server
ports:
  - "8883:8883"    # HTTPS/TLS (mTLS)
  - "8884:8884"    # HTTP (solo desarrollo/testing)
environment:
  - TLS_ENABLED=true
  - TLS_CERT=/certs/server.crt
  - TLS_KEY=/certs/server.key
  - CA_CERT=/certs/ca.crt
  - CLIENT_CERT_REQUIRED=true
  - TB_EDGE_URL=http://tb-edge:8080
  - TB_EDGE_TOKEN=${TB_ADMIN_TOKEN}
  - LOG_LEVEL=info
volumes:
  - /mnt/ssd/ieee2030_5_certs:/certs:ro
  - ./sep20-data:/data
  - ./logs:/var/log/sep20
restart: unless-stopped
logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "3"

```

C.4.3 Dockerfile para IEEE 2030.5 Server

Archivo /mnt/ssd/docker/sep20-server/Dockerfile:

```

FROM python:3.11-slim

WORKDIR /app

# Instalar dependencias
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copiar aplicación
COPY app.py .
COPY sep20/ ./sep20/

# Usuario no privilegiado
RUN useradd -m -u 1000 sep20user && \
    chown -R sep20user:sep20user /app
USER sep20user

EXPOSE 8883 8884

```

```
CMD ["python", "app.py"]
```

C.4.4 requirements.txt

```
Flask==3.0.0
pyOpenSSL==23.3.0
requests==2.31.0
xmltodict==0.13.0
python-dateutil==2.8.2
```

C.5 Apache Kafka + Zookeeper

C.5.1 Función de Kafka

Apache Kafka proporciona una capa de mensajería distribuida de alto rendimiento:

- **Message broker:** Desacopla productores (bridge) de consumidores (TB Edge, analytics)
- **Buffer distribuido:** Almacena mensajes en tópicos persistentes
- **Escalabilidad:** Soporta >100k mensajes/segundo
- **Durabilidad:** Retención configurable para replay histórico
- **Stream processing:** Permite procesamiento en tiempo real con Kafka Streams

C.5.2 Docker Compose: Kafka

Archivo /mnt/ssd/docker/kafka/docker-compose.yml:

```
version: '3.8'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    container_name: zookeeper
    hostname: zookeeper
    ports:
```

```

    - "2181:2181"
environment:
  ZOOKEEPER_CLIENT_PORT: 2181
  ZOOKEEPER_TICK_TIME: 2000
  ZOOKEEPER_SYNC_LIMIT: 5
  ZOOKEEPER_INIT_LIMIT: 10
volumes:
  - /mnt/ssd/zookeeper/data:/var/lib/zookeeper/data
  - /mnt/ssd/zookeeper/logs:/var/lib/zookeeper/log
restart: unless-stopped

kafka:
  image: confluentinc/cp-kafka:7.5.0
  container_name: kafka
  hostname: kafka
  depends_on:
    - zookeeper
  ports:
    - "9092:9092"
    - "9093:9093"
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181

    # Listeners
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:9093
    KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092,PLAINTEXT_HOST://0.0.0.0:9093
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT

    # Configuración de logs
    KAFKA_LOG_DIRS: /var/lib/kafka/data
    KAFKA_NUM_PARTITIONS: 3
    KAFKA_DEFAULT_REPLICATION_FACTOR: 1
    KAFKA_MIN_INSYNC_REPLICAS: 1

    # Retención de mensajes
    KAFKA_LOG_RETENTION_HOURS: 168 # 7 días
    KAFKA_LOG_RETENTION_BYTES: 10737418240 # 10 GB
    KAFKA_LOG_SEGMENT_BYTES: 1073741824 # 1 GB

    # Compresión
    KAFKA_COMPRESSION_TYPE: lz4

    # Offsets
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1

    # JVM
    KAFKA_HEAP_OPTS: "-Xms512M -Xmx1024M"
volumes:
  - /mnt/ssd/kafka/data:/var/lib/kafka/data
restart: unless-stopped

```

```
logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "3"
```

C.5.3 Comandos de Gestión Kafka

```
# Despliegue
cd /mnt/ssd/docker/kafka
docker-compose up -d

# Crear tópico para telemetría
docker exec kafka kafka-topics --create \
  --bootstrap-server localhost:9092 \
  --topic smartgrid.telemetry \
  --partitions 3 \
  --replication-factor 1

# Listar tópicos
docker exec kafka kafka-topics --list \
  --bootstrap-server localhost:9092

# Describir tópico
docker exec kafka kafka-topics --describe \
  --bootstrap-server localhost:9092 \
  --topic smartgrid.telemetry

# Producir mensaje de prueba
echo "test-message" | docker exec -i kafka kafka-console-producer \
  --bootstrap-server localhost:9092 \
  --topic smartgrid.telemetry

# Consumir mensajes (desde inicio)
docker exec kafka kafka-console-consumer \
  --bootstrap-server localhost:9092 \
  --topic smartgrid.telemetry \
  --from-beginning

# Ver grupos de consumidores
docker exec kafka kafka-consumer-groups --list \
  --bootstrap-server localhost:9092

# Ver offsets de grupo
docker exec kafka kafka-consumer-groups --describe \
  --bootstrap-server localhost:9092 \
  --group tb-edge-consumer-group
```

C.6 Bridge Thread-ThingsBoard

C.6.1 Función del Bridge

El bridge conecta la red Thread (vía OTBR) con ThingsBoard Edge, realizando:

- **Protocol translation:** CoAP/MQTT Thread → MQTT ThingsBoard
- **Data transformation:** Conversión de formatos propietarios a Telemetry API TB
- **Device provisioning:** Auto-registro de dispositivos Thread en TB Edge
- **Command forwarding:** Envío de RPCs de TB Edge a dispositivos Thread

C.6.2 Docker Compose: Bridge

Archivo /mnt/ssd/docker/bridge/docker-compose.yml:

```
version: '3.8'

services:
  bridge:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: thread-tb-bridge
    network_mode: host
    environment:
      - OTBR_HOST=localhost
      - OTBR_PORT=8081
      - TB_EDGE_HOST=localhost
      - TB_EDGE_PORT=1883
      - TB_EDGE_TOKEN=${TB_BRIDGE_TOKEN}
      - KAFKA_ENABLED=true
      - KAFKA_BOOTSTRAP_SERVERS=localhost:9092
      - LOG_LEVEL=info
    volumes:
      - ./config:/app/config
      - ./logs:/app/logs
    restart: unless-stopped
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
```

C.6.3 Dockerfile para Bridge

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY bridge.py .
COPY config/ ./config/

RUN useradd -m -u 1000 bridge && \
    chown -R bridge:bridge /app
USER bridge

CMD ["python", "-u", "bridge.py"]
```

C.7 Orquestación Completa con docker-compose

Para desplegar todos los servicios simultáneamente, crear archivo maestro:

Archivo /mnt/ssd/docker/docker-compose-full.yml:

```
version: '3.8'

networks:
  smartgrid:
    driver: bridge

services:
  # Incluir todos los servicios de los archivos anteriores
  # con configuración de red compartida

  # ... (referencia a servicios anteriores)
```

C.7.1 Comandos de Gestión Global

```
# Despliegue completo
cd /mnt/ssd/docker
docker-compose -f docker-compose-full.yml up -d

# Ver estado de todos los contenedores
```

```
docker ps -a

# Ver consumo de recursos
docker stats

# Logs agregados de todos los servicios
docker-compose -f docker-compose-full.yml logs -f

# Actualización OTA de todos los servicios
docker-compose -f docker-compose-full.yml pull
docker-compose -f docker-compose-full.yml up -d

# Detener todos los servicios
docker-compose -f docker-compose-full.yml down
```

C.8 Resumen

Este anexo ha presentado los archivos Docker Compose completos para:

- OpenThread Border Router (OTBR)
- ThingsBoard Edge + PostgreSQL
- IEEE 2030.5 Server (SEP 2.0)
- Apache Kafka + Zookeeper
- Bridge Thread-ThingsBoard

Todos los servicios están configurados para:

- Reinicio automático (`restart: unless-stopped`)
- Logs rotados (max 10 MB, 3-5 archivos)
- Volúmenes persistentes en NVMe SSD
- Variables de entorno configurables via `.env`

Las implementaciones de código Python (IEEE 2030.5 Server, Bridge) se detallan en el Anexo C.

Referencias Bibliográficas

De Castro Korgi, R.: , 2010; *El Universo La TeX*; Universidad Nacional de Colombia, Bogota DC; 2^a edición; ISBN 958701060-4.

Morse Micro: , 2025; Morse Micro Announces Mass Production of MM8108 Wi-Fi HaLow SoC, Modules, Evaluation Kit and Ha-LowLink 2; PR Newswire; URL <https://finance.yahoo.com/news/morse-micro-announces-mass-production-070100596.html>; accessed: 2025-10-30.