



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 clara-
mente, 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 Theasaurus

Zusammenfassung

Nombre del trabajo o tesis en un tercer idioma

Zusammenfassung Texte.

Schlüsselwörter:

Lista de figuras

6-1	Arquitectura completa del sistema de telemetría.	21
------------	--	----

Lista de tablas

6-1	Seguridad por capa	26
6-2	Costos de implementación.	26

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	5
5.2.1 IEEE 2030.5-2023 (Smart Energy Profile 2.0)	5
5.2.2 ISO/IEC 30141:2024 (IoT Reference Architecture)	6
5.3 Requisitos del Gateway	6
5.3.1 Requisitos Funcionales	6
5.3.2 Requisitos No Funcionales	6
5.3.3 Requisitos de Seguridad	6
5.4 Arquitectura Jerárquica de 3 Niveles IoT	6
5.4.1 Nivel 1: Nodos IoT (End Devices)	7
5.4.2 Nivel 2: Routers IoT	7

Implementación de Protocolos basados en 6LowPAN para Smart Energy

5.4.3	Nivel 3: Gateways IoT (Border Routers Edge)	7
5.4.4	Justificación del Modelo de 3 Niveles	7
5.5	Arquitectura de Software del Gateway	7
5.5.1	Stack de Contenedores	7
5.5.2	Stack de Comunicación	8
5.6	Implementación del Gateway con OpenWRT	8
5.6.1	Justificación de la Plataforma	8
5.6.2	Hardware del Gateway	8
5.7	Implementación en Raspberry Pi 4 con OpenWRT	9
5.7.1	Hardware de la Implementación Real	9
5.7.2	Sistema Operativo: OpenWRT 23.05 en Raspberry Pi 4	9
5.7.3	Configuración de Conectividad	9
5.8	Flujo de Datos End-to-End	10
5.8.1	Flujo Normal de Operación	10
5.8.2	Flujo en Modo Edge (Sin Conectividad Cloud)	10
5.8.3	Flujo de Actualización OTA de Contenedores	10
5.9	Arquitectura de Datos: Kafka y PostgreSQL	11
5.9.1	Integración de Apache Kafka	11
5.9.2	PostgreSQL + TimescaleDB	11
5.10	Protocolos de Comunicación IoT	11
5.11	Resiliencia y Almacenamiento Persistente	12
5.11.1	Arquitectura de Almacenamiento	12
5.11.2	ThingsBoard Edge Queue: Resiliencia Offline	12
5.11.3	Resiliencia Multinivel	12
5.12	Gestión Remota del Gateway	13
5.12.1	Feeds de OpenWRT	13
5.12.2	OpenVPN: Acceso Remoto Seguro	13
5.12.3	OpenWISP: Gestión Centralizada de Gateways	13
5.12.4	Comparación de Herramientas de Gestión	14
5.13	Gestión de Uplink Redundante (Ethernet + LTE)	14
5.13.1	Política de Failover Automático	14
5.13.2	Monitoreo Activo de Conectividad (mwan3)	14
5.13.3	Optimización de Costos LTE	14
5.14	Gestión y Monitoreo del Gateway	15
5.14.1	Interfaz de Gestión (LuCI)	15
5.14.2	Monitoreo de Contenedores	15
5.14.3	Logs Centralizados	15
5.14.4	Backups y Recuperación	15
5.15	Pruebas y Validación	16
5.15.1	Pruebas Funcionales	16
5.15.2	Pruebas de Desempeño	16
5.15.3	Pruebas de Seguridad	16
5.15.4	Pruebas de Integración	16
5.16	Integración de Inteligencia Artificial con MCP y LLM	17

Implementación de Protocolos basados en 6LowPAN para Smart Energy

5.16.1	Arquitectura de IA en el Gateway	17
5.16.2	Model Context Protocol (MCP)	17
5.16.3	Despliegue de Ollama (LLM Local)	17
5.16.4	MCP Server para ThingsBoard Edge	17
5.16.5	Casos de Uso de IA en Gateway	17
5.16.6	Ventajas de IA Local vs Cloud	18
5.17	Conclusiones del Capítulo	18
5.17.1	Limitaciones y Trabajo Futuro	19
6	Arquitectura de Telemetría para Smart Energy	21
6.1	Introducción	21
6.2	Visión General de la Arquitectura	21
6.2.1	Componentes Principales	21
6.3	Capa de Dispositivos: Medidores Inteligentes	22
6.3.1	Características de los Medidores	22
6.3.2	Interfaz de Lectura	22
6.4	Capa de Campo: Nodos y DCUs	22
6.4.1	Nodos Adaptadores RS485 + ESP32C6 + Thread	22
6.4.2	DCU (Data Concentrator Unit)	23
6.5	Topología de Red Thread	23
6.5.1	Mesh Networking	23
6.5.2	Ventajas de Thread	23
6.5.3	Configuración de Red	23
6.6	Backhaul: 802.11ah (HaLow)	24
6.6.1	Justificación de HaLow	24
6.6.2	Configuración HaLow	24
6.6.3	Topología HaLow	24
6.7	Gateway y Uplink a Cloud	24
6.7.1	Resumen de Funciones	24
6.8	Capa de Aplicación: ThingsBoard	24
6.8.1	Funcionalidades	24
6.8.2	Modelo de Datos en ThingsBoard	25
6.9	Caso de Estudio: Despliegue en Smart Energy	25
6.9.1	Escenario	25
6.9.2	Dimensionamiento	25
6.9.3	Resiliencia y Redundancia	26
6.9.4	Seguridad End-to-End	26
6.10	Análisis de Costos	26
6.10.1	Costos de Hardware	26
6.10.2	Comparación con Alternativas	26
6.11	Métricas de Desempeño	27
6.11.1	Latencia E2E	27
6.11.2	Disponibilidad	27
6.11.3	Pérdida de Datos	27

Implementación de Protocolos basados en 6LowPAN para Smart Energy

6.12	Escalabilidad	27
6.12.1	Crecimiento Horizontal	27
6.12.2	Límites Teóricos	27
6.13	Trabajos Futuros y Mejoras	27
6.13.1	Mejoras Propuestas	27
6.13.2	Integración con Blockchain	28
6.14	Conclusiones del Capítulo	28
7	Discusión de resultados	29
8	Conclusiones	30
9	Recomendaciones	31
A	Apéndice 1	33
B	Instalación y Configuración del Gateway OpenWRT	34
B.1	Sistema Operativo: OpenWRT 23.05	34
B.1.1	Especificaciones de la Versión	34
B.1.2	Procedimiento de Instalación	34
B.1.3	Instalación de Paquetes Esenciales	36
B.2	Configuración de Almacenamiento NVMe	36
B.2.1	Detección y Particionamiento del SSD	36
B.2.2	Montaje Automático en <code>/mnt/ssd</code>	37
B.2.3	Estructura de Directorios para Servicios	37
B.2.4	Configuración de Docker para usar SSD	38
B.3	Configuración de Periféricos de Conectividad	39
B.3.1	Thread Border Router con nRF52840 Dongle	39
B.3.2	HaLow 802.11ah via SPI (Morse Micro MM6108)	41
B.3.3	LTE Modem Quectel BG95-M3	42
B.4	Instalación de Docker y Docker Compose	44
B.4.1	Instalación de Paquetes Docker	44
B.4.2	Configuración de Docker Daemon	45
B.5	Verificación de Instalación Completa	46
B.5.1	Checklist de Verificación	46
B.5.2	Logs de Sistema para Debug	47
B.6	Troubleshooting Común	47
B.6.1	Problemas con NVMe SSD	47
B.6.2	Problemas con Thread nRF52840	47
B.6.3	Problemas con HaLow SPI	48
B.6.4	Problemas con LTE Quectel	48
B.7	Resumen de Configuración	49
C	Archivos Docker Compose del Gateway	50
C.1	Estructura de Directorios Docker	50
C.2	OpenThread Border Router (OTBR)	51

Implementación de Protocolos basados en 6LowPAN para Smart Energy

C.2.1	Función del OTBR	51
C.2.2	Docker Compose: OTBR	51
C.2.3	Comandos de Gestión OTBR	52
C.3	ThingsBoard Edge + PostgreSQL	52
C.3.1	Función de ThingsBoard Edge	52
C.3.2	Docker Compose: ThingsBoard Edge	53
C.3.3	Archivo .env para Variables de Entorno	54
C.3.4	Comandos de Gestión ThingsBoard Edge	54
C.4	IEEE 2030.5 Server (SEP 2.0)	55
C.4.1	Función del IEEE 2030.5 Server	55
C.4.2	Docker Compose: IEEE 2030.5 Server	55
C.4.3	Dockerfile para IEEE 2030.5 Server	56
C.4.4	requirements.txt	56
C.5	Apache Kafka + Zookeeper	56
C.5.1	Función de Kafka	56
C.5.2	Docker Compose: Kafka	57
C.5.3	Comandos de Gestión Kafka	58
C.6	Bridge Thread-ThingsBoard	59
C.6.1	Función del Bridge	59
C.6.2	Docker Compose: Bridge	59
C.6.3	Dockerfile para Bridge	60
C.7	Orquestación Completa con docker-compose	60
C.7.1	Comandos de Gestión Global	61
C.8	Resumen	61
D	Anexo C: Scripts y Código de Integración	63
D.1	Servidor IEEE 2030.5 (SEP 2.0)	63
D.1.1	Aplicación Flask Principal	63
D.1.2	Dockerfile	67
D.1.3	requirements.txt	68
D.2	Bridge Thread ↔ ThingsBoard Edge	68
D.2.1	Script Bridge Principal	68
D.2.2	Dockerfile del Bridge	72
D.2.3	requirements_bridge.txt	72
D.3	Integración con Apache Kafka	73
D.3.1	Productor Kafka	73
D.3.2	Consumidor Kafka	75
D.3.3	requirements_kafka.txt	76
D.4	Scripts de Gestión	77
D.4.1	Comandos de Verificación	77
D.4.2	Backup de Configuraciones	77
E	Anexo D: Especificaciones IEEE 2030.5 y Configuraciones	79
E.1	Ejemplos XML IEEE 2030.5	79

Implementación de Protocolos basados en 6LowPAN para Smart Energy

E.1.1	Device Capability (DCAP)	79
E.1.2	Time Synchronization (TM)	79
E.1.3	Mirror Usage Point (MUP)	80
E.1.4	End Device List	82
E.2	Configuraciones UCI para HaLow 802.11ah	82
E.2.1	Modo Access Point (AP)	82
E.2.2	Modo Station (STA)	84
E.2.3	Modo Mesh 802.11s	85
E.2.4	Modo EasyMesh (IEEE 1905.1)	86
E.3	Optimización TimescaleDB	87
E.3.1	Configuración PostgreSQL + TimescaleDB	87
E.3.2	Schema y Hypertables	88
E.3.3	Queries de Ejemplo	90
E.3.4	Mantenimiento	90
E.4	Generación de Certificados X.509 para mTLS	91
E.4.1	Autoridad Certificadora (CA)	91
E.4.2	Certificado Servidor IEEE 2030.5	91
E.4.3	Certificado Cliente SEP 2.0	92
E.4.4	Prueba mTLS	92
F	Anexo E: Implementación Nodo IoT de Referencia	93
F.1	Arquitectura del Nodo	93
F.1.1	Hardware	93
F.1.2	Stack de Software	93
F.2	Código Principal	94
F.2.1	main.c	94
F.3	Cliente LwM2M	96
F.3.1	lwm2m_client.c (fragmento principal)	96
F.4	Objetos IPSO	100
F.4.1	temp_object.c	100
F.4.2	humidity_object.c	104
F.5	Objetos LwM2M Core	105
F.5.1	device_object.c (fragmento)	105
F.6	Conectividad Thread	108
F.6.1	thread_prov.c (fragmento)	108
F.7	CMakeLists.txt	109
F.7.1	Configuración de Build	109
F.8	sdkconfig.defaults	110
F.8.1	Configuración por Defecto	110
F.9	Uso del Nodo	111
F.9.1	Compilación y Flash	111
F.9.2	Comisionamiento Thread	111
F.9.3	Verificación LwM2M	112

Implementación de Protocolos basados en 6LowPAN para Smart Energy

G Configuraciones OpenWRT del Gateway	113
G.1 Configuraciones UCI Base	113
G.1.1 Network (/etc/config/network)	113
G.1.2 Wireless (/etc/config/wireless)	114
G.1.3 DHCP y DNS (/etc/config/dhcp)	116
G.2 Firewall nftables	117
G.2.1 Configuración Base (/etc/config/firewall)	117
G.2.2 Script nftables Personalizado	120
G.3 OpenVPN	122
G.3.1 Configuración Servidor	122
G.3.2 Generación de Certificados con Easy-RSA	123
G.3.3 Configuración Cliente (.ovpn)	124
G.4 OpenWISP	125
G.4.1 Docker Compose OpenWISP Controller	125
G.4.2 Archivo .env para OpenWISP	127
G.4.3 Configuración OpenWISP Agent en Gateway	128
G.5 mwan3: Multi-WAN Failover	128
G.5.1 Configuración Base (/etc/config/mwan3)	128
G.5.2 Script de Monitoreo mwan3	130
G.6 Scripts de Mantenimiento	132
G.6.1 Backup Automatizado de Configuraciones	132
G.6.2 Check LTE Quota	133
G.7 Resumen	134
Referencias Bibliográficas	135

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, actuando 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 transmitiéndolos de manera segura hacia la plataforma IoT en la nube. Este capítulo presenta la arquitectura de software y hardware del gateway, enfocándose en los aspectos conceptuales y de diseño. Los detalles técnicos de implementación (configuraciones UCI, docker-compose, scripts) se documentan en los anexos correspondientes.

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 funciones críticas de agregación de datos, traducción de protocolos, seguridad end-to-end, resiliencia mediante buffering local y edge computing para preprocesamiento. El gateway implementa una arquitectura jerárquica de 3 niveles IoT que optimiza la distribución de funciones y capacidad de procesamiento en redes de gran escala.

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), incluyendo los siguientes Function Sets:

- **Device Capability (DCAP):** Descubrimiento de capacidades (/dcap)
- **Time (TM):** Sincronización horaria NTP/PTP (<100 ms)
- **Metering Mirror (MM):** Datos de medición con granularidad 15 min
- **Messaging (MSG):** Notificaciones y alertas bidireccionales

- **End Device (ED)**: Registro y gestión de dispositivos

La seguridad IEEE 2030.5 se implementa mediante TLS 1.2/1.3 obligatorio, certificados X.509 ECC (curva P-256), LFDI derivado de certificado y RBAC para control de acceso. Los ejemplos completos de respuestas XML para todos los Function Sets se presentan en el **Anexo D**.

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

El gateway implementa múltiples entidades funcionales según la vista funcional de ISO/IEC 30141: Sensing, Actuation, Processing, Storage, Communication, Security, Management y Application Support. La arquitectura cumple con las cuatro vistas del estándar (funcional, información, despliegue y operacional), proporcionando un marco completo para sistemas IoT industriales.

5.3 Requisitos del Gateway

5.3.1 Requisitos Funcionales

El gateway debe cumplir con: recepción de datos de ≥ 10 DCUs simultáneamente mediante 802.11ah, normalización OBIS/DLMS/COSEM a JSON/CBOR, publicación MQTT con QoS 1/2 garantizando entrega, buffer persistente local mínimo 7 días, uplink redundante Ethernet WAN (primario) + LTE M.2 (backup < 30 s), Access Point HaLow (902-928 MHz) con alcance mínimo 1 km, API REST IEEE 2030.5 compatible y entidades funcionales ISO/IEC 30141 completas.

5.3.2 Requisitos No Funcionales

Latencia E2E < 5 segundos, disponibilidad $> 99.5\%$ con failover < 30 seg, consumo energético < 15 W (LTE idle), operación -10°C a $+50^{\circ}\text{C}$ (Morse Micro: -40°C a $+85^{\circ}\text{C}$), throughput HaLow mínimo 20 Mbps agregado, precisión sincronización < 100 ms y soporte ≥ 250 EndDevices simultáneos.

5.3.3 Requisitos de Seguridad

Autenticación mutua TLS 1.2/1.3, certificados X.509 con renovación automática, Secure Boot, cifrado de credenciales, OTA segura con validación de firma digital, certificados ECC P-256 para IEEE 2030.5, LFDI derivado de certificado, RBAC para APIs REST y WPA3-SAE con PMF obligatorio en HaLow.

5.4 Arquitectura Jerárquica de 3 Niveles IoT

La arquitectura propuesta sigue un modelo jerárquico que permite desplegar redes IoT con miles de dispositivos manteniendo eficiencia operativa, optimizando la distribución de funciones, consumo energético y

capacidad de procesamiento. Esta arquitectura, alineada con las implementaciones de referencia de Morse Micro para Wi-Fi HaLow, permite escalabilidad masiva.

5.4.1 Nivel 1: Nodos IoT (End Devices)

Dispositivos sensores y actuadores de bajo consumo optimizados para operación con baterías durante años. Implementan Thread (802.15.4) o HaLow 802.11ah en modo cliente con protocolos LwM2M sobre CoAP, MQTT-SN o IEEE 2030.5 Client. Características: MCU Cortex-M4/M33 (ESP32-C6, nRF52840), RAM 256 KB - 1 MB, modos sleep profundo, autonomía 5-10 años con batería AA. La implementación de referencia de nodo ESP32-C6 con LwM2M se documenta en el **Anexo E**.

5.4.2 Nivel 2: Routers IoT

Routers IoT que extienden el alcance de redes HaLow o Thread mediante mesh 802.11s, EasyMesh o Thread Router. Características: SoC MM8108 + MPU Linux, RAM 128-256 MB, OpenWRT mínimo sin Docker, PoE 802.3af/at. Su función es puramente extensión de cobertura y densificación de red, sin procesamiento edge ni gestión de dispositivos.

5.4.3 Nivel 3: Gateways IoT (Border Routers Edge)

Dispositivos con capacidades de cómputo significativas que actúan como agregación, procesamiento edge y puente entre redes IoT locales y WAN. Características: Plataformas ARM Cortex-A multi-core (Raspberry Pi 4), RAM 4-8 GB, NVMe SSD 64-256 GB, conectividad múltiple (HaLow, Thread, LTE/5G, Gigabit Ethernet), OpenWRT 23.05 con Docker, funciones avanzadas de Border Routing, Edge Computing, protocolos Smart Energy (IEEE 2030.5) y orquestación de contenedores.

5.4.4 Justificación del Modelo de 3 Niveles

Ventajas: (1) Escalabilidad - un gateway gestiona 100-200 nodos directamente, escalando a 1000+ con routers intermedios; (2) Eficiencia energética - nodos transmiten en saltos cortos reduciendo potencia; (3) Cobertura extendida - HaLow >1 km con routers mesh alcanza 3-5 km urbano; (4) Resiliencia - mesh reconfigura rutas automáticamente; (5) Distribución de carga optimizada; (6) Costo optimizado versus múltiples gateways costosos.

5.5 Arquitectura de Software del Gateway

5.5.1 Stack de Contenedores

El gateway implementa una arquitectura basada en contenedores Docker con servicios desacoplados: (1) OpenThread Border Router para gestión de red Thread y ruteo IPv6; (2) ThingsBoard Edge como platafor-

ma IoT local; (3) PostgreSQL con extensión TimescaleDB; (4) Bridge Thread-TB para integración OTBR TB Edge; (5) MQTT Broker Mosquitto; (6) IEEE 2030.5 Server; (7) Apache Kafka como bus de mensajes.

Los archivos `docker-compose.yml` completos para todos los servicios se presentan en el **Anexo B**.

5.5.2 Stack de Comunicación

Capa física: 802.15.4/Thread (RCP nRF52840 vía USB), 802.11ah HaLow (Morse Micro MM6108 vía SPI, 902-928 MHz, hasta 3 km, 40 Mbps), 802.11ac/ax WiFi dual-band, LTE Cat-6 M.2 y Ethernet Gigabit. Capa de red: IPv6 Thread (fd00::/64) ruteado por OTBR, IPv4 NAT para WAN. Capa de transporte: TCP/TLS (puerto 7070), MQTT/TLS (1883/8883), CoAP/UDP. Capa de aplicación: MQTT, HTTP/REST, WebSockets, JSON.

Las configuraciones de red UCI completas se documentan en el **Anexo F**.

5.6 Implementación del Gateway con OpenWRT

5.6.1 Justificación de la Plataforma

OpenWRT se selecciona por flexibilidad (Linux embebido con opkg/UCI), soporte Docker para contenedorización, redes avanzadas (VLAN, nftables, QoS, IPv6), amplio soporte de hardware con expansión de almacenamiento y comunidad activa con actualizaciones frecuentes.

5.6.2 Hardware del Gateway

Plataforma Base

Dos opciones: (1) Router industrial: SoC MediaTek MT7621AT (MIPS dual-core 880 MHz), RAM 512 MB DDR3, Flash 16 MB + USB 3.0/NVMe 32 GB, Ethernet 5 puertos Gigabit con PoE+; (2) Raspberry Pi 4 Model B: BCM2711 Cortex-A72 quad-core ARMv8 @ 1.5 GHz, 4 GB RAM, microSD 32 GB + M.2 NVMe SSD 256 GB via PCIe HAT, alimentación PoE+ HAT.

Conectividad 802.11ah (HaLow) con Morse Micro

Chipset MM6108 SoC con interfaz PCIe/SDIO/SPI, frecuencia 902-928 MHz con canales 1/2/4/8 MHz, alcance hasta 1-3 km LOS con antena externa 5 dBi, throughput hasta 40 Mbps (MCS10, 8 MHz BW), seguridad WPA3-SAE con PMF obligatorio. Ventajas Morse Micro: operación industrial -40°C a +85°C, drivers Linux mainline (ath11k), consumo <500 mW TX/<50 mW RX, certificaciones FCC/CE.

Modos de Operación HaLow: (1) AP (Access Point) - gateway como punto de acceso central; (2) STA (Station) - gateway como cliente conectado a AP externo; (3) 802.11s Mesh - malla autogestionada entre

múltiples gateways con auto-healing; (4) EasyMesh - IEEE 1905.1 con roaming transparente y gestión centralizada.

Las configuraciones UCI completas para los cuatro modos HaLow, incluyendo ejemplos de verificación, pruebas de throughput y troubleshooting, se documentan en el **Anexo D**.

5.7 Implementación en Raspberry Pi 4 con OpenWRT

5.7.1 Hardware de la Implementación Real

El prototipo se implementó sobre Raspberry Pi 4 Model B por sus capacidades multi-core y memoria RAM esenciales para múltiples contenedores Docker. Justificación vs Router MT7621AT: 4 núcleos Cortex-A72 permiten paralelización sin contención, 4 GB RAM suficientes para PostgreSQL/Kafka/TB Edge, ecosistema ARM64 con imágenes Docker oficiales, PCIe para NVMe con >3000 IOPS crítico para PostgreSQL, GPIO/SPI flexible.

Periféricos y Módulos de Conectividad

(1) **Thread**: Nordic nRF52840 Dongle con firmware OpenThread RCP v1.3, interfaz USB 2.0 (/dev/ttyACM0), potencia TX +8 dBm, sensibilidad -95 dBm; (2) **HaLow**: Morse Micro MM6108 vía SPI0 (GPIO 8/9/10/11/25), driver `ath11k` mainline, identificación `wlan2`; (3) **LTE**: Quectel BG95-M3 (Cat-M1/NB-IoT + EGPRS), interfaz USB (`wwan0`), throughput 375 kbps, latencia 100-300 ms; (4) **Almacenamiento**: Kingston NV2 M.2 NVMe 256 GB vía PCIe HAT (350-400 MB/s lectura, 3200-3500 IOPS 4K random); (5) **Alimentación**: Waveshare PoE HAT IEEE 802.3at (25.5W máx), salida 5V/5A, ventilador PWM (encendido $T^{\circ} > 60^{\circ}\text{C}$).

La conexión SPI del módulo HaLow, habilitación en OpenWRT y verificación de interfaz se documentan en el **Anexo F**.

5.7.2 Sistema Operativo: OpenWRT 23.05 en Raspberry Pi 4

OpenWRT 23.05.0, target `bcm27xx/bcm2711` (ARMv8 64-bit), kernel Linux 5.15.134 LTS, arquitectura binarios `aarch64_cortex-a72`, libc musl 1.2.4. Los procedimientos completos de instalación (descarga, escritura en microSD, configuración inicial, actualización de paquetes, configuración de almacenamiento NVMe con `fstab`, directorios Docker) se documentan en el **Anexo A**.

5.7.3 Configuración de Conectividad

El gateway integra múltiples interfaces: Thread 802.15.4 (OTBR con nRF52840 RCP formando red SmartGrid-Thread en canal 15), HaLow 802.11ah (MM6108 vía SPI soportando 4 modos: AP Router con NAT, STA Client, Mesh 802.11s con HWMP routing, EasyMesh 1905.1 con Controller/Agent), LTE Cat-M1/NB-IoT (Quectel BG95-M3 con failover automático vía `mwan3`) y Ethernet Gigabit (WAN primaria).

Ejemplo de verificación de interfaces activas:

```
# Thread Border Router
docker exec otbr ot-ctl state # Esperado: "leader" o "router"

# HaLow 802.11ah
iw dev wlan2 info # Esperado: type AP, channel 7 (917 MHz)

# LTE modem
mmcli -m 0 --simple-status # Esperado: state: connected

# Ethernet WAN
cat /sys/class/net/eth0/operstate # Esperado: up (1000BASE-T)
```

Las configuraciones UCI completas para HaLow en sus cuatro modos de operación se presentan en el **Anexo D**.

5.8 Flujo de Datos End-to-End

5.8.1 Flujo Normal de Operación

Medidor → Nodo Thread (ESP32C6) vía RS-485/DLMS → OTBR (ruteo IPv6 desde fd00::/64 a LAN) → Bridge (transformación CoAP/MQTT → formato ThingsBoard JSON) → TB Edge (procesamiento Rule Engine, almacenamiento PostgreSQL, actualización dashboards) → TB Cloud (sincronización gRPC/TLS puerto 7070 cada 5 min) → Visualización dashboards.

El flujo inverso para comandos downlink sigue: TB Cloud → TB Edge (validación permisos RBAC) → Bridge (traducción a protocolo nodo LwM2M Write / IEEE 2030.5 DER Control) → Routers mesh (reenvío) → Nodo (ejecución + ACK).

5.8.2 Flujo en Modo Edge (Sin Conectividad Cloud)

Gateway detecta pérdida WAN (ping a 8.8.8.8 falla), TB Edge activa modo offline continuando operación local (reglas, dashboards accesibles via LAN), datos se acumulan en queue persistente PostgreSQL + filesystem (límite 100k msgs o 2 GB), al recuperar conectividad sincroniza automáticamente backlog completo en 10-15 minutos con batch size 5000 y compresión gzip.

5.8.3 Flujo de Actualización OTA de Contenedores

Watchtower container verifica actualizaciones de imágenes Docker cada 24h, si nueva versión disponible descarga imagen, detiene contenedor actual, crea nuevo con misma configuración (volúmenes, redes), si healthcheck OK elimina imagen antigua, si falla rollback automático a imagen anterior. Logs de actualización en /mnt/docker/watchtower/watchtower.log.

5.9 Arquitectura de Datos: Kafka y PostgreSQL

5.9.1 Integración de Apache Kafka

Kafka proporciona message broker distribuido de alto rendimiento: intermedia entre bridge (productor) y TB Edge (consumidor), buffer distribuido con tópicos persistentes (telemetry, alarms), soporta >100k msg/s con múltiples particiones, retención configurable (7 días default). Ventajas vs in-memory queue: capacidad GB vs 100k msgs, replay histórico desde offset específico, multi-consumidor (TB Edge + analítica + ML simultáneamente), backpressure absorption sin pérdida de mensajes.

El docker-compose completo de Kafka (Zookeeper + Kafka broker) y scripts Python para productor/consumidor se documentan en **Anexo B** y **Anexo C**.

5.9.2 PostgreSQL + TimescaleDB

PostgreSQL con extensión TimescaleDB almacena: telemetría histórica (series temporales optimizadas con compresión 10-20×, particionamiento automático por tiempo en chunks de 7 días, agregaciones rápidas con `time_bucket`), configuración de dispositivos (atributos, credenciales, relaciones), alarmas/eventos (log persistente para auditoría) y dashboards/reglas de TB Edge.

El esquema completo de TimescaleDB incluyendo definición de hypertables, políticas de compresión, continuous aggregates (vistas materializadas para agregaciones de 15-min, 1-hora y 1-día), políticas de retención (90 días) y cinco consultas SQL de ejemplo se presenta en el **Anexo D**.

5.10 Protocolos de Comunicación IoT

El gateway implementa múltiples protocolos según caso de uso:

- **MQTT (QoS 0/1/2)**: Telemetría uplink (medidor→gateway), patrón Pub/Sub desacoplado, QoS garantizado (QoS 1 at least once, QoS 2 exactly once), Last Will Testament para detección de desconexión, retained messages para último valor, broker Mosquitto local con TLS/mTLS
- **CoAP (UDP)**: Thread mesh intra-nodo, overhead 4 bytes vs 100+ HTTP, Observe para suscripciones, DTLS+PSK para seguridad, block-wise transfer para mensajes >1024 bytes, métodos RESTful (GET/POST/PUT/DELETE)
- **HTTP/REST**: APIs gestión (TB Edge puerto 8080, IEEE 2030.5 puerto 8883, LuCI puerto 80, Ollama puerto 11434), webhooks para integraciones, consultas cloud
- **LwM2M**: Device management (bootstrap, firmware OTA), objetos estándar OMA SpecWorks (Security 0, Server 1, Device 3, Connectivity 4, Firmware Update 5), operaciones Read/Write/Execute/Observe/Discover, transporte CoAP sobre UDP (binding U) o SMS/NB-IoT (binding S), DTLS eficiente (PSK 16 bytes vs X.509 2 KB)

La selección de protocolo por caso de uso se documenta en tabla comparativa en el documento original. La implementación completa de referencia de un nodo IoT ESP32-C6 con cliente LwM2M AVSystems Anjay se documenta en el **Anexo E**.

5.11 Resiliencia y Almacenamiento Persistente

5.11.1 Arquitectura de Almacenamiento

Estrategia de almacenamiento de alta resiliencia: Flash interna 128 MB (sistema OpenWRT + configuración UCI), SSD M.2 NVMe 256 GB (datos persistentes Docker/PostgreSQL/queue TB Edge), USB 3.0 opcional (backups periódicos). Ventajas SSD NVMe vs microSD/USB: durabilidad >1M ciclos E/W (MTBF >1.5M horas), desempeño >3000 IOPS escritura (latencia <0.1ms vs 5-20ms SD), fiabilidad con ECC interno, power-loss protection (PLP) y SMART monitoring.

5.11.2 ThingsBoard Edge Queue: Resiliencia Offline

TB Edge implementa cola de mensajes persistente garantizando resiliencia ante pérdida de conectividad cloud. Arquitectura: queue storage en PostgreSQL + filesystem (`/mnt/ssd/docker/queue`), capacidad hasta 100k mensajes (500 MB CBOR), política FIFO con priorización de alarmas críticas sobre telemetría histórica.

Modo Online (conectividad cloud activa): TB Edge sincroniza cada 5 minutos batch de 1000 mensajes con TB Cloud vía gRPC (puerto 7070), al confirmar ACK elimina mensajes de la cola.

Modo Offline (sin conectividad cloud): TB Edge detecta pérdida de conexión (timeout gRPC >30s), cambia a modo offline continuando procesamiento local, mensajes se acumulan en queue persistente, dashboards locales permanecen funcionales (`http://<gateway-ip>:8080`), alarmas se ejecutan localmente, queue crece hasta límite configurado (100k msgs o 2 GB).

Recuperación de Conectividad (catch-up sync): TB Edge detecta reconexión (gRPC handshake exitoso), inicia sincronización acelerada con batch size 5000 mensajes, prioriza alarmas/eventos críticos, comprime datos con gzip (40-60 % reducción), sincroniza backlog completo de 100k msgs en 10-15 minutos, retorna a modo normal (batch 1000, intervalo 5 min).

Protección contra Desbordamiento: Script de monitoreo ejecutado vía cron cada hora elimina telemetría histórica >7 días, comprime eventos no críticos con gzip, notifica operador si queue >1.8 GB (90 % del límite).

La configuración completa de queue (archivo `tb-edge.yml` con parámetros de `sync_interval`, `batch_size`, `compression`, `retry_policy`, `persistent_queue`) y scripts de monitoreo se documentan en **Anexo B** y **Anexo C**.

5.11.3 Resiliencia Multinivel

Seis niveles de resiliencia con Recovery Time Objective (RTO): L1 Hardware (SSD NVMe con ECC/PLP/SMART, RTO 0s), L2 Filesystem (ext4 con journaling/fsck automático, RTO <30s), L3 Base de datos (PostgreSQL WAL/autovacuum/replication slots, RTO <60s), L4 Aplicación (TB Edge Queue con persistent queue/retry policy/compression, RTO <300s), L5 Red (mwan3 WAN failover Ethernet primario/LTE backup con tracking activo, RTO <30s), L6 Container (Docker healthchecks/restart policy/Watchtower auto-updates, RTO <120s).

5.12 Gestión Remota del Gateway

5.12.1 Feeds de OpenWRT

OpenWRT utiliza feeds (repositorios de paquetes) para extender funcionalidad: feeds oficiales (base, packages, luci, routing, telephony con >10k paquetes) + feeds custom para aplicaciones propietarias Smart Grid. Gestión con opkg: `opkg update`, `opkg find`, `opkg install`, `opkg upgrade`, `opkg list-installed`.

La configuración de feeds custom incluyendo estructura de directorios, ejemplo de Makefile para paquete personalizado (`tb-edge-connector`) y hosting vía nginx se documenta en el **Anexo F**.

5.12.2 OpenVPN: Acceso Remoto Seguro

OpenVPN proporciona túnel VPN cifrado para gestión remota: acceso SSH seguro desde NOC, LuCI web UI sin exponer puerto 80/443 a internet, debugging remoto (logs, tcpdump, análisis performance), túnel permanente hub-spoke. Arquitectura: NOC Server VPN (10.8.0.0/24) → Gateway 1 (10.8.0.100) / Gateway 2 (10.8.0.101) / ... / Gateway N (10.8.0.199) + Admin PC (10.8.0.50).

Configuración cliente OpenVPN: certificados PKI (ca.crt, gateway-001.crt, gateway-001.key, ta.key), compresión lzo adaptive, keepalive 10/120 (detectar desconexión en 120s), persistencia de túnel, logging, pull routes desde servidor, reconexión automática, usuario sin privilegios (nobody/nogroup).

Configuración servidor VPN: puerto 1194 UDP, certificados (ca/server/dh2048), client-to-client (permitir gateways comunicarse), push routes a clientes (red NOC 10.10.0.0/24), keepalive, logging, client-config-dir (CCD) para IPs fijas por gateway y push de rutas específicas.

Las configuraciones completas UCI (`/etc/config/openvpn`), archivos .conf y CCD se documentan en el **Anexo F**.

5.12.3 OpenWISP: Gestión Centralizada de Gateways

OpenWISP es plataforma open-source para gestión masiva (100-1000 gateways): Controller Django (backend), Config agente en gateway, Monitoring (colección de métricas CPU/RAM/tráfico), Firmware Upgrader (actualizaciones OTA masivas), Network Topology (visualización).

Funcionalidades: templates UCI con variables (`{{apn}}`, `{{halow_channel}}`), push configuración remota vía HTTPS con aplicación automática (`uci commit && reload_config`), actualizaciones OTA programadas (inmediata o ventana de mantenimiento 3 AM) con actualización segura dual-partition (escribir Partition B, reiniciar, si falla rollback automático a Partition A), monitoreo de uptime/CPU/RAM/storage/interfaces/Docker, alertas configurables (email/SMS/webhook) para Gateway Offline, High CPU, Low Disk, LTE Failover.

La instalación completa de OpenWISP Config en gateway, despliegue de OpenWISP Controller en Docker (docker-compose.yml con PostgreSQL/Redis/Dashboard/Celery), gestión de configuraciones con templates JSON, firmware OTA workflow y configuración de alertas se documentan en el **Anexo F**.

5.12.4 Comparación de Herramientas de Gestión

LuCI (local) para gestión individual sin gestión masiva, OpenVPN+SSH para <10 gateways con CLI manual, OpenWISP completo para 100-10,000 gateways con templates/push automático/Firmware OTA scheduler/monitoring/alertas/zero-touch provisioning, todo open-source (\$0).

5.13 Gestión de Uplink Redundante (Ethernet + LTE)

5.13.1 Política de Failover Automático

OpenWRT implementa failover basado en route metrics: Ethernet WAN metric=10 (prioridad alta), LTE metric=20 (backup). Kernel selecciona ruta con menor métrica (Ethernet), si falla (link down) cambia automáticamente a LTE, al recuperar Ethernet restaura ruta principal, tiempo de conmutación <30 segundos incluyendo renegotiación TCP.

Las configuraciones UCI de interfaces `wan_eth` y `wan_lte` con protocolo dhcp/modemmanager y métricas se documentan en el **Anexo F**.

5.13.2 Monitoreo Activo de Conectividad (mwan3)

Paquete mwan3 proporciona tracking proactivo de enlaces WAN: ping periódico a 8.8.8.8 y 1.1.1.1, reliability de 2 pings perdidos para declarar fallo (failover), count 3 / timeout 2 / interval 5, políticas de balanceo (75 % Ethernet / 25 % LTE), reglas específicas por servicio (MQTT puerto 8883 solo por Ethernet). Verificación con `mwan3 status` y `mwan3 interfaces`.

La configuración completa de mwan3 (`/etc/config/mwan3` con interfaces, policies, rules) se documenta en el **Anexo F**.

5.13.3 Optimización de Costos LTE

Estrategias para minimizar consumo celular: (1) Compresión CBOR vs JSON (reducción 40-60 % en tamaño payload); (2) Batching - TB Edge acumula 5 min de telemetría y envía en un solo paquete HTTP/2; (3) Compresión gzip para payloads >1 KB; (4) Políticas de tráfico por WAN - script hotplug `/etc/hotplug.d/iface/99-wan-monitor` detecta si LTE activo y adapta comportamiento (detener Watchtower, aumentar intervalo sync TB Edge de 5 min a 1h); (5) Monitoreo consumo con vnstat (`vnstat -m -i wwan0`), alarma si >10 GB/mes deshabilitando LTE y enviando alerta a TB Edge.

Los scripts hotplug `99-wan-monitor` y `check-lte-quota.sh` se documentan en el **Anexo C**.

5.14 Gestión y Monitoreo del Gateway

5.14.1 Interfaz de Gestión (LuCI)

LuCI proporciona interfaz web en `http://<gateway-ip>:80` con módulos: Network (configuración interfaces WAN/LAN, WiFi, firewall, DHCP), System (estado CPU/RAM/storage, logs, backups), Docker (gestión contenedores vía `luci-app-dockerman`: start/stop, logs, stats), Services (configuración servicios dnsmasq, dropbear SSH, uhttpd).

5.14.2 Monitoreo de Contenedores

Docker stats para visualización en tiempo real de CPU %/MEM USAGE/MEM %/NET I/O por contenedor con `docker stats --no-stream`. Healthchecks en `docker-compose.yml`: `test (curl -f http://localhost:8080/api/health) interval 30s, timeout 10s, retries 3, start_period 120s`. Verificación con `docker ps --filter "health=unhealthy"`.

5.14.3 Logs Centralizados

Consulta logs por contenedor con `docker logs -f --tail=100 tb-edge` o `docker logs --since 1h otbr | grep ERROR`. Syslog integration: configurar log-driver syslog en `/etc/docker/daemon.json` para enviar a servidor remoto UDP 514 con tag `gateway-Name`.

5.14.4 Backups y Recuperación

Backup OpenWRT vía LuCI (System > Backup/Flash Firmware > Generate archive) o CLI `sysupgrade -b /tmp/backup-$(date +%Y%m%d).tar.gz`. Backup volúmenes Docker con script diario ejecutado vía cron (0 2 * * *): `tar czf de tb-edge-data, postgres-data, otbr-config`, retención 7 días (`find -mtime +7 -delete`).

Disaster recovery: restaurar OpenWRT (flash imagen + restaurar backup configuración), montar volumen de datos (`mount /dev/sda1 /mnt/docker`), restaurar volúmenes desde backup si necesario, desplegar contenedores (`docker-compose up -d`), verificar healthchecks (`docker ps`), sincronizar TB Edge con cloud (automático al conectar).

Los scripts de backup automatizado `backup.sh` se documentan en el **Anexo C**.

5.15 Pruebas y Validación

5.15.1 Pruebas Funcionales

Validaciones clave: (1) Formación red Thread - verificar OTBR leader/router con `docker exec otbr ot-ctl state` y `ot-ctl child table`; (2) Conexión HaLow - asociación DCUs con `iw dev wlan2 station dump`, señal >-70 dBm, throughput >20 Mbps con `iperf3`; (3) Validación 4 modos HaLow - AP con `hostapd_cli all_sta`, STA con `iw link`, Mesh 802.11s con `iw mpath dump` y test multi-hop ping6, EasyMesh con `ubus call map.controller dump_topology` y test roaming/band steering; (4) Failover Ethernet/LTE - `ifdown wan_eth`, verificar `mwan3 status`, reconectar; (5) Publicación MQTT con `mosquitto_pub`, sincronización cloud con `docker logs tb-edge | grep "Cloud synchronization"`, comando `downlink`.

5.15.2 Pruebas de Desempeño

Latencia E2E objetivo <5 s percentil 95 con timestamps en payload + análisis en TB Edge. Throughput HaLow: 10 DCUs @ 2 Mbps = 20 Mbps agregado, pérdida $<0.1\%$ con señal >-65 dBm, rango verificar conectividad 1 km LoS y 500 m NLOS. Throughput MQTT: 10 dispositivos publicando cada 15 seg = 40 msg/min, escalar hasta observar pérdida o latencia >5 s. Consumo energético con PoE meter: idle <5 W, carga media <12 W, carga alta <18 W (límite PoE+ 25W). Resiliencia offline: 24h sin WAN, buffer >28 k mensajes (300 medidores \times 96 lecturas/día), sincronización completa <10 min al reconectar. Tiempo failover WAN: ping continuo a 8.8.8.8, objetivo <30 segundos.

5.15.3 Pruebas de Seguridad

Validaciones: (1) Firewall - escaneo `nmap -sS -p- <gateway-wan-ip>`, esperado solo puertos explícitos (22 SSH, 443 HTTPS); (2) HaLow WPA3-SAE - validar `iw dev wlan2 info | grep PMF` esperado "PMF: required", intentar asociación con estación WPA2-only rechazada; (3) TLS/mTLS - `openssl s_client -connect <tb-cloud>:7070 -CAfile ca.crt`, verificar return code 0; (4) Inyección MQTT - `mosquitto_pub -h localhost -p 1883 -t test -m "unauthorized"`, esperado Connection refused; (5) Container escape - `docker inspect tb-edge | grep '"Privileged": false'` excepto OTBR; (6) LTE APN security - `grep -r .pn.*password/var/log/`, esperado sin resultados; (7) Actualizaciones automáticas - `docker logs watchtower | grep "Updated"`.

5.15.4 Pruebas de Integración

Comisionado Thread vía OTBR web UI, reglas TB Edge con alarmas (crear regla consumo >5 kW, verificar activación), dashboard en tiempo real con latencia <2 s, API REST consultas (`curl -X GET http://localhost:8080/api/te` -H "X-Authorization: Bearer \$TOKEN"), resiliencia offline 24h con generación de 28,800 mensajes, verificar queue size 150-200 MB con compresión, reconectar WAN, monitorear catch-up sync esperando 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 es protocolo estándar Anthropic para comunicación entre aplicaciones y servicios de IA (Claude, GPT, modelos locales). Ollama permite ejecutar modelos open-source localmente (Llama 3.2, Mistral, Phi-3) sin enviar datos a cloud externo. Integración vía Rule Engine TB Edge para análisis en tiempo real.

5.16.2 Model Context Protocol (MCP)

MCP permite conectarse a múltiples proveedores de IA, proporcionar contexto estructurado a modelos (herramientas, datos, prompts) y ejecutar acciones en sistemas externos desde respuestas de LLM. Componentes: MCP Server (expone herramientas y recursos al LLM, ej. consultar TB Edge API), MCP Client (aplicación que consume servicios de IA, ej. dashboard analítico), protocolo JSON-RPC 2.0 sobre stdio/SSE/WebSocket.

5.16.3 Despliegue de Ollama (LLM Local)

Docker Compose para Ollama: imagen `ollama/ollama:latest`, puerto 11434 API REST, volumen `./models:/root/.ollama`, 8 GB RAM mínimo. Descargar modelo Llama 3.2:3b (2 GB) con `docker exec ollama ollama pull llama3.2:3b` o Phi-3:mini (1.3 GB optimizado edge). Prueba de inferencia con `curl http://localhost:11434/api/generate -d '{"model":"llama3.2:3b","prompt":"Analiza consumo..."}'`.

El docker-compose completo de Ollama se documenta en el **Anexo B**.

5.16.4 MCP Server para ThingsBoard Edge

Implementación de MCP Server Python que expone API TB Edge al LLM con herramientas `get_device_telemetry` (obtiene telemetría histórica) y `get_device_alarms` (obtiene alarmas activas), procesa solicitudes MCP JSON-RPC 2.0 con métodos `tools/list` y `tools/call`, loop stdio principal para comunicación (`for line in sys.stdin`).

El código completo del MCP Server `tb_mcp_server.py` y configuración MCP Client `mcp_config.json` se documentan en el **Anexo C**.

5.16.5 Casos de Uso de IA en Gateway

(1) **Análisis de anomalías en consumo:** Prompt `Analiza consumo medidor METER-001 últimas 24h, identifica patrones anómalos (fraude, falla, consumo irregular)`, LLM invoca `get_device_telemetry` vía MCP, analiza serie temporal (100 puntos, intervalo 15 min), detecta pico 500 kWh a las 3 AM (vs promedio

50 kWh), genera respuesta .^Anomalía detectada: consumo 10× superior, posible bypass medidor o falla CT, recomendar inspección física"; (2) **Mantenimiento predictivo**: Prompt .^Evalúa 50 medidores zona Norte, predice fallas próximos 30 días basándote en alarmas históricas", LLM consulta alarmas de 50 dispositivos, identifica 5 con >10 alarmas en 7 días, analiza telemetría con alta varianza, genera ranking prioridad mantenimiento; (3) **Asistente de operación (Chatbot)**: Dashboard TB Edge con chatbot integrado responde consultas en lenguaje natural ("¿Cuántos medidores offline?" → LLM consulta TB Edge API → .^Atualmente 3 offline: METER-042/089/123, última comunicación hace 2h, posible problema Thread").

5.16.6 Ventajas de IA Local vs Cloud

IA Local (Gateway Ollama): latencia <500 ms, privacidad alta (datos no salen del gateway), costo \$0 (hardware local), disponibilidad offline 100 %, modelos open-source (Llama 3B-7B, Phi-3), capacidad análisis media (3B-7B params), consumo +5W CPU/+15W GPU. IA Cloud (GPT-4/Claude): latencia 2-5s, privacidad baja (envío a cloud), costo \$0.01-0.10/consulta, disponibilidad offline 0 % (requiere internet), modelos propietarios (100B+ params), capacidad análisis alta.

Recomendación: IA local 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 Smart Energy:

- **Flexibilidad**: Contenedores Docker permiten actualizar/escalar servicios independientemente
- **Edge Computing**: ThingsBoard Edge procesa datos localmente reduciendo latencia y dependencia cloud
- **Conectividad robusta multimodal**: HaLow (Morse Micro MM6108) 1-3 km hasta 40 Mbps con 4 modos (AP/STA/Mesh/EasyMesh) + LTE Cat-6 redundante con failover <30s
- **Escalabilidad Arquitectónica**: Estrella (2,500 endpoints / 3 km), Mesh 802.11s (7,500 endpoints / 9 km auto-healing), EasyMesh (12,500 endpoints / roaming transparente)
- **Reducción CAPEX/OPEX**: Mesh 66 % ahorro infraestructura WAN, \$3,240/año ahorro planes LTE con backhaul HaLow sin costo recurrente
- **Interoperabilidad**: OpenThread Border Router con soporte Thread 1.3 multi-vendor compatible
- **Resiliencia**: SSD NVMe (>1M ciclos E/W, >3000 IOPS, <0.1ms latencia), queue persistente TB Edge (100k msgs, 2 GB, sincronización catch-up <15 min con batch 5000 + gzip), 6 niveles resiliencia hardware/filesystem/DB/aplicación/red/containers (RTO <5 min), mesh auto-healing (<10s reconvergencia HWMP eliminando single point of failure)
- **Inteligencia Artificial (Roadmap Futuro)**: MCP + Ollama para análisis local (latencia <500 ms, privacidad 100 % datos no salen), requiere optimización térmica RPi 4, alternativa servidor dedicado para análisis batch offline

- **Arquitectura de Datos Distribuida:** Kafka (>100k msg/s, buffer 7 días, replay histórico, multi-consumidor, backpressure), PostgreSQL+TimescaleDB (compresión 10-20×, particionamiento automático, >3000 IOPS en NVMe, agregaciones time_bucket)
- **Protocolos Multiprotocolo:** MQTT (QoS 0/1/2 Pub/Sub), CoAP (UDP 4 bytes overhead Observe), HTTP/REST (APIs gestión), LwM2M (OTA firmware, objetos OMA estándar, DTLS eficiente PSK 16B vs X.509 2KB)
- **Seguridad multicapa:** Firewall nftables (puertos explícitos), container isolation (namespaces), TLS/mTLS cloud (puerto 7070 gRPC), Thread AES-128-CCM, HaLow WPA3-SAE+PMF (Morse Micro), OpenVPN (túnel permanente NOC sin exponer puertos internet)
- **Mantenibilidad:** OpenWRT Feeds (opkg custom packages Smart Grid), OpenVPN (túnel VPN permanente hub-spoke IPs fijas 10.8.0.100-199), OpenWISP (gestión masiva 100-1000 GWs templates UCI push remoto, Firmware OTA scheduler dual-partition rollback, monitoring CPU/RAM/Interfaces/Docker alertas email/SMS), Watchtower (OTA contenedores), backups automatizados cron
- **Escalabilidad:** 10 DCUs × 250 nodos Thread = 2,500 endpoints AP. Mesh/EasyMesh multiplican 3-5× capacidad sin rediseño arquitectónico
- **Costo-efectividad:** Hardware propósito general (router OpenWRT + módulos M.2 estándar) reduce CAPEX vs propietarios, optimización LTE 3.7 GB/mes (vs 20-30 GB sin compresión CBOR 40-60 %), Mesh HaLow elimina 60-70 % backhaul dedicado
- **Conformidad Estándares:** IEEE 2030.5-2023 (Function Sets DCAP/TM/MM/MSG/ED, API REST XML, X.509 ECC P-256, LFDI, RBAC), ISO/IEC 30141:2024 (arquitectura IoT referencia 8 entidades funcionales, 4 vistas funcional/información/despliegue/operacional), cumplimiento regulatorio CREG Colombia para medición inteligente

5.17.1 Limitaciones y Trabajo Futuro

Validación performance (mediciones CPU/RAM bajo carga completa, benchmarks temperatura con ventilador activo objetivo <75°C, test throughput E2E nodo Thread → OTBR → HaLow → TB Edge → PostgreSQL, stress test 1000 msg/s durante 24h validar estabilidad térmica y resiliencia SSD), conectividad HaLow via USB (Morse Micro Q2 2026 USB 2.0 High-Speed simplifica integración elimina complejidad SPI), IA local (Ollama Llama 3.2 1B o Phi-3 mini en RPi 4 8 GB RAM, validar casos uso detección anomalías fraude bypass CT y mantenimiento predictivo ranking dispositivos alarmas, alternativa Ollama servidor x86 para análisis batch offline datos PostgreSQL), rendimiento I/O (RAID-1 NVMe para >500 dispositivos requiere Compute Module 4 dual M.2), alta disponibilidad (par gateways RPi 4 activo-pasivo VRRP/keepalived, en mesh configurar 2 gateways uplink LTE root bridges redundantes RSTP), RPi vs hardware industrial (migración CM4 carrier board DIN-rail -40°C a +85°C dual Ethernet dual M.2 NVMe certificaciones industriales vibración EMI/EMC, alternativa x86 industrial Intel Atom/Celeron N5105 8 GB RAM dual NIC PCIe mayor costo \$200-300 vs \$55 RPi 4), 5G RedCap (Quectel RG500U latencia <50ms vs 100-300ms LTE-M throughput 100 Mbps vs 375 kbps crítico comandos RPC downlink tiempo real), agregación enlaces (MPTCP Ethernet+LTE simultáneos failover <1s sin pérdida TCP), mesh avanzado (802.11r fastroaming <50ms EasyMesh handoff crítico vehículos eléctricos movimiento carga dinámica V2G), HaLow+LoRaWAN híbrido (sensores ultra-low-power <10 mW batería 10 años LoRaWAN 915 MHz con HaLow backhaul gateways LoRa concentradores Semtech SX1302), quantum-safe crypto (algoritmos post-cuánticos Kyber-768 Dilithium-3 en certificados X.509 protección largo plazo NIST PQC Round 4 2025+ crítico infraestructura Smart Grid vida útil >20 años).

Próximo capítulo: Arquitectura completa del sistema integrando nodos Thread (ESP32-C6), DCUs con Thread Border Router, gateway Raspberry Pi 4 + OpenWRT con HaLow multimodal (AP/STA/Mesh/EasyMesh), Quectel BG95 LTE-M y nRF52840 Thread RCP, y plataforma cloud ThingsBoard, con caso de estudio de

despliegue real para 900 medidores residenciales en infraestructura colombiana con topología mesh 802.11s (3 gateways \times 9 km cobertura \times 300 medidores por gateway).

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

Los medidores inteligentes implementan los estándares IEC 62052/62053 (clase 1 o 2 según precisión requerida) con interfaz DLMS/COSEM sobre RS-485 o puerto óptico IEC 62056-21. Registran perfiles de carga, eventos y parámetros instantáneos utilizando códigos OBIS estándar. Opcionalmente incorporan detección de manipulación (tamper) y capacidad de corte/reconexión remota.

6.3.2 Interfaz de Lectura

Cada medidor expone tres tipos de información:

- **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

Los nodos adaptadores actúan como puente entre el medidor (RS-485) y la red Thread (802.15.4), realizando lectura periódica del medidor vía DLMS/COSEM, encapsulación de datos en paquetes IPv6/6LoWPAN, y transmisión al DCU por radio 802.15.4.

Hardware

La implementación de hardware utiliza el microcontrolador ESP32C6 con radio 802.15.4 integrado, transceptor RS-485 (MAX485 o SP485) con aislamiento galvánico, alimentación de 5V desde medidor o batería con supercapacitor, y antena PCB o externa para 2.4 GHz. Los detalles completos de diseño de hardware se documentan en el Anexo E.

Software

El software incluye el stack Thread (OpenThread en ESP-IDF), cliente DLMS simplificado para lectura de códigos OBIS configurables, y modos de bajo consumo energético. La implementación completa del firmware se presenta en el Anexo E.

6.4.2 DCU (Data Concentrator Unit)

Función

El DCU cumple cuatro roles críticos: actúa como Thread Border Router terminando la red Thread y conectándola a IP, agrega datos de hasta 100 nodos Thread, realiza preprocesamiento (validación, filtrado de duplicados, compresión), y transmite datos agregados al Gateway por 802.11ah.

Hardware

El hardware del DCU utiliza ESP32C6 (dual radio: Thread + WiFi), módulo HaLow (Newracom NRC7292 o similar vía SPI/SDIO), alimentación PoE 802.3af (13W) o AC/DC con batería de respaldo, y opcionalmente SD card para buffer extendido. Las especificaciones detalladas se documentan en el Anexo E.

Software

La arquitectura de software incluye OpenThread Border Router (OTBR), stack WiFi nativo de ESP-IDF, driver HaLow integrado en FreeRTOS, y cola de mensajes con persistencia en SPIFFS/SD. Los detalles de implementación y configuración se presentan en el Anexo C.

6.5 Topología de Red Thread

6.5.1 Mesh Networking

Thread implementa una red mallada auto-organizante con tres tipos de nodos: Leader (coordina la red, elegido automáticamente), Routers (enrután tráfico de otros nodos), y End Devices (nodos de bajo consumo como los adaptadores de medidor).

6.5.2 Ventajas de Thread

Las principales ventajas incluyen auto-healing (reconfiguración automática ante fallos), IPv6 nativo con direccionamiento global único, seguridad mediante AES-128 CCM en capa de enlace y DTLS en aplicación, y escalabilidad hasta 250+ nodos por red Thread.

6.5.3 Configuración de Red

La configuración básica incluye canal 2.4 GHz (canales 15-26 evitando interferencia WiFi), PAN ID único para identificar la red Thread, y Network Key de 128 bits compartida vía preconfiguración o commissioning. Los procedimientos detallados de configuración se documentan en el Anexo D.

6.6 Backhaul: 802.11ah (HaLow)

6.6.1 Justificación de HaLow

HaLow (802.11ah) ofrece ventajas significativas sobre WiFi tradicional: alcance hasta 1 km en línea de vista (vs. 100m WiFi 2.4 GHz), mejor penetración en interiores (banda sub-1 GHz), menor consumo mediante modos de ahorro energético (TIM, RAW), y soporte de miles de clientes por AP.

6.6.2 Configuración HaLow

La configuración opera en banda 902-928 MHz (ISM, región dependiente) con ancho de canal 1-8 MHz configurable según regulación, seguridad WPA3-SAE resistente a ataques de diccionario, y QoS WMM para priorizar tráfico de telemetría crítica. Los parámetros completos de configuración se detallan en el Anexo D.

6.6.3 Topología HaLow

El Gateway actúa como Access Point HaLow con hasta 10 DCUs asociados simultáneamente. Alternativamente, se puede implementar Mesh HaLow para mayor cobertura si los módulos lo soportan. Los modos de operación y configuraciones específicas se documentan en el Anexo D.

6.7 Gateway y Uplink a Cloud

Ver Capítulo 3 para detalles completos de implementación del Gateway.

6.7.1 Resumen de Funciones

El Gateway realiza recepción de datos de DCUs por 802.11ah, normalización y agregación, publicación MQTT/TLS a ThingsBoard (puerto 8883), y buffer offline con reconexión automática.

6.8 Capa de Aplicación: ThingsBoard

6.8.1 Funcionalidades

ThingsBoard proporciona ingesta de telemetría mediante suscripción a topics MQTT con persistencia en base de datos, visualización en dashboards en tiempo real con gráficos de consumo y alarmas, reglas y alertas para detección de anomalías (consumo excesivo, caída de tensión), API REST para integración con

sistemas externos (facturación, ERP), y control remoto con comandos de corte/reconexión hacia medidores (downlink).

6.8.2 Modelo de Datos en ThingsBoard

Entidades

El modelo incluye tres tipos de entidades: Device (cada medidor con ID único), Asset (grupo lógico de medidores por transformador o zona geográfica), y Customer (cliente/usuario final que consulta su consumo).

Atributos y Telemetría

Los Atributos almacenan metadatos estáticos (ubicación, tipo de medidor, tarifa), mientras que la Telemetría registra series temporales de consumo, tensión, corriente, etc. Las estructuras de datos y esquemas completos se documentan en el Anexo D.

6.9 Caso de Estudio: Despliegue en Smart Energy

6.9.1 Escenario

El caso de estudio contempla despliegue en zona residencial de 300 viviendas divididas en 3 sectores: Sector 1 con 100 medidores conectados a DCU-1, Sector 2 con 100 medidores a DCU-2, Sector 3 con 100 medidores a DCU-3, y Gateway ubicado en punto central con línea de vista a los 3 DCUs.

6.9.2 Dimensionamiento

Tráfico Esperado

Con lecturas cada 15 minutos, el sistema genera 96 lecturas/día/medidor, totalizando 28,800 lecturas/día para 300 medidores. Con tamaño de mensaje de 200 bytes (JSON), el tráfico diario es aproximadamente 5.5 MB/día (carga muy baja).

Capacidad de Red

La capacidad de red Thread (250 kbps efectivos) soporta 100 nodos por DCU con holgura. HaLow con 1 MHz y MCS0 proporciona 150 kbps, suficiente para 3 DCUs. El uplink WiFi (54 Mbps mínimo 802.11g) no representa cuello de botella.

6.9.3 Resiliencia y Redundancia

El sistema implementa tres niveles de buffer: DCU con buffer local de 48h en SD card, Gateway con buffer local de 24h en flash, y ThingsBoard replicado con PostgreSQL HA (3 nodos). Los detalles de configuración de alta disponibilidad se documentan en el Anexo B.

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

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

La solución propuesta resulta significativamente más económica que alternativas: Celular NB-IoT requiere \$10/mes/dispositivo (\$36,000/año, inviable), PLC (G3-PLC/PRIME) tiene mayor costo de nodos (\$30-40) sin ventajas claras, y LoRaWAN presenta mayor latencia (clase A) y menor throughput aunque alcance similar.

6.11 Métricas de Desempeño

6.11.1 Latencia E2E

La latencia end-to-end Medidor → ThingsBoard es menor a 5 segundos (promedio 3s medido en piloto), con desglose: Lectura DLMS (0.5s) + Thread (0.5s) + HaLow (1s) + MQTT/TLS (1s).

6.11.2 Disponibilidad

El objetivo de disponibilidad es 99.5 % (downtime máximo 43h/año). En piloto se alcanzó 99.7 % (26h downtime en 12 meses, principalmente por cortes de energía).

6.11.3 Pérdida de Datos

Con QoS 1 la pérdida es menor a 0.01 % (1 mensaje perdido cada 10,000). Sin buffer, la pérdida alcanza 2 % en escenarios de desconexión frecuente.

6.12 Escalabilidad

6.12.1 Crecimiento Horizontal

El sistema permite agregar más DCUs sin modificar gateway (hasta 10 DCUs por gateway) y agregar más gateways sin modificar ThingsBoard (clúster horizontal).

6.12.2 Límites Teóricos

Los límites teóricos son: 250 nodos Thread por DCU (límite de protocolo), 10 DCUs HaLow por Gateway (límite de asociación simultánea), e ilimitado por sistema (ThingsBoard clúster + load balancer).

6.13 Trabajos Futuros y Mejoras

6.13.1 Mejoras Propuestas

Se proponen cuatro mejoras principales: Edge Analytics para detección de anomalías en DCU/Gateway reduciendo tráfico cloud, Compresión mediante CBOR o Protocol Buffers para reducir tamaño de mensajes,

Multicast usando downlink multicast en Thread para comandos broadcast (sincronización de hora), e IPv6 E2E extendiendo IPv6 desde medidor hasta cloud eliminando traducción en DCU.

6.13.2 Integración con Blockchain

Se contempla el uso de ledger distribuido para auditoría inmutable de lecturas y smart contracts para liquidación automática de facturación peer-to-peer. Los detalles de arquitectura blockchain y casos de uso se presentan en el Anexo G (trabajo futuro).

6.14 Conclusiones del Capítulo

La arquitectura propuesta es:

- **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

//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 /
# nvme0n1    259:0    0 238.5G 0 disk
# nvme0n1p1 259:1    0 238.5G 0 part

# Si el SSD no está particionado, crear tabla GPT
fdisk /dev/nvme0n1
# 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/nvme0n1p1

# Verificar filesystem creado
blkid /dev/nvme0n1p1
# Esperado: /dev/nvme0n1p1: 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-opts": {
    "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/

# Descargar firmware RCP OpenThread (versión estable)
wget https://github.com/openthread/ot-nrf528xx/releases/download/\
thread-reference-20230706/ot-rcp-ot-nrf52840-dongle.hex

# Poner nRF52840 en modo bootloader DFU:
# 1. Presionar botón RESET en dongle
# 2. LED debe parpadear en rojo (modo DFU activo)

# Flash firmware RCP
nrfjprog --program ot-rcp-ot-nrf52840-dongle.hex \
        --chiperase --verify --reset

# Verificar programación exitosa
# 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
```

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

```
lsusb | grep "Nordic"
# Esperado: Bus 001 Device 003: ID 1915:521f Nordic Semiconductor ASA
#           Open Thread RCP

# Verificar interfaz serial
ls -la /dev/ttyACM*
# Esperado: /dev/ttyACM0 (puede variar si hay otros dispositivos USB serial)

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

# Crear archivo de configuración wpantund
cat > /etc/wpantund.conf <<EOF
Config:NCP:SocketPath "/dev/ttyACM0"
Config:NCP:SocketBaud 115200
Config:TUN:InterfaceName wpan0
Config:IPv6:Prefix fd00::/64
Config:Daemon:PrivDropToUser nobody
Config:Daemon:PIDFile /var/run/wpantund.pid
EOF

# Habilitar y arrancar wpantund
/etc/init.d/wpantund enable
/etc/init.d/wpantund start

# Verificar interfaz wpan0 creada
ip link show wpan0
# Esperado:
# 5: wpan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1280 qdisc ...

# Verificar status de Thread network
wpanctl 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)
wpanctl form "SmartGrid-Thread" -c 15 -T router

# O unirse a red Thread existente con credenciales
wpanctl join "SmartGrid-Thread" -c 15 -T router \
  --panid 0xABCD --xpanid 0x1234567812345678 \
  --key 00112233445566778899aabbccddeeff

# Verificar que el gateway es Border Router activo
wpanctl status
# Esperado:
```

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

```
# "NCP:State" => "associated"
# "Network:Name" => "SmartGrid-Thread"
# "Network:PANID" => "0xABCD"
# "Network:NodeType" => "router"

# Habilitar prefix delegation para IPv6
wpanctl 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 via interfaz SPI del GPIO y requiere habilitación de SPI en Device Tree y carga de driver ath11k modificado.

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

# Después 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
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/ttyUSB0 - 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
```

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

```
# 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-opts": {
    "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: `wpantctl 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=00112233445566778899aabbccddeeff
    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.

D Anexo C: Scripts y Código de Integración

Este anexo presenta el código fuente completo de los componentes de software desarrollados para la integración de protocolos y servicios en el gateway. Incluye la implementación del servidor IEEE 2030.5, el bridge de traducción Thread-ThingsBoard, y los productores/consumidores Kafka.

D.1 Servidor IEEE 2030.5 (SEP 2.0)

D.1.1 Aplicación Flask Principal

Implementación del servidor RESTful IEEE 2030.5 en Python con Flask, proporcionando los Function Sets DCAP, Time y Metering Mirror.

app.py

```
from flask import Flask, Response, request
import requests
import json
import time
import os

app = Flask(__name__)

# Configuración ThingsBoard Edge
TB_EDGE_URL = os.getenv('TB_EDGE_URL', 'http://tb-edge:8080')
TB_EDGE_TOKEN = os.getenv('TB_EDGE_TOKEN', '')

# Namespace IEEE 2030.5
SEP_NS = 'urn:ieee:std:2030.5:ns'

@app.route('/dcap', methods=['GET'])
def device_capability():
    """
    IEEE 2030.5 Device Capability (DCAP)
```

```

    Endpoint de descubrimiento que expone los Function Sets disponibles.
    """
    xml = f'''<?xml version="1.0" encoding="UTF-8"?>
<DeviceCapability xmlns="{SEP_NS}">
  <href>/dcap</href>
  <TimeLink href="/tm"/>
  <MirrorUsagePointListLink href="/mup" all="0"/>
  <MessagingProgramListLink href="/msg" all="0"/>
  <EndDeviceListLink href="/edev" all="0"/>
  <SelfDeviceLink href="/sdev"/>
</DeviceCapability>'''
    return Response(xml, mimetype='application/sep+xml')

@app.route('/tm', methods=['GET'])
def time_sync():
    """
    IEEE 2030.5 Time (TM)
    Sincronización horaria para clientes SEP 2.0.
    Calidad 7 = máxima precisión (< 100ms via NTP).
    """
    current_time = int(time.time())
    xml = f'''<?xml version="1.0" encoding="UTF-8"?>
<Time xmlns="{SEP_NS}">
  <currentTime>{current_time}</currentTime>
  <dstEndTime>0</dstEndTime>
  <dstOffset>0</dstOffset>
  <dstStartTime>0</dstStartTime>
  <localTime>{current_time}</localTime>
  <quality>7</quality>
  <tzOffset>-18000</tzOffset>
</Time>'''
    return Response(xml, mimetype='application/sep+xml')

@app.route('/mup', methods=['GET'])
def mirror_usage_point_list():
    """
    IEEE 2030.5 Mirror Usage Point List
    Lista de dispositivos con datos de medición disponibles.
    """
    # Consultar dispositivos en ThingsBoard Edge
    try:
        resp = requests.get(
            f"{TB_EDGE_URL}/api/tenant/devices?pageSize=100",
            headers={"X-Authorization": f"Bearer {TB_EDGE_TOKEN}"},
            timeout=5
        )
        devices = resp.json().get('data', [])

        device_links = []
        for idx, device in enumerate(devices):
            device_id = device['id']['id']
            device_links.append(
                f'  <MirrorUsagePoint href="/mup/{device_id}"/>'
            )
    
```

```

        xml = f'<?xml version="1.0" encoding="UTF-8"?>
<MirrorUsagePointList xmlns="{SEP_NS}" all="{len(devices)}">
{chr(10).join(device_links)}
</MirrorUsagePointList>'
        return Response(xml, mimetype='application/sep+xml')

except Exception as e:
    app.logger.error(f"Error fetching devices: {e}")
    return Response('Error fetching devices', status=500)

@app.route('/mup/<device_id>', methods=['GET'])
def mirror_usage_point(device_id):
    """
    IEEE 2030.5 Mirror Usage Point (individual device)
    Telemetría de medición reflejada desde ThingsBoard Edge.
    Granularidad: 15 minutos (900 segundos).
    """
    try:
        # Obtener últimas lecturas de telemetría
        resp = requests.get(
            f"{TB_EDGE_URL}/api/plugins/telemetry/DEVICE/{device_id}"
            "/values/timeseries?keys=energy_kwh,power_w,voltage_v",
            headers={"X-Authorization": f"Bearer {TB_EDGE_TOKEN}"},
            timeout=5
        )
        data = resp.json()

        # Extraer valores (último timestamp)
        energy_entry = data.get('energy_kwh', [{}])[0]
        power_entry = data.get('power_w', [{}])[0]
        voltage_entry = data.get('voltage_v', [{}])[0]

        energy_kwh = energy_entry.get('value', 0.0)
        power_w = power_entry.get('value', 0.0)
        voltage_v = voltage_entry.get('value', 0.0)
        timestamp = energy_entry.get('ts', int(time.time() * 1000)) // 1000

        # Convertir kWh a Wh (IEEE 2030.5 usa Wh entero)
        energy_wh = int(energy_kwh * 1000)

        xml = f'<?xml version="1.0" encoding="UTF-8"?>
<MirrorUsagePoint xmlns="{SEP_NS}">
  <mRID>{device_id}</mRID>
  <deviceLFDI>{device_id[:16].upper()}</deviceLFDI>
  <MirrorMeterReading>
    <mRID>mr_{device_id}</mRID>
    <Reading>
      <value>{energy_wh}</value>
      <localID>1</localID>
      <timePeriod>
        <duration>900</duration>
        <start>{timestamp}</start>
      </timePeriod>
    </Reading>
  </MirrorMeterReading>
</MirrorUsagePoint>'

```

```

        </Reading>
        <ReadingType>
            <powerOfTenMultiplier>0</powerOfTenMultiplier>
            <uom>72</uom>
        </ReadingType>
    </MirrorMeterReading>
    <MirrorMeterReading>
        <mRID>mr_p_{device_id}</mRID>
        <Reading>
            <value>{int(power_w)}</value>
            <localID>2</localID>
            <timePeriod>
                <duration>900</duration>
                <start>{timestamp}</start>
            </timePeriod>
        </Reading>
        <ReadingType>
            <powerOfTenMultiplier>0</powerOfTenMultiplier>
            <uom>38</uom>
        </ReadingType>
    </MirrorMeterReading>
</MirrorUsagePoint>'''
    return Response(xml, mimetype='application/sep+xml')

except Exception as e:
    app.logger.error(f"Error fetching telemetry for {device_id}: {e}")
    return Response('Device not found or telemetry unavailable',
                    status=404)

@app.route('/msg', methods=['GET'])
def messaging_program_list():
    """
    IEEE 2030.5 Messaging Program List
    Lista de programas de mensajería para alertas y notificaciones.
    """
    xml = f'''<?xml version="1.0" encoding="UTF-8"?>
<MessagingProgramList xmlns="{SEP_NS}" all="1">
    <MessagingProgram href="/msg/1">
        <mRID>msg-grid-alerts</mRID>
        <description>Grid Alerts and Notifications</description>
    </MessagingProgram>
</MessagingProgramList>'''
    return Response(xml, mimetype='application/sep+xml')

@app.route('/edev', methods=['GET'])
def end_device_list():
    """
    IEEE 2030.5 End Device List
    Lista de dispositivos registrados en el sistema.
    """
    try:
        resp = requests.get(
            f"{TB_EDGE_URL}/api/tenant/devices?pageSize=100",
            headers={"X-Authorization": f"Bearer {TB_EDGE_TOKEN}"},

```

```

        timeout=5
    )
    devices = resp.json().get('data', [])

    device_entries = []
    for device in devices:
        device_id = device['id']['id']
        device_name = device.get('name', 'Unknown')
        device_entries.append(f''' <EndDevice href="/edev/{device_id}">
<lFDI>{device_id[:16].upper()}</lFDI>
<sFDI>{device_id[:8]}</sFDI>
</EndDevice>''')

    xml = f'''<?xml version="1.0" encoding="UTF-8"?>
<EndDeviceList xmlns="{SEP_NS}" all="{len(devices)}">
{chr(10).join(device_entries)}
</EndDeviceList>'''
    return Response(xml, mimetype='application/sep+xml')

except Exception as e:
    app.logger.error(f"Error fetching devices: {e}")
    return Response('Error fetching devices', status=500)

if __name__ == '__main__':
    # Configuración TLS/mTLS
    cert_file = os.getenv('TLS_CERT', '/certs/server.crt')
    key_file = os.getenv('TLS_KEY', '/certs/server.key')

    app.run(
        host='0.0.0.0',
        port=8883,
        ssl_context=(cert_file, key_file),
        debug=False
    )

```

D.1.2 Dockerfile

```

FROM python:3.11-slim

WORKDIR /app

# Dependencias del sistema
RUN apt-get update && apt-get install -y --no-install-recommends \
    ca-certificates \
    && rm -rf /var/lib/apt/lists/*

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

# Código de aplicación
COPY app.py .

```

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

EXPOSE 8883

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

D.1.3 requirements.txt

```
Flask==3.0.0
requests==2.31.0
pyOpenSSL==23.3.0
Werkzeug==3.0.1
```

D.2 Bridge Thread ↔ ThingsBoard Edge

D.2.1 Script Bridge Principal

Traductor de protocolos que convierte mensajes CoAP/MQTT desde dispositivos Thread a formato ThingsBoard.

bridge.py

```
import paho.mqtt.client as mqtt
import json
import time
import logging
import os

# Configuración de logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Configuración MQTT
THREAD_BROKER = os.getenv('THREAD_BROKER', 'localhost')
THREAD_PORT = int(os.getenv('THREAD_PORT', '1883'))
THREAD_TOPIC = os.getenv('THREAD_TOPIC', 'thread/telemetry/#')

TB_BROKER = os.getenv('TB_BROKER', 'localhost')
TB_PORT = int(os.getenv('TB_PORT', '1883'))
```

```

TB_ACCESS_TOKEN = os.getenv('TB_ACCESS_TOKEN', '')

# Cliente MQTT para dispositivos Thread
thread_client = mqtt.Client(client_id='thread_bridge')

# Cliente MQTT para ThingsBoard Edge
tb_client = mqtt.Client(client_id='tb_bridge')

# Contador de mensajes procesados
message_count = 0
last_log_time = time.time()

def on_thread_connect(client, userdata, flags, rc):
    """Callback al conectar con broker Thread"""
    if rc == 0:
        logger.info(f"Connected to Thread MQTT broker at {THREAD_BROKER}")
        client.subscribe(THREAD_TOPIC)
        logger.info(f"Subscribed to {THREAD_TOPIC}")
    else:
        logger.error(f"Failed to connect to Thread broker, code {rc}")

def on_tb_connect(client, userdata, flags, rc):
    """Callback al conectar con ThingsBoard Edge"""
    if rc == 0:
        logger.info(f"Connected to ThingsBoard Edge at {TB_BROKER}")
    else:
        logger.error(f"Failed to connect to TB Edge, code {rc}")

def transform_telemetry(thread_data):
    """
    Transforma datos de Thread a formato ThingsBoard.

    Thread input format:
    {
        "device_id": "esp32c6_001",
        "timestamp": 1730000000,
        "temperature_c": 25.3,
        "humidity_pct": 65.8,
        "energy_kwh": 12.456,
        "power_w": 1250,
        "voltage_v": 230.5
    }

    ThingsBoard output format:
    {
        "ts": 1730000000000, # Milliseconds
        "values": {
            "temperature": 25.3,
            "humidity": 65.8,
            "energy": 12.456,
            "power": 1250,
            "voltage": 230.5
        }
    }
    """

```

```

"""
try:
    # Convertir timestamp a milisegundos
    ts_ms = int(thread_data.get('timestamp', time.time())) * 1000

    # Mapear campos a formato TB
    telemetry = {
        "ts": ts_ms,
        "values": {}
    }

    # Mapeo de campos comunes
    field_mapping = {
        'temperature_c': 'temperature',
        'humidity_pct': 'humidity',
        'energy_kwh': 'energy',
        'power_w': 'power',
        'voltage_v': 'voltage',
        'current_a': 'current',
        'frequency_hz': 'frequency',
        'pf': 'powerFactor'
    }

    for thread_key, tb_key in field_mapping.items():
        if thread_key in thread_data:
            telemetry['values'][tb_key] = thread_data[thread_key]

    return telemetry

except Exception as e:
    logger.error(f"Error transforming telemetry: {e}")
    return None

def on_thread_message(client, userdata, msg):
    """
    Callback al recibir mensaje de dispositivos Thread.
    Transforma y publica a ThingsBoard Edge.
    """
    global message_count, last_log_time

    try:
        # Decodificar payload
        payload_str = msg.payload.decode('utf-8')
        thread_data = json.loads(payload_str)

        logger.debug(f"Received from Thread: {thread_data}")

        # Extraer device_id del mensaje o del topic
        device_id = thread_data.get('device_id')
        if not device_id:
            # Extraer de topic: thread/telemetry/device123 -> device123
            topic_parts = msg.topic.split('/')
            if len(topic_parts) >= 3:
                device_id = topic_parts[2]

```



```

        else:
            logger.warning("No device_id found in message or topic")
            return

    # Transformar datos
    tb_telemetry = transform_telemetry(thread_data)
    if not tb_telemetry:
        return

    # Publicar a ThingsBoard Edge
    tb_topic = f"v1/devices/{device_id}/telemetry"
    tb_payload = json.dumps(tb_telemetry)

    result = tb_client.publish(tb_topic, tb_payload, qos=1)

    if result.rc == mqtt.MQTT_ERR_SUCCESS:
        message_count += 1

    # Log estadísticas cada 100 mensajes
    if message_count % 100 == 0:
        elapsed = time.time() - last_log_time
        rate = 100 / elapsed if elapsed > 0 else 0
        logger.info(f"Processed {message_count} messages "
                    f"({rate:.1f} msg/s)")
        last_log_time = time.time()
    else:
        logger.error(f"Failed to publish to TB: {result.rc}")

except json.JSONDecodeError as e:
    logger.error(f"Invalid JSON from Thread: {e}")
except Exception as e:
    logger.error(f"Error processing Thread message: {e}")

def main():
    """Función principal del bridge"""
    logger.info("Starting Thread-ThingsBoard Bridge...")

    # Configurar callbacks Thread
    thread_client.on_connect = on_thread_connect
    thread_client.on_message = on_thread_message

    # Configurar callbacks ThingsBoard
    tb_client.on_connect = on_tb_connect
    tb_client.username_pw_set(TB_ACCESS_TOKEN)

    # Conectar a ambos brokers
    try:
        logger.info(f"Connecting to Thread broker {THREAD_BROKER}:{THREAD_PORT}")
        thread_client.connect(THREAD_BROKER, THREAD_PORT, keepalive=60)

        logger.info(f"Connecting to TB Edge {TB_BROKER}:{TB_PORT}")
        tb_client.connect(TB_BROKER, TB_PORT, keepalive=60)

    # Iniciar loops en threads separados

```

```

        thread_client.loop_start()
        tb_client.loop_start()

    logger.info("Bridge is running. Press Ctrl+C to stop.")

    # Mantener vivo
    while True:
        time.sleep(1)

except KeyboardInterrupt:
    logger.info("Shutting down bridge...")
except Exception as e:
    logger.error(f"Fatal error: {e}")
finally:
    thread_client.loop_stop()
    tb_client.loop_stop()
    thread_client.disconnect()
    tb_client.disconnect()
    logger.info("Bridge stopped.")

if __name__ == '__main__':
    main()

```

D.2.2 Dockerfile del Bridge

```

FROM python:3.11-slim

WORKDIR /app

# Dependencias Python
COPY requirements_bridge.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Script bridge
COPY bridge.py .

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

CMD ["python", "bridge.py"]

```

D.2.3 requirements_bridge.txt

```
paho-mqtt==1.6.1
```

D.3 Integración con Apache Kafka

D.3.1 Productor Kafka

Versión mejorada del bridge que publica telemetría a Kafka para procesamiento distribuido.

kafka_producer.py

```
from kafka import KafkaProducer
import paho.mqtt.client as mqtt
import json
import time
import logging
import os

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Configuración Kafka
KAFKA_BOOTSTRAP = os.getenv('KAFKA_BOOTSTRAP', 'localhost:9092')
KAFKA_TOPIC = os.getenv('KAFKA_TOPIC', 'telemetry')
KAFKA_COMPRESSION = os.getenv('KAFKA_COMPRESSION', 'lz4')

# Configuración MQTT Thread
THREAD_BROKER = os.getenv('THREAD_BROKER', 'localhost')
THREAD_PORT = int(os.getenv('THREAD_PORT', '1883'))
THREAD_TOPIC = os.getenv('THREAD_TOPIC', 'thread/telemetry/#')

# Inicializar productor Kafka
producer = KafkaProducer(
    bootstrap_servers=KAFKA_BOOTSTRAP.split(','),
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    compression_type=KAFKA_COMPRESSION,
    acks='all', # Confirmación de todas las réplicas
    retries=3,
    max_in_flight_requests_per_connection=5,
    linger_ms=100, # Batching: esperar 100ms para agrupar mensajes
    batch_size=16384 # 16 KB batch size
)

# Cliente MQTT
mqtt_client = mqtt.Client(client_id='kafka_producer')

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        logger.info(f"Connected to Thread MQTT at {THREAD_BROKER}")
        client.subscribe(THREAD_TOPIC)
    else:
        logger.error(f"MQTT connection failed: {rc}")
```

```

def on_message(client, userdata, msg):
    """Recibir de Thread, publicar a Kafka"""
    try:
        payload = json.loads(msg.payload.decode('utf-8'))

        # Enriquecer con metadata
        kafka_message = {
            'device_id': payload.get('device_id', 'unknown'),
            'timestamp': int(time.time() * 1000), # ms
            'source_topic': msg.topic,
            'data': payload
        }

        # Publicar a Kafka
        future = producer.send(KAFKA_TOPIC, kafka_message)

        # Callback opcional para confirmar
        future.add_callback(lambda metadata:
            logger.debug(f"Sent to {metadata.topic}:{metadata.partition} "
                f"offset {metadata.offset}"))
        future.add_errback(lambda e:
            logger.error(f"Kafka send failed: {e}"))

    except Exception as e:
        logger.error(f"Error processing message: {e}")

def main():
    logger.info(f"Kafka Producer starting...")
    logger.info(f"Kafka: {KAFKA_BOOTSTRAP} | Topic: {KAFKA_TOPIC}")
    logger.info(f"MQTT: {THREAD_BROKER}:{THREAD_PORT} | Topic: {THREAD_TOPIC}")

    mqtt_client.on_connect = on_connect
    mqtt_client.on_message = on_message

    try:
        mqtt_client.connect(THREAD_BROKER, THREAD_PORT, keepalive=60)
        mqtt_client.loop_start()

        logger.info("Producer running. Press Ctrl+C to stop.")
        while True:
            time.sleep(1)

    except KeyboardInterrupt:
        logger.info("Shutting down...")
    finally:
        producer.flush()
        producer.close()
        mqtt_client.loop_stop()
        mqtt_client.disconnect()

if __name__ == '__main__':
    main()

```

D.3.2 Consumidor Kafka

Consumidor que lee de Kafka y publica a ThingsBoard Edge.

kafka_consumer.py

```
from kafka import KafkaConsumer
import paho.mqtt.client as mqtt
import json
import logging
import os

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Configuración Kafka
KAFKA_BOOTSTRAP = os.getenv('KAFKA_BOOTSTRAP', 'localhost:9092')
KAFKA_TOPIC = os.getenv('KAFKA_TOPIC', 'telemetry')
KAFKA_GROUP_ID = os.getenv('KAFKA_GROUP_ID', 'tb-edge-consumer')

# Configuración ThingsBoard
TB_BROKER = os.getenv('TB_BROKER', 'localhost')
TB_PORT = int(os.getenv('TB_PORT', '1883'))
TB_ACCESS_TOKEN = os.getenv('TB_ACCESS_TOKEN', '')

# Consumer Kafka
consumer = KafkaConsumer(
    KAFKA_TOPIC,
    bootstrap_servers=KAFKA_BOOTSTRAP.split(','),
    group_id=KAFKA_GROUP_ID,
    value_deserializer=lambda m: json.loads(m.decode('utf-8')),
    auto_offset_reset='earliest', # Procesar desde el inicio si es nuevo
    enable_auto_commit=True,
    auto_commit_interval_ms=5000
)

# Cliente MQTT ThingsBoard
tb_client = mqtt.Client(client_id='kafka_consumer')
tb_client.username_pw_set(TB_ACCESS_TOKEN)

def on_tb_connect(client, userdata, flags, rc):
    if rc == 0:
        logger.info(f"Connected to ThingsBoard Edge at {TB_BROKER}")
    else:
        logger.error(f"TB connection failed: {rc}")

def main():
    logger.info(f"Kafka Consumer starting...")
    logger.info(f"Kafka: {KAFKA_BOOTSTRAP} | Topic: {KAFKA_TOPIC} | "
                f"Group: {KAFKA_GROUP_ID}")
    logger.info(f"ThingsBoard: {TB_BROKER}:{TB_PORT}")
```

```

tb_client.on_connect = on_tb_connect
tb_client.connect(TB_BROKER, TB_PORT, keepalive=60)
tb_client.loop_start()

try:
    logger.info("Consuming messages from Kafka...")
    for message in consumer:
        try:
            kafka_data = message.value
            device_id = kafka_data.get('device_id', 'unknown')
            payload = kafka_data.get('data', {})

            # Transformar a formato TB
            tb_telemetry = {
                'ts': kafka_data.get('timestamp'),
                'values': payload
            }

            # Publicar a TB Edge
            tb_topic = f"v1/devices/{device_id}/telemetry"
            tb_client.publish(tb_topic, json.dumps(tb_telemetry), qos=1)

            logger.debug(f"Forwarded device {device_id} to TB Edge")

        except Exception as e:
            logger.error(f"Error processing Kafka message: {e}")

except KeyboardInterrupt:
    logger.info("Shutting down...")
finally:
    consumer.close()
    tb_client.loop_stop()
    tb_client.disconnect()

if __name__ == '__main__':
    main()

```

D.3.3 requirements_kafka.txt

```

kafka-python==2.0.2
paho-mqtt==1.6.1

```

D.4 Scripts de Gestión

D.4.1 Comandos de Verificación

verify_services.sh

```
#!/bin/bash
# Script para verificar estado de servicios del gateway

echo "=== Gateway Services Status ==="

# Docker containers
echo -e "\n[Docker Containers]"
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

# OpenThread Border Router
echo -e "\n[OpenThread RCP]"
docker exec -it otbr ot-ctl state 2>/dev/null || echo "OTBR not running"

# ThingsBoard Edge
echo -e "\n[ThingsBoard Edge]"
curl -s http://localhost:8080/api/auth/token -o /dev/null && \
    echo "TB Edge: Running" || echo "TB Edge: Not accessible"

# IEEE 2030.5 Server
echo -e "\n[IEEE 2030.5 Server]"
curl -k -s https://localhost:8883/dcap -o /dev/null && \
    echo "SEP 2.0 Server: Running" || echo "SEP 2.0 Server: Not accessible"

# Kafka
echo -e "\n[Kafka Topics]"
docker exec -it kafka kafka-topics --list \
    --bootstrap-server localhost:9092 2>/dev/null || \
    echo "Kafka not running"

# Network interfaces
echo -e "\n[Network Interfaces]"
ip -br addr show | grep -E 'wlan|wpan|wwan|eth'

echo -e "\n=== End of Status Check ==="
```

D.4.2 Backup de Configuraciones

backup_config.sh

```
#!/bin/bash
# Backup de configuraciones del gateway
```

```
BACKUP_DIR="/mnt/ssd/backups"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/gateway_backup_${TIMESTAMP}.tar.gz"

mkdir -p "$BACKUP_DIR"

echo "Creating gateway configuration backup..."

tar -czf "$BACKUP_FILE" \
  /etc/config \
  /mnt/ssd/docker/*/docker-compose.yml \
  /mnt/ssd/docker/*/*.py \
  /mnt/ssd/docker/*/certs \
  2>/dev/null

if [ $? -eq 0 ]; then
  echo "Backup created: $BACKUP_FILE"
  ls -lh "$BACKUP_FILE"

  # Mantener solo últimos 7 backups
  ls -t "$BACKUP_DIR"/gateway_backup_*.tar.gz | tail -n +8 | xargs rm -f
else
  echo "Backup failed"
  exit 1
fi
```


E Anexo D: Especificaciones IEEE 2030.5 y Configuraciones

Este anexo documenta las especificaciones completas de configuración para los componentes del gateway, incluyendo ejemplos XML IEEE 2030.5, comandos UCI para HaLow, y optimizaciones para TimescaleDB.

E.1 Ejemplos XML IEEE 2030.5

E.1.1 Device Capability (DCAP)

Documento XML completo del endpoint de descubrimiento de capacidades:

```
<?xml version="1.0" encoding="UTF-8"?>
<DeviceCapability xmlns="urn:ieee:std:2030.5:ns">
  <href>/dcap</href>
  <pollRate>900</pollRate>
  <TimeLink href="/tm"/>
  <MirrorUsagePointListLink href="/mup" all="0"/>
  <MessagingProgramListLink href="/msg" all="0"/>
  <EndDeviceListLink href="/edev" all="0"/>
  <DERProgramListLink href="/derp" all="0"/>
  <SelfDeviceLink href="/sdev"/>
</DeviceCapability>
```

E.1.2 Time Synchronization (TM)

Respuesta de sincronización horaria con calidad máxima:

```
<?xml version="1.0" encoding="UTF-8"?>
<Time xmlns="urn:ieee:std:2030.5:ns">
  <currentTime>1730000000</currentTime>
  <dstEndTime>1698627600</dstEndTime>
  <dstOffset>3600</dstOffset>
```

```

    <dstStartTime>1710046800</dstStartTime>
    <localTime>1730000000</localTime>
    <quality>7</quality>
    <tzOffset>-18000</tzOffset>
  </Time>

```

Campos importantes:

- **currentTime:** Tiempo UNIX en segundos (UTC).
- **quality:** 0-7, donde 7 indica sincronización NTP con precisión <100 ms.
- **tzOffset:** Offset en segundos desde UTC (Colombia: -18000 = UTC-5).
- **dstOffset:** Offset adicional durante horario de verano (si aplica).

E.1.3 Mirror Usage Point (MUP)

Ejemplo de telemetría de medición reflejada:

```

<?xml version="1.0" encoding="UTF-8"?>
<MirrorUsagePoint xmlns="urn:ieee:std:2030.5:ns">
  <mRID>0123456789ABCDEF0123456789ABCDEF</mRID>
  <deviceLFDI>0123456789ABCDEF</deviceLFDI>
  <MirrorMeterReading>
    <mRID>mr_energy_001</mRID>
    <description>Active Energy Delivered</description>
    <Reading>
      <consumptionBlock>0</consumptionBlock>
      <qualityFlags>0</qualityFlags>
      <timePeriod>
        <duration>900</duration>
        <start>1730000000</start>
      </timePeriod>
      <touTier>0</touTier>
      <value>123456789</value>
      <localID>1</localID>
    </Reading>
    <ReadingType>
      <accumulationBehaviour>4</accumulationBehaviour>
      <commodity>1</commodity>
      <dataQualifier>0</dataQualifier>
      <flowDirection>1</flowDirection>
      <intervalLength>900</intervalLength>
      <kind>12</kind>
      <phase>0</phase>
      <powerOfTenMultiplier>0</powerOfTenMultiplier>
      <timeAttribute>0</timeAttribute>
      <uom>72</uom>
    </ReadingType>
  </MirrorMeterReading>

```

```

<MirrorMeterReading>
  <mRID>mr_power_001</mRID>
  <description>Instantaneous Active Power</description>
  <Reading>
    <qualityFlags>0</qualityFlags>
    <timePeriod>
      <duration>900</duration>
      <start>1730000000</start>
    </timePeriod>
    <value>1250</value>
    <localID>2</localID>
  </Reading>
  <ReadingType>
    <accumulationBehaviour>0</accumulationBehaviour>
    <commodity>1</commodity>
    <dataQualifier>0</dataQualifier>
    <flowDirection>1</flowDirection>
    <intervalLength>0</intervalLength>
    <kind>12</kind>
    <phase>0</phase>
    <powerOfTenMultiplier>0</powerOfTenMultiplier>
    <timeAttribute>0</timeAttribute>
    <uom>38</uom>
  </ReadingType>
</MirrorMeterReading>
<MirrorMeterReading>
  <mRID>mr_voltage_001</mRID>
  <description>RMS Voltage</description>
  <Reading>
    <qualityFlags>0</qualityFlags>
    <timePeriod>
      <duration>900</duration>
      <start>1730000000</start>
    </timePeriod>
    <value>2305</value>
    <localID>3</localID>
  </Reading>
  <ReadingType>
    <accumulationBehaviour>0</accumulationBehaviour>
    <commodity>1</commodity>
    <dataQualifier>0</dataQualifier>
    <flowDirection>1</flowDirection>
    <intervalLength>0</intervalLength>
    <kind>12</kind>
    <phase>0</phase>
    <powerOfTenMultiplier>-1</powerOfTenMultiplier>
    <timeAttribute>0</timeAttribute>
    <uom>29</uom>
  </ReadingType>
</MirrorMeterReading>
</MirrorUsagePoint>

```

ReadingType - Unidades de Medida (uom):

- 38: Watts (W) - Potencia activa
- 72: Watt-hours (Wh) - Energía activa
- 29: Voltage (V) - Voltaje RMS
- 5: Current (A) - Corriente RMS
- 63: Volt-Ampere Reactive (VAr) - Potencia reactiva

E.1.4 End Device List

Lista de dispositivos registrados con identificadores LFDI/SFDI:

```
<?xml version="1.0" encoding="UTF-8"?>
<EndDeviceList xmlns="urn:ieee:std:2030.5:ns" all="3">
  <EndDevice href="/edev/001">
    <changedTime>1730000000</changedTime>
    <enabled>true</enabled>
    <lfdi>0123456789ABCDEF</lfdi>
    <sfdi>01234567</sfdi>
    <FunctionSetAssignmentsListLink href="/edev/001/fsa" all="4"/>
    <RegistrationLink href="/edev/001/rg"/>
  </EndDevice>
  <EndDevice href="/edev/002">
    <changedTime>1730001000</changedTime>
    <enabled>true</enabled>
    <lfdi>FEDCBA9876543210</lfdi>
    <sfdi>FEDCBA98</sfdi>
    <FunctionSetAssignmentsListLink href="/edev/002/fsa" all="4"/>
    <RegistrationLink href="/edev/002/rg"/>
  </EndDevice>
  <EndDevice href="/edev/003">
    <changedTime>1730002000</changedTime>
    <enabled>true</enabled>
    <lfdi>1234567890ABCDEF</lfdi>
    <sfdi>12345678</sfdi>
    <FunctionSetAssignmentsListLink href="/edev/003/fsa" all="4"/>
    <RegistrationLink href="/edev/003/rg"/>
  </EndDevice>
</EndDeviceList>
```

E.2 Configuraciones UCI para HaLow 802.11ah

E.2.1 Modo Access Point (AP)

Configuración completa del gateway como AP HaLow:

```
# Interfaz inalámbrica HaLow (wlan2)
uci set wireless.halow=wifi-device
uci set wireless.halow.type='mac80211'
uci set wireless.halow.path='platform/soc/1e140000.pcie/pci0000:00/0000:00:00.0/0000:01:00.0'
uci set wireless.halow.channel='7'          # 917 MHz (S1G)
uci set wireless.halow.bandwidth='8'        # 8 MHz (opciones: 1, 2, 4, 8, 16)
uci set wireless.halow.hwmode='11ah'
uci set wireless.halow.country='US'
uci set wireless.halow.txpower='20'         # 20 dBm = 100 mW
uci set wireless.halow.legacy_rates='0'
uci set wireless.halow.mu_beamformer='0'
uci set wireless.halow.mu_beamformee='0'

# Interfaz virtual AP
uci set wireless.halow_ap=wifi-iface
uci set wireless.halow_ap.device='halow'
uci set wireless.halow_ap.mode='ap'
uci set wireless.halow_ap.network='halow_lan'
uci set wireless.halow_ap.ssid='SmartGrid-HaLow-AP'
uci set wireless.halow_ap.encryption='sae'
uci set wireless.halow_ap.key='<WPA3-PSK-SECURE-KEY>'
uci set wireless.halow_ap.ieee80211w='2'    # PMF obligatorio
uci set wireless.halow_ap.sae_pwe='2'      # Hash-to-Element (H2E)
uci set wireless.halow_ap.wpa_disable_eapol_key_retries='1'
uci set wireless.halow_ap.max_inactivity='600' # 10 min timeout
uci set wireless.halow_ap.disassoc_low_ack='0'
uci set wireless.halow_ap.skip_inactivity_poll='0'

# Red virtual para HaLow
uci set network.halow_lan=interface
uci set network.halow_lan.proto='static'
uci set network.halow_lan.ipaddr='192.168.100.1'
uci set network.halow_lan.netmask='255.255.255.0'
uci set network.halow_lan.ip6assign='64'
uci set network.halow_lan.ip6hint='100'

# DHCP server para clientes HaLow
uci set dhcp.halow=dhcp
uci set dhcp.halow.interface='halow_lan'
uci set dhcp.halow.start='100'
uci set dhcp.halow.limit='150'
uci set dhcp.halow.leasetime='12h'
uci set dhcp.halow.dhcpv6='server'
uci set dhcp.halow.ra='server'
uci set dhcp.halow.ra_management='1'

# Firewall zone
uci set firewall.halow_zone=zone
uci set firewall.halow_zone.name='halow'
uci set firewall.halow_zone.input='ACCEPT'
uci set firewall.halow_zone.output='ACCEPT'
uci set firewall.halow_zone.forward='ACCEPT'
uci set firewall.halow_zone.network='halow_lan'
```

```
uci set firewall.halow_lan_forwarding=forwarding
uci set firewall.halow_lan_forwarding.src='halow'
uci set firewall.halow_lan_forwarding.dest='lan'
```

```
uci set firewall.halow_wan_forwarding=forwarding
uci set firewall.halow_wan_forwarding.src='halow'
uci set firewall.halow_wan_forwarding.dest='wan'
```

```
# Aplicar configuración
```

```
uci commit wireless
```

```
uci commit network
```

```
uci commit dhcp
```

```
uci commit firewall
```

```
# Reiniciar servicios
```

```
wifi reload
```

```
/etc/init.d/network restart
```

```
/etc/init.d/firewall restart
```

E.2.2 Modo Station (STA)

Configuración del gateway para conectarse a AP HaLow remoto:

```
# Interfaz HaLow como Station
```

```
uci set wireless.halow=wifi-device
```

```
uci set wireless.halow.type='mac80211'
```

```
uci set wireless.halow.channel='auto'      # Auto-scan
```

```
uci set wireless.halow.bandwidth='8'
```

```
uci set wireless.halow.hwmode='11ah'
```

```
uci set wireless.halow.country='US'
```

```
uci set wireless.halow.disabled='0'
```

```
uci set wireless.halow_sta=wifi-iface
```

```
uci set wireless.halow_sta.device='halow'
```

```
uci set wireless.halow_sta.mode='sta'
```

```
uci set wireless.halow_sta.network='wan_halow'
```

```
uci set wireless.halow_sta.ssid='SmartGrid-HaLow-Backhaul'
```

```
uci set wireless.halow_sta.encryption='sae'
```

```
uci set wireless.halow_sta.key='<WPA3-PSK-BACKHAUL>'
```

```
uci set wireless.halow_sta.ieee80211w='2'
```

```
# Red WAN via HaLow
```

```
uci set network.wan_halow=interface
```

```
uci set network.wan_halow.proto='dhcp'
```

```
uci set network.wan_halow.metric='20'      # Métrica menor = mayor prioridad
```

```
# Agregar a mwan3 para failover
```

```
uci set mwan3.wan_halow=interface
```

```
uci set mwan3.wan_halow.enabled='1'
```

```
uci set mwan3.wan_halow.family='ipv4'
```

```
uci set mwan3.wan_halow.track_ip='8.8.8.8'
```

```
uci set mwan3.wan_halow.track_ip='1.1.1.1'
uci set mwan3.wan_halow.track_method='ping'
uci set mwan3.wan_halow.reliability='1'
uci set mwan3.wan_halow.count='1'
uci set mwan3.wan_halow.size='56'
uci set mwan3.wan_halow.max_ttl='60'
uci set mwan3.wan_halow.timeout='2'
uci set mwan3.wan_halow.interval='5'
uci set mwan3.wan_halow.down='3'
uci set mwan3.wan_halow.up='3'

uci commit wireless
uci commit network
uci commit mwan3

wifi reload
/etc/init.d/network restart
/etc/init.d/mwan3 restart
```

E.2.3 Modo Mesh 802.11s

Configuración para red mesh sin controlador centralizado:

```
# Interfaz HaLow Mesh
uci set wireless.halow=wifi-device
uci set wireless.halow.type='mac80211'
uci set wireless.halow.channel='7'
uci set wireless.halow.bandwidth='8'
uci set wireless.halow.hwmode='11ah'
uci set wireless.halow.country='US'
uci set wireless.halow.txpower='20'

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.mesh_fwding='1'
uci set wireless.halow_mesh.mesh_ttl='31'
uci set wireless.halow_mesh.mesh_rssi_threshold='-80'
uci set wireless.halow_mesh.encryption='sae'
uci set wireless.halow_mesh.key='<MESH-KEY>'
uci set wireless.halow_mesh.network='mesh_lan'

# Red mesh
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='10.100.0.1'
uci set network.bat0.netmask='255.255.0.0'
uci set network.bat0.ip6assign='64'

# Batman-adv
uci set batman-adv.bat0=mesh
uci set batman-adv.bat0.aggregated_ogms='1'
uci set batman-adv.bat0.ap_isolation='0'
uci set batman-adv.bat0.bonding='0'
uci set batman-adv.bat0.fragmentation='1'
uci set batman-adv.bat0.gw_mode='server'
uci set batman-adv.bat0.log_level='0'
uci set batman-adv.bat0.orig_interval='5000'
uci set batman-adv.bat0.bridge_loop_avoidance='1'
uci set batman-adv.bat0.distributed_arp_table='1'
uci set batman-adv.bat0.multicast_mode='1'

uci commit wireless
uci commit network
uci commit batman-adv

# Cargar módulo kernel
modprobe batman-adv

wifi reload
/etc/init.d/network restart
```

E.2.4 Modo EasyMesh (IEEE 1905.1)

Configuración para mesh gestionado con controlador y agentes:

```
# Controlador EasyMesh (Gateway principal)
uci set easymesh.config=easymesh
uci set easymesh.config.enabled='1'
uci set easymesh.config.role='controller'

# Interfaz backhaul HaLow
uci set wireless.halow_backhaul=wifi-iface
uci set wireless.halow_backhaul.device='halow'
uci set wireless.halow_backhaul.mode='ap'
uci set wireless.halow_backhaul.network='backhaul'
uci set wireless.halow_backhaul.ssid='mesh-backhaul-5g'
uci set wireless.halow_backhaul.encryption='sae'
uci set wireless.halow_backhaul.key='<BACKHAUL-KEY>'
uci set wireless.halow_backhaul.multi_ap='2' # Backhaul BSS
uci set wireless.halow_backhaul.ieee80211w='2'
uci set wireless.halow_backhaul.hidden='1'

# Interfaz frontal para clientes
uci set wireless.halow_front=wifi-iface
uci set wireless.halow_front.device='halow'
```



```
uci set wireless.halow_front.mode='ap'
uci set wireless.halow_front.network='lan'
uci set wireless.halow_front.ssid='SmartGrid-HaLow'
uci set wireless.halow_front.encryption='sae'
uci set wireless.halow_front.key='<CLIENT-KEY>'
uci set wireless.halow_front.multi_ap='1' # Fronthaul BSS
uci set wireless.halow_front.ieee80211w='2'

# Red backhaul
uci set network.backhaul=interface
uci set network.backhaul.proto='static'
uci set network.backhaul.ipaddr='192.168.200.1'
uci set network.backhaul.netmask='255.255.255.0'

# Servicios EasyMesh
uci set ieee1905.ieee1905=ieee1905
uci set ieee1905.ieee1905.enabled='1'
uci set ieee1905.ieee1905.al_interface='eth0'
uci set ieee1905.ieee1905.management_interface='br-lan'

uci commit easymesh
uci commit wireless
uci commit network
uci commit ieee1905

/etc/init.d/easymesh enable
/etc/init.d/easymesh start
wifi reload
```

E.3 Optimización TimescaleDB

E.3.1 Configuración PostgreSQL + TimescaleDB

Optimizaciones para almacenamiento de series temporales de alta frecuencia:

```
# postgresql.conf (dentro del contenedor)
# Ubicación: /var/lib/postgresql/data/postgresql.conf

# --- Memoria ---
shared_buffers = 2GB          # 25% de RAM (para RPi4 8GB)
effective_cache_size = 6GB    # 75% de RAM
work_mem = 16MB               # Por operación de sort/hash
maintenance_work_mem = 512MB  # Para VACUUM, CREATE INDEX

# --- Escritura ---
wal_buffers = 16MB
checkpoint_completion_target = 0.9
max_wal_size = 4GB
min_wal_size = 1GB
```

```
wal_compression = on

# --- Checkpoints (reducir I/O en SSD) ---
checkpoint_timeout = 30min
checkpoint_warning = 5min

# --- Queries ---
random_page_cost = 1.1           # SSD, no HDD
effective_io_concurrency = 200    # Para NVMe
max_worker_processes = 4         # CPUs disponibles
max_parallel_workers_per_gather = 2
max_parallel_workers = 4

# --- Logging ---
logging_collector = on
log_destination = 'csvlog'
log_directory = 'log'
log_filename = 'postgresql-%Y-%m-%d.log'
log_rotation_age = 1d
log_rotation_size = 100MB
log_min_duration_statement = 1000 # Log queries > 1s

# --- TimescaleDB ---
shared_preload_libraries = 'timescaledb'
timescaledb.max_background_workers = 4
```

E.3.2 Schema y Hypertables

Creación de tablas optimizadas para telemetría:

```
-- Crear extensión TimescaleDB
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Tabla principal de telemetría
CREATE TABLE telemetry (
    time          TIMESTAMPTZ NOT NULL,
    device_id     TEXT NOT NULL,
    metric        TEXT NOT NULL,
    value         DOUBLE PRECISION,
    unit          TEXT,
    quality       SMALLINT DEFAULT 0
);

-- Convertir a hypertable (particionado automático por tiempo)
SELECT create_hypertable('telemetry', 'time',
    chunk_time_interval => INTERVAL '1 day');

-- Índices para queries frecuentes
CREATE INDEX idx_telemetry_device_time ON telemetry (device_id, time DESC);
CREATE INDEX idx_telemetry_metric_time ON telemetry (metric, time DESC);
```

```
-- Compresión automática (chunks > 7 días)
ALTER TABLE telemetry SET (
    timescaledb.compress,
    timescaledb.compress_segmentby = 'device_id,metric',
    timescaledb.compress_orderby = 'time DESC'
);

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

-- Retención automática (eliminar datos > 1 año)
SELECT add_retention_policy('telemetry', INTERVAL '365 days');

-- Continuous Aggregates (vistas materializadas)
CREATE MATERIALIZED VIEW telemetry_15min
WITH (timescaledb.continuous) AS
SELECT time_bucket('15 minutes', time) AS bucket,
    device_id,
    metric,
    AVG(value) AS avg_value,
    MAX(value) AS max_value,
    MIN(value) AS min_value,
    COUNT(*) AS sample_count
FROM telemetry
GROUP BY bucket, device_id, metric
WITH NO DATA;

-- Refrescar cada 5 minutos
SELECT add_continuous_aggregate_policy('telemetry_15min',
    start_offset => INTERVAL '1 hour',
    end_offset => INTERVAL '5 minutes',
    schedule_interval => INTERVAL '5 minutes');

-- Vista agregada horaria
CREATE MATERIALIZED VIEW telemetry_hourly
WITH (timescaledb.continuous) AS
SELECT time_bucket('1 hour', time) AS bucket,
    device_id,
    metric,
    AVG(value) AS avg_value,
    MAX(value) AS max_value,
    MIN(value) AS min_value,
    STDDEV(value) AS stddev_value,
    COUNT(*) AS sample_count
FROM telemetry
GROUP BY bucket, device_id, metric
WITH NO DATA;

SELECT add_continuous_aggregate_policy('telemetry_hourly',
    start_offset => INTERVAL '1 day',
    end_offset => INTERVAL '1 hour',
    schedule_interval => INTERVAL '1 hour');
```

E.3.3 Queries de Ejemplo

```
-- Telemetría reciente de un dispositivo (últimos 15 min)
SELECT time, metric, value, unit
FROM telemetry
WHERE device_id = 'meter_001'
      AND time > NOW() - INTERVAL '15 minutes'
ORDER BY time DESC;

-- Consumo energético diario agregado
SELECT time_bucket('1 day', time) AS day,
       device_id,
       MAX(value) - MIN(value) AS daily_energy_kwh
FROM telemetry
WHERE metric = 'energy_kwh'
      AND time > NOW() - INTERVAL '30 days'
GROUP BY day, device_id
ORDER BY day DESC;

-- Potencia promedio por hora (usando continuous aggregate)
SELECT bucket AS hour,
       device_id,
       avg_value AS avg_power_w,
       max_value AS peak_power_w
FROM telemetry_hourly
WHERE metric = 'power_w'
      AND bucket > NOW() - INTERVAL '7 days'
ORDER BY bucket DESC, device_id;

-- Alertas: voltaje fuera de rango (207-242V, RETIE Colombia)
SELECT time, device_id, value AS voltage_v
FROM telemetry
WHERE metric = 'voltage_v'
      AND time > NOW() - INTERVAL '1 hour'
      AND (value < 207.0 OR value > 242.0)
ORDER BY time DESC;

-- Dispositivos con mayor consumo (últimas 24h)
SELECT device_id,
       MAX(value) - MIN(value) AS energy_consumed_kwh
FROM telemetry
WHERE metric = 'energy_kwh'
      AND time > NOW() - INTERVAL '24 hours'
GROUP BY device_id
ORDER BY energy_consumed_kwh DESC
LIMIT 10;
```

E.3.4 Mantenimiento

```
-- Ver tamaño de hypertables y chunks
SELECT hypertable_name,
```

```

pg_size_pretty(hypertable_size(format('%I.%I', hypertable_schema, hypertable_name))) AS size
FROM timescaledb_information.hypertables
ORDER BY hypertable_size(format('%I.%I', hypertable_schema, hypertable_name)) DESC;

-- Ver chunks comprimidos
SELECT chunk_schema, chunk_name,
       pg_size_pretty(before_compression_total_bytes) AS before,
       pg_size_pretty(after_compression_total_bytes) AS after,
       round((1 - after_compression_total_bytes::numeric / before_compression_total_bytes::numeric) * 100) AS compression_ratio
FROM timescaledb_information.compressed_chunk_stats
ORDER BY before_compression_total_bytes DESC;

-- Forzar compresión manual de chunks antiguos
SELECT compress_chunk(i)
FROM show_chunks('telemetry', older_than => INTERVAL '7 days') i;

-- Actualizar estadísticas para optimizador de queries
ANALYZE telemetry;
ANALYZE telemetry_15min;
ANALYZE telemetry_hourly;

-- Vacuuming manual (liberar espacio)
VACUUM ANALYZE telemetry;

```

E.4 Generación de Certificados X.509 para mTLS

E.4.1 Autoridad Certificadora (CA)

```

#!/bin/bash
# Crear CA para IEEE 2030.5 mTLS

# CA privada
openssl ecparam -name prime256v1 -genkey -noout -out ca.key
chmod 600 ca.key

# Certificado CA (válido 10 años)
openssl req -new -x509 -sha256 -key ca.key -out ca.crt -days 3650 \
    -subj "/C=CO/ST=Antioquia/L=Medellin/O=SmartGrid CA/CN=SmartGrid Root CA"

# Verificar CA
openssl x509 -in ca.crt -text -noout

```

E.4.2 Certificado Servidor IEEE 2030.5

```

# Key privada servidor
openssl ecparam -name prime256v1 -genkey -noout -out server.key

# CSR (Certificate Signing Request)

```

```

openssl req -new -sha256 -key server.key -out server.csr \
  -subj "/C=CO/ST=Antioquia/L=Medellin/O=SmartGrid/CN=gateway.local"

# Extensiones SAN (Subject Alternative Name)
cat > server_ext.cnf <<EOF
subjectAltName = DNS:gateway.local,DNS:*.gateway.local,IP:192.168.1.1
extendedKeyUsage = serverAuth
EOF

# Firmar con CA (válido 2 años)
openssl x509 -req -sha256 -in server.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out server.crt -days 730 -extfile server_ext.cnf

# Verificar cadena
openssl verify -CAfile ca.crt server.crt

```

E.4.3 Certificado Cliente SEP 2.0

```

# Key privada cliente
openssl ecparam -name prime256v1 -genkey -noout -out client.key

# CSR cliente
openssl req -new -sha256 -key client.key -out client.csr \
  -subj "/C=CO/ST=Antioquia/L=Medellin/O=SmartGrid/CN=meter001"

# Extensiones cliente
cat > client_ext.cnf <<EOF
extendedKeyUsage = clientAuth
EOF

# Firmar con CA
openssl x509 -req -sha256 -in client.csr -CA ca.crt -CAkey ca.key \
  -CAcreateserial -out client.crt -days 730 -extfile client_ext.cnf

# LFDI (Long Form Device Identifier) = SHA256 del certificado
openssl x509 -in client.crt -outform DER | openssl dgst -sha256 -binary | xxd -p -c 32

```

E.4.4 Prueba mTLS

```

# Curl con autenticación mutua
curl -v --cacert ca.crt --cert client.crt --key client.key \
  https://gateway.local:8883/dcap

# OpenSSL s_client test
openssl s_client -connect gateway.local:8883 \
  -CAfile ca.crt -cert client.crt -key client.key \
  -showcerts

```

F Anexo E: Implementación Nodo IoT de Referencia

Este anexo documenta la implementación de referencia de un nodo IoT sensor basado en ESP32-C6, utilizando el protocolo LwM2M (Lightweight M2M) sobre Thread, con integración a ThingsBoard Edge vía el gateway. El código fuente completo está disponible en el repositorio [jsebgiraldo/Tesis-app](#) en la ruta `projects/lwm2m/esp-idf/thingsboard_lwm2m_temperature_humidity`.

F.1 Arquitectura del Nodo

F.1.1 Hardware

- **MCU:** ESP32-C6 (RISC-V, 160 MHz, 512 KB SRAM)
- **Radio:** IEEE 802.15.4 (Thread 1.3) integrado
- **Sensores:** DHT22 simulado (temperatura/humedad)
- **Alimentación:** Batería Li-Ion 18650 3.7V + regulador 3.3V
- **Modos de bajo consumo:** Deep sleep (<20 μ A), light sleep (800 μ A)

F.1.2 Stack de Software

- **Framework:** ESP-IDF 5.1+ (FreeRTOS)
- **Pila Thread:** OpenThread (Joiner commissioning)
- **Pila LwM2M:** AVSystems Anjay 3.x (cliente LwM2M 1.1)
- **Objetos IPSO:** Temperature (3303), Humidity (3304)
- **Objetos LwM2M:** Device (3), Connectivity Monitoring (4), Location (6)
- **Transporte:** CoAP sobre UDP/IPv6 (Thread)

F.2 Código Principal

F.2.1 main.c

Punto de entrada de la aplicación con inicialización de subsistemas:

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_sleep.h"
#include "driver/gpio.h"

// Módulos locales
#include "wifi_provisioning.h"
#include "thread_prov.h"
#include "led_status.h"

void lwm2m_client_start(void);

static const char *TAG = "lwm2m_main";

// GPIO para botón de factory reset (ESP32-C6: GPIO9 típico)
#define CONFIG_BOARD_BOOT_BUTTON_GPIO 9
#define CONFIG_FACTORY_RESET_HOLD_MS 5000

static inline bool is_deep_sleep_wake_capable_gpio(gpio_num_t gpio)
{
    // En ESP32-C6, GPIO0-GPIO7 son LP GPIOs (wake from deep sleep)
    return (gpio >= GPIO_NUM_0 && gpio <= GPIO_NUM_7);
}

static void factory_reset_task(void* arg)
{
    const gpio_num_t btn = (gpio_num_t)CONFIG_BOARD_BOOT_BUTTON_GPIO;
    const TickType_t hold_ticks = pdMS_TO_TICKS(CONFIG_FACTORY_RESET_HOLD_MS);

    gpio_config_t io_conf = {
        .pin_bit_mask = (1ULL << btn),
        .mode = GPIO_MODE_INPUT,
        .pull_up_en = GPIO_PULLUP_ENABLE,
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        .intr_type = GPIO_INTR_DISABLE
    };
    gpio_config(&io_conf);

    while (1) {
        if (gpio_get_level(btn) == 0) { // Botón presionado (activo bajo)
            TickType_t press_start = xTaskGetTickCount();
```



```

        while (gpio_get_level(btn) == 0) {
            TickType_t elapsed = xTaskGetTickCount() - press_start;
            if (elapsed >= hold_ticks) {
                ESP_LOGW(TAG, "Factory reset triggered! Erasing NVS...");

                // Parpadeo LED rápido para indicar reset
                led_status_factory_reset();

                // Borrar partición NVS
                nvs_flash_erase();
                nvs_flash_init();

                ESP_LOGW(TAG, "Factory reset complete. Rebooting...");
                vTaskDelay(pdMS_TO_TICKS(1000));
                esp_restart();
            }
            vTaskDelay(pdMS_TO_TICKS(100));
        }
        vTaskDelay(pdMS_TO_TICKS(200));
    }
}

void app_main(void)
{
    ESP_LOGI(TAG, "=== LwM2M Temperature/Humidity Node ===");
    ESP_LOGI(TAG, "ESP-IDF version: %s", esp_get_idf_version());

    // Inicializar NVS (almacenamiento persistente)
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES ||
        ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    // Inicializar LED de estado
    led_status_init();
    led_status_set(LED_STATUS_BOOTING);

    // Iniciar tarea de factory reset en background
    xTaskCreate(factory_reset_task, "factory_rst", 2048, NULL,
                tskIDLE_PRIORITY + 1, NULL);

#ifdef CONFIG_LWM2M_NETWORK_USE_THREAD
    ESP_LOGI(TAG, "Starting Thread Provisioning...");
    thread_provisioning_init();

    ESP_LOGI(TAG, "Waiting for Thread network attachment...");
    thread_provisioning_wait_connected();

    ESP_LOGI(TAG, "Thread connected! Starting LwM2M client...");
    led_status_set(LED_STATUS_CONNECTED);

```

```

    lwm2m_client_start();

#elif CONFIG_LWM2M_NETWORK_USE_WIFI
    ESP_LOGI(TAG, "Starting WiFi Provisioning...");
    wifi_provisioning_init();

    ESP_LOGI(TAG, "Waiting for WiFi connection...");
    wifi_provisioning_wait_connected();

    ESP_LOGI(TAG, "WiFi connected! Starting LwM2M client...");
    led_status_set(LED_STATUS_CONNECTED);
    lwm2m_client_start();

#else
    ESP_LOGE(TAG, "No network backend enabled. "
                "Enable Thread or WiFi in menuconfig.");
    led_status_set(LED_STATUS_ERROR);
#endif
}

```

F.3 Cliente LwM2M

F.3.1 lwm2m_client.c (fragmento principal)

Cliente Anjay con registro de objetos IPSO y manejo de eventos:

```

#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_log.h"
#include "esp_event.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_netif.h"
#include <string.h>
#include <stdlib.h>

// Objetos LwM2M
#include "device_object.h"
#include "firmware_update.h"
#include "temp_object.h"
#include "humidity_object.h"
#include "onoff_object.h"
#include "connectivity_object.h"
#include "location_object.h"

// AVSystems Anjay
#include <anjay/anjay.h>
#include <anjay/security.h>

```

```

#include <anjay/server.h>
#include <avsystem/commons/avs_time.h>
#include <avsystem/commons/avs_log.h>

static const char *TAG = "lwm2m_client";

// Endpoint name (único por dispositivo, basado en MAC)
static char g_endpoint_name[32] = {0};

static void resolve_endpoint_name(void)
{
    if (strlen(g_endpoint_name) > 0) {
        return; // Ya resuelto
    }

#ifdef CONFIG_LWM2M_ENDPOINT_NAME
    strncpy(g_endpoint_name, CONFIG_LWM2M_ENDPOINT_NAME,
            sizeof(g_endpoint_name) - 1);
#else
    // Generar desde MAC address
    uint8_t mac[6];
    esp_efuse_mac_get_default(mac);
    snprintf(g_endpoint_name, sizeof(g_endpoint_name),
            "esp32c6_%02x%02x%02x", mac[3], mac[4], mac[5]);
#endif
}

static int setup_security(anjay_t *anjay)
{
    // Servidor LwM2M (ThingsBoard Edge en gateway Thread)
    const anjay_security_instance_t security = {
        .ssid = 123, // Server Short ID
        .server_uri = CONFIG_LWM2M_SERVER_URI, // coap://[fd00::1]:5683
        .security_mode = ANJAY_SECURITY_NOSEC, // Sin DTLS (red Thread confiable)
        .bootstrap_server = false
    };

    anjay_iid_t security_iid = ANJAY_ID_INVALID;
    int result = anjay_security_object_add_instance(anjay, &security,
                                                    &security_iid);

    if (result) {
        ESP_LOGE(TAG, "Failed to add Security instance: %d", result);
        return result;
    }

    ESP_LOGI(TAG, "Security object configured: URI=%s SSID=%d",
            security.server_uri, security.ssid);
    return 0;
}

static int setup_server(anjay_t *anjay)
{
    const anjay_server_instance_t server = {
        .ssid = 123,

```

```

        .lifetime = 300,                // 5 min
        .default_min_period = 1,        // Notificaciones: mín 1s
        .default_max_period = -1,       // Servidor define máximo
        .disable_timeout = -1,
        .binding = "U"                  // UDP
    };

    anjay_iid_t server_iid = ANJAY_ID_INVALID;
    int result = anjay_server_object_add_instance(anjay, &server,
                                                &server_iid);

    if (result) {
        ESP_LOGE(TAG, "Failed to add Server instance: %d", result);
        return result;
    }

    ESP_LOGI(TAG, "Server object configured: Lifetime=%ds Binding=%s",
              server.lifetime, server.binding);
    return 0;
}

static void lwm2m_client_task(void *arg)
{
    avs_log_set_default_level(AVS_LOG_DEBUG);

    const anjay_dm_object_def_t **dev_obj = NULL;
    const anjay_dm_object_def_t **loc_obj = NULL;

    resolve_endpoint_name();
    ESP_LOGI(TAG, "Lwm2M Endpoint: %s", g_endpoint_name);

    // Configuración Anjay
    anjay_configuration_t cfg = {
        .endpoint_name = g_endpoint_name,
        .in_buffer_size = CONFIG_LWM2M_IN_BUFFER_SIZE,    // 4096
        .out_buffer_size = CONFIG_LWM2M_OUT_BUFFER_SIZE, // 4096
        .msg_cache_size = CONFIG_LWM2M_MSG_CACHE_SIZE,    // 4096
    };

#ifdef ANJAY_WITH_LWM2M11
    // Forzar Lwm2M 1.1 para compatibilidad con ThingsBoard
    static const anjay_lwm2m_version_config_t ver_11 = {
        .minimum_version = ANJAY_LWM2M_VERSION_1_1,
        .maximum_version = ANJAY_LWM2M_VERSION_1_1
    };
    cfg.lwm2m_version_config = &ver_11;
#endif

    anjay_t *anjay = anjay_new(&cfg);
    if (!anjay) {
        ESP_LOGE(TAG, "Could not create Anjay instance");
        vTaskDelete(NULL);
    }

    // Instalar objetos Security/Server

```

```

    if (anjay_security_object_install(anjay) ||
        anjay_server_object_install(anjay)) {
        ESP_LOGE(TAG, "Could not install Security/Server objects");
        goto cleanup;
    }

    if (setup_security(anjay) || setup_server(anjay)) {
        goto cleanup;
    }

    // Registrar objetos IPSO
    if (anjay_register_object(anjay, temp_object_def())) {
        ESP_LOGE(TAG, "Could not register Temperature (3303)");
        goto cleanup;
    }

    if (anjay_register_object(anjay, humidity_object_def())) {
        ESP_LOGE(TAG, "Could not register Humidity (3304)");
        goto cleanup;
    }

    if (anjay_register_object(anjay, connectivity_object_def())) {
        ESP_LOGE(TAG, "Could not register Connectivity (4)");
        goto cleanup;
    }

    // Registrar objeto Device (3)
    dev_obj = device_object_create(g_endpoint_name);
    if (!dev_obj || anjay_register_object(anjay, dev_obj)) {
        ESP_LOGE(TAG, "Could not register Device (3)");
        goto cleanup;
    }

    // Registrar objeto Location (6)
    loc_obj = location_object_create();
    if (!loc_obj || anjay_register_object(anjay, loc_obj)) {
        ESP_LOGE(TAG, "Could not register Location (6)");
        goto cleanup;
    }

    ESP_LOGI(TAG, "Starting Anjay event loop");

    // Notificar objetos al servidor al inicio
    anjay_notify_instances_changed(anjay, 3303); // Temperature
    anjay_notify_instances_changed(anjay, 3304); // Humidity
    anjay_notify_instances_changed(anjay, 4);    // Connectivity

    // Instalar Firmware Update (OTA)
    ESP_LOGI(TAG, "Installing Firmware Update object...");
    int fw_result = fw_update_install(anjay);
    if (fw_result) {
        ESP_LOGW(TAG, "Firmware Update install failed: %d", fw_result);
    } else {
        ESP_LOGI(TAG, "Firmware Update object ready");
    }

```

```

    }

    // Loop principal
    const avs_time_duration_t max_wait =
        avs_time_duration_from_scalar(100, AVS_TIME_MS);

    while (1) {
        anjay_event_loop_run(anjay, max_wait);

        // Actualizar objetos cada 100ms
        device_object_update(anjay, dev_obj);
        temp_object_update(anjay);
        humidity_object_update(anjay);
        onoff_object_update(anjay);
        connectivity_object_update(anjay);
        location_object_update(anjay, loc_obj);

        // Verificar si hay OTA pendiente
        if (fw_update_requested()) {
            ESP_LOGW(TAG, "Firmware update ready, rebooting...");
            vTaskDelay(pdMS_TO_TICKS(1000));
            fw_update_reboot();
        }
    }
}

cleanup:
    if (dev_obj) device_object_release(dev_obj);
    if (loc_obj) location_object_release(loc_obj);
    anjay_delete(anjay);
    vTaskDelete(NULL);
}

void lwm2m_client_start(void)
{
    xTaskCreate(lwm2m_client_task, "lwm2m",
        CONFIG_LWM2M_TASK_STACK_SIZE, // 8192
        NULL, tskIDLE_PRIORITY + 2, NULL);
}

```

F.4 Objetos IPSO

F.4.1 temp_object.c

Implementación del objeto Temperature (3303):

```

#include "temp_object.h"
#include <math.h>
#include <stdbool.h>
#include <freertos/FreeRTOS.h>

```

```

#include <freertos/task.h>
#include <anjay/io.h>
#include <esp_log.h>

#define OID_TEMPERATURE 3303
#define IID_DEFAULT 0

// Resource IDs (según OMA SpecWorks IPSO)
#define RID_SENSOR_VALUE 5700
#define RID_SENSOR_UNITS 5701
#define RID_MIN_MEASURED 5601
#define RID_MAX_MEASURED 5602
#define RID_RESET_MIN_MAX 5605

#define TEMP_SAMPLE_INTERVAL_MS 1000
#define TEMP_DELTA_EPS 0.01f

static const char *TAG = "temp_obj";

// Estado interno
static float g_current_value = 0.0f;
static float g_min_measured = 100.0f;
static float g_max_measured = -100.0f;
static TickType_t g_last_sample_tick = 0;

static float read_temperature_sensor(void)
{
    // Simulación: senoidal 20-30°C con ruido
    TickType_t ticks = xTaskGetTickCount();
    float base = 25.0f;
    float phase = (float)(ticks % 10000) / 250.0f;
    float delta = 5.0f * sinf(phase);
    float noise = ((float)(esp_random() % 100) / 1000.0f) - 0.05f;

    return base + delta + noise;
}

static void ensure_sample(void)
{
    if (g_last_sample_tick == 0) {
        float value = read_temperature_sensor();
        g_current_value = value;
        g_min_measured = value;
        g_max_measured = value;
        g_last_sample_tick = xTaskGetTickCount();

        ESP_LOGD(TAG, "init sample: value=%.3fC min=%.3f max=%.3f",
                 g_current_value, g_min_measured, g_max_measured);
    }
}

static int temp_list_instances(anjay_t *anjay,
                              const anjay_dm_object_def_t *const *def,
                              anjay_dm_list_ctx_t *ctx) {

```

```

    (void) anjay; (void) def;
    anjay_dm_emit(ctx, IID_DEFAULT);
    return 0;
}

static int temp_list_resources(anjay_t *anjay,
                             const anjay_dm_object_def_t *const *def,
                             anjay_iid_t iid,
                             anjay_dm_resource_list_ctx_t *ctx) {
    (void) anjay; (void) def; (void) iid;
    anjay_dm_emit_res(ctx, RID_MIN_MEASURED, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_MAX_MEASURED, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_RESET_MIN_MAX, ANJAY_DM_RES_E,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_SENSOR_VALUE, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    anjay_dm_emit_res(ctx, RID_SENSOR_UNITS, ANJAY_DM_RES_R,
                      ANJAY_DM_RES_PRESENT);
    return 0;
}

static int temp_read(anjay_t *anjay,
                    const anjay_dm_object_def_t *const *def,
                    anjay_iid_t iid,
                    anjay_rid_t rid,
                    anjay_riid_t riid,
                    anjay_output_ctx_t *ctx) {
    (void) anjay; (void) def; (void) iid; (void) riid;
    ensure_sample();

    switch (rid) {
    case RID_SENSOR_VALUE:
        ESP_LOGD(TAG, "read Temperature -> %.3f C", g_current_value);
        return anjay_ret_float(ctx, g_current_value);

    case RID_SENSOR_UNITS:
        return anjay_ret_string(ctx, "Cel"); // Celsius

    case RID_MIN_MEASURED:
        ESP_LOGD(TAG, "read Min -> %.3f C", g_min_measured);
        return anjay_ret_float(ctx, g_min_measured);

    case RID_MAX_MEASURED:
        ESP_LOGD(TAG, "read Max -> %.3f C", g_max_measured);
        return anjay_ret_float(ctx, g_max_measured);

    default:
        return ANJAY_ERR_METHOD_NOT_ALLOWED;
    }
}

static int temp_execute(anjay_t *anjay,

```



```

        const anjay_dm_object_def_t *const *def,
        anjay_iid_t iid,
        anjay_rid_t rid,
        anjay_execute_ctx_t *ctx) {
(void) anjay; (void) def; (void) iid; (void) ctx;

if (rid == RID_RESET_MIN_MAX) {
    ESP_LOGI(TAG, "Resetting min/max values");
    g_min_measured = g_current_value;
    g_max_measured = g_current_value;

    // Notificar cambios al servidor
    anjay_notify_changed(anjay, OID_TEMPERATURE, IID_DEFAULT,
                        RID_MIN_MEASURED);
    anjay_notify_changed(anjay, OID_TEMPERATURE, IID_DEFAULT,
                        RID_MAX_MEASURED);
    return 0;
}

return ANJAY_ERR_METHOD_NOT_ALLOWED;
}

static const anjay_dm_object_def_t OBJ_DEF = {
    .oid = OID_TEMPERATURE,
    .version = "1.1",
    .handlers = {
        .list_instances = temp_list_instances,
        .list_resources = temp_list_resources,
        .resource_read = temp_read,
        .resource_execute = temp_execute
    }
};

static const anjay_dm_object_def_t *const OBJ_DEF_PTR = &OBJ_DEF;

const anjay_dm_object_def_t *const *temp_object_def(void) {
    ensure_sample();
    return &OBJ_DEF_PTR;
}

void temp_object_update(anjay_t *anjay) {
    if (!anjay) {
        return;
    }

    TickType_t now = xTaskGetTickCount();
    if (g_last_sample_tick == 0 ||
        (now - g_last_sample_tick) >= pdMS_TO_TICKS(TEMP_SAMPLE_INTERVAL_MS)) {

        g_last_sample_tick = now;
        bool min_changed = false;
        bool max_changed = false;

        float new_value = read_temperature_sensor();

```

```

// Actualizar min/max
if (new_value < g_min_measured) {
    g_min_measured = new_value;
    min_changed = true;
}
if (new_value > g_max_measured) {
    g_max_measured = new_value;
    max_changed = true;
}

// Solo notificar si cambi6 significativamente
if (fabsf(new_value - g_current_value) > TEMP_DELTA_EPS) {
    ESP_LOGD(TAG, "Temperature changed: %.3f -> %.3f C",
              g_current_value, new_value);
    g_current_value = new_value;
    anjay_notify_changed(anjay, OID_TEMPERATURE, IID_DEFAULT,
                        RID_SENSOR_VALUE);
}

if (min_changed) {
    anjay_notify_changed(anjay, OID_TEMPERATURE, IID_DEFAULT,
                        RID_MIN_MEASURED);
}
if (max_changed) {
    anjay_notify_changed(anjay, OID_TEMPERATURE, IID_DEFAULT,
                        RID_MAX_MEASURED);
}
}
}

```

F.4.2 humidity_object.c

Implementaci6n del objeto Humidity (3304), an6logo a Temperature:

```

#include "humidity_object.h"
#include <math.h>
#include <stdbool.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <anjay/io.h>
#include <esp_log.h>

#define OID_HUMIDITY 3304
#define IID_DEFAULT 0
#define RID_SENSOR_VALUE 5700
#define RID_SENSOR_UNITS 5701
#define RID_MIN_MEASURED 5601
#define RID_MAX_MEASURED 5602
#define RID_RESET_MIN_MAX 5605

```

```

#define HUM_SAMPLE_INTERVAL_MS 1000
#define HUM_DELTA_EPS 0.01f

static const char *TAG = "humid_obj";

static float g_current_value = 0.0f;
static float g_min_measured = 100.0f;
static float g_max_measured = 0.0f;
static TickType_t g_last_sample_tick = 0;

static float read_humidity_sensor(void)
{
    // Simulación: senoidal 45-75%RH
    TickType_t ticks = xTaskGetTickCount();
    float base = 55.0f;
    float phase = (float)(ticks % 12000) / 300.0f;
    float delta = 10.0f * sinf(phase);
    float noise = ((float)(esp_random() % 100) / 1000.0f) - 0.05f;

    float value = base + delta + noise;

    // Clamp 0-100%
    if (value < 0.0f) value = 0.0f;
    if (value > 100.0f) value = 100.0f;

    return value;
}

// [Resto de funciones similar a temp_object.c]
// list_instances, list_resources, resource_read, resource_execute
// con lógica adaptada para humedad

static const anjay_dm_object_def_t OBJ_DEF = {
    .oid = OID_HUMIDITY,
    .version = "1.1",
    .handlers = {
        .list_instances = hum_list_instances,
        .list_resources = hum_list_resources,
        .resource_read = hum_read,
        .resource_execute = hum_execute
    }
};

// Implementaciones análogas...

```

F.5 Objetos LwM2M Core

F.5.1 device_object.c (fragmento)

Objeto Device (3) con métricas del dispositivo:

```

#include "device_object.h"
#include "sdkconfig.h"
#include <anjay/anjay.h>
#include <anjay/io.h>
#include <esp_system.h>
#include <esp_log.h>
#include <esp_heap_caps.h>
#include <esp_idf_version.h>

#define RID_MANUFACTURER 0
#define RID_MODEL_NUMBER 1
#define RID_SERIAL_NUMBER 2
#define RID_FIRMWARE_VERSION 3
#define RID_REBOOT 4
#define RID_BATTERY_LEVEL 9
#define RID_MEMORY_FREE 10
#define RID_ERROR_CODE 11
#define RID_CURRENT_TIME 13

#define DEVICE_MANUFACTURER "Universidad Nacional"
#define DEVICE_MODEL "ESP32-C6 LwM2M Node"
#define DEVICE_TYPE "Temperature/Humidity Sensor"

static const char *TAG = "device_obj";

typedef struct {
    const anjay_dm_object_def_t *def;
    char serial_number[32];
    int32_t battery_level;
    int32_t power_voltage_mv;
    int32_t power_current_ma;
    TickType_t last_update_tick;
    bool do_reboot;
} device_object_t;

static int resource_read(anjay_t *anjay,
                        const anjay_dm_object_def_t *const *obj_ptr,
                        anjay_iid_t iid,
                        anjay_rid_t rid,
                        anjay_riid_t riid,
                        anjay_output_ctx_t *ctx) {
    device_object_t *obj = get_obj(obj_ptr);

    switch (rid) {
    case RID_MANUFACTURER:
        return anjay_ret_string(ctx, DEVICE_MANUFACTURER);

    case RID_MODEL_NUMBER:
        return anjay_ret_string(ctx, DEVICE_MODEL);

    case RID_SERIAL_NUMBER:
        return anjay_ret_string(ctx, obj->serial_number);

    case RID_FIRMWARE_VERSION:

```

```

        return anjay_ret_string(ctx, esp_get_idf_version());

    case RID_BATTERY_LEVEL:
        return anjay_ret_i32(ctx, obj->battery_level);

    case RID_MEMORY_FREE:
        return anjay_ret_i32(ctx, (int32_t)esp_get_free_heap_size());

    case RID_CURRENT_TIME:
        return anjay_ret_i64(ctx, (int64_t)time(NULL));

    default:
        return ANJAY_ERR_NOT_FOUND;
    }
}

static int resource_execute(anjay_t *anjay,
                           const anjay_dm_object_def_t *const *obj_ptr,
                           anjay_iid_t iid,
                           anjay_rid_t rid,
                           anjay_execute_ctx_t *ctx) {
    device_object_t *obj = get_obj(obj_ptr);

    if (rid == RID_REBOOT) {
        ESP_LOGW(TAG, "Reboot requested via LwM2M");
        obj->do_reboot = true;
        return 0;
    }

    return ANJAY_ERR_METHOD_NOT_ALLOWED;
}

void device_object_update(anjay_t *anjay,
                        const anjay_dm_object_def_t *const *def) {
    device_object_t *obj = get_obj(def);

    if (obj->do_reboot) {
        ESP_LOGW(TAG, "Rebooting...");
        esp_restart();
    }

    // Actualizar nivel de batería simulado cada 10s
    TickType_t now = xTaskGetTickCount();
    if ((now - obj->last_update_tick) >= pdMS_TO_TICKS(10000)) {
        obj->last_update_tick = now;

        // Simulación: batería 70-100% con lenta descarga
        obj->battery_level -= 1;
        if (obj->battery_level < 70) obj->battery_level = 100;

        anjay_notify_changed(anjay, 3, 0, RID_BATTERY_LEVEL);
    }
}

```

F.6 Conectividad Thread

F.6.1 thread_prov.c (fragmento)

Provisioning de red Thread con OpenThread Joiner:

```
#include "thread_prov.h"
#include <string.h>
#include <esp_log.h>
#include <esp_openthread.h>
#include <esp_openthread_lock.h>
#include <openthread/thread.h>
#include <openthread/joiner.h>

static const char *TAG = "thread_prov";

static void ot_joiner_callback(otError error, void *context)
{
    if (error == OT_ERROR_NONE) {
        ESP_LOGI(TAG, "Joiner success! Attached to Thread network");

        esp_openthread_lock_acquire(portMAX_DELAY);
        otThreadSetEnabled(esp_openthread_get_instance(), true);
        esp_openthread_lock_release();
    } else {
        ESP_LOGE(TAG, "Joiner failed: %d", error);
    }
}

void thread_provisioning_init(void)
{
    ESP_LOGI(TAG, "Initializing OpenThread...");

    // Configuración Thread por defecto
    esp_openthread_platform_config_t config = {
        .radio_config = ESP_OPENTHREAD_DEFAULT_RADIO_CONFIG(),
        .host_config = ESP_OPENTHREAD_DEFAULT_HOST_CONFIG(),
        .port_config = ESP_OPENTHREAD_DEFAULT_PORT_CONFIG(),
    };

    ESP_ERROR_CHECK(esp_openthread_init(&config));

    otInstance *instance = esp_openthread_get_instance();

    // Iniciar Joiner con PSKd (pre-shared key for device)
    esp_openthread_lock_acquire(portMAX_DELAY);

    const char *pskd = CONFIG_THREAD_JOINER_PSKD; // "JO1NME"
    otError error = otJoinerStart(instance, pskd, NULL, PACKAGE_NAME,
                                   NULL, NULL, NULL,
                                   ot_joiner_callback, NULL);
```

```

    esp_openthread_lock_release();

    if (error != OT_ERROR_NONE) {
        ESP_LOGE(TAG, "Failed to start Joiner: %d", error);
    } else {
        ESP_LOGI(TAG, "Joiner started with PSKd");
    }
}

void thread_provisioning_wait_connected(void)
{
    ESP_LOGI(TAG, "Waiting for Thread attachment...");

    while (1) {
        esp_openthread_lock_acquire(portMAX_DELAY);
        otInstance *instance = esp_openthread_get_instance();
        otDeviceRole role = otThreadGetDeviceRole(instance);
        esp_openthread_lock_release();

        if (role >= OT_DEVICE_ROLE_CHILD) {
            ESP_LOGI(TAG, "Thread attached! Role: %d", role);
            break;
        }

        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

```

F.7 CMakeLists.txt

F.7.1 Configuración de Build

```

idf_component_register(
    SRCS
        "main.c"
        "lwm2m_client.c"
        "device_object.c"
        "temp_object.c"
        "humidity_object.c"
        "onoff_object.c"
        "connectivity_object.c"
        "firmware_update.c"
        "location_object.c"
        "wifi_provisioning.c"
        "thread_prov.c"
        "led_status.c"

    INCLUDE_DIRS
        "."

```

```

"${IDF_PATH}/components/app_update/include"

REQUIRES
    freertos
    esp_netif
    esp_wifi
    nvs_flash
    lwip
    anjay-esp-idf
    wifi_provisioning
    openthread
    driver
    app_update
    led_strip

PRIV_REQUIRES
    app_update
)

# Asegurar headers app_update visibles
target_include_directories(${COMPONENT_LIB} PRIVATE
    "${IDF_PATH}/components/app_update/include")

```

F.8 sdkconfig.defaults

F.8.1 Configuración por Defecto

```

# Lwm2m Server URI (gateway Thread border router)
CONFIG_LWM2M_SERVER_URI="coap://[fd00::1]:5683"
CONFIG_LWM2M_ENDPOINT_NAME="esp32c6_temphumid"

# Buffer sizes
CONFIG_LWM2M_IN_BUFFER_SIZE=4096
CONFIG_LWM2M_OUT_BUFFER_SIZE=4096
CONFIG_LWM2M_MSG_CACHE_SIZE=4096
CONFIG_LWM2M_TASK_STACK_SIZE=8192

# Thread Joiner
CONFIG_LWM2M_NETWORK_USE_THREAD=y
CONFIG_THREAD_JOINER_PSKD="J01NME"

# OpenThread
CONFIG_OPENTHREAD_ENABLED=y
CONFIG_OPENTHREAD_COMMISSIONER=n
CONFIG_OPENTHREAD_JOINER=y
CONFIG_OPENTHREAD_NETWORK_NAME="SmartGrid-Thread"
CONFIG_OPENTHREAD_NETWORK_CHANNEL=15
CONFIG_OPENTHREAD_NETWORK_PANID=0x1234
CONFIG_OPENTHREAD_NETWORK_EXTPANID="1111111122222222"

```



```
# Anjay
CONFIG_ANJAY_WITH_ATTR_STORAGE=y
CONFIG_ANJAY_WITH_LWM2M11=y

# FreeRTOS
CONFIG_FREERTOS_HZ=1000
CONFIG_FREERTOS_UNICORE=n

# ESP32-C6
CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ_160=y
CONFIG_ESP_PHY_RF_CAL_FULL=y

# Power Management
CONFIG_PM_ENABLE=y
CONFIG_PM_DFS_INIT_AUTO=y
CONFIG_PM_POWER_DOWN_CPU_IN_LIGHT_SLEEP=y
CONFIG_PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP=y

# Logging
CONFIG_LOG_DEFAULT_LEVEL_INFO=y
CONFIG_LOG_MAXIMUM_LEVEL_DEBUG=y
```

F.9 Uso del Nodo

F.9.1 Compilación y Flash

```
# Desde directorio del proyecto
cd projects/lwm2m/esp-idf/thingsboard_lwm2m_temperature_humidity

# Configurar (opcional, solo primera vez)
idf.py menuconfig

# Compilar
idf.py build

# Flash al ESP32-C6
idf.py -p COM3 flash monitor # Windows
idf.py -p /dev/ttyUSB0 flash monitor # Linux

# Solo monitor
idf.py -p COM3 monitor
```

F.9.2 Comisionamiento Thread

En el gateway OTBR:

```
# Habilitar comisionado
```

```
docker exec -it otbr ot-ctl commissioner start
docker exec -it otbr ot-ctl commissioner joiner add * J01NME

# Verificar dispositivo unido
docker exec -it otbr ot-ctl child table
# Output esperado: Child ID | RLOC16 | Timeout | ... | IPv6 Address
```

F.9.3 Verificación LwM2M

En ThingsBoard Edge:

1. Navegar a *Devices* → se debe crear automáticamente `esp32c6_XXXXXX`
2. *Latest Telemetry* mostrará: temperature, humidity, battery_level, memory_free
3. *Attributes* mostrará: manufacturer, model, fw_version
4. Configurar *Observe* en recursos 3303/0/5700 y 3304/0/5700 para notificaciones automáticas

G Configuraciones OpenWRT del Gateway

Este anexo documenta las configuraciones completas del sistema operativo OpenWRT en el gateway IoT, incluyendo archivos UCI, reglas de firewall nftables, configuración OpenVPN, despliegue de OpenWISP, y políticas de failover con mwan3.

G.1 Configuraciones UCI Base

G.1.1 Network (/etc/config/network)

Configuración completa de interfaces de red:

```
config interface 'loopback'
    option device 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'fd00::/48'
    option packet_steering '1'

config device
    option name 'br-lan'
    option type 'bridge'
    list ports 'eth0'

config interface 'lan'
    option device 'br-lan'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'
    option ip6hint '1'
```

```
# Interfaz Ethernet WAN
config interface 'wan'
    option device 'eth1'
    option proto 'dhcp'
    option peerdns '0'
    option dns '1.1.1.1 8.8.8.8'
    option metric '10'

config interface 'wan6'
    option device 'eth1'
    option proto 'dhcpv6'
    option reqaddress 'try'
    option reqprefix 'auto'
    option peerdns '0'
    option dns '2606:4700:4700::1111 2001:4860:4860::8888'

# Interfaz LTE (Quectel BG95-M3)
config interface 'lte'
    option device '/dev/ttyUSB2'
    option proto 'qmi'
    option apn 'internet.movistar.co'
    option auth 'none'
    option delay '10'
    option metric '20'
    option peerdns '0'
    option dns '8.8.8.8 8.8.4.4'
    option ipv6 'auto'

# HaLow backhaul station
config interface 'halow_wan'
    option proto 'dhcp'
    option metric '15'
    option peerdns '0'
    option dns '1.1.1.1'

# Thread Border Router
config interface 'thread_br'
    option device 'wpan0'
    option proto 'static'
    option ipaddr '192.168.100.1'
    option netmask '255.255.255.0'
    option ip6assign '64'
    option ip6hint '100'

# VPN OpenVPN
config interface 'vpn0'
    option proto 'none'
    option device 'tun0'
```

G.1.2 Wireless (/etc/config/wireless)

Configuración WiFi 2.4 GHz y HaLow 802.11ah:

```
# WiFi 2.4 GHz (BCM43455 integrado en RPi4)
config wifi-device 'radio0'
    option type 'mac80211'
    option path 'platform/soc/fe300000.mmcnr/mmc_host/mmc1/mmc1:0001/mmc1:0001:1'
    option channel '6'
    option band '2g'
    option htmode 'HT40'
    option country 'C0'
    option txpower '20'
    option legacy_rates '0'
    option cell_density '0'

config wifi-iface 'default_radio0'
    option device 'radio0'
    option mode 'ap'
    option network 'lan'
    option ssid 'SmartGrid-Gateway'
    option encryption 'sae-mixed'
    option key '<WIFI-PASSWORD>'
    option ieee80211w '1'
    option wpa_disable_eapol_key_retries '1'
    option max_inactivity '300'

# HaLow 802.11ah (Morse Micro MM6108-EK03 SPI)
config wifi-device 'halow'
    option type 'mac80211'
    option path 'platform/soc/fe204000.spi/spi_master/spi0/spi0.0'
    option channel '7'
    option bandwidth '8'
    option hwmode '11ah'
    option country 'US'
    option txpower '20'
    option legacy_rates '0'
    option mu_beamformer '0'
    option mu_beamformee '0'
    option slg_long '1'
    option slg_short '0'

# HaLow AP para DCUs
config wifi-iface 'halow_ap'
    option device 'halow'
    option mode 'ap'
    option network 'halow_lan'
    option ssid 'SmartGrid-HaLow-Backhaul'
    option encryption 'sae'
    option key '<HALOW-AP-KEY>'
    option ieee80211w '2'
    option sae_pwe '2'
    option wpa_disable_eapol_key_retries '1'
    option max_inactivity '600'
    option disassoc_low_ack '0'
    option skip_inactivity_poll '0'
    option max_listen_interval '65535'
    option dtim_period '10'
```

```
# Red virtual HaLow LAN
config interface 'halow_lan'
    option proto 'static'
    option ipaddr '192.168.200.1'
    option netmask '255.255.255.0'
    option ip6assign '64'
    option ip6hint '200'
```

G.1.3 DHCP y DNS (/etc/config/dhcp)

```
config dnsmasq
    option domainneeded '1'
    option boguspriv '1'
    option filterwin2k '0'
    option localise_queries '1'
    option rebind_protection '1'
    option rebind_localhost '1'
    option local '/lan/'
    option domain 'lan'
    option expandhosts '1'
    option nonegcache '0'
    option cachesize '1000'
    option authoritative '1'
    option readethers '1'
    option leasefile '/tmp/dhcp.leases'
    option resolvfile '/tmp/resolv.conf.d/resolv.conf.auto'
    option nonwildcard '1'
    option localservice '1'
    option ednspacket_max '1232'

config dhcp 'lan'
    option interface 'lan'
    option start '100'
    option limit '150'
    option leasetime '12h'
    option dhcpv4 'server'
    option dhcpv6 'server'
    option ra 'server'
    option ra_slaac '1'
    list ra_flags 'managed-config'
    list ra_flags 'other-config'

config dhcp 'wan'
    option interface 'wan'
    option ignore '1'

config dhcp 'halow_lan'
    option interface 'halow_lan'
    option start '10'
    option limit '50'
    option leasetime '24h'
```

```
option dhcpv4 'server'
option dhcpv6 'server'
option ra 'server'

config dhcp 'thread_br'
option interface 'thread_br'
option start '50'
option limit '200'
option leasetime '12h'
option dhcpv4 'server'
option dhcpv6 'server'
option ra 'server'

# Entradas estáticas para DCUs
config host
option name 'dcu1'
option dns '1'
option mac 'AA:BB:CC:DD:EE:01'
option ip '192.168.200.10'

config host
option name 'dcu2'
option dns '1'
option mac 'AA:BB:CC:DD:EE:02'
option ip '192.168.200.11'

config host
option name 'dcu3'
option dns '1'
option mac 'AA:BB:CC:DD:EE:03'
option ip '192.168.200.12'
```

G.2 Firewall nftables

G.2.1 Configuración Base (/etc/config/firewall)

```
config defaults
option input 'REJECT'
option output 'ACCEPT'
option forward 'REJECT'
option synflood_protect '1'
option drop_invalid '1'
option tcp_syncookies '1'
option tcp_ecn '0'
option tcp_window_scaling '1'
option accept_redirects '0'
option accept_source_route '0'
option flow_offloading '1'
option flow_offloading_hw '0'
```

```
# Zona LAN
config zone
    option name 'lan'
    option input 'ACCEPT'
    option output 'ACCEPT'
    option forward 'ACCEPT'
    list network 'lan'

# Zona WAN
config zone
    option name 'wan'
    option input 'REJECT'
    option output 'ACCEPT'
    option forward 'REJECT'
    option masq '1'
    option mtu_fix '1'
    list network 'wan'
    list network 'wan6'
    list network 'lte'
    list network 'halow_wan'

# Zona HaLow backhaul
config zone
    option name 'halow'
    option input 'ACCEPT'
    option output 'ACCEPT'
    option forward 'ACCEPT'
    list network 'halow_lan'

# Zona Thread
config zone
    option name 'thread'
    option input 'ACCEPT'
    option output 'ACCEPT'
    option forward 'ACCEPT'
    list network 'thread_br'

# Zona VPN
config zone
    option name 'vpn'
    option input 'ACCEPT'
    option output 'ACCEPT'
    option forward 'ACCEPT'
    option masq '0'
    list network 'vpn0'

# Forwarding LAN -> WAN
config forwarding
    option src 'lan'
    option dest 'wan'

# Forwarding HaLow -> LAN
config forwarding
    option src 'halow'
```



```
    option dest 'lan'

# Forwarding HaLow -> WAN
config forwarding
    option src 'halow'
    option dest 'wan'

# Forwarding Thread -> LAN
config forwarding
    option src 'thread'
    option dest 'lan'

# Forwarding Thread -> WAN
config forwarding
    option src 'thread'
    option dest 'wan'

# Forwarding VPN -> LAN
config forwarding
    option src 'vpn'
    option dest 'lan'

# Forwarding LAN -> VPN
config forwarding
    option src 'lan'
    option dest 'vpn'

# Permitir SSH desde WAN (puerto no estándar)
config rule
    option name 'Allow-SSH-WAN'
    option src 'wan'
    option proto 'tcp'
    option dest_port '2222'
    option target 'ACCEPT'

# Permitir HTTPS Web UI desde WAN
config rule
    option name 'Allow-HTTPS-WAN'
    option src 'wan'
    option proto 'tcp'
    option dest_port '443'
    option target 'ACCEPT'

# Permitir OpenVPN desde WAN
config rule
    option name 'Allow-OpenVPN'
    option src 'wan'
    option proto 'udp'
    option dest_port '1194'
    option target 'ACCEPT'

# Permitir ICMP ping desde WAN (para mwan3 tracking)
config rule
    option name 'Allow-Ping-WAN'
```

```

    option src 'wan'
    option proto 'icmp'
    option icmp_type 'echo-request'
    option family 'ipv4'
    option target 'ACCEPT'

# Rate limit ICMP para prevenir flood
config rule
    option name 'Limit-ICMP'
    option src 'wan'
    option proto 'icmp'
    option family 'ipv4'
    option limit '10/second'
    option limit_burst '20'
    option target 'ACCEPT'

# Bloquear acceso directo a Docker desde WAN
config rule
    option name 'Block-Docker-WAN'
    option src 'wan'
    option dest 'lan'
    option dest_ip '172.17.0.0/16'
    option target 'REJECT'

# Permitir LwM2M CoAP desde Thread
config rule
    option name 'Allow-LwM2M-Thread'
    option src 'thread'
    option proto 'udp'
    option dest_port '5683 5684'
    option target 'ACCEPT'

# Permitir MQTT desde HaLow (DCUs)
config rule
    option name 'Allow-MQTT-HaLow'
    option src 'halow'
    option proto 'tcp'
    option dest_port '1883 8883'
    option target 'ACCEPT'

```

G.2.2 Script nftables Personalizado

Ubicación: /etc/nftables.d/custom_rules.nft

```

#!/usr/sbin/nft -f
# Reglas nftables personalizadas para gateway SmartGrid

table inet smartgrid {
    # Set de IPs permitidas para administración
    set admin_ips {
        type ipv4_addr

```

```

        flags interval
        elements = {
            192.168.1.0/24,
            10.0.0.0/8,
            172.16.0.0/12
        }
    }

# Set de puertos Docker a proteger
set docker_ports {
    type inet_service
    elements = { 8080, 5432, 9092, 2181, 8883 }
}

# Rate limiting para conexiones SSH
chain ssh_ratelimit {
    type filter hook input priority filter; policy accept;

    tcp dport 2222 ct state new \
        limit rate over 3/minute \
        counter drop comment "SSH brute-force protection"
}

# Protección DDoS básica
chain ddos_protection {
    type filter hook input priority filter; policy accept;

    # SYN flood protection
    tcp flags syn tcp flags & (fin|syn|rst|ack) == syn \
        ct state new \
        limit rate over 100/second burst 150 packets \
        counter drop comment "SYN flood protection"

    # Invalid packets
    ct state invalid counter drop

    # Fragmentos pequeños (posible ataque)
    ip frag-off & 0x1fff != 0 \
        limit rate over 10/second \
        counter drop comment "IP fragment attack"
}

# NAT para Docker containers (bypass masquerade)
chain postrouting_docker {
    type nat hook postrouting priority srcnat; policy accept;

    # No hacer SNAT para tráfico Docker interno
    oifname "docker0" counter accept

    # SNAT para containers hacia WAN
    ip saddr 172.17.0.0/16 oifname { "eth1", "wwan0", "wlan2" } \
        counter masquerade comment "Docker to WAN"
}

```

```
# Log de intentos de acceso a servicios críticos
chain log_critical {
    type filter hook input priority filter - 1; policy accept;

    tcp dport @docker_ports ip saddr != @admin_ips \
        limit rate 1/minute \
        log prefix "Blocked Docker access: " level warn
}
}
```

Para activar:

```
# Cargar reglas personalizadas
nft -f /etc/nftables.d/custom_rules.nft

# Hacer persistente (agregar a /etc/rc.local)
echo "nft -f /etc/nftables.d/custom_rules.nft" >> /etc/rc.local
```

G.3 OpenVPN

G.3.1 Configuración Servidor

Archivo: /etc/openvpn/server.conf

```
# Puerto y protocolo
port 1194
proto udp
dev tun

# Certificados y llaves (PKI con Easy-RSA)
ca /etc/openvpn/pki/ca.crt
cert /etc/openvpn/pki/issued/server.crt
key /etc/openvpn/pki/private/server.key
dh /etc/openvpn/pki/dh.pem
tls-auth /etc/openvpn/pki/ta.key 0

# Cifrado
cipher AES-256-GCM
auth SHA256
tls-version-min 1.2
tls-cipher TLS-ECDHE-RSA-WITH-AES-256-GCM-SHA384

# Red VPN
server 10.8.0.0 255.255.255.0
topology subnet
ifconfig-pool-persist /tmp/openvpn-ipp.txt

# Rutas hacia LAN y redes Thread/HaLow
```

```
push "route 192.168.1.0 255.255.255.0"
push "route 192.168.100.0 255.255.255.0"
push "route 192.168.200.0 255.255.255.0"
push "route fd00::/48"
```

```
# DNS interno
push "dhcp-option DNS 192.168.1.1"
push "dhcp-option DOMAIN lan"
```

```
# Seguridad
client-to-client
keepalive 10 120
comp-lzo no
max-clients 10
user nobody
group nogroup
persist-key
persist-tun
```

```
# Logging
status /tmp/openvpn-status.log
log-append /var/log/openvpn.log
verb 3
mute 20
```

G.3.2 Generación de Certificados con Easy-RSA

```
#!/bin/bash
# Script de inicialización PKI para OpenVPN

cd /etc/openvpn

# Descargar Easy-RSA
wget https://github.com/OpenVPN/easy-rsa/releases/download/v3.1.7/EasyRSA-3.1.7.tgz
tar xzf EasyRSA-3.1.7.tgz
mv EasyRSA-3.1.7 easyrsa
cd easyrsa

# Inicializar PKI
./easyrsa init-pki

# Crear CA (ingresar contraseña segura cuando se solicite)
./easyrsa build-ca

# Generar certificado y llave del servidor
./easyrsa gen-req server nopass
./easyrsa sign-req server server

# Generar parámetros Diffie-Hellman (tarda varios minutos)
./easyrsa gen-dh

# Generar llave TLS-Auth para HMAC
```

```
openvpn --genkey secret pki/ta.key

# Crear certificado para cliente (ej. admin)
./easyrsa gen-req client1 nopass
./easyrsa sign-req client client1

# Copiar archivos al directorio OpenVPN
cp pki/ca.crt pki/issued/server.crt pki/private/server.key \
  pki/dh.pem pki/ta.key /etc/openvpn/

echo "PKI creada exitosamente en /etc/openvpn/easyrsa/pki"
```

G.3.3 Configuración Cliente (.ovpn)

Archivo: client1.ovpn (distribuir a administradores)

```
client
dev tun
proto udp
remote <GATEWAY-PUBLIC-IP> 1194

resolv-retry infinite
nobind
persist-key
persist-tun

# Cifrado (debe coincidir con servidor)
cipher AES-256-GCM
auth SHA256
tls-version-min 1.2

# Compresión
comp-lzo no

verb 3

<ca>
-----BEGIN CERTIFICATE-----
[Contenido de ca.crt]
-----END CERTIFICATE-----
</ca>

<cert>
-----BEGIN CERTIFICATE-----
[Contenido de client1.crt]
-----END CERTIFICATE-----
</cert>

<key>
-----BEGIN PRIVATE KEY-----
[Contenido de client1.key]
```

```

-----END PRIVATE KEY-----
</key>

<tls-auth>
-----BEGIN OpenVPN Static key V1-----
[Contenido de ta.key]
-----END OpenVPN Static key V1-----
</tls-auth>

key-direction 1

```

G.4 OpenWISP

G.4.1 Docker Compose OpenWISP Controller

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

```

version: '3.8'

services:
  postgres:
    image: postgis/postgis:15-3.3-alpine
    container_name: openwisp-postgres
    environment:
      POSTGRES_DB: openwisP_db
      POSTGRES_USER: openwisP
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - /mnt/ssd/openwisP/postgres:/var/lib/postgresql/data
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U openwisP"]
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7-alpine
    container_name: openwisP-redis
    command: redis-server --appendonly yes
    volumes:
      - /mnt/ssd/openwisP/redis:/data
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s
      timeout: 3s
      retries: 3

```

```

openwisp:
  image: openwisp/openwisp-dashboard:latest
  container_name: openwisp-dashboard
  depends_on:
    postgres:
      condition: service_healthy
    redis:
      condition: service_healthy
  environment:
    DB_ENGINE: django.contrib.gis.db.backends.postgis
    DB_NAME: openwisp_db
    DB_USER: openwisp
    DB_PASSWORD: ${POSTGRES_PASSWORD}
    DB_HOST: postgres
    DB_PORT: 5432

    REDIS_HOST: redis
    REDIS_PORT: 6379

    DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
    DJANGO_ALLOWED_HOSTS: "*"
    DJANGO_CORS_ORIGIN_WHITELIST: "http://localhost,https://gateway.local"

    EMAIL_BACKEND: django.core.mail.backends.smtp.EmailBackend
    EMAIL_HOST: smtp.gmail.com
    EMAIL_PORT: 587
    EMAIL_USE_TLS: 1
    EMAIL_HOST_USER: ${EMAIL_USER}
    EMAIL_HOST_PASSWORD: ${EMAIL_PASSWORD}

    OPENWIS_ORGANIZATI_UUID: ${ORG_UUID}
    OPENWIS_SHARED_SECRET: ${SHARED_SECRET}
  ports:
    - "8000:8000"
  volumes:
    - /mnt/ssd/openwisp/media:/opt/openwisp/media
    - /mnt/ssd/openwisp/static:/opt/openwisp/static
  restart: unless-stopped
  logging:
    driver: "json-file"
    options:
      max-size: "10m"
      max-file: "3"

celery:
  image: openwisp/openwisp-dashboard:latest
  container_name: openwisp-celery
  depends_on:
    - openwisp
    - redis
  environment:
    DB_ENGINE: django.contrib.gis.db.backends.postgis
    DB_NAME: openwisp_db
    DB_USER: openwisp

```



```

    DB_PASSWORD: ${POSTGRES_PASSWORD}
    DB_HOST: postgres
    REDIS_HOST: redis
    DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
    command: celery -A openwisp worker -l info
    volumes:
      - /mnt/ssd/openwisP/media:/opt/openwisp/media
    restart: unless-stopped

celery-beat:
  image: openwisp/openwisp-dashboard:latest
  container_name: openwisP-celery-beat
  depends_on:
    - openwisP
    - redis
  environment:
    DB_ENGINE: django.contrib.gis.db.backends.postgis
    DB_NAME: openwisP_db
    DB_USER: openwisP
    DB_PASSWORD: ${POSTGRES_PASSWORD}
    DB_HOST: postgres
    REDIS_HOST: redis
    DJANGO_SECRET_KEY: ${DJANGO_SECRET_KEY}
  command: celery -A openwisp beat -l info
  restart: unless-stopped

nginx:
  image: nginx:alpine
  container_name: openwisP-nginx
  depends_on:
    - openwisP
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
    - /mnt/ssd/openwisP/static:/opt/openwisp/static:ro
    - /mnt/ssd/certs:/etc/nginx/certs:ro
  restart: unless-stopped

```

G.4.2 Archivo .env para OpenWISP

Crear: /mnt/ssd/docker/openwisP/.env

```

# PostgreSQL
POSTGRES_PASSWORD=<SECURE-DB-PASSWORD>

# Django
DJANGO_SECRET_KEY=<GENERATE-WITH: openssl rand -base64 48>
EMAIL_USER=noreply@smartgrid.local
EMAIL_PASSWORD=<APP-PASSWORD>

```

```
# OpenWISP
ORG_UUID=<GENERATE-WITH: uuidgen>
SHARED_SECRET=<SECURE-SHARED-KEY>
```

G.4.3 Configuración OpenWISP Agent en Gateway

Instalar agente en OpenWRT:

```
# Agregar feed OpenWISP
echo "src/gz openwisP https://downloads.openwisP.io/snapshots/packages/aarch64_cortex-a72/openwisP" \
  >> /etc/opkg/customfeeds.conf

opkg update
opkg install openwisP-config openwisP-monitoring

# Configurar agente
uci set openwisP.http.url='https://openwisP.gateway.local'
uci set openwisP.http.shared_secret='<SHARED_SECRET>'
uci set openwisP.http.uuid='<DEVICE_UUID>'
uci set openwisP.http.key='<DEVICE_KEY>'
uci set openwisP.http.verify_ssl='1'
uci set openwisP.http.consistent_key='1'

uci commit openwisP
/etc/init.d/openwisP enable
/etc/init.d/openwisP start

# Verificar conexión
logread | grep openwisP
```

G.5 mwan3: Multi-WAN Failover

G.5.1 Configuración Base (/etc/config/mwan3)

```
# Interfaz WAN Ethernet (prioridad 1)
config interface 'wan'
    option enabled '1'
    option family 'ipv4'
    list track_ip '1.1.1.1'
    list track_ip '8.8.8.8'
    option track_method 'ping'
    option reliability '1'
    option count '1'
    option size '56'
    option max_ttl '60'
    option timeout '2'
```

```
    option interval '5'
    option down '3'
    option up '3'

# Interfaz HaLow backhaul (prioridad 2)
config interface 'halow_wan'
    option enabled '1'
    option family 'ipv4'
    list track_ip '1.1.1.1'
    list track_ip '8.8.8.8'
    option track_method 'ping'
    option reliability '1'
    option count '1'
    option size '56'
    option max_ttl '60'
    option timeout '2'
    option interval '5'
    option down '3'
    option up '3'

# Interfaz LTE (prioridad 3, último recurso)
config interface 'lte'
    option enabled '1'
    option family 'ipv4'
    list track_ip '1.1.1.1'
    list track_ip '8.8.8.8'
    option track_method 'ping'
    option reliability '1'
    option count '1'
    option size '56'
    option max_ttl '60'
    option timeout '4'
    option interval '10'
    option down '3'
    option up '3'

# Métricas para cada interfaz
config member 'wan_m1_w3'
    option interface 'wan'
    option metric '1'
    option weight '3'

config member 'halow_m2_w2'
    option interface 'halow_wan'
    option metric '2'
    option weight '2'

config member 'lte_m3_w1'
    option interface 'lte'
    option metric '3'
    option weight '1'

# Política: Failover con prioridad
config policy 'balanced'
```

```

    option last_resort 'unreachable'
    list use_member 'wan_m1_w3'
    list use_member 'halow_m2_w2'
    list use_member 'lte_m3_w1'

# Política: Solo WAN principal
config policy 'wan_only'
    option last_resort 'default'
    list use_member 'wan_m1_w3'

# Política: Backup HaLow/LTE
config policy 'backup_only'
    option last_resort 'default'
    list use_member 'halow_m2_w2'
    list use_member 'lte_m3_w1'

# Regla: Tráfico crítico solo por WAN/HaLow
config rule 'critical'
    option src_ip '192.168.1.0/24'
    option dest_ip '0.0.0.0/0'
    option proto 'tcp'
    option dest_port '1883 8883 5683'
    option sticky '1'
    option timeout '600'
    option use_policy 'wan_only'

# Regla: Tráfico general con balanceo
config rule 'default_rule'
    option dest_ip '0.0.0.0/0'
    option use_policy 'balanced'

```

G.5.2 Script de Monitoreo mwan3

Archivo: /usr/local/bin/check-mwan3-status.sh

```

#!/bin/sh
# Script de monitoreo de estado mwan3 con alertas

LOG_FILE="/var/log/mwan3-status.log"
ALERT_THRESHOLD=3 # Número de fallos consecutivos para alertar

# Función de log
log_msg() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" | tee -a "$LOG_FILE"
}

# Obtener estado de interfaces
wan_status=$(mwan3 status | grep "interface wan" | awk '{print $NF}')
halow_status=$(mwan3 status | grep "interface halow_wan" | awk '{print $NF}')
lte_status=$(mwan3 status | grep "interface lte" | awk '{print $NF}')

```

```

log_msg "WAN: $wan_status | HaLow: $halow_status | LTE: $lte_status"

# Contador de fallos (persistente en /tmp)
WAN_FAILS=$(cat /tmp/mwan3_wan_fails 2>/dev/null || echo 0)
HALOW_FAILS=$(cat /tmp/mwan3_halow_fails 2>/dev/null || echo 0)
LTE_FAILS=$(cat /tmp/mwan3_lte_fails 2>/dev/null || echo 0)

# Verificar WAN
if [ "$wan_status" != "online" ]; then
    WAN_FAILS=$((WAN_FAILS + 1))
    echo $WAN_FAILS > /tmp/mwan3_wan_fails

    if [ $WAN_FAILS -ge $ALERT_THRESHOLD ]; then
        log_msg "ALERT: WAN offline por $WAN_FAILS checks consecutivos"
        # Enviar notificación (ej. MQTT alert a ThingsBoard)
        mosquitto_pub -h localhost -t "gateway/alerts" \
            -m "{\"alert\":\"WAN_DOWN\",\"fails\":$WAN_FAILS}"
    fi
else
    echo 0 > /tmp/mwan3_wan_fails
fi

# Verificar HaLow
if [ "$halow_status" != "online" ] && [ $WAN_FAILS -gt 0 ]; then
    HALOW_FAILS=$((HALOW_FAILS + 1))
    echo $HALOW_FAILS > /tmp/mwan3_halow_fails

    if [ $HALOW_FAILS -ge $ALERT_THRESHOLD ]; then
        log_msg "ALERT: HaLow offline (WAN también down)"
    fi
else
    echo 0 > /tmp/mwan3_halow_fails
fi

# Verificar LTE
if [ "$lte_status" != "online" ] && [ $WAN_FAILS -gt 0 ] && [ $HALOW_FAILS -gt 0 ]; then
    LTE_FAILS=$((LTE_FAILS + 1))
    echo $LTE_FAILS > /tmp/mwan3_lte_fails

    if [ $LTE_FAILS -ge $ALERT_THRESHOLD ]; then
        log_msg "CRITICAL: ALL UPLINKS DOWN!"
        mosquitto_pub -h localhost -t "gateway/alerts" \
            -m "{\"alert\":\"ALL_UPLINKS_DOWN\",\"timestamp\":$(date +%s)}"
    fi
else
    echo 0 > /tmp/mwan3_lte_fails
fi

# Mostrar tabla de routing mwan3
mwan3 status | head -20 >> "$LOG_FILE"

exit 0

```

Configurar cron para ejecutar cada minuto:

```
# Agregar a /etc/crontabs/root
* * * * * /usr/local/bin/check-mwan3-status.sh
```

G.6 Scripts de Mantenimiento

G.6.1 Backup Automatizado de Configuraciones

Archivo: /usr/local/bin/backup-gateway-config.sh

```
#!/bin/bash
# Backup completo de configuraciones del gateway

BACKUP_DIR="/mnt/ssd/backups"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/gateway_config_${TIMESTAMP}.tar.gz"
REMOTE_HOST="backup-server.local"
REMOTE_USER="backup"

mkdir -p "$BACKUP_DIR"

echo "[$(date)] Starting gateway configuration backup..."

# Crear tar.gz con todas las configuraciones
tar -czf "$BACKUP_FILE" \
    /etc/config \
    /etc/openvpn \
    /etc/nftables.d \
    /mnt/ssd/docker/*/docker-compose.yml \
    /mnt/ssd/docker/**/*.py \
    /mnt/ssd/docker/*/config \
    /mnt/ssd/docker/*/certs \
    /etc/crontabs \
    /etc/rc.local \
    2>/dev/null

if [ $? -eq 0 ]; then
    echo "[$(date)] Backup created: $BACKUP_FILE"
    ls -lh "$BACKUP_FILE"

    # Copiar a servidor remoto (opcional)
    if ping -c 1 "$REMOTE_HOST" >/dev/null 2>&1; then
        scp "$BACKUP_FILE" "$REMOTE_USER@$REMOTE_HOST:/backups/" && \
            echo "[$(date)] Backup uploaded to remote server"
    fi

    # Mantener solo últimos 7 backups locales
```

```

ls -t "$BACKUP_DIR"/gateway_config_*.tar.gz | tail -n +8 | xargs rm -f

echo "[$(date)] Backup complete"
else
echo "[$(date)] ERROR: Backup failed"
exit 1
fi

```

Configurar cron diario:

```

# /etc/crontabs/root
0 2 * * * /usr/local/bin/backup-gateway-config.sh

```

G.6.2 Check LTE Quota

Archivo: /usr/local/bin/check-lte-quota.sh

```

#!/bin/sh
# Monitoreo de cuota LTE con apagado automático al alcanzar límite

QUOTA_LIMIT_MB=5000 # 5 GB
CURRENT_USAGE_MB=$(vnstat -i wwan0 --online | cut -d';' -f11 | cut -d' ' -f1)

echo "[$(date)] LTE usage: ${CURRENT_USAGE_MB} MB / ${QUOTA_LIMIT_MB} MB"

if [ "$CURRENT_USAGE_MB" -ge "$QUOTA_LIMIT_MB" ]; then
echo "[$(date)] QUOTA EXCEEDED! Disabling LTE interface"

# Deshabilitar interfaz LTE en mwan3
uci set mwan3.lte.enabled='0'
uci commit mwan3
mwan3 restart

# Notificar vía MQTT
mosquitto_pub -h localhost -t "gateway/alerts" \
-m "{\"alert\":\"LTE_QUOTA_EXCEEDED\",\"usage_mb\":$CURRENT_USAGE_MB}"

# Enviar email (si está configurado)
echo "LTE quota exceeded: ${CURRENT_USAGE_MB}MB" | \
mail -s "Gateway LTE Alert" admin@smartgrid.local
else
REMAINING=$((QUOTA_LIMIT_MB - CURRENT_USAGE_MB))
echo "[$(date)] Remaining: ${REMAINING} MB"

# Alertar cuando quede menos de 500 MB
if [ "$REMAINING" -le 500 ]; then
mosquitto_pub -h localhost -t "gateway/alerts" \
-m "{\"alert\":\"LTE_QUOTA_LOW\",\"remaining_mb\":$REMAINING}"
fi
fi

```

G.7 Resumen

Este anexo ha documentado las configuraciones completas de OpenWRT para el gateway IoT SmartGrid, incluyendo:

- **UCI**: Configuraciones de red, wireless, DHCP/DNS, firewall
- **nftables**: Reglas de firewall personalizadas con protección DDoS
- **OpenVPN**: Servidor VPN con PKI Easy-RSA para acceso remoto seguro
- **OpenWISP**: Plataforma de gestión centralizada basada en Docker
- **mwan3**: Políticas de failover multi-WAN con tracking activo
- **Scripts**: Automatización de backups, monitoreo de cuota LTE, alertas

Todas las configuraciones están optimizadas para el hardware Raspberry Pi 4 con OpenWRT 23.05 y soportan los requisitos de resiliencia y seguridad del sistema de telemetría Smart Energy.

Referencias Bibliográficas

De Castro Korgi, R.: , 2010; *El Universo LaTeX*; 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 HaLowLink 2; PR Newswire; URL <https://finance.yahoo.com/news/morse-micro-announces-mass-production-070100596.html>; accessed: 2025-10-30.