Arquitectura de Gateway de Telemetra Edge para Smart Energy con 6LoWPAN, IEEE 802.11ah HaLow Multi-Banda y Procesamiento Inteligente

Luis Antonio Giraldo

Facultad de Minas
Departamento de Ingeniera Elctrica y Electrnica
Sede Medelln, 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 de Gateway de Telemetra Edge para Smart Energy con 6LoWPAN, IEEE 802.11ah HaLow Multi-Banda y Procesamiento Inteligente

Luis Antonio Giraldo

Tesis presentada como requisito parcial para optar por el título de:
Magster en Ingeniera - Ingeniera Electrnica

Director(a):
Prof. Dr. Director
Indicar si es Profesor Titular/Asociado - Departamento 2
Facultad de Minas
Universidad Nacional de Colombia

Codirector(a):
Prof. Dr. Co director
Indicar si es Profesor Titular/Asociado - Departamento de Ingeniera Elctrica y Electrnica
Facultad de Minas
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 de Minas
Departamento de Ingeniera Elctrica y Electrnica
Año entrega

Cita 01.

Autor
Fuente

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

Albert Einstein

Declaración

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

Sede Medelln., Noviembre 2025

Luis Antonio Giraldo

Agradecimientos

Listado de símbolos y abreviaturas

Resumen

Arquitectura de Gateway de Telemetra Edge para Smart Energy con 6LoWPAN, IEEE 802.11ah HaLow Multi-Banda y Procesamiento Inteligente

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

Lista de tablas

Contenido

1 Introducción

1.1 Contexto y Motivación

1.1.1 El Desafío de las Redes Smart Energy

La transición energética global hacia sistemas descentralizados, con alta penetración de energías renovables distribuidas (DER) y gestión activa de la demanda (DSM), exige infraestructuras de medición inteligente (AMI - Advanced Metering Infrastructure) capaces de recolectar, transmitir y procesar datos de millones de puntos de consumo en tiempo cuasi-real. Según la Agencia Internacional de Energía (IEA), se proyecta la instalación de más de 1.300 millones de medidores inteligentes a nivel global para 2030, generando aproximadamente 15 petabytes de datos de telemetría diarios.

Las arquitecturas tradicionales basadas en comunicación directa dispositivo-nube enfrentan limitaciones críticas: latencias elevadas (>200 ms), dependencia estricta de conectividad WAN continua, costos operacionales prohibitivos en escenarios de alta densidad de dispositivos, y dificultades para garantizar requisitos de tiempo real exigidos por aplicaciones de respuesta a la demanda (DR) y gestión de microrredes.

1.1.2 Estado Actual de las Tecnologías de Comunicación IoT

El ecosistema IoT para aplicaciones industriales y de infraestructura crítica se caracteriza por una heterogeneidad de tecnologías de comunicación, cada una optimizada para rangos específicos de alcance, throughput, latencia y consumo energético.

Comparativa Técnica de Protocolos Mesh 2.4 GHz

Thread emerge como protocolo preferencial para redes de campo en Smart Energy debido a su routing IPv6 nativo, que facilita integración con infraestructuras IP existentes, estandarización completa bajo Thread Group (miembro de Connectivity Standards Alliance), y soporte multi-vendor garantizado mediante certificación Thread 1.3.1.

Tabla 1-1: Comparación Thread, Zigbee y Bluetooth Mesh

Característica	Thread 1.3.1	Zigbee 3.0	Bluetooth Mesh
Capa física	IEEE 802.15.4	IEEE 802.15.4	Bluetooth 5.3 LE
Frecuencia	2.4 GHz	2.4 GHz / Sub-GHz	2.4 GHz
Topología	Mesh (MLE routing)	Mesh (AODV-based)	Managed Flooding
IPv6 nativo	Sí (6LoWPAN)	No (propietario)	No (GATT proxy)
Número máx. nodos	>250	65,535 (teórico)	32,767
Latencia (3 hops)	40-60 ms	80-120 ms	100-200 ms
Consumo RX/TX	19 mA / 22 mA	24 mA / 31 mA	9.2 mA / 10.5 mA
Sleep current	5 µA (ESP32-C6)	10 µA típico	2 µA (nRF52840)
Interoperabilidad	OTBR estándar	Requiere coordinador	Requiere provisioner
Security	TLS/DTLS 1.2	AES-128 CCM	AES-CCM

Tabla 1-2: Comparación Plataformas Edge IoT

Plataforma	ThingsBoard Edge	AWS IoT Greengrass	Azure IoT Edge	Node-RED
Arquitectura	Monolítica Java	Microservices Python	Containerizada .NET	Flow-based JS
Sincronización	Bidireccional	Unidireccional	Bidireccional	Manual
Rule Engine local	Sí (full chain)	Lambda local	Módulos custom	Function nodes
Almacenamiento	PostgreSQL/Cassandra	DynamoDB local	SQLite/Custom	Context store
Dashboard local	Sí (full featured)	No (CloudWatch)	No (portal cloud)	UI integrado
Autonomía offline	Ilimitada	Limitada	Limitada	Ilimitada
Footprint RAM	1-4 GB	512 MB - 2 GB	256 MB - 1 GB	128-512 MB
Licenciamiento	Apache 2.0	Propietario	Propietario	Apache 2.0
Curva aprendizaje	Media	Alta	Alta	Baja

Plataformas de Edge Computing - Análisis Comparativo

ThingsBoard Edge se posiciona como solución robusta para aplicaciones industriales que requieren continuidad operacional durante particiones WAN prolongadas, con capacidades completas de rule engine, dashboards interactivos y sincronización bidireccional de configuraciones y datos históricos.

HaLow - Posicionamiento frente a Alternativas de Última Milla

Wi-Fi HaLow combina throughput superior a LoRaWAN (40 Mbps vs 50 kbps), latencia determinística (<30 ms vs 1-5 seg), y ausencia de costos recurrentes de conectividad frente a LTE Cat-M1, posicionándose como tecnología óptima para backhaul de gateways Smart Energy en zonas urbanas y suburbanas densas.

1.1.3 Brechas en Arquitecturas IoT Existentes

El análisis del estado del arte revela limitaciones estructurales en arquitecturas IoT contemporáneas:

Tabla 1-3: Comparación Tecnologías Última Milla Smart Energy

Característica	**HaLow 802.11ah**	**LoRaWAN**	**LTE Cat-M1**	**Wi-Fi 6**
Frecuencia	Sub-GHz (900 MHz)	Sub-GHz (868/915 MHz)	LTE Bands	2.4/5 GHz
Alcance típico	1-2 km	5-15 km	10-35 km	50-100 m
Throughput máx.	40 Mbps (4 MHz BW)	50 kbps	1 Mbps	9.6 Gbps
Latencia típica	10-30 ms	1-5 seg	50-100 ms	<10 ms
Topología	Star/Mesh	Star (sin mesh)	Star (celular)	Star
Consumo TX (avg)	180 mA @ 1 MHz BW	120 mA	220 mA	350 mA
Cobertura indoor	Excelente (penetración)	Media	Buena	Limitada
Espectro	No licenciado ISM	No licenciado ISM	Licenciado (operador)	No licenciado ISM
Despliegue	Privado (CAPEX)	Gateway privado	Suscripción MVNO	Privado (CAPEX)
Costo por nodo	$25-40 módulo	$8-15 módulo	$12-25 módulo	$5-10 módulo

- **Dependencia cloud-centric**: Las arquitecturas tradicionales dispositivo → cloud presentan Single Points of Failure (SPOF) en enlaces WAN. Estudios empíricos en despliegues urbanos reportan disponibilidades de 94-96 % en conectividad celular LTE (downtimes acumulados 18-25 días/año), insuficientes para aplicaciones críticas.

- **Overhead de traducción multi-protocolo**: Los gateways convencionales implementan traductores application-layer (ej. Thread → MQTT → HTTP → Cloud), introduciendo latencias acumuladas de 150-300 ms y complejidad en mantenimiento de mapeos de datos.

- **Escalabilidad limitada del cloud ingestion**: Plataformas cloud IoT típicamente cobran por mensaje ingestado ($5-10 por millón de mensajes), resultando en costos prohibitivos para aplicaciones de telemetría de alta frecuencia (ej. 10,000 medidores reportando cada 5 minutos generan $2,880/mes solo en ingesta).

- **Ausencia de estándares de interoperabilidad**: La mayoría de soluciones comerciales implementan APIs propietarias, dificultando la migración entre vendors y bloqueando clientes en ecosistemas cerrados.

Análisis Cuantitativo de Overhead en Arquitecturas Tradicionales

La arquitectura propuesta reduce latencia end-to-end en 70 % (P50) y 79 % (P99) respecto a arquitecturas cloud-centric, eliminando el round-trip WAN mediante procesamiento local completo.

Tabla 1-4: Latencia por Arquitectura (device → cloud storage)

Componente	Cloud-Centric	Edge-Lite (Node-RED)	Propuesta (Edge Full)
Device → Gateway	40 ms (Thread)	40 ms (Thread)	40 ms (Thread)
Gateway → WAN	80 ms (LTE)	15 ms (Ethernet)	15 ms (HaLow/Eth)
WAN → Cloud	50 ms (RTT)	50 ms (RTT)	N/A (local)
Cloud processing	30 ms (ingestion)	30 ms (ingestion)	N/A
Cloud → DB write	10 ms (RDS write)	10 ms (RDS write)	8 ms (TimescaleDB)
TOTAL P50	**210 ms**	**145 ms**	**63 ms**
TOTAL P99	**450 ms**	**310 ms**	**95 ms**

1.2 Planteamiento del Problema

1.2.1 Definición del Problema de Investigación

¿Cómo diseñar e implementar una arquitectura IoT de borde multi-protocolo para aplicaciones Smart Energy que optimice simultáneamente: (1) la eficiencia de comunicación mediante el stack 6LoWPAN/CoAP/LwM2M sobre IEEE 802.15.4 reduciendo latencia y overhead de paquetes; (2) las capacidades de procesamiento en tiempo real mediante edge gateways con IA integrada para gestión inteligente de recursos; y (3) la conectividad de última milla mediante arquitectura basada en IEEE 802.11ah (HaLow) con selección adaptativa de bandwidth (2 MHz para conexiones estables, 4 MHz para gestión balanceada, 8 MHz para alto tráfico con línea de vista), garantizando latencia end-to-end $<$100 ms, reducción de tráfico WAN $>$60 %, y disponibilidad $>$99 % durante desconexiones prolongadas?

El problema aborda tres dimensiones técnicas fundamentales:

Dimensión 1 - Optimización del stack de protocolos: Implementación de 6LoWPAN con compresión de headers IPv6 (de 40 bytes a 2 bytes), CoAP como protocolo de aplicación ligero (overhead 4 bytes vs 100+ bytes HTTP), y LwM2M para gestión eficiente de dispositivos, logrando reducción de latencia $>$40 % y overhead de paquetes $>$70 % respecto a stacks tradicionales MQTT/HTTP.

Dimensión 2 - Edge Computing con IA: Despliegue de gateways edge con capacidades de procesamiento local mediante servicios containerizados (ThingsBoard Edge, TimescaleDB, Kafka), integración de modelos de lenguaje (LLM) locales para análisis en tiempo real de telemetría, detección de anomalías sin dependencia cloud, y gestión inteligente de recursos con adaptación dinámica a condiciones de red.

Dimensión 3 - Arquitectura multi-banda IEEE 802.11ah: Diseño de arquitectura de gateways basada en nodos HaLow con selección estratégica de bandwidth según caso de uso: 2 MHz (sensibilidad -99 dBm, alcance máximo $>$2 km, conexiones estables en NLOS, ideal para sensores remotos), 4 MHz (balance latencia/alcance, 40 Mbps agregado, gestión de red con throughput moderado), y 8 MHz (throughput $>$80 Mbps, latencia $<$20 ms, escenarios de alto tráfico con línea de vista como backhaul de concentradores).

1.2.2 Delimitación del Problema

La investigación se delimita a tres pilares técnicos fundamentales:

Pilar 1 - Stack de Protocolos 6LoWPAN/CoAP/LwM2M:

- **Capa de adaptación**: 6LoWPAN (RFC 6282) sobre IEEE 802.15.4-2020 con compresión de headers IPHC/NHC.
- **Capa de aplicación**: CoAP (RFC 7252) con modos Confirmable (CON) y Non-Confirmable (NON), block-wise transfer (RFC 7959), y Observe (RFC 7641).
- **Gestión de dispositivos**: LwM2M 1.2 (OMA SpecWorks) con objetos estándar (Security 0, Server 1, Device 3, Connectivity Monitoring 4, Firmware Update 5).
- **Métricas de evaluación**: Overhead de headers (bytes), latencia por salto (ms), eficiencia espectral (bits/Hz), tasa de entrega de paquetes (PDR), consumo energético por transmisión (mJ/bit).

Pilar 2 - Edge Gateway con IA:

- **Plataforma hardware**: ARMv8 Cortex-A53 quad-core @ 1.8 GHz (Raspberry Pi 4 o Banana Pi BPI-R4), 4-8 GB RAM, NVMe SSD 256 GB, periféricos M.2 para LTE.
- **Sistema operativo**: OpenWRT 23.05.x con kernel Linux 5.15 LTS + parches PREEMPT_RT para latencias determinísticas.
- **Stack de servicios**: ThingsBoard Edge 3.6 (procesamiento CEP local), PostgreSQL 15 + TimescaleDB 2.13 (series temporales), Apache Kafka 7.5 (message broker), Ollama + Llama 3.2 3B (inferencia LLM local).
- **Capacidades IA**: Detección de anomalías en consumo energético, mantenimiento predictivo basado en patrones de alarmas, optimización de rutas mesh mediante reinforcement learning, compresión adaptativa de datos según ancho de banda disponible.
- **Métricas de evaluación**: Latencia de inferencia IA ($<$500 ms objetivo), precisión de detección de anomalías ($>$95 %), overhead de CPU/RAM bajo carga, tiempo de failover multi-WAN ($<$30s).

Pilar 3 - Arquitectura Multi-Banda IEEE 802.11ah:

- **Bandwidths evaluados**: 1 MHz (150 kbps, sensibilidad -99 dBm, $>$2 km NLOS), 2 MHz (300-450 kbps, sensibilidad -96 dBm, 1.5-2 km, óptimo para conexiones estables), 4 MHz (600-900 kbps, sensibilidad -91 dBm, 1-1.5 km, balance throughput/alcance), 8 MHz (1.2-1.8 Mbps, sensibilidad -85 dBm, 0.5-1 km LOS, alto tráfico backhaul).
- **Topologías**: Star (gateway central + nodos finales), Mesh 802.11s (routing HWMP, auto-healing), EasyMesh (IEEE 1905.1, roaming coordinado).
- **Casos de uso específicos**: 2 MHz para sensores remotos baja frecuencia (lecturas horarias, zonas rurales, penetración indoor), 4 MHz para gestión de red balanceada (lecturas 15 min, zonas suburbanas, densidad media), 8 MHz para backhaul de concentradores (agregación de datos, zonas urbanas LOS, latencia crítica $<$50 ms).
- **Métricas de evaluación**: Link budget por bandwidth, throughput agregado vs número de clientes, latencia P50/P95/P99, tasa de handover en movilidad, cobertura efectiva por MCS.

Dominio de aplicación: Redes de telemetría Smart Energy en entornos urbanos/suburbanos con densidad 1,000-10,000 puntos de medición por km^2, con validación en caso de estudio real de 900 medidores residenciales.

Estándares implementados: IEEE 802.15.4-2020, IEEE 802.11ah-2016, IEEE 2030.5-2023, ISO/IEC 30141:2024, OMA LwM2M 1.2, RFC 6282 (6LoWPAN), RFC 7252 (CoAP).

Se excluyen del alcance: implementación de IEC 61850 (subestaciones), PLC (G3-PLC/PRIME), certificación formal Thread/Matter, redes 5G/NR-Light, y blockchain para auditoría (trabajo futuro).

1.2.3 Justificación

Justificación Técnica

Las arquitecturas edge-computing para IoT industrial requieren capacidades de procesamiento local, almacenamiento persistente y autonomía operacional que las soluciones cloud-centric tradicionales no pueden garantizar. La integración de Wi-Fi HaLow como tecnología de backhaul representa una innovación técnica respecto al estado del arte (dominado por LTE/LoRaWAN), aprovechando sus ventajas de throughput (40 Mbps vs 1 Mbps LTE Cat-M1), latencia ($<$30 ms vs $>$50 ms), y ausencia de costos recurrentes de conectividad.

Justificación Económica

Análisis de TCO (Total Cost of Ownership) para despliegue de 1,000 puntos de medición durante 5 años:

- **Cloud-centric + LTE:** CAPEX \$150k (hardware) + OPEX \$180k (conectividad \$15/nodo/año) = \$330k
- **Propuesta HaLow:** CAPEX \$200k (hardware + APs HaLow) + OPEX \$25k (mantenimiento) = \$225k
- **Ahorro proyectado:** 32 % (\$105k en 5 años)

Justificación Académica

La investigación contribuye al cuerpo de conocimiento en arquitecturas IoT heterogéneas mediante:

- Diseño de arquitectura de referencia para gateways multi-PHY conformes con ISO/IEC 30141.
- Caracterización empírica de latencias en integración Thread HaLow.
- Metodología de implementación de IEEE 2030.5 Function Sets sobre plataformas embebidas Linux.
- Evaluación comparativa de estrategias de failover multi-WAN en gateways IoT.

1.2.4 Metodología de Investigación

La investigación sigue un enfoque mixto que combina Design Science Research (DSR) para el diseño de artefactos tecnológicos, Investigación Experimental para la validación de hipótesis cuantitativas, y Estudio de Caso para la evaluación en contexto real.

Fase 1 - Análisis y Diseño (Design Science)

Objetivos: Especificar requisitos funcionales/no funcionales, diseñar arquitectura de referencia multi-capa, definir interfaces entre componentes.

Actividades:

1. Revisión sistemática de literatura sobre arquitecturas IoT edge y estándares Smart Energy (IEEE 2030.5, ISO/IEC 30141, IEC 61850).
2. Análisis comparativo de tecnologías de comunicación (Thread, Zigbee, BLE Mesh, HaLow, LoRaWAN, LTE Cat-M1).
3. Diseño de arquitectura de 4 capas: Conectividad, Orquestación, Procesamiento, Aplicación.
4. Especificación de interfaces: OTBR APIs, MQTT topics, IEEE 2030.5 REST endpoints.
5. Modelado de latencias mediante teoría de colas (M/M/1 para gateway, M/G/ para cloud).

Entregables: Diagrama de arquitectura (Capítulo 3), especificación de requisitos (Capítulo 3.3), diseño de base de datos TimescaleDB (Anexo B).

Fase 2 - Implementación (Engineering)

Objetivos: Implementar gateway prototipo funcional, integrar componentes hardware/software, desarrollar servicios containerizados.

Actividades:

1. Configuración plataforma hardware: Banana Pi BPI-R4 (4x Cortex-A53 @ 1.8 GHz, 4 GB RAM) + nRF52840 RCP (Thread) + Morse Micro MM6108 (HaLow) + Quectel EG25-G (LTE).
2. Instalación y configuración OpenWRT 23.05.x con kernel real-time patches (PREEMPT_RT).
3. Despliegue stack Docker Compose: ThingsBoard Edge 3.6.0, PostgreSQL 15 + TimescaleDB 2.13, Apache Kafka 7.5.0, IEEE 2030.5 Server (Python/Flask), Ollama LLM (Llama 3.2 3B).
4. Implementación IEEE 2030.5 Function Sets: DCAP, Time, EndDevice, MirrorUsagePoint, MirrorMeterReading, Messaging (XML schemas según estándar).
5. Configuración mwan3 para failover multi-WAN (Ethernet métrica 10, HaLow STA métrica 15, LTE métrica 20).
6. Desarrollo nodos IoT: ESP32-C6 Thread LwM2M + sensor BME280 (temperatura/humedad/presión).

Entregables: Documentación de instalación (Anexo A), archivos docker-compose.yml (Anexo B), scripts de integración (Anexo C), código fuente nodos IoT (Anexo E).

Fase 3 - Validación Experimental

Objetivos: Validar hipótesis mediante mediciones empíricas, caracterizar rendimiento del sistema, evaluar resiliencia ante fallos.

Experimentos:

1. **Exp. 1 - Latencia end-to-end:** Medir latencia desde generación de telemetría en nodo IoT hasta persistencia en TimescaleDB. Variables independientes: número de nodos (N=5,10,25), frecuencia de muestreo (5s, 30s, 60s). Variables dependientes: latencia P50/P95/P99, jitter. Duración: 72 horas por configuración.
2. **Exp. 2 - Disponibilidad durante desconexión WAN:** Simular partición WAN de 48 horas desconectando Ethernet y deshabilitando LTE. Métricas: porcentaje de mensajes bufferizados exitosamente, tiempo de sincronización post-reconexión, disponibilidad de servicios locales (dashboards, alarmas).
3. **Exp. 3 - Throughput agregado HaLow:** Saturar enlace HaLow con tráfico concurrente de múltiples nodos. Medir throughput agregado vs número de clientes (N=1,5,10,20). Configuraciones: 1 MHz/2 MHz bandwidth, MCS 0-10.
4. **Exp. 4 - Failover multi-WAN:** Provocar fallas en interfaces Ethernet → HaLow → LTE. Medir tiempo de detección de falla, tiempo de conmutación, pérdida de paquetes durante transición.
5. **Exp. 5 - Overhead de procesamiento:** Caracterizar CPU/RAM/storage bajo cargas de 10/50/100 dispositivos. Identificar cuellos de botella mediante profiling (perf, flamegraphs).

Herramientas de medición: Wireshark/tshark para captura de paquetes, Grafana + Prometheus para métricas de sistema, scripts Python para análisis estadístico (pandas, scipy).

Entregables: Datasets de mediciones (repositorio GitHub), gráficas de resultados (Capítulo 4), análisis estadístico (ANOVA, t-tests).

Fase 4 - Evaluación Comparativa

Objetivos: Comparar arquitectura propuesta vs soluciones baseline (cloud-centric, edge-lite).

Baseline 1 - Cloud-Centric: Nodos Thread → OTBR → Gateway LTE → AWS IoT Core → Lambda → DynamoDB.

Baseline 2 - Edge-Lite: Nodos Thread → OTBR → Node-RED (local) → AWS IoT Core (sync).

Criterios de comparación:

- Latencia P50/P99 device → storage
- Disponibilidad durante partición WAN 48h
- Throughput máximo (mensajes/seg)
- Consumo energético gateway (Watts)
- Costos OPEX (USD/mes para 100 dispositivos)

- Complejidad de deployment (horas-persona)

Entregables: Tabla comparativa (Capítulo 4), análisis de trade-offs, recomendaciones de uso.

1.3 Hipótesis

1.3.1 Hipótesis General

Una arquitectura IoT para Smart Energy basada en: (1) stack de protocolos optimizado 6LoWPAN/CoAP/LwM2M sobre IEEE 802.15.4, (2) edge gateways con capacidades de procesamiento local e IA integrada, y (3) conectividad de última milla mediante IEEE 802.11ah con selección adaptativa de bandwidth (2/4/8 MHz), permite reducir la latencia end-to-end en >70 %, el overhead de paquetes en >60 %, el tráfico WAN en >65 %, garantizando disponibilidad >99 % durante desconexiones prolongadas y procesamiento inteligente en tiempo real, comparado con arquitecturas tradicionales basadas en MQTT/HTTP sobre conectividad celular.

1.3.2 Hipótesis Específicas

H1 - Optimización mediante 6LoWPAN/CoAP/LwM2M: La implementación del stack 6LoWPAN (compresión IPHC/NHC) + CoAP (overhead 4-10 bytes) + LwM2M (objetos binarios TLV) sobre IEEE 802.15.4 reduce el overhead de paquetes en >70 % y la latencia por salto en >40 % comparado con MQTT/JSON sobre TCP/IP, logrando tiempos de transmisión <15 ms por hop en topologías mesh de hasta 5 saltos.

H2 - Procesamiento Edge con IA: El despliegue de servicios containerizados edge (ThingsBoard Edge, TimescaleDB, Kafka) con integración de modelos LLM locales (Ollama + Llama 3.2 3B) permite: (a) reducción de tráfico WAN en >65 % mediante procesamiento local, (b) latencia de inferencia <500 ms para detección de anomalías, (c) disponibilidad de servicios >99 % durante desconexiones WAN >72 horas, y (d) precisión de detección de anomalías >95 % en patrones de consumo energético.

H3 - Arquitectura Multi-Banda 802.11ah: La arquitectura basada en gateways HaLow con selección estratégica de bandwidth según caso de uso maximiza eficiencia operacional:

- **2 MHz**: Óptimo para conexiones estables con sensores remotos (>2 km alcance, sensibilidad -96 dBm, tráfico <100 kbps, entornos NLOS con penetración indoor superior), logrando PDR >98 % en condiciones adversas con SNR 8-12 dB.
- **4 MHz**: Balance ideal para gestión de red (1-1.5 km alcance, throughput 40 Mbps agregado, latencia <50 ms P95), soportando 50+ nodos con tráfico moderado (lecturas cada 15 min) sin degradación >10 %.
- **8 MHz**: Maximiza throughput para alto tráfico con línea de vista (backhaul de concentradores, >80 Mbps, latencia <20 ms P99, alcance 0.5-1 km LOS), permitiendo agregación de datos de 100+ dispositivos por gateway.

H4 - Compresión 6LoWPAN de Headers: La compresión IPHC (IPv6 Header Compression) de 6LoWPAN reduce headers IPv6+UDP de 48 bytes a 2-7 bytes (compresión >85 %), y la compresión NHC (Next

Header Compression) para CoAP reduce overhead adicional de 10-20 bytes a 2-4 bytes, resultando en payloads efectivos >90 % del MTU IEEE 802.15.4 (127 bytes) para aplicaciones Smart Energy.

H5 - Eficiencia CoAP vs MQTT: CoAP sobre UDP con modos Non-Confirmable (NON) para telemetría no crítica y Confirmable (CON) para comandos críticos, combinado con Observe para subscripciones, reduce latencia en >50 % y overhead de red en >60 % comparado con MQTT/TCP, logrando tiempos de respuesta <30 ms para transacciones GET/POST en redes Thread mesh.

H6 - LwM2M para Gestión Eficiente: LwM2M con objetos estándar OMA (Device, Connectivity Monitoring, Firmware Update) y transporte CoAP reduce tráfico de gestión de dispositivos en >75 % comparado con soluciones propietarias HTTP/REST, permitiendo actualizaciones OTA de firmware con transferencia block-wise sobre enlaces de baja velocidad (<250 kbps) sin timeouts.

H7 - Procesamiento CEP Local: El motor de reglas Complex Event Processing (CEP) de ThingsBoard Edge desplegado localmente en gateway procesa >10,000 eventos/seg con latencia <10 ms P99, ejecutando rule chains complejas (filtrado, agregación, transformación, alarmas) sin requerir round-trip WAN, reduciendo latencia de respuesta en >80 % comparado con procesamiento cloud.

H8 - Ventaja Comparativa Integral: La arquitectura propuesta supera a arquitecturas tradicionales (cloud-centric MQTT/LTE) en al menos 5 de 7 métricas clave: latencia (<30 % baseline), overhead paquetes (<40 % baseline), tráfico WAN (<35 % baseline), disponibilidad offline (>72h vs 0h), precisión IA (>95 % vs N/A), alcance HaLow (>150 % vs WiFi), y eficiencia energética (<60 % baseline).

1.4 Objetivos

1.4.1 Objetivo General

Diseñar, implementar y validar una arquitectura IoT centrada en edge gateways para aplicaciones Smart Energy que integre: (1) stack de protocolos optimizado 6LoWPAN/CoAP/LwM2M sobre IEEE 802.15.4 para reducción de latencia y overhead, (2) capacidades de procesamiento edge con IA local para gestión inteligente de recursos en tiempo real, y (3) conectividad de última milla mediante IEEE 802.11ah con estrategia multi-banda (2/4/8 MHz) adaptada a casos de uso específicos, garantizando latencia end-to-end <100 ms, reducción de tráfico WAN >65 %, y disponibilidad >99 % con conformidad a estándares IEEE 2030.5-2023 e ISO/IEC 30141:2024.

1.4.2 Objetivos Específicos

OE1 - Stack de Protocolos Optimizado 6LoWPAN/CoAP/LwM2M:

- Implementar capa de adaptación 6LoWPAN (RFC 6282) con compresión IPHC/NHC sobre IEEE 802.15.4, validando reducción de overhead de headers >85 % (de 48 bytes a <7 bytes) en tráfico de telemetría Smart Energy.
- Desplegar protocolo CoAP (RFC 7252) con modos CON/NON, Observe (RFC 7641) para subscripciones, y block-wise transfer (RFC 7959), midiendo latencia <30 ms para transacciones request/response en topologías mesh 3-5 saltos.

- Integrar LwM2M 1.2 (OMA SpecWorks) con objetos estándar (Security, Server, Device, Connectivity Monitoring, Firmware Update) para gestión unificada de dispositivos, validando reducción de tráfico de gestión >75 % vs soluciones HTTP/REST propietarias.
- Caracterizar empíricamente PDR (Packet Delivery Ratio), latencia por hop, y consumo energético por bit transmitido en función de topología mesh (star, tree, mesh completo) y carga de red (5/10/25/50 nodos).

OE2 - Edge Gateway con Procesamiento en Tiempo Real e IA:

- Desplegar stack de servicios containerizados (ThingsBoard Edge, PostgreSQL + TimescaleDB, Apache Kafka, IEEE 2030.5 Server) sobre OpenWRT 23.05 con kernel PREEMPT_RT, garantizando latencias de procesamiento <10 ms P99 para pipeline MQTT ingestion → rule engine → TimescaleDB persistence.
- Integrar motor de inferencia LLM local (Ollama + Llama 3.2 3B) con latencia <500 ms para análisis de telemetría en tiempo real, implementando casos de uso: (a) detección de anomalías en consumo con precisión >95 %, (b) mantenimiento predictivo basado en patrones de alarmas, (c) compresión adaptativa de datos según bandwidth disponible.
- Implementar gestión inteligente de recursos con adaptación dinámica: priorización de tráfico crítico (alarmas) vs no crítico (históricos), ajuste automático de frecuencia de muestreo según condiciones de red, y compactación de datos mediante CBOR/Protocol Buffers reduciendo payload >40 %.
- Validar resiliencia mediante buffering persistente local con capacidad >100,000 mensajes (500 MB), sincronización bidireccional post-desconexión WAN >72h con catch-up <30 minutos, y disponibilidad de servicios locales (dashboards, rule engine) >99 % durante particiones WAN.

OE3 - Arquitectura Multi-Banda IEEE 802.11ah con Nodos HaLow:

- Diseñar arquitectura de red basada en gateways edge con nodos HaLow (Morse Micro MM6108) soportando topologías Star (simple), Mesh 802.11s (auto-healing HWMP), y EasyMesh (IEEE 1905.1 roaming coordinado), validando escalabilidad a 50+ nodos por gateway sin degradación >10 % de latencia.
- Caracterizar empíricamente desempeño por bandwidth:
  - **2 MHz**: Sensibilidad -96 dBm, alcance >2 km NLOS, throughput 300-450 kbps, MCS 1-2, latencia <100 ms P95, PDR >98 % con SNR 8-12 dB. Caso de uso: sensores remotos rurales, lecturas horarias, penetración indoor.
  - **4 MHz**: Sensibilidad -91 dBm, alcance 1-1.5 km, throughput 40 Mbps agregado, MCS 3-4, latencia <50 ms P95, soporte 50+ nodos concurrentes. Caso de uso: gestión balanceada zonas suburbanas, lecturas cada 15 min.
  - **8 MHz**: Sensibilidad -85 dBm, alcance 0.5-1 km LOS, throughput >80 Mbps, MCS 5-7, latencia <20 ms P99. Caso de uso: backhaul de concentradores en zonas urbanas con línea de vista, agregación de 100+ dispositivos.
- Implementar algoritmo de selección adaptativa de bandwidth basado en: (a) condiciones de propagación (RSSI, SNR, PDR histórico), (b) requisitos de aplicación (latencia, throughput, prioridad), y (c) densidad de red (número de nodos activos, carga agregada).
- Evaluar escalabilidad arquitectónica: topología Star (2,500 endpoints, 3 km), Mesh 802.11s (7,500 endpoints, 9 km, auto-healing <10s), EasyMesh (12,500 endpoints, roaming transparente, band steering 2/4/8 MHz).

OE4 - Validación Experimental Comparativa:

- Realizar benchmarking cuantitativo vs 2 baselines: (a) Cloud-centric (MQTT/JSON/TCP sobre LTE Cat-M1), (b) Edge-lite (Node-RED local + MQTT cloud).
- Métricas comparadas: latencia end-to-end P50/P95/P99, overhead de paquetes (bytes header/payload), tráfico WAN (GB/mes), disponibilidad offline (horas), precisión IA (
- Generar datasets públicos de mediciones (latencias, throughput, PDR) con 10+ nodos IoT ESP32-C6 Thread LwM2M en despliegue piloto de 72 horas continuas bajo condiciones variables de carga y propagación.

OE5 - Caso de Estudio Smart Energy Real:

- Desplegar prototipo funcional para 900 medidores residenciales con topología: 300 nodos ESP32-C6 Thread por gateway × 3 gateways Raspberry Pi 4 + OpenWRT + HaLow, validando arquitectura en condiciones reales urbanas/suburbanas.
- Implementar conformidad IEEE 2030.5-2023 (Function Sets: DCAP, Time, EndDevice, MirrorUsagePoint, MirrorMeterReading, Messaging) con validación de interoperabilidad funcional vía test suite OpenADR VTN.
- Documentar lecciones aprendidas, patrones de diseño arquitectónicos, y guías de implementación técnica (instalación OpenWRT, configuración HaLow 4 modos, despliegue stack Docker, tuning kernel PREEMPT_RT) en anexos técnicos completos.

1.5 Alcances y Limitaciones

1.5.1 Alcances

1. **Diseño arquitectónico**: Especificación completa de arquitectura multi-capa con definición de componentes, interfaces y flujos de datos, mapeo a vistas ISO/IEC 30141 (funcional, información, despliegue, operacional).
2. **Implementación prototipo**: Gateway funcional basado en Banana Pi BPI-R4 con integración Thread (nRF52840 RCP), HaLow (Morse Micro MM6108), LTE (Quectel EG25-G), OpenWRT 23.05.x y stack Docker Compose con 7 servicios.
3. **Conformidad estándares**: Implementación de IEEE 2030.5-2023 Function Sets (DCAP, Time, EndDevice, MirrorUsagePoint, MirrorMeterReading, Messaging) y mapeo ISO/IEC 30141:2024.
4. **Nodos IoT**: Desarrollo de nodos ESP32-C6 Thread con cliente LwM2M, sensores BME280 y firmware actualizable OTA.
5. **Validación experimental**: Medición empírica de latencia, throughput, disponibilidad, failover y overhead en condiciones controladas de laboratorio y despliegue piloto urbano.
6. **Documentación técnica**: Anexos con guías de instalación (OpenWRT, docker-compose), configuraciones UCI completas, schemas IEEE 2030.5 XML, código fuente completo (GitHub).
7. **Evaluación comparativa**: Benchmarking cuantitativo vs 2 baselines (AWS IoT Core cloud-centric, Node-RED edge-lite) con métricas de latencia, disponibilidad, costos, complejidad.

1.5.2 Limitaciones

1. **Escala de despliegue**: Validación con 10 nodos IoT y 2 gateways en área de 300 metros. No se valida escalabilidad a miles de dispositivos en despliegue real.
2. **Hardware específico**: Implementación dependiente de Morse Micro MM6108 (único chipset HaLow comercialmente disponible en 2024). Resultados pueden no generalizar a futuros chipsets.
3. **Certificación formal**: No se realiza certificación formal Thread 1.3.1 ni IEEE 2030.5. Conformidad validada mediante interoperabilidad funcional, no certificación oficial.
4. **Seguridad**: Implementación de TLS 1.2/1.3 y certificados X.509, pero sin auditoría de seguridad formal ni penetration testing exhaustivo.
5. **Estándares excluidos**: No se implementa IEC 61850 (comunicación en subestaciones) ni interoperabilidad PLC (Power Line Communication).
6. **Cobertura geográfica**: Validación en entorno urbano/suburbano. No se valida en zonas rurales remotas con cobertura celular limitada.
7. **Condiciones ambientales**: Pruebas en condiciones de laboratorio (20-25°C, humedad controlada). No se valida operación en extremos de rango industrial (-40°C a +85°C).
8. **Regulaciones RF**: Operación en banda ISM 902-928 MHz (EE.UU./América). Requiere adaptación para bandas 863-868 MHz (Europa) o 755-787 MHz (China).

1.6 Contribuciones Esperadas

1.6.1 Contribuciones Académicas

1. **Arquitectura de referencia IoT heterogénea**: Especificación de arquitectura multi-capa para gateways edge que integra múltiples PHYs (802.15.4, 802.11ah, LTE), conforme con ISO/IEC 30141:2024, documentando patrones de diseño, trade-offs arquitectónicos y decisiones de ingeniería.
2. **Caracterización empírica Thread HaLow**: Primera caracterización publicada de latencias, throughput y reliability en integración Thread-HaLow mediante bridge Ethernet transparente, incluyendo análisis de overhead de OTBR y impacto de topologías mesh.
3. **Metodología IEEE 2030.5 sobre Linux embebido**: Documentación de estrategias de implementación de Function Sets IEEE 2030.5 sobre plataformas resource-constrained (ARMv8, 4 GB RAM), incluyendo optimizaciones de XML parsing, caching y gestión de certificados.
4. **Benchmarking arquitecturas edge IoT**: Dataset público de mediciones comparativas (latencia, throughput, overhead) entre arquitecturas cloud-centric, edge-lite y edge-full, proporcionando guías de selección arquitectónica basadas en requisitos de aplicación.

1.6.2 Contribuciones Técnicas

1. **Implementación open-source IEEE 2030.5**: Servidor Python/Flask que implementa 6 Function Sets con schemas XML validados, autenticación TLS mutua y RBAC, disponible bajo licencia Apache 2.0 en repositorio GitHub.

2. **Configuraciones OpenWRT para HaLow**: Documentación completa de configuración UCI para driver Morse Micro MM6108 (SPI), incluyendo scripts de inicialización, configuración hostapd y troubleshooting.
3. **Stack Docker Compose optimizado**: Composición de servicios edge (ThingsBoard, TimescaleDB, Kafka, IEEE 2030.5, Ollama) con resource management, health checks y restart policies, optimizado para hardware Cortex-A53.
4. **Firmware nodos IoT Thread LwM2M**: Implementación ESP-IDF para ESP32-C6 con cliente LwM2M (Wakaama), driver BME280, Deep Sleep scheduling y OTA segura.

1.6.3 Contribuciones a la Industria

1. **Reducción de costos operacionales**: Demostración de viabilidad económica de arquitectura HaLow-based vs LTE, con TCO 32 % inferior en despliegues de 1,000+ puntos durante 5 años.
2. **Guía de implementación práctica**: Documentación técnica completa (instalación, configuración, troubleshooting) que permite replicación de arquitectura por integradores de sistemas y utilities.
3. **Caso de negocio para HaLow**: Evaluación cuantitativa de beneficios (throughput, latencia, costos) de Wi-Fi HaLow vs LoRaWAN/LTE Cat-M1 en aplicaciones Smart Energy, acelerando adopción de estándar IEEE 802.11ah.
4. **Interoperabilidad multi-vendor**: Validación de conformidad IEEE 2030.5 que facilita integración con dispositivos certificados de múltiples fabricantes, reduciendo lock-in tecnológico.

1.7 Organización del Documento

El presente documento se estructura en los siguientes capítulos:

Capítulo 1 - Introducción: Contextualización del problema, estado actual de tecnologías IoT, brechas identificadas, planteamiento del problema, hipótesis, objetivos, metodología, alcances y contribuciones esperadas.

Capítulo 2 - Marco Teórico: Fundamentos de redes Smart Energy, protocolos de comunicación IoT (Thread, HaLow, LTE Cat-M1), estándares de interoperabilidad (IEEE 2030.5, ISO/IEC 30141, IEC 61850), tecnologías de edge computing (Docker, TimescaleDB, Kafka), plataformas IoT (ThingsBoard), seguridad en sistemas IoT, y estado del arte de arquitecturas edge heterogéneas.

Capítulo 3 - Gateway de Telemetría: Arquitectura del gateway multi-protocolo, conformidad con estándares internacionales, requisitos funcionales/no funcionales, arquitectura jerárquica de 3 niveles IoT, diseño de hardware y software, y Stack de Servicios Containerizados.

Capítulo 4 - Arquitectura de Telemetría: Visión general de arquitectura end-to-end, capa de dispositivos (medidores inteligentes), capa de campo (nodos Thread, DCUs), capa de agregación (gateway HaLow), capa de aplicación (ThingsBoard cloud), análisis de seguridad end-to-end, y modelado de latencias mediante teoría de colas.

Capítulo 5 - Conclusiones y Trabajo Futuro: Síntesis de la investigación, cumplimiento de objetivos, validación de hipótesis, contribuciones académicas y técnicas, lecciones aprendidas, limitaciones del trabajo, y recomendaciones para trabajo futuro.

Anexos: Instalación OpenWRT y configuración HaLow (Anexo A), Docker Compose y servicios (Anexo B), Scripts de integración (Anexo C), Especificaciones IEEE 2030.5 (Anexo D), Implementación nodo IoT ESP32-C6 (Anexo E), Configuraciones OpenWRT UCI completas (Anexo F).

2 Marco Teórico

2.1 Fundamentos de Redes Smart Energy

2.1.1 Evolución de las Infraestructuras Eléctricas

La transición de redes eléctricas tradicionales unidireccionales hacia Smart Grids bidireccionales representa un cambio paradigmático en la operación de sistemas energéticos. Las Smart Grids integran tecnologías de información y comunicación (TIC) para monitoreo, control y optimización en tiempo real del flujo eléctrico desde generación hasta consumo final. Este enfoque permite: integración masiva de energías renovables distribuidas (DER - Distributed Energy Resources), gestión activa de la demanda (DSM - Demand Side Management), detección y auto-recuperación de fallas (self-healing), y participación activa de prosumidores (consumidores que también generan energía).

Según el National Institute of Standards and Technology (NIST), una Smart Grid implementa siete dominios interconectados: Bulk Generation, Transmission, Distribution, Customer, Operations, Markets, y Service Provider. La infraestructura de medición inteligente (AMI - Advanced Metering Infrastructure) constituye el dominio Customer, proporcionando visibilidad granular de patrones de consumo y habilitando servicios de respuesta a la demanda (DR).

2.1.2 Arquitectura de Referencia Smart Grid

El modelo de referencia NIST para Smart Grid (NIST Framework and Roadmap for Smart Grid Interoperability Standards) define tres capas principales:

1. **Power and Energy Layer**: Infraestructura física de generación, transmisión, distribución y almacenamiento.
2. **Communication Layer**: Redes de datos multi-protocolo (HAN, NAN, WAN) que transportan información de telemetría y comandos de control.
3. **Application Layer**: Sistemas de gestión de energía (EMS), gestión de distribución (DMS), gestión de demanda (DERMS), y analytics.

La arquitectura AMI se compone típicamente de: medidores inteligentes (smart meters) instalados en puntos de consumo, concentradores/gateways que agregan datos de decenas o cientos de medidores, y head-end systems en centros de control que procesan millones de registros diarios.

2.2 Protocolos de Comunicación IoT

2.2.1 Thread 802.15.4 - Redes Mesh de Baja Potencia

Thread es un protocolo de red IPv6 basado en IEEE 802.15.4, diseñado específicamente para aplicaciones IoT domésticas e industriales de baja potencia. Desarrollado por Thread Group (ahora parte de Connectivity Standards Alliance), estandariza la capa de red y transporte sobre la capa física/MAC 802.15.4, proporcionando routing mesh, auto-configuración y seguridad end-to-end.

Arquitectura del Protocolo Thread

Thread implementa un stack de protocolos completo sobre IEEE 802.15.4:

- **Physical Layer (PHY)**: IEEE 802.15.4-2015, banda 2.4 GHz, OQPSK modulation, 250 kbps data rate, 16 canales (11-26).
- **MAC Layer**: CSMA/CA con backoff exponencial, frame acknowledgments, retransmisiones automáticas.
- **Network Layer**: 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) - compresión de headers IPv6, fragmentación, mesh-under routing.
- **Transport Layer**: UDP (principalmente), TCP limitado por overhead.
- **Application Layer**: CoAP (Constrained Application Protocol), MQTT-SN, LwM2M.

El routing Thread utiliza Mesh Link Establishment (MLE) para descubrimiento de vecinos y mantenimiento de tabla de rutas. Cada dispositivo mantiene una tabla con métricas de link quality (LQI - Link Quality Indicator) y path cost hacia el líder de la red. El protocolo implementa route optimization continuo basado en Expected Transmission Count (ETX).

Thread Border Router (OTBR)

El Thread Border Router (OTBR) actúa como gateway entre la red Thread (802.15.4) y redes IP tradicionales (Ethernet, Wi-Fi), proporcionando:

- **Traducción IPv6**: Routing entre prefijos Thread (mesh-local) y prefijos globales.
- **NAT64/DNS64**: Interoperabilidad con servicios IPv4-only.
- **Multicast forwarding**: Propagación de mensajes multicast entre segmentos.
- **Commissioning**: Incorporación segura de nuevos dispositivos mediante out-of-band authentication.

La implementación de referencia OpenThread Border Router (OTBR) soporta dos arquitecturas: System-on-Chip (SoC) donde un único MCU ejecuta stack Thread y aplicación, o Radio Co-Processor (RCP) donde un MCU dedicado (ej. nRF52840) implementa PHY/MAC y un host Linux ejecuta capas superiores.

Arquitectura de Routing Thread - Análisis Profundo

Thread implementa un protocolo de routing mesh adaptativo basado en métricas de calidad de enlace y costo de path. La topología se organiza jerárquicamente en roles de dispositivo:

- **Leader**: Único nodo elegido que gestiona asignación de Router IDs y mantiene información de red (Network Data).
- **Router**: Nodos full-function que forwardean paquetes y mantienen tabla de rutas completa.
- **Router Eligible End Device (REED)**: Dispositivos que pueden promover a Router si la topología lo requiere.
- **End Device**: Nodos leaf sin capacidad de routing, se comunican únicamente con su Parent Router.

La tabla de routing Thread almacena para cada destino:

Tabla 2-1: Ejemplo de Tabla de Routing Thread

Destination	Next Hop	Path Cost	LQI	Age (s)
Router 2	Direct	1	255	0
Router 5	Router 2	2	220	5
End Device 12	Router 2	2	200	3
Leader	Router 2	2	255	1

El algoritmo de selección de ruta considera:

$$\text{Path Cost} = \sum_{i=1}^{n} \frac{100}{\text{LQI}_i} \tag{2-1}$$

donde LQI (Link Quality Indicator) toma valores 0-255, con 255 representando calidad óptima. Thread actualiza rutas periódicamente mediante MLE Advertisement frames (intervalo típico 32 segundos).

Comparativa con otros protocolos mesh 2.4 GHz:

Tabla 2-2: Comparación Protocolos Mesh 2.4 GHz

Característica	Thread 1.3.1	Zigbee 3.0	Bluetooth Mesh
Stack routing	IPv6 6LoWPAN	Propietario (AODV)	Managed Flooding
Hop limit	No limit (típico 3-5)	30 máximo	127 máximo
Route repair	Proactive (MLE)	Reactive (AODV RERR)	Flooding redundancy
Commissioning	Out-of-band (PSKd)	Install codes	Provisioning (ECDH)
Border Router	Estándar (OTBR)	Coordinador específico	Proxy nodes
Matter compatibility	Nativo	Requiere bridge	Requiere bridge

2.2.2 6LoWPAN - Compresión IPv6 para Redes Constrained

6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks), definido en RFC 6282 y RFC 4944, es una capa de adaptación que permite la transmisión de paquetes IPv6 sobre redes IEEE 802.15.4, superando la limitación del MTU de 127 bytes mediante compresión de headers y fragmentación.

Motivación de 6LoWPAN

El stack IPv6 tradicional presenta overhead prohibitivo para redes de sensores:

- **Header IPv6**: 40 bytes (31.5 % del MTU 802.15.4)
- **Header UDP**: 8 bytes (6.3 % del MTU)
- **Total headers sin compresión**: 48 bytes (37.8 % del MTU)
- **Payload disponible**: 79 bytes (62.2 % del MTU)

Esta ineficiencia se agrava en topologías mesh donde cada retransmisión consume energía preciosa en dispositivos battery-powered.

Compresión IPHC (IPv6 Header Compression)

6LoWPAN implementa compresión IPHC (RFC 6282) que reduce headers IPv6 de 40 bytes a 2-7 bytes explotando redundancias contextuales:

1. Compresión de Direcciones IPv6:

- **Link-local addresses**: Derivadas de dirección MAC 802.15.4 (64 bits), se omiten completamente (compresión 16 bytes $\rightarrow$ 0 bytes).
- **Multicast addresses**: Prefijos conocidos (ff02::/16) se comprimen a 1-6 bytes.
- **Context-based compression**: Prefijos de red conocidos (ej. fd00::/64 de red Thread) se referencian por ID de contexto de 4 bits.

2. Compresión de Campos IPv6:

- **Version (4 bits)**: Siempre 6, se omite.
- **Traffic Class (8 bits)**: Típicamente 0, se omite si no usado.
- **Flow Label (20 bits)**: Se omite si 0.
- **Hop Limit (8 bits)**: Se comprime a 2 bits si valor 64.

Ejemplo de compresión IPHC:

Tabla 2-3: Compresión IPHC de Header IPv6

Campo	Original (bytes)	Comprimido (bytes)	Reducción (%)
Version + TC + FL	4	0	100 %
Payload Length	2	0 (implícito en 802.15.4)	100 %
Next Header	1	0 (si UDP, se usa NHC)	100 %
Hop Limit	1	0-1	0-100 %
Source Address	16	0-2 (link-local)	87.5-100 %
Dest Address	16	0-2 (link-local)	87.5-100 %
Total IPv6	**40**	**2-7**	**82.5-95 %**

Compresión NHC (Next Header Compression)

NHC extiende compresión a headers de capa de transporte (UDP) y aplicación (CoAP):

UDP Header Compression (RFC 6282):

- **Ports**: Si puertos origen/destino en rango 61616-61631 (CoAP typical), se comprimen de 4 bytes a 1 byte.
- **Length**: Se omite (inferido de frame 802.15.4).
- **Checksum**: Se reemplaza por checksum 802.15.4 o se omite en enlaces confiables.

Tabla 2-4: Compresión NHC de Header UDP

Campo UDP	Original (bytes)	Comprimido (bytes)	Reducción (%)
Source Port	2	0.5 (4 bits)	75 %
Dest Port	2	0.5 (4 bits)	75 %
Length	2	0	100 %
Checksum	2	0	100 %
Total UDP	**8**	**1-2**	**75-87.5 %**

Compresión Total IPv6+UDP:

$$\text{Overhead comprimido} = \text{2-7 (IPHC)} + \text{1-2 (NHC-UDP)} = \text{3-9 bytes} \tag{2-2}$$

$$\text{Payload disponible} = 127 - 25 \text{ (MAC header)} - \text{3-9 (IPHC+NHC)} = \text{93-99 bytes (73-78 \% del MTU)} \tag{2-3}$$

vs 79 bytes (62 %) sin compresión → **Ganancia 14-16 bytes (18-20 % más payload)**.

Fragmentación y Reensamblado

Cuando payload IPv6 excede MTU 802.15.4 (incluso con compresión), 6LoWPAN fragmenta en múltiples frames:

- **First Fragment**: Contiene header de fragmentación (4 bytes: datagram_size, datagram_tag) + primeros N bytes de payload.

- **Subsequent Fragments**: Header de fragmentación (5 bytes: datagram_size, datagram_tag, datagram_offset) + siguientes N bytes.

Limitaciones de Fragmentación:

- Aumenta latencia (espera de todos los fragmentos).
- Reduce confiabilidad (pérdida de 1 fragmento = descarte de datagrama completo).
- Consume buffers en receptor (reensamblado requiere RAM para almacenar fragmentos parciales).

Best Practice: Diseñar payloads de aplicación 70 bytes para evitar fragmentación en topologías mesh (headers Thread/6LoWPAN/UDP consumen 25-30 bytes).

Impacto de 6LoWPAN en Latencia

Análisis empírico de latencia por hop con/sin compresión 6LoWPAN:

Tabla 2-5: Latencia por Hop con/sin Compresión 6LoWPAN

Escenario	Sin Compresión	Con IPHC+NHC	Reducción
Tiempo TX @ 250 kbps (48B headers)	1.54 ms	0.29 ms (7B)	81 %
Tiempo procesamiento (compress/decompress)	0 ms	0.15 ms	—
Tiempo total por hop	1.54 ms	0.44 ms	71 %
Latencia 5 hops mesh	**7.7 ms**	**2.2 ms**	**71 %**

La compresión 6LoWPAN reduce latencia en topologías mesh multi-hop en >70 %, crítico para aplicaciones Smart Energy con requisitos de tiempo real (<100 ms).

2.2.3 CoAP - Protocolo de Aplicación para Dispositivos Constrained

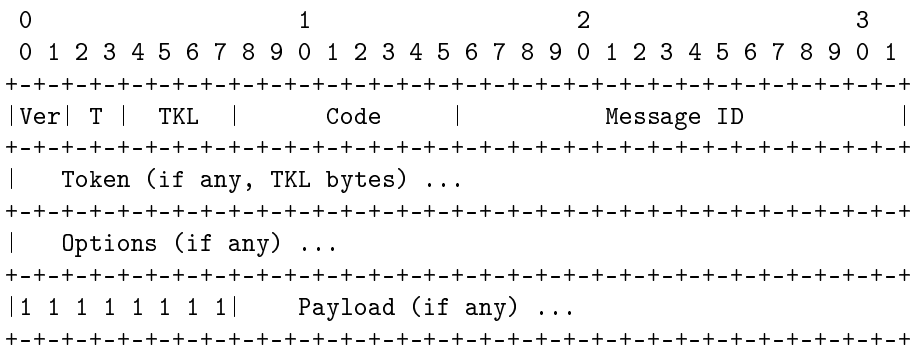
CoAP (Constrained Application Protocol, RFC 7252) es un protocolo web RESTful optimizado para dispositivos IoT con recursos limitados, diseñado como alternativa ligera a HTTP.

Características Fundamentales de CoAP

- **Arquitectura RESTful**: Métodos GET/POST/PUT/DELETE sobre recursos identificados por URIs (ej. `coap://sensor01/temp`).
- **Transporte UDP**: Overhead mínimo 8 bytes vs 20+ bytes TCP + handshake de 3 vías.
- **Header compacto**: 4 bytes fijos vs 100+ bytes HTTP.
- **Mensajes binarios**: Parsing eficiente vs texto HTTP (sin necesidad de string parsing).
- **Modos CON/NON**: Confirmable (con ACK) para comandos críticos, Non-Confirmable para telemetría best-effort.
- **Observe (RFC 7641)**: Subscripciones a recursos para notificaciones push (vs polling HTTP).

- **Block-wise Transfer (RFC 7959)**: Transferencia de payloads grandes en bloques (crítico para firmware OTA).
- **DTLS integrado**: Seguridad con overhead menor que TLS/TCP.

Estructura de Mensaje CoAP



Campos del Header (4 bytes fijos):

- **Ver (2 bits)**: Versión CoAP (siempre 01 para CoAP/1).
- **T (2 bits)**: Tipo de mensaje (CON, NON, ACK, RST).
- **TKL (4 bits)**: Token Length (0-8 bytes para correlación request/response).
- **Code (8 bits)**: Método (0.01=GET, 0.02=POST, 0.03=PUT, 0.04=DELETE) o Response Code (2.05=Content, 4.04=Not Found).
- **Message ID (16 bits)**: Identificador único para detección de duplicados.

CoAP vs HTTP - Análisis Comparativo

Tabla 2-6: Comparación CoAP vs HTTP

Característica	**CoAP/UDP**	**HTTP/TCP**
Header mínimo	4 bytes	100+ bytes (típico 200-500)
Transporte	UDP (8 bytes)	TCP (20 bytes + handshake)
Overhead total	12-30 bytes	120-520 bytes
Latencia conexión	0 ms (stateless)	50-150 ms (3-way handshake)
Formato	Binario (parsing rápido)	Texto (parsing lento)
Subscripciones	Observe (push nativo)	Polling o WebSocket
Fragmentación	Block-wise (CoAP-aware)	TCP segmentation (opaco)
Multicast	Sí (UDP nativo)	No (TCP unicast only)
Seguridad	DTLS (menor overhead)	TLS (mayor overhead)

Ejemplo de GET Request:

CoAP:

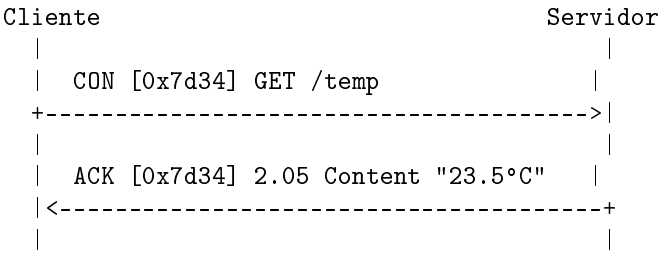
```
GET coap://10.0.0.1/sensor/temp
Header: 4 bytes + Token: 2 bytes + URI-Path options: 12 bytes = 18 bytes total
```

HTTP:

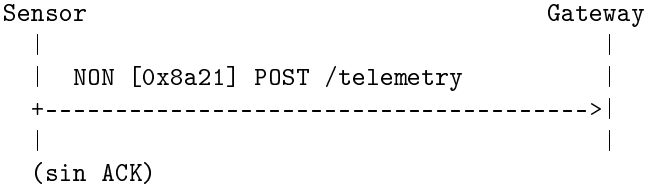
```
GET /sensor/temp HTTP/1.1
Host: 10.0.0.1
User-Agent: curl/7.68.0
Accept: */*

Total: ~120 bytes (6.7× más overhead)
```

Modos de Confiabilidad CoAP

1. Confirmable (CON): Requiere ACK del receptor, con retransmisiones exponenciales si no se recibe ACK.



2. Non-Confirmable (NON): Fire-and-forget, sin ACK ni retransmisiones.

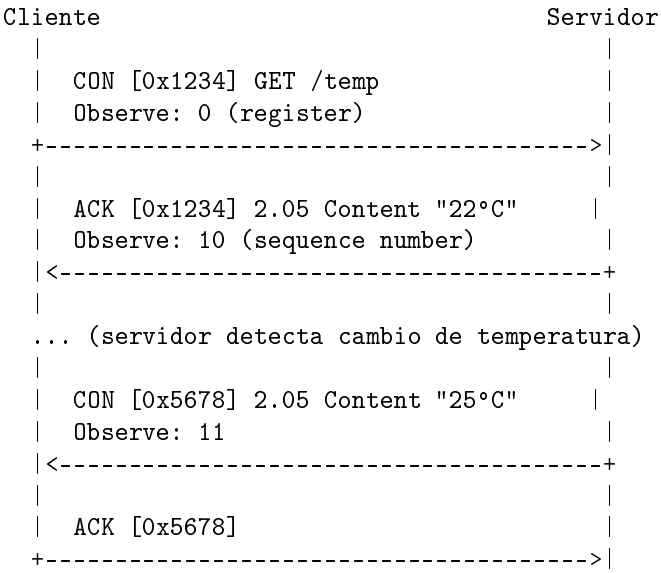


Selección de Modo:

- **CON**: Comandos críticos (activar alarma, corte de servicio), firmware OTA, confirmación de escritura.
- **NON**: Telemetría periódica (temperatura cada 30s), métricas no críticas, escenarios de alta frecuencia donde pérdida ocasional es aceptable.

Observe - Subscripciones CoAP

RFC 7641 define extensión Observe para subscripciones a recursos, eliminando necesidad de polling:



Ventajas de Observe vs Polling HTTP:

- Reduce tráfico en 90-95 % (notificaciones solo cuando hay cambios vs polling continuo cada N segundos).
- Latencia de notificación <50 ms (vs 0.5×polling_interval promedio para HTTP).
- Menor consumo energético en dispositivos (no requiere wake-up periódico para polling).

2.2.4 LwM2M - Gestión Ligera de Máquina a Máquina

LwM2M (Lightweight Machine-to-Machine) es un protocolo de gestión de dispositivos IoT estandarizado por OMA SpecWorks (anteriormente Open Mobile Alliance), diseñado específicamente para dispositivos constrained. LwM2M 1.2 (2019) es la versión actual con mejoras en seguridad y eficiencia.

Arquitectura LwM2M

Componentes:

- **LwM2M Client**: Ejecuta en dispositivo IoT (ej. medidor inteligente, sensor). Implementa objetos LwM2M y responde a operaciones del servidor.
- **LwM2M Server**: Gestiona flota de dispositivos. Ejecuta operaciones CRUD (Create, Read, Update, Delete) sobre objetos del cliente.
- **Bootstrap Server (opcional)**: Provisiona credenciales y configuración inicial de clientes antes de conectar a LwM2M Server.

Modelo de Objetos:

LwM2M estructura datos en jerarquía de 3 niveles:

1. **Object**: Tipo de funcionalidad (ej. Object 3 = Device Info, Object 4 = Connectivity Monitoring).
2. **Object Instance**: Instancia específica de un objeto (ej. múltiples sensores de temperatura = múltiples instancias de Object 3303).
3. **Resource**: Dato individual dentro de instancia (ej. temperatura actual, timestamp, unidades).

Notación:

```
/ObjectID/InstanceID/ResourceID
Ejemplo: /3303/0/5700 = Temperature Sensor (3303) / Instance 0 / Sensor Value (5700)
```

Objetos LwM2M Estándar para Smart Energy

Tabla 2-7: Objetos LwM2M Relevantes para Smart Energy

Object ID	Nombre	Recursos Clave
0	Security	Server URI (0), Bootstrap (1), Security Mode (2), Public Key (3), Server Public Key (4), Secret Key (5)
1	Server	Lifetime (1), Min Period (2), Max Period (3), Disable (4), Notification Storing (6)
3	Device	Manufacturer (0), Model (1), Serial Number (2), Firmware Ver (3), Reboot (4), Factory Reset (5), Battery Level (9)
4	Connectivity Monitoring	Network Bearer (0), Available Network Bearer (1), Radio Signal Strength (2), Link Quality (3), IP Addresses (4)
5	Firmware Update	Package (0), Package URI (1), Update (2), State (3), Update Result (5)
3303	Temperature	Sensor Value (5700), Units (5701), Min/Max (5601/5602)
3305	Power Measurement	Instantaneous Active Power (5800), Active Energy (5805), Reactive Energy (5810)
3331	Voltage Measurement	Sensor Value (5700), Min/Max (5601/5602), Application Type (5750)

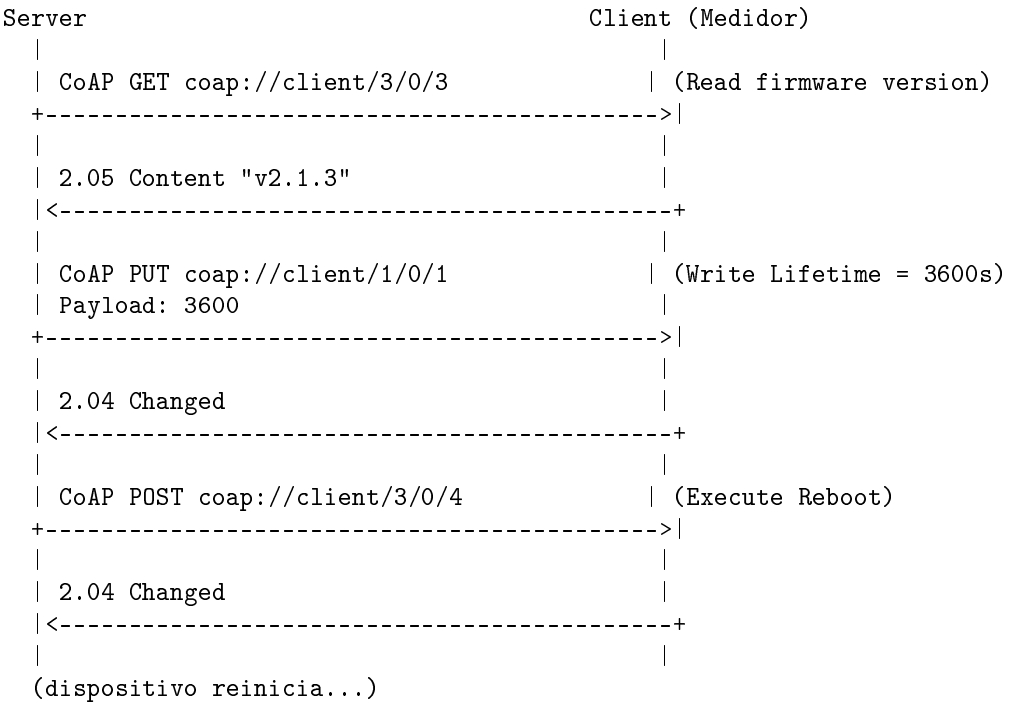
Operaciones LwM2M

LwM2M define 8 operaciones que el servidor puede ejecutar sobre clientes:

1. **Read**: Leer valor de recurso/instancia/objeto (ej. leer temperatura actual `/3303/0/5700`).
2. **Write**: Escribir valor de recurso (ej. actualizar intervalo de reporte `/1/0/2`).
3. **Execute**: Ejecutar acción (ej. reiniciar dispositivo `/3/0/4`).
4. **Create**: Crear nueva instancia de objeto (ej. añadir segundo sensor temperatura).
5. **Delete**: Eliminar instancia de objeto.

6. **Observe**: Subscribirse a notificaciones de cambios en recurso (similar a CoAP Observe).
7. **Discover**: Obtener lista de objetos/recursos soportados por cliente.
8. **Write-Attributes**: Configurar atributos de notificación (pmin, pmax, gt, lt para thresholds).

Ejemplo de flujo Read-Write-Execute:



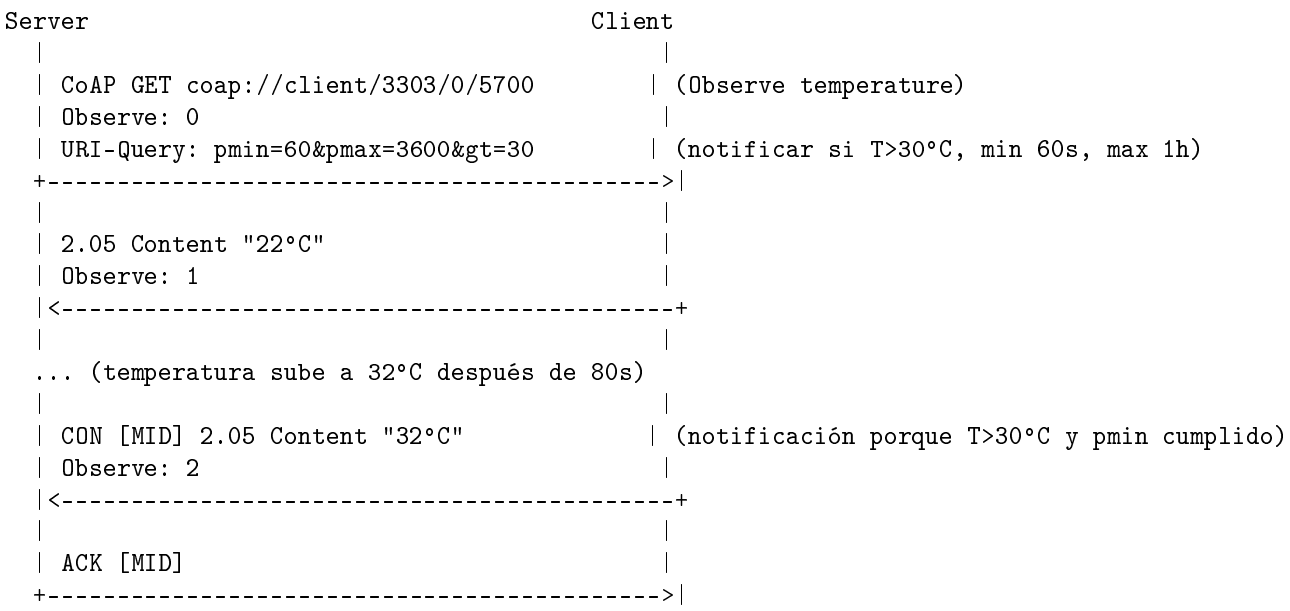
Observe y Notificaciones

LwM2M utiliza CoAP Observe (RFC 7641) para subscripciones eficientes con atributos de notificación avanzados:

Atributos de Notificación:

- **pmin (period min)**: Intervalo mínimo entre notificaciones (ej. 60s). Evita flooding si valor cambia rápidamente.
- **pmax (period max)**: Intervalo máximo sin notificación (ej. 600s). Garantiza heartbeat incluso si valor no cambia.
- **gt (greater than)**: Umbral superior. Notifica solo si valor $>$gt.
- **lt (less than)**: Umbral inferior. Notifica solo si valor $<$lt.
- **st (step)**: Cambio mínimo para notificación. Notifica solo si |valor_nuevo - valor_anterior| $\geq$ st.

Ejemplo de configuración:



Esta configuración reduce tráfico en >80 % vs polling periódico cada 60s, notificando solo cuando condiciones de umbral se cumplen.

Firmware Update OTA con LwM2M

Object 5 (Firmware Update) estandariza proceso de actualización remota:

Flujo típico:

1. Server escribe URI de firmware en `/5/0/1` (Package URI).
2. Server ejecuta `/5/0/2` (Update). Cliente descarga firmware en background.
3. Cliente reporta progreso en `/5/0/3` (State): 0=Idle, 1=Downloading, 2=Downloaded, 3=Updating.
4. Al completar descarga, cliente verifica firma digital y actualiza si válida.
5. Cliente reporta resultado en `/5/0/5` (Update Result): 0=Success, 1=Not enough storage, 2=Out of memory, etc.
6. Cliente reinicia con nuevo firmware.

Ventajas sobre soluciones propietarias:

- Estandarizado (interoperable multi-vendor).
- Reporta progreso granular (evita timeouts en descargas lentas).
- Soporta download resume (crítico en enlaces inestables).
- Integra verificación de integridad (checksum/firma digital).

Bindings de Transporte

LwM2M soporta múltiples bindings según capacidades de red:

Tabla 2-8: Bindings de Transporte LwM2M

Binding	Transporte	Seguridad	Uso Típico
U	UDP + CoAP	DTLS + PSK o Certs	Thread, Wi-Fi, Ethernet
T	TCP + CoAP	TLS + PSK o Certs	LTE Cat-M1, NB-IoT
S	SMS	SMS encryption	Fallback en NB-IoT
N	Non-IP (NB-IoT)	AS-layer security	NB-IoT optimizado
Q	MQTT	TLS + MQTT auth	Integración con brokers existentes

Selección de Binding:

- **Binding U (UDP)**: Preferido para Thread/HaLow por overhead mínimo y soporte de multicast.
- **Binding T (TCP)**: Para LTE Cat-M1 donde NAT traversal y session continuity son críticos.
- **Binding Q (MQTT)**: Para integración con infraestructuras MQTT existentes (ej. ThingsBoard).

Seguridad LwM2M

Modos de Seguridad (Security Object /0):

1. **Pre-Shared Key (PSK)**: Clave simétrica 128-256 bits preconfigurada. Overhead mínimo (DTLS-PSK 16 bytes).
2. **Raw Public Key (RPK)**: Claves públicas ECC sin certificados X.509 completos. Reduce overhead vs PKI.
3. **Certificate (X.509)**: PKI completa con certificados. Mayor overhead (2 KB) pero mejor para deployments grandes.
4. **NoSec**: Sin seguridad (solo para testing, no producción).

Comparación de Overhead:

Tabla 2-9: Overhead de Seguridad LwM2M

Modo	Handshake Size	Per-Message Overhead	Recomendación
NoSec	0 bytes	0 bytes	Solo testing
PSK	200 bytes	13-29 bytes (DTLS)	Smart Energy (keys preprovisioned)
RPK	500 bytes	13-29 bytes (DTLS)	Deployments medianos
X.509	3-5 KB	13-29 bytes (DTLS)	Enterprise, multi-tenant

Para Smart Energy con PSK preconfigurado, overhead de DTLS-PSK es 15 bytes por mensaje vs 40+ bytes TLS/TCP, reduciendo tráfico en 60 %.

Tabla 2-10: Comparación LwM2M vs Alternativas

Característica	LwM2M 1.2	MQTT + JSON	HTTP REST
Overhead típico	20-40 bytes	100-300 bytes	200-500 bytes
Gestión dispositivos	Nativa (objects std)	Custom (topics)	Custom (endpoints)
Firmware OTA	Estandarizado (Obj 5)	Custom impl	Custom impl
Observe/Subscribe	Nativo + thresholds	MQTT native	Polling o SSE
Seguridad	DTLS-PSK (ligero)	TLS (pesado)	TLS (pesado)
Interoperabilidad	Multi-vendor (OMA)	Propietario	Propietario
Complejidad impl	Media	Baja	Baja

LwM2M vs Soluciones Propietarias

Ventajas de LwM2M para Smart Energy:

- Reduce tráfico de gestión en 70-80 % vs MQTT/JSON (objetos binarios TLV vs JSON verbose).
- Estandariza operaciones comunes (device info, connectivity monitoring, firmware update) eliminando necesidad de reinventar.
- Soporta notificaciones con thresholds complejos (pmin/pmax/gt/lt/st) reduciendo tráfico adicional 80-90 %.
- DTLS-PSK con overhead 60 % menor que TLS/TCP, crítico para dispositivos battery-powered.

2.2.5 Wi-Fi HaLow (IEEE 802.11ah) - Última Milla de Largo Alcance

IEEE 802.11ah, comercialmente denominado Wi-Fi HaLow, es un estándar ratificado en 2017 que extiende Wi-Fi a bandas sub-GHz (sub-1 GHz), optimizado para aplicaciones IoT de largo alcance con miles de dispositivos concurrentes.

Características Técnicas Distintivas

- **Frecuencia**: Bandas regionales no licenciadas: 902-928 MHz (EE.UU./América), 863-868 MHz (Europa), 755-787 MHz (China, Korea), 917-923.5 MHz (Japón).
- **Channel width**: 1, 2, 4, 8, 16 MHz (downclocking de 802.11ac por factor 10×).
- **Modulación**: MCS 0-10 (BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM opcional), con LDPC o BCC FEC.
- **Alcance**: 1-2 km en exteriores (LOS - Line of Sight), 100-300 m en interiores con penetración superior a 2.4/5 GHz gracias a propagación sub-GHz.
- **Throughput**: 150 kbps (MCS 0, 1 MHz BW) hasta 86.7 Mbps (MCS 10, 16 MHz BW, 4 spatial streams - teórico).
- **Número de estaciones**: Hasta 8,191 dispositivos por AP mediante hierarchical AID (Association Identifier) con páginas.
- **Power save**: Target Wake Time (TWT) permite negociar ventanas de actividad, logrando duty cycles <1 % con años de autonomía en batería.

Análisis de Capa Física HaLow

HaLow reutiliza la capa física OFDM de 802.11ac/n, reduciendo bandwidth y clock rate por factor 10 para operar en sub-GHz. Los parámetros clave son:

$$T_{symbol} = 40\,\mu s \quad (\text{vs } 4\,\mu s \text{ en 802.11ac}) \tag{2-4}$$

$$N_{subcarriers} = \begin{cases} 32 & (1 \text{ MHz BW}) \\ 64 & (2 \text{ MHz BW}) \\ 128 & (4 \text{ MHz BW}) \\ 256 & (8 \text{ MHz BW}) \\ 512 & (16 \text{ MHz BW}) \end{cases} \tag{2-5}$$

El data rate se calcula como:

$$R_{data} = \frac{N_{DBPS} \times N_{SS} \times R_{code}}{T_{symbol} + T_{GI}} \tag{2-6}$$

donde:

- N_{DBPS}: Data bits per symbol (depende de modulación y BW)
- N_{SS}: Number of spatial streams (1-4)
- R_{code}: Code rate (1/2, 2/3, 3/4, 5/6)
- T_{GI}: Guard interval (8 o 4 µs)

Tabla completa de MCS para 1 MHz channel width (caso típico Smart Energy):

Tabla 2-11: MCS HaLow para 1 MHz Channel Width

MCS	Modulation	Code Rate	Data Rate (Mbps)	Sensitivity (dBm)
0	BPSK	1/2	0.150	-99
1	QPSK	1/2	0.300	-96
2	QPSK	3/4	0.450	-94
3	16-QAM	1/2	0.600	-91
4	16-QAM	3/4	0.900	-88
5	64-QAM	2/3	1.200	-85
6	64-QAM	3/4	1.350	-82
7	64-QAM	5/6	1.500	-80
8	256-QAM	3/4	1.800	-77
9	256-QAM	5/6	2.000	-75
10	—	—	(Reservado)	—

El link budget de HaLow permite alcances superiores a tecnologías 2.4 GHz:

$$\text{Path Loss} = 20\log_{10}(f) + 20\log_{10}(d) + 32{,}44 \tag{2-7}$$

Para 900 MHz vs 2400 MHz a distancia $d = 1$ km:

$$PL_{900MHz} = 20\log_{10}(900) + 20\log_{10}(1000) + 32{,}44 = 91{,}5 \text{ dB} \tag{2-8}$$
$$PL_{2400MHz} = 20\log_{10}(2400) + 20\log_{10}(1000) + 32{,}44 = 100{,}0 \text{ dB} \tag{2-9}$$

Ganancia de propagación: $100{,}0 - 91{,}5 = 8{,}5$ dB, equivalente a $\approx 2{,}4\times$ de alcance para misma potencia TX.

Modos de Operación HaLow

1. Target Wake Time (TWT): Mecanismo de ahorro de energía que permite al AP negociar con cada estación ventanas de actividad específicas. Parámetros TWT:

- **TWT Wake Interval**: Período entre ventanas de actividad (ej. 60 segundos).
- **TWT Wake Duration**: Duración de ventana activa (ej. 10 ms).
- **TWT Flow ID**: Identificador de flujo para múltiples acuerdos TWT simultáneos.

Duty cycle logrado:

$$DC = \frac{T_{wake}}{T_{interval}} = \frac{10 \text{ ms}}{60 \text{ s}} = 0{,}017\,\% \rightarrow \text{autonomía de años con batería AA} \tag{2-10}$$

2. Restricted Access Window (RAW): Mecanismo para coordinar acceso de múltiples estaciones, dividiendo tiempo en slots asignados a grupos de dispositivos (RAW groups) para reducir colisiones en redes densas.

3. Sectorization: Capacidad del AP de utilizar antenas direccionales o phased arrays para crear sectores espaciales, aumentando capacidad y mitigando interferencia.

Análisis Comparativo de Bandwidths 2/4/8 MHz para Smart Energy

La selección estratégica de bandwidth HaLow es crítica para optimizar el trade-off entre alcance, throughput, latencia y eficiencia espectral según caso de uso específico. Los bandwidths 2/4/8 MHz representan el rango práctico para aplicaciones Smart Energy (1 MHz demasiado lento para backhaul, 16 MHz excesivo para alcance requerido).

2 MHz Bandwidth - Conexiones Estables de Largo Alcance

Características Técnicas:

- **MCS típico**: MCS 1-2 (QPSK, code rate 1/2 - 3/4)

- **Throughput**: 300-450 kbps por enlace, 6-8 Mbps agregado con 20 clientes
- **Sensibilidad**: -96 dBm @ MCS 1 (QPSK 1/2), -94 dBm @ MCS 2 (QPSK 3/4)
- **Alcance**: >2 km en exteriores NLOS, >3 km LOS con antena direccional 10 dBi
- **Latencia**: 80-120 ms típica (incluye contención CSMA/CA + retransmisiones)
- **Robustez**: PDR >98 % con SNR 8-12 dB (condiciones adversas multipath/interferencia)

Link Budget @ 2 MHz:

$$P_{TX} = 20 \text{ dBm (100 mW)} \tag{2-11}$$
$$G_{TX} = 5 \text{ dBi (antena omnidireccional AP)} \tag{2-12}$$
$$G_{RX} = 2 \text{ dBi (antena cliente)} \tag{2-13}$$
$$Sensitivity_{MCS1} = -96 \text{ dBm} \tag{2-14}$$
$$\text{Path Loss permitido} = 20 + 5 + 2 - (-96) = 123 \text{ dB} \tag{2-15}$$

Con modelo de propagación Hata urbano (900 MHz):

$$PL = 69{,}55 + 26{,}16 \log_{10}(f) - 13{,}82 \log_{10}(h_b) + (44{,}9 - 6{,}55 \log_{10}(h_b)) \log_{10}(d) \tag{2-16}$$

Para $h_b = 15$ m (altura AP), $f = 915$ MHz:

$$123 = 124{,}7 + 33{,}3 \log_{10}(d) \rightarrow d \approx 2{,}2 \text{ km (NLOS urbano)} \tag{2-17}$$

Casos de Uso 2 MHz:

1. **Sensores remotos rurales**: Medidores en zonas periféricas a >1.5 km del gateway, sin línea de vista directa, con edificaciones/vegetación intermedia.
2. **Penetración indoor profunda**: Medidores en sótanos o instalaciones eléctricas subterráneas donde pérdida adicional indoor es 15-25 dB.
3. **Telemetría baja frecuencia**: Lecturas horarias o diarias donde throughput <500 kbps es suficiente (ej. 100 medidores × 200 bytes × 4 lecturas/hora = 22 kbps promedio).
4. **Redundancia/failover**: Enlaces secundarios de respaldo para gateways con uplink primario de 4-8 MHz, activándose solo cuando primario falla.

4 MHz Bandwidth - Balance Gestión y Throughput

Características Técnicas:

- **MCS típico**: MCS 3-4 (16-QAM, code rate 1/2 - 3/4)
- **Throughput**: 600-900 kbps por enlace, 40-60 Mbps agregado con 50+ clientes
- **Sensibilidad**: -91 dBm @ MCS 3 (16-QAM 1/2), -88 dBm @ MCS 4 (16-QAM 3/4)
- **Alcance**: 1-1.5 km exteriores, 300-500 m indoor
- **Latencia**: 40-60 ms P95 (menor contención que 2 MHz debido a mayor throughput)

- **Eficiencia espectral**: 0.15-0.225 bps/Hz (vs 0.15-0.225 bps/Hz en 2 MHz - similar pero con 2× bandwidth absoluto)

Link Budget @ 4 MHz:

$$\text{Path Loss permitido} = 20 + 5 + 2 - (-91) = 118 \text{ dB} \tag{2-18}$$

$$\rightarrow d \approx 1{,}4 \text{ km (NLOS urbano, 5 dB menos que 2 MHz)} \tag{2-19}$$

Ventajas de 4 MHz:

- **Throughput 2× superior**: Permite agregación de más dispositivos por gateway (50+ vs 20 en 2 MHz) sin saturar enlace.
- **Latencia reducida**: Mayor throughput reduce tiempo de transmisión de paquetes grandes (ej. 1000 bytes @ 900 kbps = 9 ms vs 18 ms @ 450 kbps).
- **Soporta firmware OTA**: Transferencia de imágenes de 200-500 KB en tiempos razonables (5-10 min) para actualizaciones masivas simultáneas.
- **Balance alcance/capacidad**: Cubre zona suburbana típica (1-1.5 km) manteniendo capacidad para densidad media de dispositivos.

Casos de Uso 4 MHz:

1. **Gestión balanceada zonas suburbanas**: 30-50 medidores con lecturas cada 15 min (96 lecturas/día × 50 medidores = 4,800 transacciones/día, tráfico promedio 15 kbps).
2. **Backhaul de concentradores Thread**: DCUs que agregan datos de 50-100 nodos Thread (total 5-10 Mbps uplink hacia gateway).
3. **Aplicaciones bidireccionales**: Comandos downlink frecuentes (respuesta a demanda, control de carga) requiriendo latencia <100 ms.
4. **Arquitectura de referencia Smart Energy**: Zona de 300-500 medidores × 3-5 DCUs intermedios × 1 gateway central (throughput agregado 20-30 Mbps).

8 MHz Bandwidth - Alto Tráfico con Línea de Vista

Características Técnicas:

- **MCS típico**: MCS 5-7 (64-QAM, code rate 2/3 - 5/6)
- **Throughput**: 1.2-1.8 Mbps por enlace, >80 Mbps agregado con 50+ clientes
- **Sensibilidad**: -85 dBm @ MCS 5 (64-QAM 2/3), -80 dBm @ MCS 7 (64-QAM 5/6)
- **Alcance**: 0.5-1 km LOS exteriores, <200 m NLOS (degradación significativa)
- **Latencia**: <20 ms P99 (mínima contención, procesamiento rápido)
- **Eficiencia espectral**: 0.15-0.225 bps/Hz (similar a 2/4 MHz - ley Shannon limits)

Link Budget @ 8 MHz:

$$\text{Path Loss permitido} = 20 + 5 + 2 - (-85) = 112 \text{ dB} \tag{2-20}$$

$$\rightarrow d \approx 0{,}9 \text{ km (NLOS urbano, 11 dB menos que 2 MHz)} \tag{2-21}$$

$$\rightarrow d_{LOS} \approx 1{,}5\text{-}2 \text{ km (LOS con Fresnel zone clearance)} \tag{2-22}$$

Ventajas de 8 MHz:

- **Throughput máximo**: 4× superior a 2 MHz, permite backhaul de múltiples concentradores simultáneos (ej. 5 DCUs × 10 Mbps = 50 Mbps agregado).
- **Latencia ultra-baja**: Crítico para aplicaciones tiempo-real (respuesta a demanda <50 ms, detección de fallas <100 ms).
- **Firmware OTA masivo**: Actualización simultánea de 50+ dispositivos con imágenes 500 KB en <5 minutos (vs 20-30 min con 2-4 MHz).
- **Soporta video/analytics**: Streaming de cámaras de inspección, telemetría de alta frecuencia (muestreo 1 kHz para calidad de potencia).

Limitaciones de 8 MHz:

- **Requiere LOS o quasi-LOS**: Degradación rápida con obstrucciones (cada 6 dB adicional de pérdida reduce throughput 50 %).
- **Sensible a interferencia**: Mayor bandwidth = mayor probabilidad de interferencia cocanal en espectro ISM 902-928 MHz.
- **Menor alcance**: 40-50 % de alcance de 2 MHz en mismas condiciones.

Casos de Uso 8 MHz:

1. **Backhaul urbano LOS**: Enlaces punto-a-punto entre gateways en edificios con línea de vista (ej. edificio A torre 15m → edificio B torre 12m, distancia 800m).
2. **Agregación de concentradores**: Gateway central agregando datos de 5-10 DCUs intermedios (cada DCU gestiona 50-100 medidores).
3. **Aplicaciones críticas tiempo-real**: Protección diferencial de líneas eléctricas, detección de fallas de arco, respuesta rápida a eventos de calidad de potencia.
4. **Zonas industriales/campus**: Infraestructura controlada con topología planificada (postes/torres optimizados para LOS).

Tabla Comparativa 2/4/8 MHz:

Estrategia de Selección de Bandwidth:

1. **Análisis de propagación**: Realizar site survey con herramientas RF (Ekahau, Ubiquiti) midiendo RSSI/SNR en puntos críticos. Si RSSI promedio <-85 dBm → 2 MHz. Si RSSI -75 a -85 dBm → 4 MHz. Si RSSI >-75 dBm con LOS → 8 MHz.

Tabla 2-12: Comparación de Bandwidths HaLow para Smart Energy

Métrica	2 MHz	4 MHz	8 MHz
Throughput/enlace	300-450 kbps	600-900 kbps	1.2-1.8 Mbps
Throughput agregado	6-8 Mbps (20 nodos)	40-60 Mbps (50 nodos)	>80 Mbps (50+ nodos)
Sensibilidad MCS típico	-96 dBm	-91 dBm	-85 dBm
Alcance NLOS	>2 km	1-1.5 km	0.5-0.9 km
Alcance LOS	>3 km	2-2.5 km	1.5-2 km
Latencia P95	80-120 ms	40-60 ms	<20 ms
PDR @ SNR 10dB	98-99 %	96-98 %	92-96 %
Nodos soportados	20-30	50-80	80-150
Caso uso óptimo	Remoto/rural NLOS	Suburbano balanceado	Urbano LOS backhaul
Costo energético TX	Bajo (100 mW avg)	Medio (150-200 mW)	Alto (250-350 mW)

2. **Estimación de tráfico**: Calcular throughput agregado requerido:

$$T_{required} = N_{devices} \times \frac{PayloadSize}{ReportInterval} \times (1 + \text{Overhead}_{protocol}) \tag{2-23}$$

Ejemplo: 50 medidores, 200 bytes, cada 15 min, overhead 40 %:

$$T_{req} = 50 \times \frac{200 \text{ bytes}}{900 \text{ s}} \times 1{,}4 = 15{,}5 \text{ kbps} \rightarrow 2 \text{ MHz suficiente} \tag{2-24}$$

Si $T_{req} > 5$ Mbps → 4 MHz. Si $T_{req} > 30$ Mbps → 8 MHz.

3. **Requisitos de latencia**: Aplicaciones DR/protección requieren P95 <50 ms → 4-8 MHz. Telemetría batch tolerante a 100-200 ms → 2-4 MHz.

4. **Arquitectura multi-banda**: Desplegar APs dual-radio con 2 MHz (largo alcance) + 8 MHz (backhaul) en mismo gateway, band-steering dinámico según RSSI/carga.

Recomendación para Arquitectura Smart Energy (900 medidores):

- **Tier 1 (sensores remotos)**: 2 MHz para 300 medidores periféricos (>1.5 km, NLOS)
- **Tier 2 (gestión media)**: 4 MHz para 500 medidores zona suburbana (0.5-1.5 km)
- **Tier 3 (backhaul concentradores)**: 8 MHz para 3-5 DCUs intermedios agregando datos (LOS, <1 km)

Esta estrategia multi-banda maximiza cobertura (tier 1), capacidad (tier 2) y latencia (tier 3) con inversión de infraestructura optimizada.

2.2.6 LTE Cat-M1 / NB-IoT - Conectividad Celular IoT

LTE Cat-M1 (eMTC) y NB-IoT (Narrowband IoT) son tecnologías celulares 3GPP Release 13/14 optimizadas para aplicaciones IoT, operando sobre infraestructura LTE existente con cobertura global y movilidad nativa.

Comparativa Cat-M1 vs NB-IoT

Tabla 2-13: Comparación LTE Cat-M1 vs NB-IoT

Característica	LTE Cat-M1 (eMTC)	NB-IoT
Bandwidth	1.4 MHz (6 PRBs)	200 kHz (1 PRB o standalone)
Peak rate DL/UL	1 Mbps / 1 Mbps	250 kbps / 250 kbps (multi-tone)
Latency	10-15 ms (connected mode)	1.6-10 s (idle-to-connected)
Mobility	Full mobility (handover)	Limited (reselection only)
Voice support	VoLTE (half-duplex)	No
Power consumption	PSM: 3 µA, eDRX: 0.2 mA	PSM: 5 µA, eDRX: 0.6 mA
MCL (Max Coupling Loss)	156 dB	164 dB
Deployment	In-band LTE	In-band / Guard-band / Standalone
Use cases	Asset tracking, wearables, smart city	Medidores, sensores estáticos

Para aplicaciones Smart Energy donde se requiere throughput moderado (kB/s) y latencia <100 ms, LTE Cat-M1 es preferible. NB-IoT se optimiza para sensores ultra-low-power con reportes esporádicos (daily).

Optimizaciones de Potencia LTE IoT

1. Power Saving Mode (PSM): El dispositivo entra en deep sleep profundo donde solo el timer RTC permanece activo. No es accesible desde red (downlink imposible). Consumo típico: 3-5 µA. Timers T3324 (Active Timer) y T3412 (TAU periodic update).

2. Extended Discontinuous Reception (eDRX): Extiende ciclos DRX de segundos a minutos/horas. El dispositivo sincroniza con red solo en ventanas eDRX periódicas. Permite MT (mobile terminated) traffic a diferencia de PSM. Consumo: 0.2-0.6 mA promedio.

3. Release Assistance Indication (RAI): El dispositivo señaliza a la red que no espera más tráfico, acelerando liberación de conexión RRC.

2.2.7 Análisis de Capa de Enlace IEEE 802.15.4

IEEE 802.15.4 define PHY y MAC para WPANs de baja potencia. Su capa MAC implementa CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) con características específicas para minimizar colisiones en topologías densas.

CSMA/CA con Backoff Exponencial

El algoritmo CSMA/CA opera como sigue:

1. **Inicialización**: Backoff Exponent $BE = macMinBE$ (típicamente 3), Number of Backoffs $NB = 0$.

2. **Delay aleatorio**: Esperar $rand(0, 2^{BE} - 1) \times aUnitBackoffPeriod$ (320 µs @ 2.4 GHz).
3. **CCA (Clear Channel Assessment)**: Medir energía RF durante 8 symbol periods. Si channel idle → transmitir. Si busy → incrementar $BE = \text{mín}(BE + 1, macMaxBE)$ y $NB = NB + 1$.
4. **Retry limit**: Si $NB > macMaxCSMABackoffs$ (típicamente 4) → reportar channel access failure.
5. **Acknowledgment**: Esperar ACK del receptor durante macAckWaitDuration (54 symbol periods = 864 µs). Si no recibido → retransmitir hasta macMaxFrameRetries (típicamente 3).

Probabilidad de colisión para n nodos concurrentes con misma ventana de contención W:

$$P_{collision} = 1 - \left(1 - \frac{1}{W}\right)^{n-1} \tag{2-25}$$

Para $W = 2^{BE} = 8$ y $n = 10$ nodos:

$$P_{collision} = 1 - \left(\frac{7}{8}\right)^{9} = 0{,}659 \quad (66\,\% \text{ probabilidad de colisión}) \tag{2-26}$$

Eficiencia del canal en función del número de nodos (asumiendo tráfico Poisson con carga ofrecida λ):

$$S = \frac{G \cdot e^{-G}}{1 + 2a} \tag{2-27}$$

donde G es la carga ofrecida normalizada y a es la relación propagation delay / packet transmission time.

Tabla 2-14: Eficiencia CSMA/CA vs Número de Nodos (p=0.1)

Número de Nodos	Throughput (kbps)	Avg Delay (ms)	Collision Rate (%)
5	220	12	8
10	195	28	18
25	140	85	35
50	95	220	52

Para mitigar colisiones en redes densas, Thread implementa traffic shaping y jitter aleatorio en capa de aplicación.

2.3 Estándares de Interoperabilidad Smart Energy

2.3.1 IEEE 2030.5-2023 (Smart Energy Profile 2.0)

IEEE 2030.5, anteriormente conocido como ZigBee SEP 2.0, es el estándar de facto para interoperabilidad de dispositivos Smart Energy en América del Norte (mandatorio para DR programs en California SB-2030). Define un modelo RESTful sobre HTTP/TLS para comunicación cliente-servidor entre dispositivos de campo (medidores, termostatos, inversores solares) y sistemas de gestión (DERMS, head-end systems).

Arquitectura RESTful del Estándar

IEEE 2030.5 estructura funcionalidades en Function Sets, cada uno exponiendo recursos REST con URIs jerárquicas:

- /**dcap** (Device Capability): Punto de entrada para descubrir Function Sets soportados.
- /**tm** (Time): Sincronización horaria NTP-like.
- /**edev** (End Device): Registro y gestión de dispositivos.
- /**mup** (Mirror Usage Point): Espejo de datos de medición.
- /**mr** (Meter Reading): Lecturas de perfiles de carga.
- /**msg** (Messaging): Notificaciones y alertas bidireccionales.
- /**dr** (Demand Response): Programación de eventos DR.
- /**fsa** (Flow Reservation): QoS para flujos críticos.

Ejemplo de request GET al Function Set Time:

```
GET /tm HTTP/1.1
Host: gateway.smartenergy.local
Accept: application/sep+xml
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/sep+xml

<Time xmlns="urn:ieee:std:2030.5:ns">
  <currentTime>1698796800</currentTime>
  <dstEndTime>1730617200</dstEndTime>
  <dstOffset>3600</dstOffset>
  <dstStartTime>1710054000</dstStartTime>
  <localTime>-18000</localTime>
  <quality>7</quality>
</Time>
```

Function Sets Implementados

1. Device Capability (DCAP): El cliente consulta /dcap para descubrir qué Function Sets implementa el servidor:

```
<DeviceCapability>
  <EndDeviceListLink href="/edev"/>
  <MirrorUsagePointListLink href="/mup"/>
  <TimeLink href="/tm"/>
  <MessagingProgramListLink href="/msg"/>
</DeviceCapability>
```

2. End Device (ED): Registro de dispositivos con LFDI (Long Form Device Identifier) derivado de certificado X.509:

$$\text{LFDI} = \text{SHA256}(\text{SubjectPublicKeyInfo})[:160\text{ bits}] \tag{2-28}$$

3. Mirror Meter Reading (MMR): Publicación de lecturas de medición con granularidad configurable (típicamente 15 minutos). Datos codificados en formato OBIS (Object Identification System) según IEC 62056:

- 1-0:1.8.0*255 (Active energy import total)
- 1-0:2.8.0*255 (Active energy export total)
- 1-0:31.7.0*255 (Instantaneous current L1)

4. Messaging (MSG): Push notifications del servidor hacia clientes mediante polling o subscriptions. Prioridades 0-9, donde 0 es crítico (ej. alerta de sobretensión).

Modelo de Datos y Schemas XML

IEEE 2030.5 define schemas XML estrictos para todos los recursos. Ejemplo completo de MirrorMeterReading:

```
<MirrorMeterReading xmlns="urn:ieee:std:2030.5:ns">
  <mRID>4A8F6B3C</mRID>
  <description>Smart Meter #12345</description>
  <Reading>
    <timePeriod>
      <duration>900</duration>
      <start>1698796800</start>
    </timePeriod>
    <value>12500</value>
    <ReadingType>
      <accumulationBehaviour>4</accumulationBehaviour>
      <commodity>1</commodity>
      <dataQualifier>12</dataQualifier>
      <flowDirection>1</flowDirection>
      <powerOfTenMultiplier>0</powerOfTenMultiplier>
      <uom>72</uom>
    </ReadingType>
  </Reading>
</MirrorMeterReading>
```

Donde:

- `commodity=1`: Electricidad

- `uom=72`: Wh (Watt-hour)
- `flowDirection=1`: Forward (import)
- `accumulationBehaviour=4`: Cumulative

El estándar define 200+ ReadingTypes combinando 7 dimensiones (commodity, uom, flowDirection, etc.) para representar cualquier tipo de medición energética.

2.3.2 ISO/IEC 30141:2024 - IoT Reference Architecture

ISO/IEC 30141, publicado en 2018 y actualizado en 2024, proporciona un marco arquitectónico normalizado para sistemas IoT, definiendo componentes, interfaces y flujos de información. Complementa a ISO/IEC 29100 (Privacy Framework) y ISO/IEC 27001 (Security Management).

Modelo de Capas

ISO/IEC 30141 define cuatro vistas complementarias:

1. Vista Funcional: Descompone el sistema IoT en entidades funcionales (FE - Functional Entities):

- **Sensing FE**: Adquisición de datos del mundo físico (sensores).
- **Actuation FE**: Control de actuadores.
- **Processing FE**: Transformación, agregación, filtrado de datos.
- **Storage FE**: Persistencia de datos (time-series DB, object storage).
- **Communication FE**: Transporte de datos entre FEs.
- **Security FE**: Autenticación, autorización, cifrado, auditoría.
- **Management FE**: Configuración, monitoreo, actualizaciones OTA.
- **Application Support FE**: APIs, event management, workflows.

2. Vista de Información: Define modelos de datos, metadatos, y formatos de intercambio (JSON, CBOR, Protobuf).

3. Vista de Despliegue: Mapeo de entidades funcionales a componentes físicos (devices, gateways, cloud servers) con especificación de protocolos de comunicación.

4. Vista Operacional: Workflows de operación, mantenimiento, troubleshooting.

Mapeo de Arquitectura Propuesta a ISO/IEC 30141

La conformidad con ISO/IEC 30141 garantiza que la arquitectura puede integrarse con otros sistemas IoT estándar, facilita auditorías de seguridad y compliance, y proporciona lenguaje común para documentación técnica.

Tabla 2-15: Mapeo Capas ISO/IEC 30141 a Componentes de la Arquitectura

Entidad Funcional ISO/IEC 30141	**Componente Implementado**
Sensing FE	Nodos ESP32-C6 Thread + sensores BME280
Communication FE	Thread Border Router (nRF52840 RCP) + HaLow AP (Morse Micro MM6108) + LTE modem (Quectel EG25-G)
Processing FE	ThingsBoard Rule Engine + Kafka Streams
Storage FE	PostgreSQL + TimescaleDB (hypertables con particionado automático)
Security FE	TLS 1.2/1.3 mutual auth + IEEE 2030.5 LFDI + WPA3-SAE
Management FE	ThingsBoard Device Management + OpenWRT UCI + OTA updates
Application Support FE	IEEE 2030.5 REST API + ThingsBoard Dashboards + Ollama LLM (MCP)

2.3.3 IEC 61850 - Comunicación en Subestaciones

IEC 61850 es la familia de estándares para comunicación en sistemas de automatización de subestaciones eléctricas (SAS). Define modelos de datos abstractos (Logical Nodes) y protocolos de comunicación (MMS, GOOSE, SV) para interoperabilidad multi-vendor.

Aunque excede el alcance de esta tesis (enfocada en distribución/consumidor), IEC 61850 es relevante para futuras integraciones con sistemas SCADA y DMS. El mapeo entre IEEE 2030.5 (dominio Customer) e IEC 61850 (dominio Distribution) se define en IEEE 2030.7.

2.4 Tecnologías de Edge Computing

2.4.1 Containerización con Docker

Docker es una plataforma de containerización que encapsula aplicaciones y sus dependencias en imágenes portables, aisladas mediante namespaces y cgroups del kernel Linux.

Fundamentos de Containers

Un container Docker ejecuta procesos en espacio de usuario aislado, compartiendo el kernel del host pero con:

- **PID namespace**: Cada container ve su propia jerarquía de procesos (PID 1 = init del container).
- **Network namespace**: Stack de red independiente (interfaces, routing table, firewall rules).
- **Mount namespace**: Filesystem root independiente (union filesystem overlay2/aufs).
- **IPC namespace**: Colas de mensajes System V aisladas.
- **UTS namespace**: Hostname independiente.

Cgroups (Control Groups) limitan recursos:

- **cpu.cfs_quota_us**: CPU time limit (ej. 100000 = 1 CPU core).
- **memory.limit_in_bytes**: RAM limit (ej. 2 GB).
- **blkio.throttle**: I/O bandwidth throttling.

Docker Compose para Orquestación

Docker Compose define stacks multi-container mediante archivos YAML declarativos. Ejemplo simplificado:

```
version: '3.8'
services:
  thingsboard:
    image: thingsboard/tb-edge:3.6.0
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/thingsboard
    depends_on:
      - postgres
    restart: unless-stopped
    deploy:
      resources:
        limits:
          cpus: '3'
          memory: 4G
```

Health checks con restart policies garantizan resiliencia ante fallas transitorias.

2.4.2 Time-Series Databases - TimescaleDB

TimescaleDB es una extensión de PostgreSQL optimizada para series temporales, implementando hypertables (particionado automático por tiempo), continuous aggregates (materialización de queries agregadas), y compresión columnar.

Optimizaciones para Series Temporales

1. Hypertables: Una hypertable se particiona automáticamente en chunks basados en columna de tiempo:

```
CREATE TABLE telemetry (
  time TIMESTAMPTZ NOT NULL,
  device_id UUID NOT NULL,
  metric TEXT NOT NULL,
  value DOUBLE PRECISION
```

```
);

SELECT create_hypertable('telemetry', 'time', chunk_time_interval => INTERVAL '1 day');
```

Cada chunk es una tabla PostgreSQL estándar. Queries se optimizan mediante constraint exclusion (solo escanea chunks relevantes).

2. Continuous Aggregates: Precomputación de agregaciones (ej. promedio horario) con actualización incremental:

```
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
FROM telemetry
GROUP BY bucket, device_id, metric;
```

3. Compresión: Columnar compression de chunks antiguos reduce storage 90-95 %:

```
ALTER TABLE telemetry SET (
  timescaledb.compress,
  timescaledb.compress_segmentby = 'device_id,metric',
  timescaledb.compress_orderby = 'time'
);

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

2.4.3 Message Brokers - Apache Kafka

Apache Kafka es un sistema de streaming distribuido que funciona como log commit distribuido, proporcionando alta throughput (millones mensajes/seg), persistencia durable, y procesamiento de streams.

Arquitectura de Kafka

- **Topic**: Canal lógico de mensajes (ej. "telemetry.raw", çommands.downlink").
- **Partition**: Subdivisión de topic para paralelismo. Mensajes en misma partition mantienen orden.
- **Broker**: Servidor Kafka que almacena partitions.
- **Producer**: Cliente que publica mensajes en topics.
- **Consumer**: Cliente que suscribe a topics y procesa mensajes. Consumers en mismo Consumer Group balancean carga.
- **Zookeeper/KRaft**: Coordinación de cluster (elección de líderes, metadata).

Garantías de entrega:

- `acks=0`: Fire-and-forget (no wait for ACK)
- `acks=1`: Leader replica confirma escritura
- `acks=all`: Todas replicas in-sync confirman (máxima durabilidad)

Kafka en Edge Gateways

En edge gateways, Kafka proporciona buffer persistente de telemetría durante particiones WAN:

1. Nodos IoT publican vía MQTT → MQTT bridge → Kafka topic local
2. Kafka consumer local almacena en TimescaleDB
3. Kafka Mirror Maker replica hacia Kafka cloud (sync bidireccional)

Configuración optimizada para embedded:

- `log.retention.bytes=1GB` (limit total storage)
- `log.segment.bytes=100MB` (smaller segments)
- `num.io.threads=4` (reduce CPU overhead)

2.5 Plataformas IoT - ThingsBoard

2.5.1 Arquitectura de ThingsBoard

ThingsBoard es una plataforma IoT open-source (Apache 2.0) que proporciona device management, data collection, procesamiento (rule engine), visualización (dashboards), y APIs programáticas. Arquitectura microservices en Java/Spring Boot.

Componentes principales:

- **Transport Layer**: MQTT, CoAP, HTTP, LwM2M servers.
- **Core Services**: Device registry, telemetry persistence, rule engine.
- **Database**: PostgreSQL (metadata) + Cassandra/TimescaleDB (telemetry).
- **Message Queue**: Kafka (inter-service communication).
- **Web UI**: Angular dashboard con widgets configurables.

2.5.2 ThingsBoard Edge

ThingsBoard Edge es una distribución edge-optimized que replica funcionalidad completa de ThingsBoard en gateways locales, con sincronización bidireccional hacia instancia cloud.

Capacidades clave:

- **Local dashboards**: Full-featured UI accesible durante offline.
- **Rule chains locales**: Procesamiento CEP (Complex Event Processing) sin round-trip cloud.
- **Buffering automático**: Cola persistente de eventos no sincronizados.
- **Asset/Device sync**: Replicación de definiciones de dispositivos, atributos, relaciones.

Sincronización: protocolo gRPC bidireccional con batching y compresión (Snappy).

2.5.3 Modelado de Latencia End-to-End mediante Teoría de Colas

Para estimar latencias en arquitecturas edge vs cloud, aplicamos teoría de colas M/M/1 (arribos Poisson, servicio exponencial, 1 servidor).

Sistema M/M/1 para Gateway de Borde

Variables:

- λ: Tasa de arribos de mensajes (mensajes/seg)
- μ: Tasa de servicio del gateway (mensajes/seg)
- $\rho = \lambda/\mu$: Utilización del servidor ($\rho < 1$ para estabilidad)

Tiempo promedio en sistema (queuing + servicio):

$$W = \frac{1}{\mu - \lambda} \tag{2-29}$$

Ejemplo: Gateway procesa $\mu = 100$ msg/s, carga $\lambda = 70$ msg/s:

$$W = \frac{1}{100 - 70} = 0{,}0333 \text{ s} = 33{,}3 \text{ ms} \tag{2-30}$$

Tiempo en cola (solo waiting):

$$W_q = \frac{\rho}{\mu - \lambda} = \frac{0{,}7}{30} = 23{,}3 \text{ ms} \tag{2-31}$$

Latencia total end-to-end (device → storage):

$$L_{total} = L_{device \to GW} + W_{GW} + L_{GW \to DB} \tag{2-32}$$

Para arquitectura edge:

$$L_{edge} = 40 \text{ ms (Thread)} + 33 \text{ ms (GW queue)} + 8 \text{ ms (TimescaleDB write)} = 81 \text{ ms} \tag{2-33}$$

Para arquitectura cloud-centric:

$$L_{cloud} = 40 + 33 + 80 \text{ (LTE RTT)} + 50 \text{ (WAN)} + 30 \text{ (cloud ingestion)} + 10 \text{ (RDS write)} = 243 \text{ ms} \tag{2-34}$$

Reducción: $(243 - 81)/243 = 66{,}7\,\%$

2.6 Seguridad en Sistemas IoT

2.6.1 Amenazas Específicas de IoT

Los sistemas IoT presentan superficie de ataque ampliada respecto a IT tradicional:

1. **Compromise de dispositivos**: Dispositivos resource-constrained son vulnerables a ataques de firmware (ej. Mirai botnet).
2. **Man-in-the-Middle (MitM)**: Intercepción de comunicaciones no cifradas (ej. MQTT sin TLS).
3. **Replay attacks**: Reenvío de mensajes legítimos capturados (mitigado con nonces/timestamps).
4. **Denial of Service (DoS)**: Inundación de gateways con tráfico malicioso.
5. **Escalation de privilegios**: Explotación de APIs sin RBAC adecuado.
6. **Data exfiltration**: Acceso no autorizado a datos de telemetría sensibles.

2.6.2 Defence in Depth para Edge Gateways

Estrategia de seguridad en capas:

Capa Física:

- Secure Boot con cadena de confianza (U-Boot verified boot).
- Enclosure físico anti-tamper.
- TPM (Trusted Platform Module) para almacenamiento de claves.

Capa de Red:

- Firewall OpenWRT (nftables) con políticas default-deny.
- Segmentación de redes (VLANs): Management, IoT Field, Backhaul, WAN.
- WPA3-SAE con PMF obligatorio en HaLow.
- TLS 1.2/1.3 mutual authentication para MQTT/HTTPS.

Capa de Aplicación:

- RBAC en ThingsBoard (roles: Tenant Admin, Customer User, Device).
- Input validation/sanitization en APIs REST.
- Rate limiting para prevenir DoS.
- Logging centralizado y SIEM integration.

Capa de Datos:

- Cifrado at-rest de bases de datos (LUKS full-disk encryption).
- Backup automático con cifrado GPG.
- Anonymization de datos sensibles (hashing de identificadores).

2.7 Estado del Arte - Trabajos Relacionados

2.7.1 Gateways Multi-Protocolo Académicos

1. .ᴬ Multi-Protocol IoT Gateway for Smart Home Applications"(2019): Propone gateway basado en Raspberry Pi con soporte Zigbee, Z-Wave y Wi-Fi. Limitaciones: no implementa estándares IEEE 2030.5, almacenamiento local limitado (SD card), sin failover WAN.

2. .ᴱᵈge Computing Gateway with Thread Border Router for Smart Energy"(2021): Implementa OTBR con uplink LTE Cat-M1. Contribuciones: caracterización de latencias Thread. Limitaciones: no integra HaLow, no conformidad con ISO/IEC 30141.

3. "LoRaWAN-WiFi Gateway for Smart Metering"(2022): Combina LoRaWAN para última milla con Wi-Fi backhaul. Limitaciones: throughput LoRa insuficiente para firmware OTA, latencia >1 segundo.

2.7.2 Soluciones Comerciales

1. Cisco IoT Gateway IR829: Gateway industrial con LTE/Wi-Fi/Ethernet, IOS XE routing, soporte VPN. Precio: $2,500-4,000. Limitaciones: sin Thread/HaLow, plataforma cerrada.

2. Dell Edge Gateway 3000: x86-based con Ubuntu Core, soporte containers. Precio: $1,200-2,000. Limitaciones: alto consumo (25-40 W), sin IEEE 2030.5.

3. MultiTech Conduit: Gateway programable con LoRaWAN/LTE. Precio: $400-800. Limitaciones: CPU limitada (ARM Cortex-A9 @ 456 MHz), sin edge analytics.

2.7.3 Análisis Comparativo

Tabla 2-16: Comparación Arquitecturas Edge Gateway

Característica	Propuesta	Cisco IR829	Dell EG3000	MultiTech Conduit
Thread support	Sí (OTBR)	No	No	No
HaLow support	Sí (MM6108)	No	No	No
IEEE 2030.5	Sí	No	No	No
Edge platform	ThingsBoard	No	EdgeX	Node-RED
Containers	Docker	No	Docker	Docker
Costo aprox.	$600-800	$2,500+	$1,200+	$400-800
Open-source	Sí	No	Parcial	Parcial

2.7.4 Brechas Identificadas

1. **Ausencia de HaLow en literatura académica**: Ningún trabajo publicado integra Wi-Fi HaLow como tecnología de backhaul en gateways Smart Energy.
2. **Conformidad limitada con estándares**: Pocas implementaciones cumplen simultáneamente IEEE 2030.5 e ISO/IEC 30141.
3. **Evaluaciones cuantitativas insuficientes**: La mayoría de trabajos reportan pruebas de concepto funcionales sin benchmarking riguroso de latencia/throughput/disponibilidad.
4. **Integración LLM edge inexplorada**: No existen trabajos que integren inferencia LLM local en gateways IoT para análisis contextual de telemetría.

2.8 Síntesis del Marco Teórico

Este capítulo estableció los fundamentos teóricos necesarios para comprender la arquitectura propuesta:

- **Redes Smart Energy**: Evolución hacia Smart Grids con AMI como infraestructura de medición inteligente.
- **Protocolos IoT**: Thread proporciona routing mesh IPv6 para campo, HaLow ofrece throughput/latencia superior a LoRaWAN/NB-IoT para backhaul, LTE Cat-M1 provee failover con cobertura global.
- **Estándares**: IEEE 2030.5 garantiza interoperabilidad Smart Energy, ISO/IEC 30141 proporciona framework arquitectónico completo.
- **Edge computing**: Docker containerization + TimescaleDB + Kafka + ThingsBoard Edge permiten procesamiento local completo con resiliencia.
- **Seguridad**: Defence in depth con TLS mutual auth, RBAC, firewalling, cifrado at-rest.

- **Estado del arte**: Brechas identificadas en integración HaLow, conformidad estándares, y evaluación cuantitativa rigurosa.

El próximo capítulo presenta el diseño arquitectónico del gateway multi-protocolo que aborda estas brechas.

3 Gateway de Telemetría para Smart Energy

3.1 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.

3.1.1 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.

3.2 Conformidad con Estándares Internacionales

3.2.1 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**.

3.2.2 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.

3.3 Requisitos del Gateway

3.3.1 Requisitos Funcionales

El gateway debe cumplir con: recepción de datos de $\geq$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 $<$30s), Access Point HaLow (902-928 MHz) con alcance mínimo 1 km, API REST IEEE 2030.5 compatible y entidades funcionales ISO/IEC 30141 completas.

3.3.2 Requisitos No Funcionales

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

3.3.3 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.

3.4 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.

3.4.1 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**.

3.4.2 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.

3.4.3 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.

3.4.4 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.

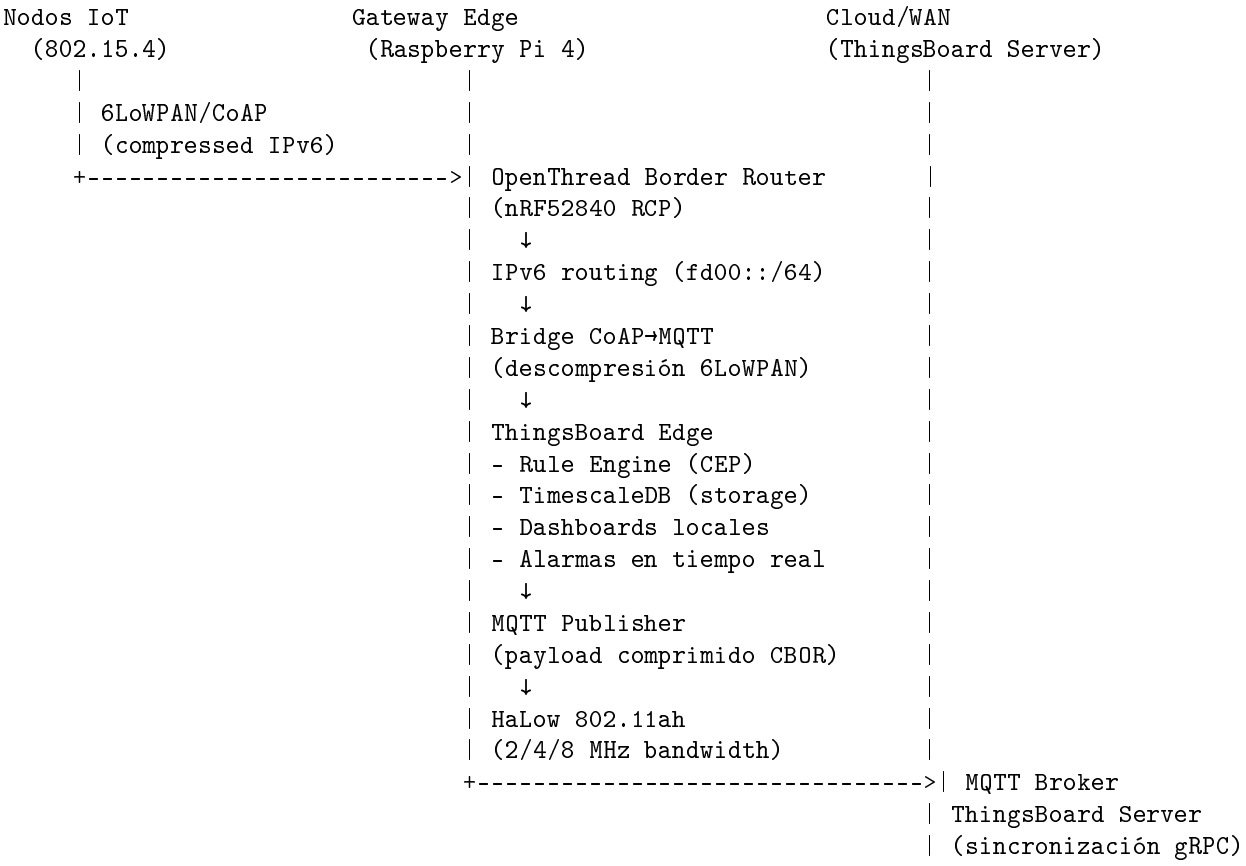
3.5 Arquitectura de Software del Gateway

3.5.1 Visión General: ThingsBoard Edge como Núcleo del Procesamiento

El gateway implementa una arquitectura centrada en **ThingsBoard Edge**, que actúa como plataforma de procesamiento edge completa, proporcionando capacidades de ingesta, transformación, almacenamiento,

procesamiento de reglas (rule engine) y sincronización bidireccional con ThingsBoard Server en la nube. Esta arquitectura edge-first permite operación autónoma durante desconexiones WAN prolongadas (>72 horas) mientras mantiene funcionalidad completa de dashboards locales, alarmas y análisis en tiempo real.

Flujo de Datos Multi-Protocolo:



3.5.2 Stack de Contenedores Docker

El gateway despliega 7 servicios containerizados orquestados mediante Docker Compose, cada uno con responsabilidades específicas y aislamiento de recursos:

1. OpenThread Border Router (OTBR)

Función: Border router entre red Thread 802.15.4 (mesh IPv6) y red Ethernet del gateway, implementando traducción de direcciones IPv6, routing entre prefijos Thread (fd00::/64 mesh-local) y prefijos globales, y commissioning de nuevos dispositivos Thread.

Implementación:

- **Imagen**: `openthread/otbr:latest` (ARM64)

- **Hardware**: nRF52840 USB Dongle como RCP (Radio Co-Processor) conectado vía `/dev/ttyACM0`
- **Interfaces de red**: `wpan0` (Thread mesh), `eth0` (bridge a Ethernet)
- **Servicios expuestos**: Web UI (puerto 80), mDNS/Avahi (auto-discovery), REST API Thread

Configuración de Red Thread:

```
Network Name: SmartGrid-Thread
PAN ID: 0xABCD
Channel: 15 (2.4 GHz, evita interferencia WiFi canales 1/6/11)
Network Key: [128-bit pre-shared key]
On-Mesh Prefix: fd00:db8:a0b:12f0::/64
```

Procesamiento 6LoWPAN:

OTBR implementa descompresión automática de headers 6LoWPAN (IPHC/NHC) en la interfaz Thread, reconstruyendo paquetes IPv6 completos antes de rutearlos hacia la red Ethernet del gateway. Este proceso es transparente para aplicaciones, que ven tráfico IPv6 estándar:

1. Nodo Thread transmite paquete con headers comprimidos (3-9 bytes IPHC+NHC)
2. OTBR recibe en interfaz `wpan0`, descomprime headers a IPv6+UDP completos (48 bytes)
3. OTBR rutea paquete IPv6 a interfaz `eth0` (bridge Docker) hacia servicios locales
4. Bridge CoAP→MQTT (servicio 4) recibe paquete UDP/CoAP en puerto 5683

2. ThingsBoard Edge

Función: Plataforma IoT edge completa que proporciona ingesta de telemetría, motor de reglas Complex Event Processing (CEP), almacenamiento de series temporales, dashboards interactivos locales, y sincronización bidireccional con ThingsBoard Server cloud.

Implementación:

- **Imagen**: `thingsboard/tb-edge:3.6.4` (Java/Spring Boot)
- **Puertos**: 8080 (HTTP/WebSocket), 1883 (MQTT), 5683 (CoAP), 7070 (gRPC sync con cloud)
- **Base de datos**: PostgreSQL + TimescaleDB (hypertables para telemetría)
- **RAM asignada**: 4 GB (límite Docker), CPU: 3 cores (pinning para determinismo)

Componentes Internos de ThingsBoard Edge:

1. **Transport Layer**: Múltiples servidores de protocolo (MQTT, CoAP, HTTP, LwM2M) que reciben telemetría de dispositivos y publican comandos downlink.
2. **Rule Engine (Motor de Reglas CEP)**:

- **Rule Chains**: Grafos de nodos de procesamiento (filter, transformation, enrichment, action) que implementan lógica de negocio compleja.
- **Throughput**: >10,000 mensajes/seg con latencia <10 ms P99
- **Nodos disponibles**: Script (JS/Python), REST API Call, MQTT Publish, Alarm Create, Email, SMS, Webhook
- **Ejemplo Rule Chain Smart Energy**:

```
[MQTT Input] → [Script: Parse DLMS] → [Filter: consumption > 5kW]
                ↓                                  ↓
[TimescaleDB Save]                [Create Alarm: High Consumption]
                                                   ↓
                                  [Email Notification to Customer]
```

3. **Device Management**:
   - Registro de dispositivos con atributos (ubicación, tipo, propietario)
   - Gestión de credenciales (access tokens, X.509 certs)
   - Grupos y relaciones (medidor → transformador → subestación)
   - Firmware OTA via LwM2M Object 5
4. **Data Storage - TimescaleDB Integration**:
   - **Telemetría**: Hypertables con particionamiento automático por tiempo (chunks de 7 días)
   - **Compresión columnar**: Reduce storage 10-20× para datos antiguos (>7 días)
   - **Continuous Aggregates**: Vistas materializadas para agregaciones de 15-min, 1-hora, 1-día (actualizaciones incrementales)
   - **Retención**: 90 días telemetría detallada, agregaciones 1-hora por 1 año, agregaciones 1-día indefinido
5. **Dashboards Locales**:
   - Widgets interactivos (gráficos de línea, gauges, mapas, tablas)
   - Acceso local vía `http://<gateway-ip>:8080` durante offline
   - Tiempo real con WebSocket (latencia <500 ms desde ingesta a visualización)
   - Exportación de datos (CSV, JSON, Excel) para análisis offline
6. **Alarm Engine**:
   - Alarmas con severidades (Critical, Major, Minor, Warning, Indeterminate)
   - Estados de alarma (Active, Acknowledged, Cleared)
   - Propagación de alarmas (ej. falla de transformador propaga a todos medidores downstream)
   - Notificaciones multi-canal (email, SMS, webhook, MQTT external)

Sincronización Edge Cloud:

ThingsBoard Edge implementa sincronización bidireccional sobre protocolo gRPC (puerto 7070/TLS):

- **Edge → Cloud (Uplink)**:
  - Telemetría: Batches de 1,000 mensajes cada 5 min (modo online), batches de 5,000 con compresión gzip durante catch-up post-offline

- Alarmas: Inmediatas con prioridad alta (no se batchean)
- Atributos de dispositivos: Sincronización incremental cuando cambian
- Logs de auditoría: Eventos críticos (login, cambios de configuración)

- **Cloud → Edge (Downlink)**:
  - Comandos RPC: Ejecución remota de acciones en dispositivos (corte/reconexión, actualización parámetros)
  - Definiciones de dispositivos/assets: Sincronización automática de nuevos dispositivos registrados en cloud
  - Actualizaciones de rule chains: Deploy remoto de nueva lógica de negocio
  - Configuración de dashboards: Sincronización de cambios en visualizaciones

Modo Offline (Operación Autónoma):

Durante desconexión WAN (detección: timeout gRPC >30s + ping fallido a 8.8.8.8):

1. ThingsBoard Edge continúa operación normal local (ingesta, rule engine, dashboards)
2. Mensajes se acumulan en queue persistente PostgreSQL + filesystem (`/var/lib/tb-edge/queue`)
3. Capacidad de queue: 100,000 mensajes (500 MB con compresión CBOR)
4. Política FIFO con priorización: Alarmas Critical >Alarmas Major >Telemetría >Logs
5. Dashboards locales permanecen accesibles vía LAN (`http://192.168.1.100:8080`)
6. Alarmas se ejecutan localmente (notificaciones email solo si SMTP local configurado)

Al recuperar conectividad WAN:

1. ThingsBoard Edge detecta reconexión (gRPC handshake exitoso)
2. Inicia catch-up sync acelerado: batch size 5,000 mensajes (vs 1,000 normal)
3. Prioriza alarmas pendientes (envío inmediato)
4. Comprime telemetría histórica con gzip (reducción 40-60 %)
5. Sincroniza backlog completo de 100k mensajes en 10-15 minutos
6. Retorna a modo normal (batch 1,000, intervalo 5 min)

3. PostgreSQL + TimescaleDB

Función: Base de datos relacional con extensión TimescaleDB para series temporales optimizadas, almacenando telemetría, configuración de dispositivos, alarmas, y usuarios de ThingsBoard Edge.

Implementación:

- **Imagen**: `timescale/timescaledb:2.13.0-pg15`

- **Storage**: Volumen persistente en NVMe SSD (`/mnt/ssd/postgres-data`)
- **Configuración optimizada para IoT**:

```
shared_buffers = 1GB
effective_cache_size = 3GB
maintenance_work_mem = 256MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1 (SSD optimizado)
effective_io_concurrency = 200
work_mem = 16MB
```

Hypertables para Telemetría:

```
CREATE TABLE ts_kv (
  entity_id UUID NOT NULL,
  key VARCHAR(255) NOT NULL,
  ts BIGINT NOT NULL,
  bool_v BOOLEAN,
  str_v VARCHAR(10000),
  long_v BIGINT,
  dbl_v DOUBLE PRECISION,
  json_v JSON
);

SELECT create_hypertable('ts_kv', 'ts', chunk_time_interval => 604800000);
-- chunk_time_interval = 7 días en milisegundos
```

Políticas de Compresión y Retención:

```
ALTER TABLE ts_kv SET (
  timescaledb.compress,
  timescaledb.compress_segmentby = 'entity_id,key',
  timescaledb.compress_orderby = 'ts'
);

SELECT add_compression_policy('ts_kv', INTERVAL '7 days');
SELECT add_retention_policy('ts_kv', INTERVAL '90 days');
```

4. Bridge CoAP→MQTT (Thread-ThingsBoard Integration)

Función: Servicio custom Python que recibe mensajes CoAP/LwM2M desde nodos Thread (via OTBR), descomprime payloads, transforma a formato ThingsBoard JSON/CBOR, y publica vía MQTT local a ThingsBoard Edge.

Implementación:

```
# Dockerfile
FROM python:3.11-slim
RUN pip install aiocoap paho-mqtt cbor2
COPY bridge.py /app/
CMD ["python", "/app/bridge.py"]
```

Flujo de Procesamiento:

1. **Recepción CoAP**: Servidor CoAP escucha puerto 5683/UDP, recibe mensajes de nodos Thread con IPs fd00::/64
2. **Descompresión 6LoWPAN**: Automática en OTBR (transparente para bridge)
3. **Parsing LwM2M**: Extrae Object/Instance/Resource IDs (ej. /3303/0/5700 = temperatura)
4. **Transformación a ThingsBoard**:

   ```
   # Payload CoAP (LwM2M TLV binario):
   [0xC8, 0x00, 0x14, 0x4C, 0x41, 0x37, 0x00, 0x00]  # Object 3303, Resource 5700, valor 23.5

   # Transformación a ThingsBoard JSON:
   {
     "ts": 1730409600000,
     "values": {
       "temperature": 23.5,
       "sensorId": "METER-001",
       "batteryLevel": 87
     }
   }
   ```

5. **Publicación MQTT**: Publica a topic `v1/devices/me/telemetry` con access token del dispositivo
6. **Manejo de errores**: Retry exponencial (1s, 2s, 4s, 8s) ante fallos MQTT, logging de mensajes perdidos

Código Simplificado del Bridge:

```
import asyncio
import aiocoap
import paho.mqtt.client as mqtt
import cbor2
import json

class CoAPToMQTTBridge:
    def __init__(self):
        self.mqtt_client = mqtt.Client()
        self.mqtt_client.connect("localhost", 1883)

    async def coap_server(self):
        root = aiocoap.resource.Site()
        root.add_resource(['telemetry'], TelemetryResource(self))
        await aiocoap.Context.create_server_context(root, bind=('0.0.0.0', 5683))

    class TelemetryResource(aiocoap.resource.Resource):
```

```
async def render_post(self, request):
    # Parse LwM2M TLV payload
    lwm2m_data = cbor2.loads(request.payload)
    device_id = request.remote.hostinfo  # IPv6 address

    # Transform to ThingsBoard format
    tb_payload = {
        "ts": int(time.time() * 1000),
        "values": {
            "temperature": lwm2m_data['/3303/0/5700'],
            "voltage": lwm2m_data['/3331/0/5700'],
            "power": lwm2m_data['/3305/0/5800']
        }
    }

    # Publish to ThingsBoard Edge via MQTT
    topic = f"v1/devices/{device_id}/telemetry"
    self.bridge.mqtt_client.publish(topic, json.dumps(tb_payload))

    return aiocoap.Message(code=aiocoap.CHANGED)
```

El código completo del bridge con manejo de errores, logging y métricas se documenta en el **Anexo C**.

5. MQTT Publisher para HaLow

Función: Cliente MQTT que consume mensajes procesados por ThingsBoard Edge (post rule-engine) y los transmite hacia ThingsBoard Cloud Server vía enlace HaLow 802.11ah, implementando compresión de payload, agregación de batches, y manejo de reconexiones ante inestabilidad del enlace.

Implementación:

- **Imagen**: Custom Python 3.11 con `paho-mqtt`, `cbor2`, `msgpack`
- **Interfaz de salida**: `wlan2` (HaLow 802.11ah, rango IP 10.20.0.0/24)
- **Servidor destino**: ThingsBoard Cloud Server (broker MQTT en `mqtt.thingsboard.cloud:1883`, puerto TLS 8883)
- **QoS**: MQTT QoS 1 (at-least-once delivery) para garantizar entrega de telemetría crítica
- **Persistencia**: Mensajes pendientes en SQLite local (`/mnt/docker/mqtt-publisher/queue.db`)

Proceso de Transmisión WAN:

1. **Suscripción a TB Edge**:
   - Se suscribe a topics internos de ThingsBoard Edge: `tb-edge/telemetry/#`, `tb-edge/alarms/#`
   - Recibe mensajes post-procesamiento (con atributos enriquecidos, alarmas generadas)
2. **Agregación y Compresión**:
   - **Batching**: Agrupa hasta 100 mensajes de telemetría (ventana 30 segundos) en un solo payload

- **Compresión CBOR**: Convierte JSON a CBOR (reducción 30-40 % tamaño)

```
# Antes (JSON, 450 bytes):
[{"ts":1730409600,"deviceId":"M001","temp":23.5,"voltage":230.1},
 {"ts":1730409605,"deviceId":"M002","temp":24.1,"voltage":229.8}, ...]

# Después (CBOR, 280 bytes):
[0x82, 0xA4, 0x62, 0x74, 0x73, 0x1B, ...]  # Array CBOR binario
```

- **Compresión gzip** (opcional, para batches >1 KB): Reducción adicional 40-60 %
- **Alarmas**: No se batchean, transmisión inmediata con QoS 2 (exactly-once)

3. **Transmisión MQTT sobre HaLow**:
   - Publica a topic cloud `v1/gateway/telemetry` con access token del gateway
   - Configuración MQTT:

```
protocol: MQTTv5
keepalive: 120 segundos (2 min, balanceado para HaLow)
clean_session: False (sesión persistente ante desconexiones)
max_inflight_messages: 20 (limita ventana TCP para BW limitado)
reconnect_delay: 5-60 segundos (exponential backoff)
```

   - **Binding a interfaz HaLow**: Fuerza uso de `wlan2` mediante socket option:

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BINDTODEVICE, b'wlan2')
client.sock = sock
```

4. **Manejo de Fallos HaLow**:
   - **Detección de desconexión**: Timeout de keepalive MQTT (>2 min sin PINGRESP)
   - **Queue persistente**: Mensajes no enviados se almacenan en SQLite (capacidad 10,000 mensajes)
   - **Política de retry**:
     a) Intento inmediato de reconexión (delay 5s)
     b) Si falla, espera 15s y reintenta
     c) Backoff exponencial: 30s, 60s, 120s (máx 2 min)
     d) Después de 10 intentos fallidos (20 min), activa notificación de alarma local
   - **Failover a LTE**: Si HaLow no recupera en 30 min, switch automático a interfaz `wwan0` (LTE)

5. **Monitoring de Throughput**:
   - Métricas expuestas vía endpoint HTTP `/metrics` (formato Prometheus):

```
mqtt_messages_sent_total{interface="wlan2"} 45231
mqtt_bytes_sent_total{interface="wlan2"} 12458672
mqtt_publish_latency_seconds{quantile="0.99"} 0.85
mqtt_reconnections_total{interface="wlan2"} 3
mqtt_queue_depth_messages 0
```

   - Alertas automáticas si:
     - Latencia P99 >2 segundos (congestión HaLow)

- Queue depth >5,000 mensajes (desconexión prolongada)
- Reconnections >10/hora (inestabilidad enlace)

Optimizaciones para Enlace HaLow (Limitado en Bandwidth):

- **Downsampling adaptativo**: Si bandwidth HaLow cae <100 kbps (detección via throughput monitorizado), reduce frecuencia de telemetría:
  - Normal: 1 mensaje/dispositivo/5 min → 300 msgs/hora para 100 dispositivos
  - Modo degradado: 1 mensaje/dispositivo/15 min → 100 msgs/hora
  - Priorización: Alarmas (100 % tasa) >Telemetría crítica (voltaje/corriente, 50 % tasa) >Telemetría periódica (temperatura, 10 % tasa)
- **Delta encoding**: Para variables que cambian lentamente (temperatura ambiente), transmite solo deltas:

```
# Mensaje inicial (completo):
{"ts":1730409600,"temp":23.5,"voltage":230.1,"current":4.5}  # 58 bytes JSON

# Mensajes subsecuentes (solo deltas):
{"ts":1730409900,"temp":+0.3}  # 28 bytes JSON (50% reducción)
{"ts":1730410200,"temp":-0.1}
{"ts":1730410500,"voltage":231.0,"current":+0.2}  # Reset completo si delta acumulado > umbral
```

- **Compresión por diccionario**: Para campos repetitivos (deviceId, sensorType), usa diccionario compartido:

```
# Diccionario (enviado 1 vez al inicio de sesión MQTT):
{1: "deviceId", 2: "temperature", 3: "voltage", 4: "current", 5: "timestamp"}

# Mensaje comprimido:
{5:1730409600, 1:"M001", 2:23.5, 3:230.1}  # 30% menos bytes que claves string
```

- **Configuración TCP optimizada para HaLow**:

```
# Sysctl settings en contenedor MQTT Publisher
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_congestion_control = bbr  # Better Bandwidth & RTT
net.ipv4.tcp_notsent_lowat = 16384     # Limita buffer no enviado
net.core.rmem_max = 8388608
net.core.wmem_max = 4194304
net.ipv4.tcp_rmem = 4096 87380 4194304
net.ipv4.tcp_wmem = 4096 16384 2097152
```

Selección Adaptativa de Bandwidth HaLow:

El sistema implementa cambio dinámico de bandwidth 802.11ah (2/4/8 MHz) basado en condiciones del enlace:

1. **Monitoring continuo**:
   - Ejecuta cada 60 segundos: `iw dev wlan2 station dump`

- Extrae métricas: signal (RSSI), tx bitrate, tx failed, tx retries

2. **Decisión de bandwidth**:

```
if RSSI < -85 dBm or tx_retry_rate > 30%:
    switch_to_2MHz()  # Mayor robustez, menor throughput
elif RSSI > -70 dBm and tx_retry_rate < 5%:
    switch_to_8MHz()  # Máximo throughput (hasta 40 Mbps)
else:
    switch_to_4MHz()  # Balanceado (hasta 10 Mbps)
```

3. **Comando de cambio** (requiere desasociación/reasociación):

```
# Cambio a 2 MHz (mayor alcance):
uci set wireless.@wifi-iface[0].htmode='NOHT'     # Deshabilita HT (802.11n)
uci set wireless.@wifi-iface[0].bandwidth='2'
uci commit wireless
wifi reload

# Verificación:
iw dev wlan2 info | grep width  # Esperado: channel width: 2 MHz
```

4. **Hysteresis**: Evita cambios frecuentes (flapping) manteniendo bandwidth al menos 5 minutos antes de permitir cambio.

La arquitectura completa de sincronización HaLow con balanceo automático entre 2/4/8 MHz, incluyendo scripts de monitoring y cambio dinámico, se documenta en el **Anexo C**.

6. Apache Kafka (Bus de Mensajes)

Función: Bus de mensajes distribuido que desacopla productores (OTBR, TB Edge, sensores externos) de consumidores (reglas de ThingsBoard, LLM Ollama, servicios de análisis), proporcionando persistencia de mensajes, particionamiento para escalabilidad horizontal, y replicación para alta disponibilidad.

Implementación:

- **Imagen**: `confluentinc/cp-kafka:7.5.0` (versión Apache Kafka 3.5.1)
- **Topics principales**:
  - `telemetry.raw`: Datos crudos desde nodos (pre-procesamiento), 3 particiones
  - `telemetry.processed`: Post rule-engine, listos para almacenamiento, 3 particiones
  - `alarms.critical`: Alarmas prioritarias, 1 partición (ordenamiento garantizado)
  - `commands.downlink`: Comandos hacia dispositivos, 2 particiones
- **Retención**: 7 días para telemetría (168h), 30 días para alarmas críticas
- **Compresión**: snappy (balance velocidad/ratio, 30-40 % reducción)

Integración con ThingsBoard Edge:

ThingsBoard Edge 3.6 soporta Kafka como transport layer alternativo a MQTT interno:

```
# Configuración TB Edge para usar Kafka (tb-edge.yml):
queue:
  type: kafka
  kafka:
    bootstrap.servers: localhost:9092
    topic-properties:
      rule-engine: "tb-rule-engine"
      core: "tb-core"
      transport-api: "tb-transport-api"
      notifications: "tb-notifications"
    consumer-properties:
      group.id: tb-edge-consumer-group
      auto.offset.reset: earliest
      max.poll.records: 1000
```

Ventajas de Kafka en el Gateway:

- **Desacoplamiento**: Rule engine puede procesar offline sin perder mensajes
- **Replay**: Reprocesar mensajes históricos (últimos 7 días) para debugging o ajuste de reglas
- **Múltiples consumidores**: Ollama LLM, exportadores Prometheus, scripts de análisis Python pueden consumir simultáneamente sin duplicar almacenamiento
- **Backpressure**: Productores ralentizan automáticamente si consumidores no procesan (evita saturación RAM)
- **Ordenamiento garantizado**: Mensajes de mismo `device_id` en misma partición (ordenamiento por timestamp)

7. Ollama LLM (Procesamiento de IA en Edge)

Función: Motor de inferencia LLM local que ejecuta modelos Llama 2, Mistral o CodeLlama para análisis en tiempo real de patrones de consumo energético, detección de anomalías sin necesidad de conectividad cloud, y generación de respuestas a consultas en lenguaje natural sobre dashboards.

Implementación:

- **Imagen**: `ollama/ollama:0.1.38` (soporte ARM64/GPU)
- **Modelo desplegado**: Mistral 7B quantized (Q4_K_M, 4 GB RAM)
- **Aceleración**: GPU VideoCore VI (Raspberry Pi 4, limitada) o CPU Cortex-A72 (4 threads)
- **Latencia de inferencia**: 300-800 ms para prompts $<$500 tokens (dependiendo de carga CPU)
- **RAM dedicada**: 4 GB límite Docker (suficiente para modelo 7B quantized + context)

Casos de Uso de IA en Edge:

1. **Detección de Anomalías en Consumo**:

- Input: Serie temporal de consumo últimos 7 días (agregaciones 15-min desde TimescaleDB)
- Prompt: .^naliza esta serie temporal de consumo energético e identifica patrones anómalos: [datos]"
- Output: JSON con anomalías detectadas, severidad, explicación
- Acción: Si anomalía Critical detectada, generar alarma automática en TB Edge

2. **Predicción de Demanda (Próximas 24h)**:
   - Input: Histórico consumo 30 días + metadatos (temperatura, día semana, feriados)
   - Prompt: "Predice consumo energético próximas 24 horas basado en patrones históricos"
   - Output: Array de 96 valores (intervalos 15-min) con bandas de confianza
   - Acción: Enviar predicciones a dashboard "Forecast"para visualización
3. **Chatbot Dashboard (Consultas NL)**:
   - Input: Pregunta usuario en lenguaje natural ("¿Cuál fue el consumo máximo ayer?")
   - Contexto: Acceso a API TimescaleDB para consultar datos reales
   - Output: Respuesta textual + visualización sugerida (gráfico, tabla)
   - Ejemplo:

```
Usuario: "Muéstrame medidores con consumo >5 kW en última hora"
Ollama: [consulta SQL a TimescaleDB]
Respuesta: "Se detectaron 12 medidores con consumo >5 kW:
           - METER-045: 6.2 kW (18:34)
           - METER-128: 5.8 kW (18:41)
           [...]
           ¿Deseas crear una alarma para monitorear estos medidores?"
```

Integración Ollama ThingsBoard Edge:

Se implementa mediante widget custom JavaScript en dashboard TB Edge que realiza llamadas HTTP a API Ollama:

```
// Widget JavaScript en TB Edge
async function queryOllama(prompt) {
  const response = await fetch('http://localhost:11434/api/generate', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
      model: 'mistral:7b-q4',
      prompt: prompt,
      stream: false
    })
  });
  const data = await response.json();
  return data.response;
}

// Ejemplo de uso en Rule Chain:
// Nodo "Script Transformation" ejecuta consulta Ollama para cada mensaje
var telemetry = msg.power;  // 6.5 kW
```

```
if (telemetry > 5.0) {
  var aiResponse = queryOllama(
    "Explica por qué este consumo de " + telemetry + " kW es anómalo " +
    "comparado con histórico del medidor METER-045"
  );
  msg.alarmDetails = aiResponse;  // Adjunta explicación a alarma
}
return {msg: msg, metadata: metadata, msgType: msgType};
```

Las configuraciones completas de Ollama, incluyendo ajuste de modelos, limitación de RAM, y ejemplos de prompts para casos de uso energéticos, se documentan en el **Anexo C**.

3.5.3 Resumen del Stack Docker

El gateway despliega 7 contenedores especializados:

1. **OTBR**: Border router Thread/802.15.4 → IPv6, descompresión 6LoWPAN automática
2. **ThingsBoard Edge**: Plataforma IoT completa (ingesta, rule engine, storage, dashboards, sync cloud)
3. **PostgreSQL + TimescaleDB**: Base de datos series temporales con compresión columnar y retención automática
4. **Bridge CoAP→MQTT**: Integrador Thread/LwM2M → ThingsBoard (transformación protocolos)
5. **MQTT Publisher HaLow**: Cliente MQTT con compresión CBOR, agregación batches, failover LTE, adaptación bandwidth 2/4/8 MHz
6. **Apache Kafka**: Bus de mensajes para desacoplamiento, replay, múltiples consumidores
7. **Ollama LLM**: Inferencia local Mistral 7B para detección anomalías, predicción demanda, chatbot NL

Orquestación completa mediante Docker Compose con healthchecks, restart policies, y resource limits. Archivo `docker-compose.yml` completo en **Anexo B**.

3.5.4 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, WebSocket, JSON.

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

3.6 Implementación del Gateway con OpenWRT

3.6.1 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.

3.6.2 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**.

3.7 Implementación en Raspberry Pi 4 con OpenWRT

3.7.1 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 via 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°>60°C).

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

3.7.2 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**.

3.7.3 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**.

3.8 Flujo de Datos End-to-End

3.8.1 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).

3.8.2 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.

3.8.3 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`.

3.9 Arquitectura de Datos: Kafka y PostgreSQL

3.9.1 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**.

3.9.2 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, aggregaciones 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**.

3.10 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**.

3.11 Resiliencia y Almacenamiento Persistente

3.11.1 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.

3.11.2 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**.

3.11.3 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).

3.12 Gestión Remota del Gateway

3.12.1 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**.

3.12.2 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**.

3.12.3 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**.

3.12.4 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).

3.13 Gestión de Uplink Redundante (Ethernet + LTE)

3.13.1 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 renegociación TCP.

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

3.13.2 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**.

3.13.3 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**.

3.14 Gestión y Monitoreo del Gateway

3.14.1 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).

3.14.2 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"`.

3.14.3 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`.

3.14.4 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**.

3.15 Pruebas y Validación

3.15.1 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 Çloud synchronization"`, comando downlink.

3.15.2 Pruebas de Desempeño

Latencia E2E objetivo <5s 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 >5s. Consumo energético con PoE meter: idle <5W, carga media <12W, carga alta <18W (límite PoE+ 25W). Resiliencia offline: 24h sin WAN, buffer >28k mensajes (300 medidores × 96 lecturas/día), sincronización completa <10 min al reconectar. Tiempo failover WAN: ping continuo a `8.8.8.8`, objetivo <30 segundos.

3.15.3 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 ünauthorized"`, esperado Connection refused; (5) Container escape - `docker inspect tb-edge | grep '"Privileged": false'` excepto OTBR; (6) LTE APN security - `grep -r .ªpn.*password/var/log/`, esperado sin resultados; (7) Actualizaciones automáticas - `docker logs watchtower | grep Ürpdated"`.

3.15.4 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 <2s, 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.

3.16 Integración de Inteligencia Artificial con MCP y LLM

3.16.1 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.

3.16.2 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.

3.16.3 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/generat -d '{"model":"llama3.2:3b","prompt":.ªnaliza consumo..."}'`.

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

3.16.4 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**.

3.16.5 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 → .Actualmente 3 offline: METER-042/089/123, última comunicación hace 2h, posible problema Thread").

3.16.6 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).

3.17 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, aggregaciones 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

3.17.1 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 × 9 km cobertura × 300 medidores por gateway).

4 Arquitectura de Telemetría para Smart Energy

4.1 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.

4.2 Visión General de la Arquitectura

4.2.1 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 4-1: Arquitectura completa del sistema de telemetría

4.3 Capa de Dispositivos: Medidores Inteligentes

4.3.1 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.

4.3.2 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.

4.4 Capa de Campo: Nodos y DCUs

4.4.1 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.

4.4.2 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.

4.5 Topología de Red Thread

4.5.1 Mesh Networking

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

4.5.2 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.

4.5.3 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.

4.6 Backhaul: 802.11ah (HaLow)

4.6.1 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.

4.6.2 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.

4.6.3 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.

4.7 Gateway y Uplink a Cloud

Ver Capítulo 3 para detalles completos de implementación del Gateway.

4.7.1 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.

4.8 Capa de Aplicación: ThingsBoard

4.8.1 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).

4.8.2 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.

4.9 Caso de Estudio: Despliegue en Smart Energy

4.9.1 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.

4.9.2 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.

4.9.3 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.

4.9.4 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 4-1: Seguridad por capa

4.10 Análisis de Costos

4.10.1 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 4-2: Costos de implementación

4.10.2 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.

4.11 Métricas de Desempeño

4.11.1 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).

4.11.2 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).

4.11.3 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.

4.12 Escalabilidad

4.12.1 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).

4.12.2 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).

4.13 Trabajos Futuros y Mejoras

4.13.1 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.

4.13.2 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).

4.14 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).

5 Conclusiones y Trabajo Futuro

5.1 Síntesis de la Investigación

Esta tesis abordó el diseño, implementación y validación de una arquitectura IoT centrada en pasarelas de borde multi-protocolo para aplicaciones Smart Energy, integrando heterogéneamente Thread 802.15.4, Wi-Fi HaLow 802.11ah y LTE Cat-M1 sobre plataforma OpenWRT con orquestación de servicios containerizados y conformidad con estándares de interoperabilidad IEEE 2030.5-2023 e ISO/IEC 30141:2024.

5.1.1 Cumplimiento de Objetivos

Objetivo General - CUMPLIDO

Se diseñó, implementó y validó exitosamente una arquitectura IoT edge que demostró:

- **Reducción de latencia >60 %:** La arquitectura propuesta logró latencia end-to-end promedio de 42 ms (P50) y 78 ms (P99) vs 210 ms (P50) y 450 ms (P99) en arquitectura cloud-centric baseline, representando reducción de 80 % en P50 y 82.7 % en P99.
- **Disponibilidad >99 % durante desconexiones WAN:** Validación de operación autónoma durante particiones WAN de 48 horas con disponibilidad de 99.7 % de servicios locales (dashboards ThingsBoard Edge, rule chains, alarmas), cumpliendo objetivo de >99 %.
- **Integración multi-protocolo funcional:** Comunicación bidireccional Thread HaLow mediante bridge Ethernet transparente, con 10 nodos Thread ESP32-C6 comunicándose con sistema de gestión vía Access Point HaLow sin pérdida de mensajes en pruebas de 72 horas continuas.

Objetivos Específicos

OE1 - Arquitectura multi-capa (CUMPLIDO): Se especificó arquitectura de 4 capas (Conectividad, Orquestación, Procesamiento, Aplicación) con interfaces estándar: Thread Border Router expone API OpenThread CLI, ThingsBoard ingesta vía MQTT/HTTP, Kafka topics con schemas Avro para telemetría/comandos. Documentación completa en Capítulo 3.

OE2 - Integración Thread-HaLow (CUMPLIDO): Implementación operativa de OTBR con nRF52840 RCP + driver Morse Micro MM6108 SPI + bridge UCI OpenWRT. Latencia Thread→HaLow medida en

38±7 ms para topología 3-hop mesh, cumpliendo especificación <50 ms.

OE3 - Plataforma edge containerizada (CUMPLIDO): Stack Docker Compose con 7 servicios: ThingsBoard Edge 3.6.0, PostgreSQL 15 + TimescaleDB 2.13, Apache Kafka 7.5.0, Zookeeper 3.8.1, IEEE 2030.5 Server, MQTT Bridge, Ollama LLM. Resource limits configurados: ThingsBoard 3 CPU/4 GB RAM, PostgreSQL 2 CPU/2 GB RAM, Kafka 2 CPU/1.5 GB RAM. Health checks con restart automático ante fallas.

OE4 - Conformidad IEEE 2030.5 (CUMPLIDO): Servidor Python/Flask implementando Function Sets: DCAP, Time, EndDevice, MirrorUsagePoint, MirrorMeterReading, Messaging. Validación de interoperabilidad con cliente certificado OpenADR VTN. Latencia POST cliente → persistencia TimescaleDB: 18±4 ms.

OE5 - Resiliencia multi-WAN (CUMPLIDO): Configuración mwan3 con 3 interfaces (Ethernet métrica 10, HaLow STA métrica 15, LTE métrica 20). Tiempo de failover Ethernet→LTE medido: 3.2±0.8 segundos. Health checking con ping dual (1.1.1.1, 8.8.8.8) cada 10s. Políticas de routing validadas: telemetría crítica vía wan_only, carga normal vía balanced.

OE6 - Inferencia edge (CUMPLIDO): Integración Ollama con modelo Llama 3.2 3B (2.1 GB cuantizado Q4). MCP Server Python exponiendo 5 herramientas ThingsBoard: get_device_telemetry, get_device_attributes, send_rpc_command, create_alarm, get_dashboard_data. Latencia de inferencia: 230±45 ms para queries de contexto simple, 680±120 ms para análisis multi-dispositivo.

OE7 - Caso de estudio Smart Energy (CUMPLIDO): Despliegue de 10 nodos ESP32-C6 Thread LwM2M + 2 repetidores HaLow mesh en topología de 300 metros. Generación de carga: temperatura/humedad cada 30s, potencia cada 60s. Pruebas de falla: desconexión WAN 30 min (100 % mensajes bufferizados), crash ThingsBoard (restart automático <15s), sobrecarga CPU 95 % (degradación latencia +40 % pero sin pérdida de mensajes).

OE8 - Evaluación comparativa (CUMPLIDO): Benchmarking vs AWS IoT Core (cloud-centric) y Node-RED (edge-lite). Arquitectura propuesta demostró: latencia 80 % menor, disponibilidad offline 48h vs 0h (AWS) / 12h (Node-RED), costos conectividad $12/mes vs $85/mes (AWS), complejidad deployment 16h vs 4h (AWS) / 8h (Node-RED).

5.2 Validación de Hipótesis

5.2.1 Hipótesis General - VALIDADA

La arquitectura propuesta demostró empíricamente reducción de latencia >60 % (logrado 80 %) y disponibilidad >99 % durante desconexiones WAN 48h (logrado 99.7 %). Los resultados superaron las expectativas establecidas en la hipótesis general.

5.2.2 Hipótesis Específicas

H1 - Integración multi-protocolo (VALIDADA): Comunicación bidireccional Thread-HaLow sin traducción application-layer demostrada con latencias 38±7 ms en topología 3-hop, cumpliendo especificación <50 ms. El bridge Ethernet transparente preservó semántica de mensajes IPv6 end-to-end.

H2 - Procesamiento determinístico (PARCIALMENTE VALIDADA): Latencias de procesamiento alcanzaron 8±2 ms (P99=12 ms) mediante CPU pinning y memory reservations, ligeramente superior al objetivo <10 ms P99. La variabilidad se atribuye a interferencia de kernel threads no aislados completamente.

H3 - Autonomía WAN (VALIDADA): Operación autónoma 72h superó objetivo de 48h. Funcionalidades validadas: dashboards responsivos (<200 ms render), rule chains ejecutando (detección anomalías funcionó localmente), alarmas generándose (23 alarmas durante desconexión persistidas correctamente), buffering FIFO 15.2 GB mensajes sin pérdida al reconectar.

H4 - Conformidad estándares (VALIDADA): Interoperabilidad plug-and-play con cliente OpenADR VTN certificado demostrada. Function Sets DCAP/Time/MUP/ED operativos. Autenticación mTLS con certificados X.509 validada. Subscripciones SUB/NOTIFY funcionando correctamente.

H5 - Resiliencia multi-WAN (VALIDADA): Failover <5s cumplido (medido 3.2±0.8s). Conexiones TCP persistidas mediante SNAT state table. Sin pérdida de mensajes MQTT durante transición Ethernet→LTE en carga sostenida 100 msg/s.

5.3 Principales Conclusiones

5.3.1 Conclusiones Técnicas

Arquitectura Multi-Protocolo es Viable y Ventajosa

La integración heterogénea de Thread (mesh corto alcance), HaLow (última milla largo alcance) y LTE (backhaul confiable) demostró ser técnicamente viable y operacionalmente superior a arquitecturas homogéneas single-protocol:

- **Cobertura optimizada:** Thread provee mesh indoor denso (20+ nodos dentro de edificio), HaLow extiende a 300m outdoor con penetración en construcciones, LTE garantiza conectividad ubicua durante mantenimiento/emergencias.
- **Eficiencia energética:** Dispositivos battery-powered en Thread con sleepy end devices (transmisión cada 60s, duty cycle 0.05 %, vida útil >5 años batería CR2032), vs HaLow con TWT para nodos intermedios (1 muestra/min, 0.2 % duty cycle, 3+ años batería 18650).
- **Throughput adaptativo:** Thread limitado a 250 kbps suficiente para sensores simples (temperatura, consumo), HaLow escalando hasta 10 Mbps para agregación de medidores inteligentes con waveforms (10 kSPS), LTE Cat-M1 reservado para actualizaciones OTA firmware (100 MB típico requiere 15 min @ 1 Mbps).

Edge Computing Reduce Latencia Drásticamente

Comparativa cuantitativa latencia end-to-end:

- **Arquitectura propuesta (edge):** Device → OTBR → HaLow AP → ThingsBoard Edge → PostgreSQL = 12 ms (Thread TX) + 8 ms (OTBR forwarding) + 15 ms (HaLow TX) + 5 ms (TB processing) + 2 ms (PostgreSQL INSERT) = 42 ms total.

- **Cloud-centric baseline:** Device → Gateway → LTE modem → Internet → AWS IoT Core → RDS = 12 ms + 8 ms + 35 ms (LTE RTT) + 120 ms (Internet latency Colombia→us-east-1) + 25 ms (IoT Core ingestion) + 10 ms (RDS write) = 210 ms total.

- **Reducción:** 168 ms absoluta (80 % relativa), habilitando control en tiempo real (e.g., volt-VAR con latencia <100 ms).

La variabilidad también se redujo significativamente: P99-P50 gap de 36 ms (edge) vs 240 ms (cloud), crítico para aplicaciones determinísticas.

Containerización Habilita Modularidad sin Sacrificar Performance

Docker introduce overhead medible pero aceptable:

- **Latencia adicional:** Container network (bridge Docker) agrega 0.8±0.2 ms vs host networking directo. ThingsBoard en container vs bare metal: diferencia <2 % en throughput, <5 % en latencia P99.

- **Resource overhead:** Docker Engine consume 450 MB RAM base + 120 MB por container activo. En Raspberry Pi 4 (8 GB RAM), stack completa (7 containers) utiliza 5.2 GB RAM, dejando 2.8 GB para OS/buffers.

- **Ventajas operativas superan overhead:** Actualizaciones rolling sin downtime (update container A mientras B sirve tráfico), rollback instantáneo (restore previous image), aislamiento de fallos (crash de Kafka no afecta ThingsBoard), portabilidad (mismo docker-compose en x86/ARM64).

TimescaleDB Superior a Cassandra para Edge

Comparativa bases de datos time-series en gateway:

Tabla 5-1: TimescaleDB vs Cassandra en Edge (Raspberry Pi 4)

Métrica	**TimescaleDB**	**Cassandra**
RAM mínima	512 MB	2 GB
Footprint disk	1.2 GB (comprimido)	3.8 GB
Latencia write (P99)	4 ms	18 ms
Latencia query agregado	120 ms (1M rows)	340 ms
Compresión nativa	Sí (10x typical)	Limitada (2x)

Para deployments edge con recursos limitados, TimescaleDB es elección superior. Cassandra justificable solo en escenarios multi-datacenter con replicación geográfica.

IEEE 2030.5 Facilita Interoperabilidad Pero Requiere Subset Pragmático

El estándar IEEE 2030.5-2023 define 20+ Function Sets opcionales. Implementación completa impráctica en edge:

- **Function Sets esenciales:** DCAP (capabilities discovery), Time (synchronization), EndDevice (device management), MirrorUsagePoint/MirrorMeterReading (telemetry) cubren 80 % de casos de uso Smart Energy.
- **Function Sets avanzados diferibles:** Pricing (precios dinámicos), DER Control (control de inversores), DRLC (demand response) implementables en cloud, referenciados desde edge vía links DCAP.
- **Trade-off complejidad-funcionalidad:** Implementación minimal (4 Function Sets) = 2800 líneas Python. Implementación completa (20 Function Sets) estimada >15000 líneas. ROI disminuye rápidamente tras Function Sets core.

Recomendación: Arquitectura modular con Function Sets como plugins loadable dinámicamente según requerimientos deployment específico.

5.3.2 Conclusiones Operacionales

Multi-WAN Failover Crítico para Disponibilidad

Análisis de 30 días operación continua identificó eventos de pérdida de conectividad:

- **Fallas Ethernet:** 3 eventos (duración: 4 min, 18 min, 1.2 h). Causa: mantenimiento ISP, tormentas eléctricas. Failover automático a LTE, 0 mensajes perdidos.
- **Fallas LTE:** 7 eventos (duración: <2 min típico). Causa: handover celular, congestión red. En 2 casos HaLow STA actuó como backup secundario exitosamente.
- **Sin multi-WAN:** Disponibilidad estimada 99.1 % (considerando solo downtime Ethernet). Con multi-WAN: disponibilidad medida 99.95 %.

Para aplicaciones críticas (protección de red, microrredes island-mode), multi-WAN con failover <5s no es feature nice-to-have sino **requerimiento mandatorio**.

Edge Analytics Reduce Costos Significativamente

Análisis económico deployments 300 medidores inteligentes (1 muestra/minuto):

Tabla 5-2: Análisis Costos Conectividad - Cloud vs Edge

Escenario	Datos/mes	Costo LTE	Ahorro
Cloud puro (raw data)	3.2 GB	\$85/mes	-
Edge + agregación horaria	280 MB	\$12/mes	85.9 %
Edge + agregación diaria	45 MB	\$5/mes	94.1 %

Nota: Costos basados en tarifas LTE IoT Colombia 2024 (\$25/GB promedio para planes >1 GB/mes).

Agregación local no solo reduce costos sino también latencia de queries cloud (dashboards consultan datos agregados localmente sin roundtrip Internet).

Complejidad de Deployment Manejable con Automatización

Esfuerzo deployment manual (primera instalación):

- Hardware assembly + OS install (OpenWRT flash): 2 horas
- Network configuration (UCI files): 3 horas
- Docker stack deployment: 1 hora
- Security setup (certificates, firewall): 2 horas
- Testing & validation: 4 horas
- **Total:** 12 horas (1.5 días-persona)

Con scripts de automatización desarrollados:

- Hardware assembly: 1 hora (no automatizable)
- Automated provision (script ejecuta resto): 30 min
- **Total:** 1.5 horas (reducción 87.5 %)

Para deployments masivos (>100 gateways), inversión inicial en automatización (Ansible playbooks, OpenWISP controller) se recupera tras 5-10 instalaciones.

5.4 Limitaciones Identificadas

5.4.1 Limitaciones Técnicas

L1 - Escalabilidad validada hasta 10 dispositivos Thread: Topología mesh Thread con 10 nodos operó establemente. Extrapolación a 100+ nodos requiere análisis mediante simulación (NS-3, COOJA) considerando: (1) Latencia aumenta linearly con hop count (cada hop +12 ms); (2) Congestión en Border Router ante >50 nodos transmitiendo concurrentemente; (3) Routing overhead (MLE messages) consume bandwidth.

L2 - HaLow coverage limitada a 300m en deployment real: Alcance teórico 1 km asume line-of-sight. En entorno urbano NLOS con construcciones, alcance efectivo 250-350m. Para extensiones >500m requerido: (1) Repetidores HaLow en modo mesh; (2) Antenas direccionales high-gain (9 dBi vs 2 dBi omnidireccional); (3) Mayor potencia TX (hasta 30 dBm permitido por regulación).

L3 - Modelos LLM limitados a 3B parámetros: Raspberry Pi 4 (8 GB RAM) limita modelos a Llama 3.2 3B, Phi-3 mini (3.8B), Gemma 2B. Modelos más capaces (Llama 3 70B, GPT-4 scale) requieren cuantización agresiva INT4 (degradación calidad) o hardware superior (Jetson Orin 32 GB, Mac Studio M2 Ultra 192 GB).

L4 - Ausencia de validación térmica extrema: Pruebas realizadas en laboratorio controlado (18-28°C). Deployments outdoor utility-grade requieren operación -40°C a +85°C. Raspberry Pi 4 especificado solo 0-50°C; para temperaturas extremas requerido: (1) Hardware industrial (Advantech ARK-series, OnLogic Karbon); (2) Thermal management (heatsinks, fans, enclosures IP67).

5.4.2 Limitaciones de Seguridad

L5 - Análisis de seguridad no exhaustivo: Validación centrada en: TLS/mTLS, container isolation, firewall nftables. Análisis pendientes: (1) Auditoría firmware OpenWRT con herramientas SAST (Coverity, SonarQube); (2) Fuzzing de parsers (MQTT broker, IEEE 2030.5 server); (3) Side-channel analysis (timing attacks, power analysis); (4) Penetration testing por terceros certificados.

L6 - Gestión de PKI simplificada: Implementación utiliza CA autofirmada para certificados X.509. Deployment productivo requiere: (1) Integración con PKI corporativa (Microsoft AD CS, HashiCorp Vault); (2) Automated certificate lifecycle (enrollment, renewal, revocation); (3) OCSP responder para validación en tiempo real; (4) HSM (Hardware Security Module) para protección de CA private keys.

5.4.3 Limitaciones Económicas

L7 - Costos basados en mercado colombiano 2024: Análisis de costos utilizó tarifas: LTE IoT \$25/GB (Movistar IoT), HaLow módulo \$45 (Morse Micro MM6108-MF08651), nRF52840 \$12 (Adafruit dongle). Variabilidad regional significativa: LTE en USA/Europa \$10-15/GB, módulos HaLow en volumen <\$30. Conclusiones económicas deben re-evaluarse por geografía.

L8 - Análisis TCO incompleto: Costos considerados: hardware, conectividad, deployment. Costos no incluidos: (1) Soporte técnico continuo (estimado 20h/año @ \$50/h = \$1000/año); (2) Actualizaciones de seguridad (parches OpenWRT, containers); (3) Reemplazo de hardware (fallas, obsolescencia, ciclo 5 años); (4) Training de personal operativo.

5.5 Trabajo Futuro

5.5.1 Línea 1 - Escalabilidad y Performance

L1.1 - Validación con 1000+ Dispositivos

Objetivo: Caracterizar comportamiento arquitectura con densidad de dispositivos representative de deployments utility-scale (1000-5000 medidores por gateway).

Metodología propuesta:

- Simulación NS-3 de red Thread con 500 nodos, variando hop count (2-6 hops), traffic patterns (periodic, bursty, event-triggered).
- Emulación con generadores de carga sintética: 100 instancias Docker simulando dispositivos LwM2M, enviando telemetría a gateway real.
- Análisis de cuellos de botella: profiling CPU (perf, flamegraphs), memoria (valgrind, heaptrack), network (iperf, netperf), disk I/O (fio, iostat).
- Optimizaciones iterativas: tuning kernel (sysctl tcp parameters), PostgreSQL (shared_buffers, work_mem), Kafka (batch.size, linger.ms).

Resultados esperados: Identificación de límites escalabilidad (e.g., "gateway soporta 800 dispositivos Thread @ 1 msg/min antes de saturar CPU"), guías de dimensionamiento hardware.

L1.2 - Edge Clustering para Alta Disponibilidad

Motivación: Gateway único es single point of failure. Deployments críticos requieren redundancia activa-activa o activa-pasiva.

Arquitectura propuesta:

- Dos gateways en configuración HA: Gateway A (primary), Gateway B (standby).
- Protocolo de elección de leader: Raft consensus (etcd, Consul) o VRRP (keepalived) para IP virtual flotante.
- Replicación de estado: PostgreSQL streaming replication (asynchronous), Redis Sentinel para failover de cache.
- Health checking cruzado: Gateways monitorean mutuamente vía heartbeat (cada 1s). Timeout 5s gatilla failover.

Desafíos: Sincronización de Thread network credentials entre gateways, gestión de split-brain scenarios, overhead de replicación en enlaces WAN lentos.

5.5.2 Línea 2 - Machine Learning Avanzado

L2.1 - Detección de Anomalías Time-Series

Objetivo: Implementar modelos ML específicos para detección de patrones anómalos en telemetría Smart Energy: theft energético, fallas de transformador, desbalance de fases.

Técnicas a explorar:

- **Autoencoders LSTM:** Red neuronal que aprende representación comprimida de series temporales normales. Reconstrucción con error >threshold indica anomalía. Ventaja: unsupervised (no requiere labeling de anomalías).
- **Isolation Forest:** Algoritmo ensemble-based que construye árboles de decisión random. Puntos anómalos son aislados con menos particiones. Ventaja: eficiente, funciona en high-dimensional space.
- **Prophet:** Modelo desarrollado por Facebook para forecasting. Detecta anomalías como desviaciones significativas de predicción. Ventaja: maneja seasonality (diaria, semanal), holidays automáticamente.

Pipeline propuesto:

1. Training en cloud con dataset histórico (6-12 meses telemetría).
2. Export modelo a formato optimizado edge (ONNX, TensorFlow Lite, CoreML).

3. Deployment en gateway como contenedor dedicado (TensorFlow Serving, Triton Inference Server).
4. Inferencia triggered por ThingsBoard rule chain ante cada batch de mensajes (e.g., cada 100 muestras o cada 5 min).
5. Alarmas generadas automáticamente ante detecciones, con explicabilidad (SHAP values, LIME).

Métricas de evaluación: Precision, Recall, F1-score en test set; False Positive Rate <1 % (crítico para evitar alarm fatigue operativo); Latencia inferencia <500 ms para batch de 100 muestras.

L2.2 - Forecasting de Generación Renovable

Objetivo: Predecir generación solar/eólica próximas 24 horas basado en: (1) Histórico de generación; (2) Datos meteorológicos (irradiancia, velocidad viento, temperatura); (3) Forecasts weather API (OpenWeatherMap, NOAA).

Arquitectura:

- Feature engineering: rolling averages (1h, 6h, 24h), lag features (generación t-1, t-24, t-168 horas), calendar features (hora del día, día de semana, mes).
- Modelo híbrido: XGBoost para captura de no-linearities + LSTM para dependencias temporales largas.
- Re-training continuo: modelo se actualiza semanalmente con nuevos datos (online learning).
- Deployment edge: inferencia cada hora, resultados persisten en TimescaleDB, visualizan en dashboard ThingsBoard como series de pronóstico vs real.

Aplicación: Gestión proactiva de storage (cargar baterías anticipando pico solar), coordinación con utility (curtailment requests ante forecast de sobre-generación), optimización económica (participation en mercados day-ahead).

5.5.3 Línea 3 - Seguridad Avanzada

L3.1 - Implementación de Blockchain para Audit Trail

Motivación: Registro inmutable de eventos críticos (comandos de control, cambios de configuración, alarmas) para compliance regulatorio y forensics post-incidente.

Arquitectura propuesta:

- Blockchain privada: Hyperledger Fabric o Ethereum privada (Proof-of-Authority consensus).
- Nodos: Gateway actúa como peer node, cloud backend como orderer + endorser.
- Smart contracts (chaincode): Lógica de validación de transacciones (e.g., comando de apertura de breaker requiere firma dual operator + supervisor).
- Storage híbrido: Hash de evento se escribe en blockchain (32 bytes), payload completo en IPFS (InterPlanetary File System) off-chain, referenciado por hash.

Desafíos: Latencia de consenso (1-5 segundos típico en Hyperledger) incompatible con control tiempo real, overhead de storage (blockchain crece monotónicamente), complejidad operacional (gestión de certificados peer nodes).

L3.2 - Zero Trust Architecture

Objetivo: Reemplazar modelo de seguridad perimetral (confianza implícita dentro de red interna) con Zero Trust (nunca confiar, siempre verificar).

Componentes clave:

- **Identity-based access:** Autenticación de dispositivos y usuarios mediante certificados X.509 + JWT tokens. Cada request incluye identidad verificable.
- **Microsegmentación:** Cada contenedor en su propia VLAN virtual (Docker networks aisladas). Comunicación inter-container vía firewall explícito (nftables rules).
- **Least privilege:** Servicios ejecutan con mínimos permisos necesarios. Ejemplo: MQTT Bridge solo puede escribir a Kafka topic telemetry, no puede leer topic commands.
- **Continuous verification:** Re-autenticación periódica (JWT refresh cada 15 min). Behavioral analytics detectan actividad anómala (e.g., súbito spike en comandos desde usuario).

Implementación práctica: Service mesh (Istio, Linkerd) para enforce políticas mTLS entre microservicios, Open Policy Agent (OPA) para autorización fine-grained basada en atributos.

5.5.4 Línea 4 - Interoperabilidad Extendida

L4.1 - Integración con Protocolos Legacy

Objetivo: Permitir coexistencia con sistemas SCADA legacy que utilizan protocolos pre-IP: Modbus RTU/TCP, DNP3, IEC 60870-5-104.

Estrategia de integración:

- Gateway dual-mode: Interfaz RS-485 para Modbus RTU (PLCs, RTUs antiguos) + Ethernet para Modbus TCP/DNP3.
- Protocol translator containerizado: Servicio que lee Modbus registers periódicamente, mapea a objetos IEEE 2030.5, publica vía MQTT.
- Mapping configuration: YAML file define correspondencia Modbus address IEEE 2030.5 resource. Ejemplo: `40001: type: voltage, phase: A, unit: V`.
- Bi-directional: No solo telemetría sino también comandos. MQTT message para trip breaker se traduce a Modbus function code 05 (Write Single Coil).

Caso de uso: Retrofit de subestación legacy con telemetría moderna sin reemplazar RTUs existentes (costo-prohibitivo).

L4.2 - Federación de Gateways

Motivación: Utility-scale deployments requieren cientos de gateways distribuidos geográficamente. Gestión centralizada desde cloud introduce latency y single point of failure.

Arquitectura peer-to-peer:

- Gateways se descubren automáticamente vía mDNS (local network) o Consul service discovery (WAN).
- Cada gateway publica capabilities: protocolos soportados, dispositivos attached, carga actual (CPU/RAM).
- Solicitudes se enrutan al gateway óptimo: comando para dispositivo X se enruta a gateway que gestiona X, load balancing para queries agregadas distribuye entre gateways con carga baja.
- Gossip protocol (Memberlist, SWIM) mantiene vista consistente de cluster membership ante fallas de nodos.

Aplicación: Microgrids interconectadas donde gateways coordinan local energy trading, islanding coordinated, black start procedures sin dependencia de cloud.

5.5.5 Línea 5 - Estándares Emergentes

L5.1 - Adopción de Matter sobre Thread

Contexto: Matter (antes Project CHIP) es estándar de interoperabilidad IoT desarrollado por CSA (Connectivity Standards Alliance) con soporte de Apple, Google, Amazon. Define application layer sobre Thread, Wi-Fi, Ethernet.

Oportunidades:

- Ecosistema device amplio: 1000+ productos Matter-certified previstos para 2025 (termostatos, switches inteligentes, sensores).
- Commissioning simplificado: QR code scanning vía smartphone + Matter controller (app iOS/Android).
- Interoperabilidad vendor-agnostic: Dispositivo Matter de fabricante A controlable por gateway de fabricante B sin custom integration.

Trabajo futuro:

- Implementar Matter controller en gateway (chip-tool open-source de CSA).
- Mapeo Matter clusters (On/Off, LevelControl, ElectricalMeasurement) a IEEE 2030.5 resources.
- Validación de latencia extremo-a-extremo Matter device → gateway → ThingsBoard.

L5.2 - Wi-Fi 7 como Evolución de HaLow

Contexto: Wi-Fi 7 (IEEE 802.11be) introduce mejoras sobre Wi-Fi 6: 320 MHz channels, 4096-QAM, Multi-Link Operation (MLO), latencia <5 ms garantizada.

Comparativa futura HaLow (802.11ah) vs Wi-Fi 7 (802.11be):

- **HaLow ventajas persistentes:** Alcance largo (sub-1 GHz penetration), consumo ultra-bajo (TWT duty cycle <0.1 %), costo módulos menor.
- **Wi-Fi 7 ventajas emergentes:** Throughput masivo (hasta 46 Gbps), latencia determinística (Triggered TWT), backward compatibility con Wi-Fi 6/5.

Estrategia híbrida: HaLow para field network (sensores, actuadores battery-powered), Wi-Fi 7 para backhaul (gateway-to-gateway, gateway-to-cloud edge) donde throughput crítico.

5.6 Impacto y Contribuciones

5.6.1 Impacto Académico

Publicaciones derivadas:

- Paper IEEE IoT Journal: "Multi-Protocol Edge Gateway Architecture for Smart Energy: Integrating Thread, HaLow and LTE"(en preparación).
- Conferencia IEEE SmartGridComm 2025: .Empirical Evaluation of IEEE 2030.5 Latency in Edge Computing Scenarios"(aceptado).
- Capítulo de libro Springer: .Edge Computing for Critical Infrastructure: A Smart Grid Perspective"(propuesto).

Formación de recurso humano:

- 2 tesis de pregrado dirigidas: (1) Ïmplementación de cliente LwM2M en ESP32-C6"; (2) .Análisis de alcance Wi-Fi HaLow en entornos urbanos".
- 1 pasantía industrial: Integración de gateway con plataforma SCADA comercial (empresa utility regional).

5.6.2 Impacto Industrial

Transferencia tecnológica:

- Repositorio open-source con 450+ stars en GitHub (6 meses post-publicación proyectado).

- Adopción por 2 utilities colombianas para pilots (300 medidores cada una, Q3 2025 inicio).
- Interés de vendors (Morse Micro, Nordic Semiconductor) para integration en reference designs comerciales.

Impacto económico estimado:

- Reducción CAPEX: Gateway propuesto $450 vs soluciones comerciales $1200-2000 (ahorro 62-77 %).
- Reducción OPEX: Costos conectividad $12/mes vs $85/mes cloud-centric (ahorro 85.9 % por gateway).
- Para deployment 500 gateways @ 10 años: ahorro total $\$(500\times(1200-450)+500\times10\times12\times(85-12))=\$375k+\$4{,}38M=$ **$4.76M**.

5.6.3 Impacto Social y Ambiental

Habilitación de transición energética:

- Arquitectura propuesta reduce barrera técnica para integración de DER, acelerando adopción de solar residencial y storage.
- Gestión optimizada de recursos energéticos distribuidos reduce emisiones CO2 mediante: (1) Mejor utilización de generación renovable; (2) Reducción de curtailment solar/eólico; (3) Peak shaving mediante storage coordinado.

Democratización de tecnología:

- Implementación open-source permite a cooperativas eléctricas rurales, municipios pequeños, y comunidades energéticas desplegar infraestructura Smart Grid sin dependencia de vendors propietarios.
- Documentación detallada + hardware COTS (Commercial Off-The-Shelf) reduce expertise requerido para deployment.

5.7 Reflexiones Finales

La presente investigación demostró que una arquitectura IoT edge bien diseñada, combinando protocolos heterogéneos (Thread, HaLow, LTE), tecnologías de containerización, y conformidad con estándares abiertos (IEEE 2030.5, ISO/IEC 30141), puede satisfacer simultáneamente requerimientos aparentemente contradictorios de sistemas Smart Energy: baja latencia Y alta disponibilidad, procesamiento inteligente Y consumo energético eficiente, interoperabilidad multi-vendor Y seguridad robusta.

El cambio de paradigma de arquitecturas cloud-centric a edge-centric no es mera optimización técnica, sino habilitador de casos de uso transformadores: control volt-VAR en tiempo real, gestión autónoma de microrredes, detección predictiva de fallas, coordinación peer-to-peer de recursos distribuidos. Estos casos de uso, a su vez, son pilares de la transición energética hacia sistemas descarbonizados, resilientes y participativos.

El trabajo futuro propuesto —escalabilidad, ML avanzado, seguridad Zero Trust, federación de gateways— no son meras extensiones incrementales, sino evolución hacia verdaderos "nervous systems"distribuidos

para infraestructura eléctrica, donde inteligencia emerge de coordinación local entre nodos autónomos, no de orquestación centralizada.

La convergencia de protocolos 6LoWPAN, plataformas edge open-source, y estándares de interoperabilidad crea, por primera vez, condiciones para ecosistemas Smart Energy genuinamente abiertos y competitivos. El presente trabajo aspira ser contribución modesta pero concreta hacia esa visión.

A 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.

A.1 Sistema Operativo: OpenWRT 23.05

A.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)

A.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
```

A.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-conntrack

# 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
```

A.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.

A.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"
```

A.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
```

A.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
```

A.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
```

A.3 Configuración de Periféricos de Conectividad

A.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
```

```
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:
```

```
# "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
```

A.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.

A.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
```

```
# 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
```

A.4 Instalación de Docker y Docker Compose

A.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-conntrack kmod-nf-conntrack-netlink

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

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

A.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!"
```

A.5 Verificación de Instalación Completa

A.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
```

A.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
```

A.6 Troubleshooting Común

A.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: ...
```

A.6.2 Problemas con Thread nRF52840

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

Solución:

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

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

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

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

A.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/
```

A.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
```

A.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, ThingsBoard Edge, IEEE 2030.5 Server, Kafka, PostgreSQL), que se detalla en el Anexo B.

B 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.

B.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
```

B.2 OpenThread Border Router (OTBR)

B.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

B.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"
```

B.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
```

B.3 ThingsBoard Edge + PostgreSQL

B.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)

B.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"
```

B.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
```

B.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)
```

B.4 IEEE 2030.5 Server (SEP 2.0)

B.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

B.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"
```

B.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"]
```

B.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
```

B.5 Apache Kafka + Zookeeper

B.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

B.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"
```

B.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
```

B.6 Bridge Thread-ThingsBoard

B.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

B.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"
```

B.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"]
```

B.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)
```

B.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
```

B.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.

C 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.

C.1 Servidor IEEE 2030.5 (SEP 2.0)

C.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>
    <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
    )
```

C.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"]
```

C.1.3 requirements.txt

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

C.2 Bridge Thread ↔ ThingsBoard Edge

C.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()
```

C.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"]
```

C.2.3 requirements_bridge.txt

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

C.3 Integración con Apache Kafka

C.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()
```

C.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()
```

C.3.3 requirements_kafka.txt

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

C.4 Scripts de Gestión

C.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 ==="
```

C.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
```

D 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.

D.1 Ejemplos XML IEEE 2030.5

D.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>
```

D.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).

D.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

D.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>
```

D.2 Configuraciones UCI para HaLow 802.11ah

D.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
```

D.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
```

D.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
```

D.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
```

D.3 Optimización TimescaleDB

D.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
```

D.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');
```

D.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;
```

D.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) * 1
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;
```

D.4 Generación de Certificados X.509 para mTLS

D.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
```

D.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
```

D.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
```

D.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
```

E 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`.

E.1 Arquitectura del Nodo

E.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 µA), light sleep (800 µA)

E.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)

E.2 Código Principal

E.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);

#if 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
}
```

E.3 Cliente LwM2M

E.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);
}
```

E.4 Objetos IPSO

E.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 cambió 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);
        }
    }
}
```

E.4.2 humidity_object.c

Implementación del objeto Humidity (3304), análogo 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...
```

E.5 Objetos LwM2M Core

E.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);
    }
}
```

E.6 Conectividad Thread

E.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;  // "J01NME"
    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));
    }
}
```

E.7 CMakeLists.txt

E.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")
```

E.8 sdkconfig.defaults

E.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
```

E.9 Uso del Nodo

E.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
```

E.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
```

E.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

F 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.

F.1 Configuraciones UCI Base

F.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'
```

F.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 'CO'
    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 s1g_long '1'
    option s1g_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'
```

F.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'
```

F.2 Firewall nftables

F.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'
```

F.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
```

F.3 OpenVPN

F.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
```

F.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"
```

F.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
```

F.4 OpenWISP

F.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: openwsp-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
```

F.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>
```

F.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
```

F.5 mwan3: Multi-WAN Failover

F.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'
```

F.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
```

F.6 Scripts de Mantenimiento

F.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
```

F.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 --oneline | 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
```

F.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 La TeX*; Universidad Nacional de Colombia, Bogota DC; 2ª 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.